Simple, Low-Cost Stereographics: VR for Everyone*

John M. Zelle
Mathematics, Computer Science, and Physics
Wartburg College
Waverly, IA 50677
john.zelle@wartburg.edu

Charles Figura
Mathematics, Computer Science, and Physics
Wartburg College
Waverly, IA 50677
charles.figura@wartburg.edu

ABSTRACT

Students are very interested in cutting-edge technologies like virtual reality (VR), and VR has many potential uses in education. However, building VR applications has proved challenging due to both cost and technical skill barriers. Through a series of experiments in "shoestring" VR, we have developed methods of bringing an important facet of VR, stereoscopic display, to our students in a simple, cost-effective way. This paper describes our approach.

Categories and Subject Descriptors: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism

General Terms: Experimentation, Human Factors

Keywords: Virtual Reality, Stereographic Display, Python

1. INTRODUCTION

Virtual reality (VR) technologies are becoming increasingly popular in diverse arenas from scientific data analysis to entertainment. An important facet of this technology is the stereographic display of computer images to produce three-dimensional visual effects. Immersive VR environments such as the CAVE [2] are exciting and incredibly useful, but have remained out of reach for most educational institutions.

We are interested in bringing aspects of virtual reality, primarily true 3D visualization, into the classroom. The possible uses of VR in education are virtually unlimited. With VR, students can explore otherwise inaccessible environments such as the surface of Mars or visualize abstractions like molecular models and magnetic fields. From the perspective of computer science education, VR offers a very attractive arena of application projects for our students to tackle.

Given what VR has to offer, one might think that everyone is doing VR, but this is not the case. The main barriers to VR have been the cost of equipment and the need for considerable technical expertise for the undertaking. We have been doing some experiments with "shoestring" VR. Somewhat to our surprise, we discovered that eye-popping stereoscopic visualizations can be produced using low-cost, off-the-shelf hardware components and programs no more complex than those written by CS1 students. The rest of this paper outlines everything you need to know to get your students started writing VR programs.

*This work was partially supported by two separate Maytag Innovation Awards for Undergraduate Research

SIGCSE'04, March 3–7, 2004, Norfolk, Virginia, USA.

2. BACKGROUND

For those who may not be familiar with stereoscopic display possibilities, we offer a bit of background. Most humans see the world through two eyes. One of the main cues that our brains use to gauge the distance to objects in our field of vision is the difference in apparent position of the object in the views obtained from the two eyes (parallax). VR technologies take advantage of this by showing each eye a slightly different view. If these two views are suitably constructed, the viewer's brain reconstructs a true three-dimensional vista. The two keys to making this work are producing the correct views for the left and right eyes and delivering each view to the appropriate eye.

One way of getting stereoscopic display is to purchase a special stereo graphics card. Externally, these cards have a connection for LCD shutter glasses. The shutter glasses alternately black out the left and right eye in quick succession. The alternation of eyes is synchronized with the display screen so that the left eye sees one image and the right sees another. Internally, stereo display is supported by having separate drawing buffers for the left and right eye. In order to support flicker-free animation, these cards provide front and back buffers for both eyes, and hence are often described as quad-buffered stereo cards. This method of providing stereo display is also called *active* stereo.

Active techniques produce excellent stereo effects and are generally used in high-end VR environments. Multi-viewer displays are possible through the use of shutter glasses synchronized by an infrared signal. Unfortunately, this is not feasible in a classroom setting, as the required projector and glasses are expensive and fragile.

As an alternative, most "mass-market" multi-viewer stereoscopic display systems employ passive stereoscopic techniques. This is the technology used in 3D movie attractions often found in amusement parks and science museums. Passive stereo works by displaying the left and right eye images simultaneously superimposed on each other. Viewers wear special filtering glasses that only allow the appropriate im-

age into each eye. The two most common passive methods are anaglyph (red-blue) and polarized.

