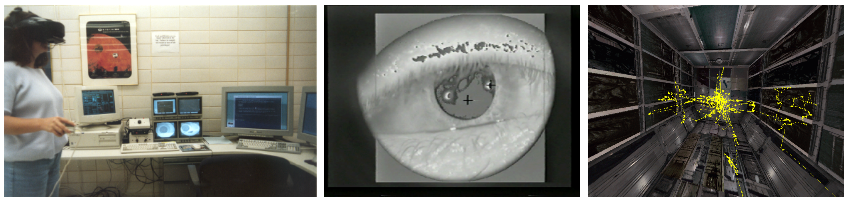
Binocular Eye Tracking in Virtual Reality for Inspection Training

Andrew T. Duchowski,[*] Vinay Shivashankaraiah, Tim Rawls
Department of Computer Science
Clemson University

Anand K. Gramopadhye,[†] Brian J. Melloy
Department of Industrial Engineering
Clemson University

Barbara Kanki[‡]
Human Factors Research and Technology Division
NASA Ames Research Center



(a) User interface. (b) Eye image from tracking device. (c) 3D *scanpaths.*

Figure 1: Recording binocular eye movements in aircraft cargo bay virtual inspection environment.

Abstract

This paper describes the development of a binocular eye tracking Virtual Reality system for aircraft inspection training. The aesthetic appearance of the environment is driven by standard graphical techniques augmented by realistic texture maps of the physical environment. A "virtual flashlight" is provided to simulate a tool used by inspectors. The user's gaze direction, as well as head position and orientation, are tracked to allow recording of the user's gaze locations within the environment. These gaze locations, or *scanpaths*, are calculated as gaze/polygon intersections, enabling comparison of fixated points with stored locations of artificially generated defects located in the environment interior. Recorded scanpaths provide a means of comparison of the performance of experts to novices, thereby gauging the effects of training.

[*] *andrewd@cs.clemson.edu*
[†] *agramop@ces.clemson.edu*
[‡] *bkanki@mail.arc.nasa.gov*

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Virtual Reality I.3.6 [Computer Graphics]: Methodology and Techniques—Interaction Techniques J.4 [Computer Applications]: Social and Behavioral Sciences—Psychology

Keywords: Eye Tracking, Virtual Reality, Visual Inspection

1 Introduction

Aircraft inspection and maintenance are an essential part of a safe, reliable air transportation system. Training has been identified as the primary intervention strategy in improving inspection performance [4]. If training is to be successful, it is clear that inspectors need to be provided with training tools to help enhance their inspection skills. In response to this need, this paper outlines a Virtual Reality (VR) inspection simulator under development at the Virtual Reality Eye Tracking (VRET) Laboratory at Clemson University.

The VR inspection system is a collaborative extension of recent efforts being pursued at the Training System Laboratory (TSL) at Clemson. Previous work at the TSL focused on the development of a computer based inspection training program—Automated System of Instruction for Specialized Training (ASSIST) [5]. The ASSIST program, developed using a task analytic methodology, features a PC-based inspection simulation of an aircraft cargo bay, where an image of a portion of the airframe is presented to the user for inspection (visual detection of defects). The image is a photo of a section of an aircraft's interior. The user is shown an image selected from a two-dimensional 4×8 grid of images, and is able to "navigate" left, right, up, and down to view the entire grid, one image at a time. Despite its advantages of being a computer-based inspection training/job-aid tool, the static, two-dimensional layout of the airframe lacks realism. To enhance the fidelity of the inspection system, an immersive, three-dimensional VR system has been developed.

The purpose of this paper is to describe the development of the Virtual Reality inspection system. The VR inspection simulator features a binocular eye tracker, built into the system's Head Mounted Display (HMD), which allows the recording of the user's dynamic point of regard within the virtual environment. User gaze directions, as well as head position and orientation, are tracked to enable navigation and post-immersive examination of the user's overt spatio-temporal focus of attention while immersed in the environment. The recorded point of regard addresses imprecision and ambiguity of the user's viewpoint in a virtual environment by explicitly providing the 3D location of the user's gaze.

