Using Texture Maps to Correct for Optical Distortion in Head-Mounted Displays

Benjamin A. Watson
Larry F. Hodges
Graphics, Visualization & Usability Center
College of Computing, Georgia Inst. Technology
Atlanta, GA 30332 USA
tel: +1-404-894-8787 fax: +1-404-853-0673
watsonb@cc.gatech.edu

ABSTRACT

This paper describes a fast method of correcting for optical distortion in head-mounted displays (HMDs). Since the distorted display surface in an HMD is not rectilinear, the shape and location of the graphics window used with the display must be chosen carefully, and some corrections made to the predistortion model. A distortion correction might be performed with optics that reverse the distortion caused by HMD lenses, but such optics can be expensive and offer a correction for only one specific HMD. Integer incremental methods or a lookup table might be used to calculate the correction, but an I/O bottleneck makes this impractical in software. Instead, a texture map may be defined that approximates the required optical correction. Recent equipment advances allow undistorted images to be input into texture mapping hardware at interactive rates. Built in filtering handles predistortion aliasing artifacts.

1. INTRODUCTION

The display screens in head-mounted-displays (HMDs) are placed within inches of a wearer's eyes. Between the wearer's eyes and the screens is an optical lens system. The purpose of the lens system is to provide an image to the user that is located at a comfortable distance for accommodation and that is magnified to provide a reasonable field-of view. Unfortunately, the optics also cause nonlinear distortions in the image so that straight lines in the model appear curved in the visual image. The most significant component of this distortion in the widely used LEEP optics is radial distortion [7, 9], which stretches the image away from the lens center. The farther an image element from this center, the more it is stretched. Image elements at the center remain unaffected. Figures 1 and 2 illustrate the effect of radial distortion.

Distortion introduces other complications as well. Because the edges of the HMD raster itself are distorted, using all of the available display space would require clipping to a curved graphics window. A slight misregistration is introduced into stereo pairs [11]. Also, since distortion changes the perceived shape and size of raster pixels, distortion can introduce a perceived blur, and complicate antialiasing. Distortion also has some beneficial side effects. Distorting the image increases field-of-view (FOV) slightly. Furthermore, because the optics do not distort the center of the image, this FOV increase does not come at the expense of central image detail, where viewer attention is usually focused. However, most of an HMD's optical FOV increase is attributable to optical magnification, not distortion.

Robinett and Rolland [14] have described an optical model for HMDs. Using this model, they were able to suggest a method of correcting for HMD distortion with a predistortion. Later, their colleague Gary Bishop implemented this method on Pixel Planes hardware [6]. Predistorting a stereo image frame took roughly 1/20 of a second Essentially, their suggested approach began with a normally generated raster image, and relocated the individual

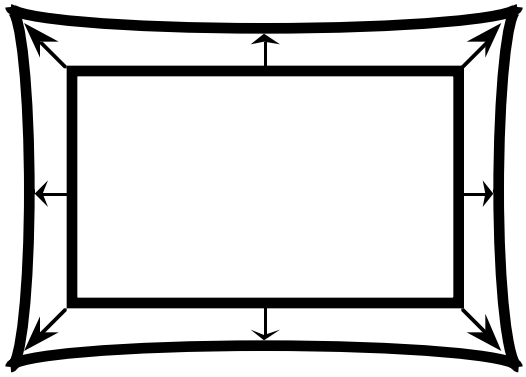

Figure 1: Radial distortion expands the extent of the display as well as the image it contains.

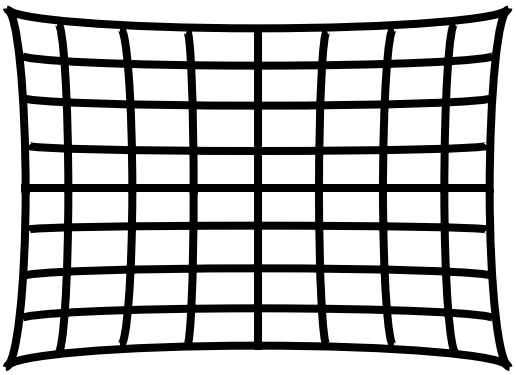

Figure 2: Severity of distortion increases with distance from the optical axis. This example shows a distorted grid.