Anaglyph stereo uses color to filter the two eye images. Typically, the left eye image is projected in red and the right eye is either blue or cyan. Viewers wear glasses having a red filter over the left eye and a blue filter over the right, so that each eye only sees the intended image. Anaglyph stereo is very easy to produce with no special equipment and can be viewed on a computer monitor or from a projected image using inexpensive cardboard glasses. The drawback is that the scene is perceived in gray-scale; the original color is lost. Also, since the eyes see the scene in competing colors, there can be considerable viewing fatigue. If red and cyan filters are used, there are techniques for adding some color back to the images, but this cannot be done in a completely general way.

A better stereo effect can be achieved using the polarization method. In this approach, the left and right images are projected through filters that polarize the two views perpendicular to each other (generally 45 and 135 degrees). The superimposed images are then viewed through inexpensive glasses with appropriate polarizing "lenses." This approach produces full color images and excellent stereo effects. The downside is that it requires the images to be projected (not viewed on a monitor).

3. SIMPLE PASSIVE STEREO

Our VR scheme uses the passive polarization technique. The setup is very simple.We have put together a computer cart containing a single CPU and two LCD projectors driven by a dual-head graphics card. Our projector resolution is 1024x768, and the dual-head display is configured to provide one 2048x768 desktop. When a window is maximized across this desktop, the left-half displays through one projector and the right-half through the other. We place an external polarizer in front of each projector and view the result through standard (polarized) 3D glasses. Projecting a stereo image is then a simple matter of placing an appropriate view in each half of the window.

A number of groups have independently developed similar approaches [1, 3]. Paul Bourke's stereography website [1] provides a wealth of good information and advice concerning passive stereo display, but along the way we have discovered some ways of simplifying the "textbook" approach.

3.1 Projector Type

A polarized passive stereo setup relies on separate projectors for left and right eye images. Although there have been tremendous advances in data projector technology in the last few years, good projectors are still relatively expensive. Since one of the goals of our project is to keep costs low, we were not excited about purchasing special projectors to dedicate to our VR cart. Since most schools and businesses already own portable data projectors, we thought it would be more cost effective to design a setup that could use existing projectors in a non-dedicated way. The idea is that, when you want to do VR, you just grab two portable projectors, drop them on the cart and go.

One potential problem with this approach is that the vast majority of current data projectors use LCD technology, which itself relies on polarization properties of light. As a result, LCD projectors emit light that is already polarized. The conventional wisdom is that this type of projector should be avoided in a passive stereo setup. A better alternative would be to use DLP projectors, which do not emit polarized light. Unfortunately we did not have any DLP projectors on hand.

We decided to try some experiments with our LCD projectors to see exactly how the polarization would affect us. At one point we even speculated that we might be able to use the inherent polarization to our advantage by not having to rely on external polarizers. When we tested our LCD projectors (various models of Sharp LCD projectors) we found that the RGB components were polarized differently. Green was at 90 degrees, while red and blue were at 180. While this nixed our plan to avoid external polarizers, it meant that using external polarizers with the standard orientations 45-135 would be possible, albeit with a loss in brightness.

Our subsequent experience has demonstrated that with modern LCD projectors, the loss in brightness from external polarization is not really a problem. Even relatively inexpensive projectors are now bright enough to be used in a lit room. Despite the loss, our VR visualizations are still bright enough to be easily viewed throughout a 40 student classroom by dimming the lights.

Our initial experiments were conducted with projectors having a maximum resolution of 800x600, and we found this resolution quite adequate for effective 3D displays. With the continual increase in modern display resolutions, it's possible that your institution may be phasing out older projectors that would be acceptable for a dedicated VR projection cart. As noted above, we are currently using non-dedicated 1024x768 projectors with very good success.

3.2 Projector Placement

Projector placement involves some design considerations. For best results, the two projectors should either be stacked or placed next to each other with parallel lines of projection to avoid any differential keystoning that would interfere with the proper alignment of the images. Typically, the projectors are stacked vertically using some sort of rigid custom designed framing. Some higher-end projectors have lens-shifting ability that allows the two images to be perfectly aligned while maintaining parallel projection. This is the optimal approach.