In Virtual Reality, the three-dimensional location of gaze serves as either a real-time or a post-immersion diagnostic indicator of the user's overt focus of attention. The collection of gaze points taken over the course of immersion, the user's three-dimensional scanpath, serves as a diagnostic tool for post-immersive reasoning about the user's actions in the environment. In our system, we conjecture recorded eye movements, or scanpaths, will enable comparison of the performance of experts to novices, thereby gauging the effects of training.

The main contribution of this paper is the presentation of real-time software techniques for the integration of the binocular eye tracker in a Virtual Reality application. Presently, it appears that the binocular eye tracker coupled with an HMD capable of vergence measurement in VR is the first of its kind to be assembled in the United States. Although binocular eye trackers integrated with HMDs have previously been proposed [8], no reports of their actual construction or operation have been found. For reference, the hardware used at the Clemson University Virtual Reality Eye Tracking lab is briefly described in Section 2. Section 3 presents the evolution of the geometric model of the virtual aircraft cargo bay inspection environment. In Section 4, technical issues of eye tracker system integration and operation are discussed, emphasizing coordinate mapping between the eye tracker and the VR application, and the calculation of the three-dimensional gaze point from binocular eye movement measurements. Concluding remarks are given in Section 5.

2 Hardware Platform

Our primary rendering engine is a dual-rack, dual-pipe, SGI Onyx2® InfiniteReality™ system with 8 raster managers and 8 MIPS® R10000™ processors, each with 4Mb secondary cache.[1] It is equipped with 3Gb of main memory and 0.5Gb of texture memory.

Multi-modal hardware components include a binocular ISCAN eye tracker mounted within a Virtual Research V8 (high resolution) Head Mounted Display (HMD). The V8 HMD offers 640×480 resolution per eye with separate left and right eye feeds. HMD position and orientation tracking is provided by an Ascension 6 Degree-Of-Freedom (6DOF) Flock Of Birds (FOB), a d.c. electromagnetic system with a 10ms latency. A 6DOF tracked, hand-held mouse provides a means to represent a virtual tool for the user in the environment. The lab is shown in Figure 2.

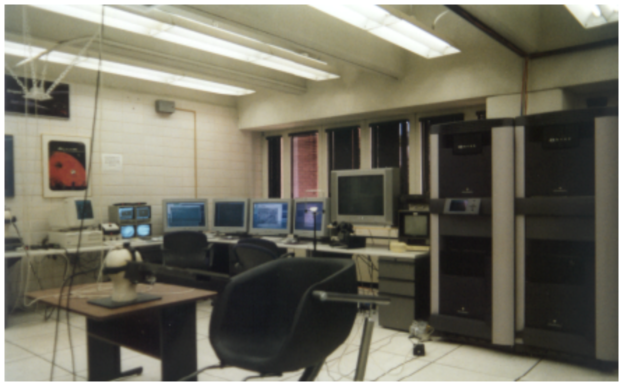


Figure 2: Virtual Reality Eye Tracking (VRET) laboratory at Clemson University.

3 Geometric Environment Modeling

The goal of the construction of the virtual environment is to match the appearance of the physical inspection environment, an aircraft cargo bay, shown in Figure 3. The phys-

[1]Silicon Graphics, Onyx2, InfiniteReality, are registered trademarks of Silicon Graphics, Inc.

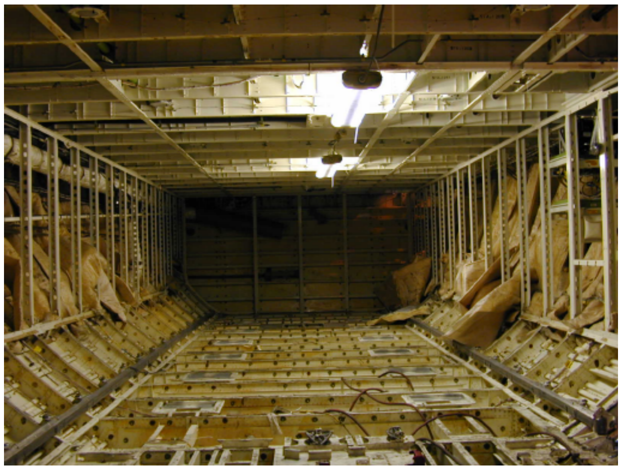


Figure 3: Aircraft cargo bay physical environment.

ical environment is a complex three-dimensional cube-like volume, with airframe components (e.g., fuselage ribs) exposed for inspection. A typical visual inspection task of the cargo bay involves carefully walking over the structural elements while searching for surface defects such as corrosion and cracks (among others).