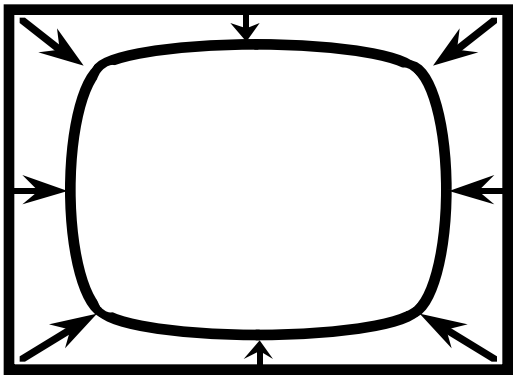

Figure 3: Predistortion contracts the image in proportion to distance from the optical axis.

pixels in the raster*. The effect of such a predistortion is illustrated in Figure 3. When this predistorted image is viewed through HMD optics, as illustrated in Figure 4, it appears to be undistorted. Note that the raster itself is still distorted.

In this paper, we will describe a real-time method of predistortion that uses the model described by Robinett & Rolland. Our goal with this research was to find a real time method for predistortion that works on standard graphics hardware.

2. WHY BOTHER CORRECTING DISTORTION?

In Sutherland's early HMD work [16, 17], distortion was not considered a pressing issue. More important was the realization of any sort of device that approximated "the ultimate display." Even later researchers such as Bryson [3], after some experiments, have concluded that compensating for distortion takes too much time, and have moved on to other research issues. Psychophysical studies have shown that the human visual system can correct for displayed distortion [20]. Yet it may not be capable of this correction when the distorting display, like most HMDs, is only intermittently used. In addition, it may be that distortion has a significant effect on the performance of HMD users, especially for tasks that require heavy use of the peripheral areas of the FOV, where distortion is most severe. Efficient correction for distortion is an important step in achieving the proper experimental control for investigating these and other questions.

Early HMDs had horizontal binocular FOVs ranging to well over 100 degrees. But they paid a price for this immersive width: severe distortion and poor resolution. Indeed, the level of visual acuity provided by many of these HMDs fits the legal definition of blindness [3]. Recently, commercial HMD manufacturers have begun to sacrifice FOV in order to achieve greater resolutions and reduced distortion. Some new HMDs have FOVs in the range of 40 degrees. Manufacturers may now have crossed a perceptual breaking point, one at which the perceptual gain of increasing resolution is offset by the loss in FOV

* This information was obtained through the author's professional correspondence with Gary Bishop of the University of North Carolina.

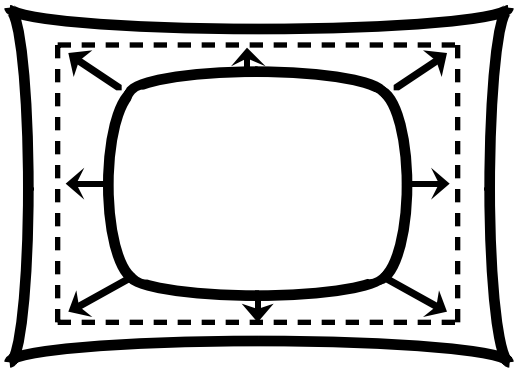

Figure 4: The predistorted image is expanded by the distorting lenses, and fills the undistorted original image's extent.

[1, 21]. If an efficient algorithmic method of eliminating distortion could be found, HMD designers could be freed from concern about the distortion that accompanies wide FOV, and could concentrate on optimizing other lens parameters. Sharp in Japan is planning to develop a wide FOV HMD, and has shown interest in just such an approach [19].