Another possibility for projectors with built-in keystone adjustment is to align the projection centers of the two images and then use the vertical keystone adjustment to align the edges of the images. A third approach is to simply keep the projectors parallel and only use the overlapping region for a cinemascope effect. In this case, the viewports of the images need to be adjusted in the viewing software so that they align properly.

Ultimately, none of the above schemes seemed most appropriate for our situation for two reasons. First our goal was to throw together an experimental setup quickly and with a minimum of fuss. Having to build a stacking apparatus was just another barrier. Second, since our projectors are not dedicated to the cart, sliding them in and out of a custom frame and having to deal with constant realignment did not seem appealing.

We decided to simply set the two projectors side-by-side and align their centers for maximum overlap. Our initial thought was that we could then build horizontal keystone correction directly into our VR software. As it turns out, we have not bothered with this because our experiments have

shown it unnecessary for our run-of-the-mill visualization needs. We were frankly surprised to find that the human visual mechanism apparently has considerable tolerance for noise in the alignment of the left and right eye images, at least when that noise is small and systematic as in the case of our horizontal keystoning distortions.

Our projectors have a built-in test pattern that can be turned on for adjustment of vertical keystoning. When we set the projectors on the cart, we simply turn on the test pattern and align the centers of the grid as carefully as possible so that images near the center are almost perfectly aligned. Horizontal keystoning distortion in this "sweet spot" is nominal but grows toward the corners of the display. If your projectors support zooming (ours do), the amount of keystone distortion can be reduced to some extent by moving the projectors farther from the screen and then reducing the projection size to fit. This reduces the amount of "toe-in" needed to align the centers, hence reducing keystoning.

Even being fairly cavalier about the projector alignment, we have not found the keystone distortion to be distracting. Part of this is because our visualizations tend to be centered in the frame, thus projecting into the area of least distortion. When we have full-width images, we use a cinemascope aspect ratio so the scene goes edge-to-edge but avoids pressing into the corners of the display. Using this technique full-width images such as 3D pictures from the Mars lander look great out to the very edge, and there is no discernible distortion or ill-effect from the slight keystoning.

3.3 Screen Considerations

For polarized passive stereo it is necessary to project onto a screen that preserves the polarization of light. Standard white projection screens will not work. What is needed is one of the higher quality "silver" screens. Portable silver screens are available from various 3D supply houses (e.g., Reel3D), however they tend to be more expensive than regular screens.We have a portable silver screen for use with our VR cart, but we have also developed a simple, inexpensive alternative.

Ordinary metallic spray paint (aluminum- or zinc- based) can be used to create a nice polarization preserving projection surface. This paint can be sprayed on cardboard for a very light screen or on a sheet of masonite or plywood for use in a more permanent setting. The current VR lab where our cart is stored employs a spray-painted masonite screen (total cost around $10). Before that, we used a room where we had simply spray-painted a rectangular region on the wall to serve as our screen. Both of these approaches gave results that equaled or exceeded the quality of picture on our portable silver screen.

3.4 Graphics Cards

The only special consideration for the graphics card is that it be a dual-head card to provide output to the two projectors. We initially used a Matrox G450 which can be purchased for less than $100. This worked very well, but we have since switched to an NVIDIA card because it has better accelerated OpenGL support under Linux. The main consideration in choosing a graphics card is driver support. After that, you can spend more for accelerated performance. We want to emphasize, however, there is no need to invest in one of the high-end quad-buffered stereo cards. The passive setup does not require this specialized stereo machinery.

3.5 3D Glasses and Polarizers

Polarized 3D viewing glasses can be purchased from scientific supply firms such as Edmund Optics or 3D outfitters such as Reel3D. Cardboard glasses with the standard 45/135 degree polarizers are available for as little as $0.40 per pair even in small lots.