Perhaps the most important use for an efficient distortion correction would be in the arena of augmented reality, in which virtual objects are superimposed onto a real world view [2, 5]. Many feel that this largely untapped research area could give rise to extremely promising training and visualization applications. One of the most difficult challenges facing augmented reality researchers is the accurate, real-time placement of virtual objects onto the real world view, commonly called the registration problem. A significant component of this problem is the static misregistration introduced by optical distortion. Techniques for eliminating this distortion could have a powerful impact in this field.

3. MAKING MAXIMUM USE OF DISPLAY AREA

Before an image can be predistorted, a graphics window must be defined, and the view of the modelled world through this window generated. To maintain psychophysical continuity and make maximum use of available display area, the boundary of this graphics window should match the boundary of the display viewport. However, because distorted HMD screens are not rectangular, achieving this continuity with HMDs is more complicated than with traditional CRT displays. We defined the graphics window as the largest rectangle that fit inside the outline of the distorted screen.

In order to fill this graphics window, it was necessary to make adjustments to the distortion correction proposed by Robinett and Rolland. That correction is an exact reversal of the modelled distortion, resulting in a graphics window that has the same bounds as the undistorted display. However, since the undistorted display is smaller than the distorted display, this correction does not make good use of the available display space (see again Figure 4) or fill the graphics window we defined. Our adjustments increased the size of the predistorted image so that, when viewed through

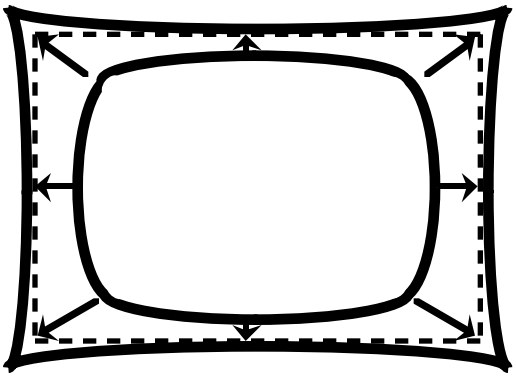

Figure 5: Adjusting the predistorting correction allows the entire distorted display extent to be utilized.

the HMD lenses, it filled our graphics window (Figure 5). We outline these adjustments below.

The boundaries of our graphics window depend on the HMD's distortion, modelled by these equations:

$$x_{vn} = x_{sn} + k_{vs}x_{sn}(x_{sn}^2 + y_{sn}^2)$$
$$y_{vn} = y_{sn} + k_{vs}y_{sn}(x_{sn}^2 + y_{sn}^2).$$

In the above notation, the distortion constant k_{vs} is a parameter of the optics being used. (x_{sn},y_{sn}) locates the pixel on the HMD raster, while (x_{vn},y_{vn}) locates the pixel location after optical distortion. For both of these coordinate systems, the origin is located at the optical axis of the HMD. If the normalized right boundary of the undistorted HMD screen is R/w_s, then the right edge of our graphics window has the distorted boundary

$$R_d = R/w_s + k_{vs}(R/w_s)^3.$$

Calculations of the distorted boundaries L_d, T_d and B_d of the other sides of the graphics window are similar. The resulting window is larger than the undistorted HMD screen, and may have a different aspect ratio from the undistorted screen. To take account of this new aspect ratio in our correction, we renormalized the graphics window boundaries by dividing each of L_d, R_d, T_d and B_d by $\max(L_d,R_d,T_d,B_d)$, and substituting them into the original model in place of the undistorted window boundaries L/w_s, R/w_s, T/w_s and B/w_s.

The severity of radial distortion increases with distance from the optical axis. Because our graphics window's boundaries are farther from the optical axis of the HMD than the boundaries of the undistorted HMD screen, distortion within our graphics window is more severe than distortion within the undistorted display screen. To take account of this increased distortion, we increased the magnitudes of the distortion constant k_{vs} and the predistortion constant k_{sv} slightly.

4. APPROACHES TO DISTORTION CORRECTION

One very straightforward approach to correcting for optical distortion is optical rather than digital. Before a normally

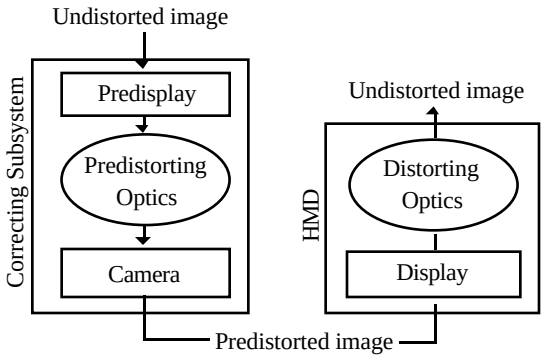

Figure 6: A hypothetical correcting subsystem would pass the undistorted image through a correcting lens before display on an HMD.

generated image is presented on a distorting display, it can be passed through a set of lenses that reverses the distortion introduced by the display. One could, for example, place a video camera with correcting lenses in front of a computer display, and send the output of the camera to an HMD. This method does not introduce any latency into the rendering pipeline* and skirts many of the filtering problems introduced by digital predistortion. However, this approach does have its drawbacks. New correcting lenses are required for each new distorting lens. The camera setup described is not very stable, and if correcting lenses were incorporated into HMD displays, they could increase the bulkiness and expense of a display that already has questionable ergonomics. Nevertheless, an HMD system with a separate predisplay and correction component (see Figure 6) might prove useful. See [4] for a description of a similar system.

A digital predistortion of an image may be performed with the use of the following equations, which approximate the continuous relocation performed by optical predistortion:

$$x_{sn} = x_{vn} + k_{sv}x_{vn}(x_{vn}^2 + y_{vn}^2) \quad (1)$$
$$y_{sn} = y_{vn} + k_{sv}y_{vn}(x_{vn}^2 + y_{vn}^2), \quad (2)$$

where k_{sv} is the predistortion constant.

Since the equations (1) and (2) are essentially parametric cubics, the predistorting relocations might be calculated dynamically and efficiently using well-known curve rasterization methods [10, 18]. For example, after perspective projection, polygon vertices could be predistorted and their edges rendered as predistorted curves. If the equation for the line defined by such an edge is

$$y_{vn} = cx_{vn} + d, \quad (3)$$

the parametric equation that defines its predistorted equivalent is obtained by substituting equation 3 into equations 1 and 2:

*Note that most modern cameras integrate light for 1/30 of a second, acting as a frame store.

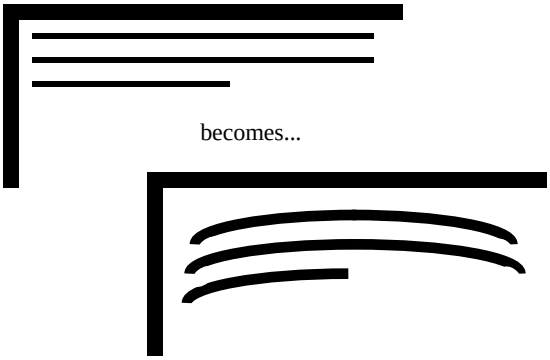

Figure 7: Undistorted scan lines might be predistorted using integer-only rasterization methods.

$$x_{sn} = (k_{sv}c^2+k_{sv})x_{vn}^3 + (2k_{sv}cd)x_{vn}^2 + (k_{sv}d^2+1)x_{vn}. \quad (4)$$

$$y_{sn} = (k_{sv}c^3+k_{sv}c)x_{vn}^3 + (3k_{sv}c^2d+k_{sv}d)x_{vn}^2 + (3ck_{sv}d^2+c)x_{vn} + (k_{sv}d^3+d). \quad (5)$$

For accuracy, polygon fill and shading routines used with this method would have to be adjusted to account for predistortion. Because this approach would have a variable complexity dependent on world geometry, we call it the geometry-dependent approach.

Rolland and Hopkins [13] are experimenting with a algorithm related to the geometry-dependent approach that uses a lookup table to predistort vertices. Rather than rendering predistorted edges, however, they minimize the edge distortion by subdividing large screen polygons and predistorting the new vertices.