The polarization of the left and right eye images from the projectors to match the glasses is accomplished with external polarizers. For our initial experiments we used inexpensive polarizing polymer sheets similar to those used in 3D glasses. Such polarizers are common equipment around physics and optics labs and can be purchased from scientific supply or 3D outfitters. The latter provide pre-cut square polarizers for around $30 a pair that are oriented in the standard 45 degree inclination, which makes them very convenient for aligning with the glasses.

We initially mounted our polarizers in simple cylindrical collars made of light cardboard that fit over the projector lens. This was cheap and easy, but we ultimately rejected this approach because the projector lenses turned when the projectors were focused and it was awkward to set up the projectors, put on the filters and get the polarizers properly aligned with the glasses. A better approach is to affix the polarizers directly to a board on the projection cart so that they are always in place and properly oriented. A a bare-bones approach is to simply put a lump of putty in front of each projector and stand the polarizer in it.

One problem that we encountered with the polymer film polarizers is that prolonged exposure to the projection beam resulted in thermal damage (warping and bubbling) to the film. This produced distortion that grew severe after tens of hours of use, and the polarizers would have to be replaced. A more durable solution was found by using glass-sandwich style polarizers. Large aperture (50mm) flat-spectrum polarizers are available from Edmund Scientific for under $35 apiece. Mounting hardware (holders and posts) for the polarizers cost about $70 per projector,resulting in a total optics cost around $200.

3.6 Hardware Summary

The bottom-line is that it takes very little investment in hardware to build an effective passive-stereo VR cart. If you have a suitable pair of projectors and a computer available, the necessary extras (glasses, polarizers, dual-head card and screen) can be purchased for less than $150 for a bare-bones experimental setup. Using higher-quality optics approximately doubles that figure. Hardware costs need not be a barrier to bringing VR into your classroom.

4. BUILDING VR APPLICATIONS

Having the equipment for stereo display is half the battle, but you also need to acquire or develop stereoscopic applications. Even a special stereo graphics card cannot automatically generate a stereo view from a standard 3D application. The software must be designed to generate and display the appropriate left and right eye views. Many existing packages designed for visualization include an option for stereo, but usually this is a mode that exploits a quad-buffered stereo card with shutter glasses and would not be useful for our passive display. Of course one of the main reasons we put our cart together was to provide our students with a platform for developing VR applications. So we were eager to start rolling our own software.

4.1 Choosing a Language

Generally, graphics applications are developed using languages like C/C++ that are very efficient and suitable for hardware-level manipulations. However, the trend toward hardware-accelerated 3D support in commodity-grade graphics cards opens up other options. Very high level languages can be used to construct applications with the heavy calculations carried out at the hardware level (or at least with calls to pre-compiled C functions).

In our experiments, we have had very good success using Python [7]. Python is a very high-level, interpreted language that is perfect for quick experimentation and rapid prototyping. A number of freely available contributed modules for Python make it a wonderful language for VR. The PyOpenGL package [6] allows access to the underlying graphics hardware for fast 3D rendering, the imaging library [8] is useful for manipulation and display of paired stereo images, while the Numeric package [4] provides high-performance numerical routines and matrix operations.

Python was a natural choice for our students, because it is also the language that we use in CS1. Our point here is not that everyone should use Python (that's another paper), but simply that your students don't have to be C++ gurus to write effective VR demonstrations.

4.2 Generating Stereo Pairs

For students with some background in computer graphics, generating stereo views should be relatively easy, but there are a couple subtleties that need to be taken into account to do the job well. Paul Bourke's web page [1] contains lucid tutorials on how to generate good stereo views. Here we will just present some highlights.

Obviously, the main idea of stereo viewing is to generate two views of a scene, one from each eye position. Generally, the position of the eyes is specified in terms of an eye-separation parameter. Any program that produces images from a 3D model can be used to generate static stereo pairs. For example, one could use the popular ray-tracing package POV-Ray [5] and generate two views of a scene from camera positions adjusted according to the eye-separation value.