Alternatively, after an undistorted image was rendered, each image scan line could be rendered as a predistorted curve (Figure 7). Because this method would have a fixed complexity dependent on image size, we call it the image-dependent approach. The predistortion of the zero-slope scan lines corrected in this approach is an especially simple case, making this image-dependent method easy to implement in hardware. By setting the slope c to 0 in equations 4 and 5, we arrive at these much simpler equations:

$$x_{sn} = k_{sv}x_{vn}^3 + (k_{sv}d^2+1)x_{vn} \quad (6)$$

$$y_{sn} = k_{sv}dx_{vn}^2 + (k_{sv}d^3+d). \quad (7)$$

Predistortion for both the geometry- and image-dependent approaches could be adjusted for different HMDs by simply passing new screen boundary and distortion severity parameters to the algorithm.

Finally, since the predistortion for the HMD being used is usually well-known, it might be more efficient to precalculate the relocation of each undistorted pixel into a table, if sufficient memory is available. Regan and Pose have implemented such an approach in hardware [12]. Their specialized architecture incorporates a pixel relocation lookup table that allows any raster image to be predistorted to correct for distortion in any sort of lens.

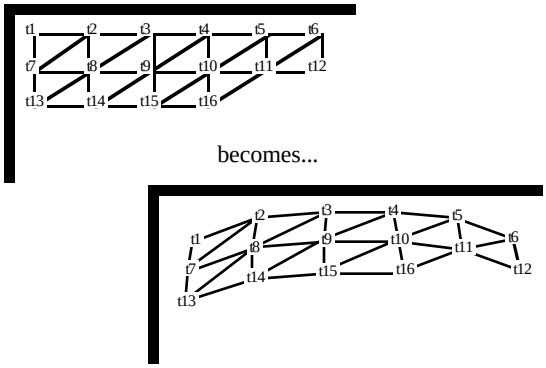

Figure 8: The vertices in a rectangular mesh textured with an undistorted image may be predistorted with equations 1 and 2. Since the texture indexes are unchanged, this predistorts the undistorted image.

5. EXPERIMENTS WITH TWO OF THESE APPROACHES

We implemented both the table-based approach and the dynamic image-dependent approach in software on a Silicon Graphics Crimson Reality Engine, and tested them with 320 x 240 images. The predistorted scan lines generated by the image-dependent method had very small slopes over most of their length, and as a result the predistorting relocation of successive scan line pixels exhibited a significant amount of coherency. We were able to take advantage of this fact to reduce the amount of memory used to store relocations in our implementation of the table-based method to 25K.

Surprisingly, the speed of these two approaches did not differ greatly. With each of these approaches we were able to predistort six graphics window frames per second, clearly inadequate for real time graphics applications. Testing showed that most CPU time was spent transferring frames between frame buffer and main memory. This explained the lack of any significant difference in speed between the two approaches, and suggested that hardware implementations of these algorithms could be quite effective.

All of the digital predistortion approaches discussed here introduce image filtering complications that worsened with the severity of the modelled distortion. We experimented with pixel by pixel blending on the predistorted image as a solution and found it inadequate. The resulting images had a snowy, speckled look. We obtained better results when we used the Z-buffer value of the undistorted image pixels to resolve relocation pixel conflicts. While we did not encounter this problem in our experimentation, this Z-buffering approach could conceivably result in binocular rivalry (right/left eye conflicts) in stereoscopic displays. Both of these filtering solutions were simple software approaches dictated by our real-time demands. A more complete solution was implemented by Regan and Pose in their specialized graphics architecture, which passes the image through a linear filter after pixel relocation.

6. PREDISTORTING WITH TEXTURE MAPS

Predistortion can be viewed as a special case of texture mapping [8]. If the undistorted image is a texture, and the

TABLE 1:
Predistorted Performance

Performance of predistorting algorithm with textured and untextured world views, and with differing undistorted source image resolutions. Predistorted views were always rendered monoscopically at 640 x 480 resolution. Reported predistortions per second are not geometry or world dependent and are highly variable. Reported frame rates include time required for rendering of undistorted source view, and so are dependent on the size of the rendered world, which contained 8211 flat shaded polygons. When texturing was used, 1588 of these polygons were mapped with 128 x 128 textures. The world contained five different textures. The predistorted view was a textured mesh with 400 vertices, filtered bilinearly. All textures were stored in a two byte internal format.

Textures?	Yes			No		
Source Res	640x480	512x256	256x256	640x480	512x256	256x256
Frames/Sec	10	14	19	11	14	19
Predist/Sec	17	20	40	19	23	77

TABLE 2:
Normal Performance

Rendering performance for an undistorted virtual world rendered monoscopically at 640 x 480 resolution. The textured and untextured worlds used for these tests were the same worlds used for the tests reported in table 1.

Textures?	Yes	No
Frames/Sec	20	25

display surface a polygon, predistortion is simply a mapping of a texture onto a polygon. One salient difference between predistortion and traditional texture mapping is that unlike most textured polygons, the display surface in predistortion must have a new undistorted image mapped onto it for each new image frame. Nevertheless, this analogy provides a useful perspective on several of the problems we discussed earlier, and suggests a new approach to performing predistortion.

We can model the undistorted image space with a flat, two-dimensional polygonal mesh. Paired with each vertex in the mesh is an index into the corresponding point in the undistorted image (the texture) itself . We then apply the predistorting equations (1) and (2) to the vertices of the mesh, leaving the texture indexes unchanged. Figure 8 illustrates this process. If the mesh is then rendered onto the display screen, a predistorted image will appear.

Until recently, it was not possible to load new textures into texturing hardware at interactive rates, and as a result, undistorted images could not be predistorted with this technique in real time. But a new and largely undocumented feature on high-end Silicon Graphics machines [15] has made this predistortion approach practical. Undistorted images can be rendered to the frame buffer and addressed directly by texturing hardware.

We have implemented this technique on a Silicon Graphics Onyx Reality Engine II, and tested it on a virtual world containing 8211 flat shaded polygons. When a textured view of the world was rendered, five 128 x 128 textures were mapped onto 1588 of these polygons. The predistorted mesh contained 400 vertices, which in our experience struck a good balance between over-sampling and over-interpolating. All textures were stored with a two byte internal format. Our results are displayed in tables 1 and 2. Our scan converter required 640 x 480 image input, and so the undistorted world was always rendered at 640 x 480 resolution. Figure 9 shows a textured and undistorted world view. While the predistorted view of the world rendered with predistortion was always rendered with 640 x 480 resolution, we varied the resolution of the source undistorted image at three levels: 640 x 480, 512 x 256, and 256 x 256. Figure 10 shows a predistorted image rendered with Figure 9 as its source texture. Figures 11 and 12 show an undistorted image rendered at 256 x 256 resolution and a matching predistorted view.

Not surprisingly, rendering an undistorted view of the world was faster than rendering a predistorted view of the world. Note that adding textures to the predistorted world had little impact on frame rate. Adding textures to the undistorted world, however, significantly impacted performance. This indicates that a large part of predistortion's impact on performance is due to its addition of texturing to an otherwise untextured world. By reducing the size of source textures, one may trade image fidelity for significant improvements in performance. This would be particularly appropriate when the resolution of the displays in the HMD being used is less than NTSC.

Filtering has long been an important concern of texture mapping researchers, and accordingly most texturing systems provide many different filtering options, at various levels of image quality and speed. We found that use of larger and more complex filters improved predistorted image quality without significantly impacting overall performance.

When considering the quality and accuracy of the images produced with this approach, it is important to realize that the equations 1 and 2, which form the basis for all the techniques discussed in this paper, are themselves only approximations of the necessary optical predistortion. The predistortion techniques discussed in sections 4 and 5 digitally sample these approximating functions. Texture-based predistortion samples the approximating functions much more sparsely than the techniques in sections 4 and 5.