Unfortunately, just pointing the viewing camera at the same point in a scene from two different positions (the "toe-in" approach) does not generate the best stereo effect. The problem is that the viewing plane onto which the 3D view is projected is usually assumed to be orthogonal to the view direction. Viewing toward the same center point from two different positions produces images that are projected into two different planes. Figure 1 depicts the situation.

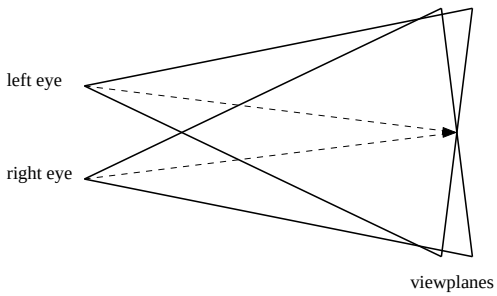

Figure 1: Viewing a point in a scene from two different camera positions produces differing view planes.

A better way to generate stereo pairs is to have the two eyes look at points in the scene that are eye-separation distance apart. Using this method, the view-direction vectors

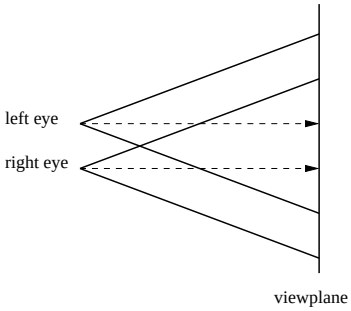

Figure 2: Using parallel (symmetric) views produces a single view plane, but images must be trimmed to area of overlap.

are parallel, and hence, the images lie in the same viewplane. However when the rendering application is not stereo aware, the resulting images must be subsequently "trimmed" to the area of actual view overlap. See Figure 2.

4.3 Using OpenGL

A lower-level approach is to actually write stereo rendering code using a graphics API such as OpenGL. OpenGL does not restrict the viewing frustum to be symmetric, so it is relatively easy to generate view-parallel left and right eye images as shown in Figure 3.

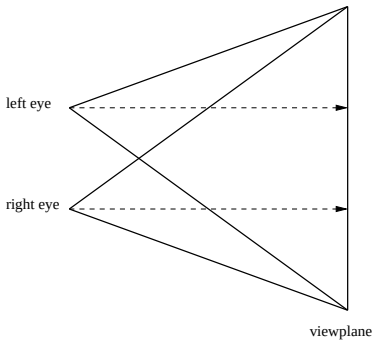

Figure 3: The "correct" approach using parallel views and asymmetric view frustums produces a single viewplane and overlapped image.

The OpenGL function `glFrustum(left, right, bottom, top, near, far)` can be used to set an appropriate projection matrix for a standard perspective projection. In OpenGL, the camera is placed on the positive z axis looking in the negative z direction. In this call, `left` and `right` specify the minimum and maximum x coordinates for the view, and `bottom` and top do the same for y. The values of `near` and `far` are the distance to the near and far clipping planes, respectively.

Generally, the code to compute a perspective projection with a symmetric frustum looks something like this:

```
top = near * tan(fov/2.0)
right = aspect * top
glFrustum(-right, right, -top, top, near, far)
```

In this example `fov` specifies the field of vision (in radians) and `aspect` is the aspect ratio of the display.

Modifying this code for stereo is simply a matter of moving the left and right boundaries in a way appropriate for the given eye:

```
off = eyeSeparation / 2.0 * (near / focalLength)
glFrustum(-right+off, right+off, -top, top, near, far)
```

The `focalLength` variable specifies how far away the viewing plane is. Objects that are closer than `focalLength` will appear to float in front of the screen while those farther away will appear to be behind the screen.