7. CONCLUSION

This paper has described a method of correcting for optical HMD distortion in real time. The method makes use of widely distributed texture mapping hardware. As texture

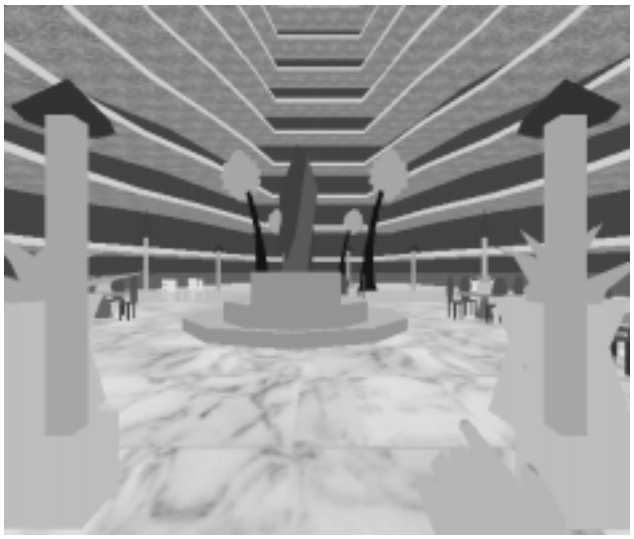

Figure 9: An undistorted view of a virtual world rendered at 640 x 480 resolution.

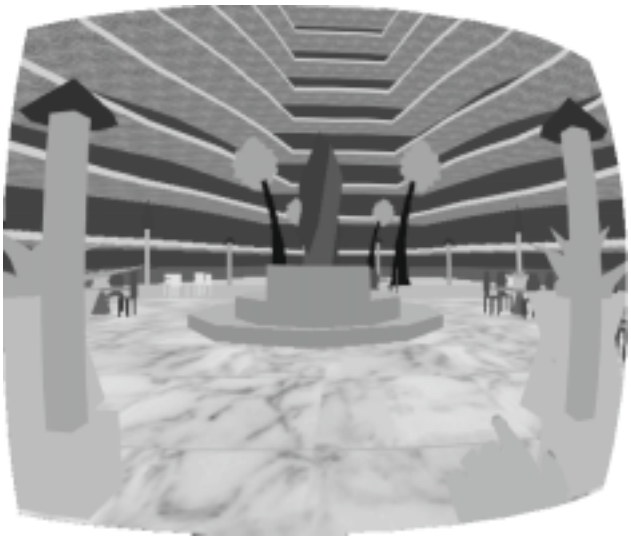

Figure 10: A predistorted view of a virtual world at 640 x 480 resolution, using the image in figure 9 as its texture.

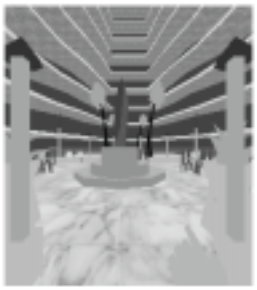

Figure 11: An undistorted view of a virtual world rendered at 256 x 256 resolution.

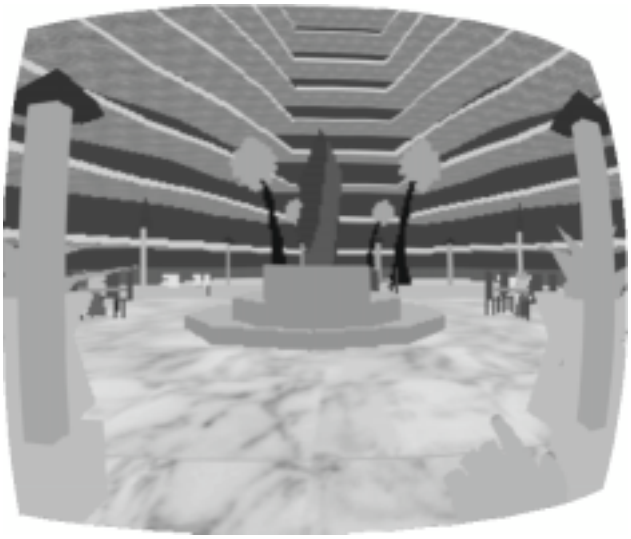

Figure 12: A predistorted view of a virtual world at 640 x 480 resolution, using the image in figure 11 as its texture.

mapping hardware improves, this technique will gain speed and utility.