All that remains is to ensure that the model-view transformation is set up so that the eyes look at appropriately offset centers. Here is one approach using the `gluLookAt` function.

```
viewpoint = viewpoint - right*eyeOffset
center = center - right*eyeOffset
gluLookAt(viewpoint[X], viewpoint[Y], viewpoint[Z],
          center[X], center[Y], center[Z],
          up[X], up[Y], up[Z])
```

In this code `viewpoint` is the vector specifying where the camera sits, `center` is the point the camera is looking at, and `right` is a unit vector specifying the direction going right from the camera. This code simply shifts both the viewpoint and the center of the scene and then uses the `gluLookAt` call to set the model-view transformation.

4.4 Stereo VR Without Graphics

Of course, many of our students do not have graphics programming experience, and we wanted a way for them to also get involved with VR applications. To this end we were very interested in the VPython visualization package [10]. VPython is a Python extension that makes 3D modeling simple enough to be included in CS1 [9].

VPython provides a basic set of 3D modeling objects like sphere, cone, box etc. that a program can create and manipulate. For example, the following (slightly modified) code from a VPython demo draws a red ball bouncing up and down on a blue floor.

```
from visual import *
floor = box(length=4, height=0.5,
            width=4, color=color.blue)
ball = sphere(pos=(0,4,0), color=color.red)
ball.velocity = vector(0,-1,0)
dt = 0.01
while True:
    ball.pos = ball.pos + ball.velocity*dt
    if ball.y < 1: ball.velocity.y = -ball.velocity.y
    else: ball.velocity.y = ball.velocity.y - 9.8*dt
```

When the program is run, VPython pops up a window and displays the scene of a ball bouncing up and down. The user is able to zoom in and out and rotate the scene using the mouse. Notice that this is all accomplished without having to know anything at all about graphics. VPython frees up programmers to write simulations, while it takes care of the visualization task.

VPython is written as a C++ extension for Python that uses OpenGL as its rendering engine. When we started using it, it contained a rudimentary mode for active stereo cards. One of the authors modified this package to support both active and passive stereo viewing modes, and these changes have now been incorporated into the VPython distribution.

Stereo viewing is controlled in VPython through two parameters: `stereo` can be set to either `"active"` or `"passive"`, and `stereodepth` is used to set the scaled focal length. By default, `stereodepth` is 0.0, and the entire scene appears behind the screen. Setting the depth to 1.0 puts the center of the scene at the screen boundary and setting the value to 2.0 puts the scene entirely in front of the screen. Values in-between can also be used. Changing the bouncing ball demonstration for use on our VR cart is as simple as adding just a couple lines of code at the top of the program:

```
scene.stereo='passive'
scene.fullscreen=True
```

VR doesn't get much easier than this!

5. CONCLUSION

We have demonstrated a simple, low-cost technique for bringing VR involving stereoscopic display into the classroom. With a combination of inexpensive hardware and simple-to-use software you and your students can also enjoy the excitement of programming eye-popping 3D visualizations.

6. REFERENCES

[1] P. Bourke. Stereographics. http://astronomy.swin.edu.au/∼pbourke/stereographics/ (accessed 11 September 2003).

[2] C. Cruz-Neira, D. Sandin, T. DeFanti, R. Kenyon, and J. Hart. The CAVE: Audiovisual experience automatic virtual environment. *Communications of the ACM*, 35(6):67–72, June 1992.

[3] S. T. Jones, S. E. Parker, and C. C. Kim. Low-cost high-performance scientific visualization. *Computing in Science and Engineering*, 3(4):12–17, July/August 2001.

[4] Numeric Python. http://numpy.sourceforge.net.

[5] POV-Ray the persistence of vision raytracer. http://www.povray.org.

[6] PyOpenGL. http://pyopengl.sourceforge.net.

[7] Python programming language. http://www.python.org.

[8] Python imaging library. http://www.pythonware.com/products/pil.

[9] C. Shannon. Another breadth-first approach to cs 1 using python. In *Thirty-fourth SIGCSE Technical Symposium on Computer Science Education*, pages 248–251, February 2003.

[10] VPython 3D programming for ordinary mortals. http://www.vpython.org.