8. ACKNOWLEDGMENTS

I am indebted to Gentaro Hirota of IBM Japan for discussions suggesting the possible application of texture maps to this problem. My reviewers made several valuable suggestions. This work was supported in part by the NSF Summer Institute in Japan program, a Link Foundation Fellowship for Advanced Simulation and Training and a Georgia Tech Foundation equipment grant.

9. REFERENCES

1. Alfano, P. and Michel, G. Restricting the field of view: perceptual and performance effects. *Perceptual and Motor Skills 70* (1990), 35-45.
2. Azuma, R. and Bishop, G. Improving static and dynamic registration in an optical see-through HMD. *Proc.Computer Graphics '94* (July 24-29, Orlando, FL), 197-204.
3. Bryson, S. Implementing virtual reality. *SIGGRAPH 1993 Course 43 Notes*, 1.1.1-1.3.26.
4. Edwards, E.K., Rolland, J.P. and Keller, K.P. Video see-through design for merging of real and virtual environments. *Proc. IEEE VRAIS '93* (September 18-22, Seattle, WA), 223-233.
5. Feiner, S., MacIntyre, B. and Seligman D. Annotating the real world with knowledge-based graphics on a see-through head-mounted display. *Proc. Graphics Interface '92* (May 11-15, Vancouver, BC), 78-85.
6. Fuchs, H. et al. Pixel Planes 5: a heterogeneous multiprocessing graphics system using processor enhanced memories. *Computer Graphics 23*, 3, (July '89) 79-88.
7. Hecht, E. and Zajac, A. *Optics*, Addison-Wesley, Reading, MA, 1974.

8. Heckbert, P. Survey of texture mapping. *IEEE Computer Graphics and Applications 6*, 10 (November '86), 56-67.
9. Howlett, E. Wide angle color photography method and system. U.S. Patent No. 4406532, 1983.
10. Klassen, R. V. Integer forward differencing of cubic polynomials. *ACM Trans. Graph. 10*, 2 (April 1991), 152-181.
11. Min, P. and Jense, H. Interactive stereoscopy optimization for head-mounted displays. *Proc. SPIE, Stereoscopic Displays and Virtual Reality Systems 2177* (1994).
12. Regan, M. and Pose, R. Priority rendering with a virtual reality address recalculation pipeline. *Proc.Computer Graphics '94* (July 24-29, Orlando, FL), 155-162.
13. Rolland, J. and Hopkins, H. A method of computational correction for optical distortion in head-mounted displays. *Univ. N. Carolina at Chapel Hill Tech Rpt. 93-045*. Available on the web at site http://www.cs.unc.edu.
14. Robinett, W. and Rolland, J. P. A computational model for the stereoscopic optics of a head-mounted display. *Presence 1*, 1 (Winter 1992), 45-62.
15. Silicon Graphics, on-line man pages on commands texdef2d and fbsubtexload on systems with IRIX 5.1.1.1 or later.
16. Sutherland, I. E. The ultimate display. *Proc. IFIPS Congress* (1965), 2, 506-508.
17. Sutherland, I. E. A head-mounted three-dimensional display. *Fall Joint Computer Conf., AFIPS Conf. Proc. 33* (1968), 757-764.
18. Watson, B. and Hodges L.F. Fast algorithms for rendering cubic curves. *Proc. Graphics Interface '92* (May 11-15 1992, Vancouver, BC), 19-28 .
19. Watson, B. A survey of virtual reality in Japan. *Presence 3*, 1 (Spring 1994), 1-18.
20. Welch, Robert B. Adaptation of space perception. In Boff, K.R., Kaufman, L. and Thomas, J.P. (1986). *Handbook of Perception and Human Performance*, ch. 24. Wiley Interscience, Chichester, UK, 1986.
21. Wells, M. and Venturino, M. Performance and head movements using a helmet-mounted display with different sized field-of-view. *Optical Engineering 29* (1989), 8, 870-877.