3.1 Model Geometry

The evolution of the virtual inspection environment began with a straightforward emulation of the two-dimensional ASSIST inspection grid. The rationale for this design was the relatively quick development of a simple texture-mapped grid to provide the user with a more natural representation and navigation of the entire inspection region. The image grid, displayed as a flat polygonal wall in VR, is shown in Figure 4. Although the VR environment provided advan-

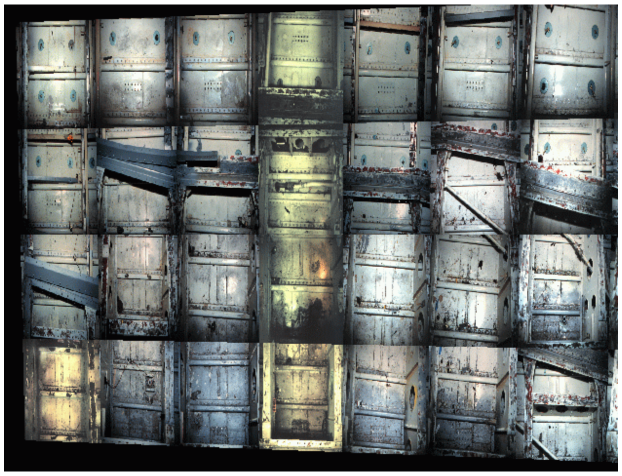


Figure 4: Initial planar 2D virtual environment.

tages over the PC-based system, several problems with its design became evident.

The main advantage of the VR system is its display of the entire side of the airframe's wall, which provides better context for the user in terms of the location of the individual panels under inspection. The obvious drawbacks of this implementation, however, are that there are noticeable discrepancies in the appearance of the images (e.g., lighting changes and positional misalignment), and the unsuitable simplicity of the geometry for VR. The simplistic 2D wall in effect defeats the immersive and natural navigational advantages offered by VR technology. Clearly, to provide an immersive environment, a three-dimensional structure was required.

The next evolution of the inspection environment was patterned after a simple three-dimensional enclosure (e.g., a cube), specified by actual dimensions of the real inspection environment, i.e., an aircraft's cargo bay. An early stage of the wireframe rendition of the cargo bay "3D box" is shown in Figure 5. The model is built entirely out of planar poly-

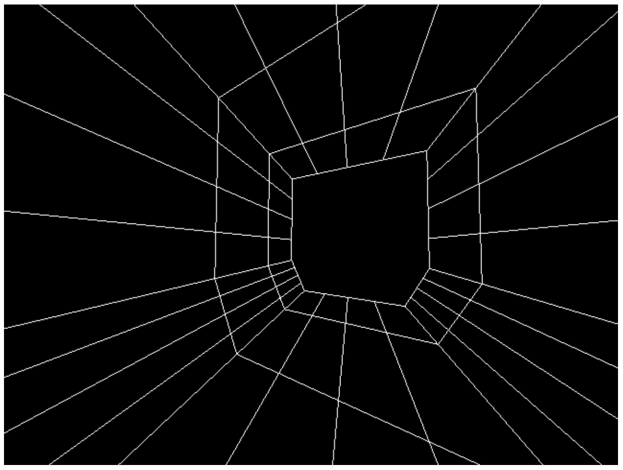


Figure 5: 3D box-like virtual environment (shown in wireframe).

gons. There are two pragmatic reasons for this design choice. First, since the representation of the true complexity of the airframe structure is avoided, fast display rates are maintained (on the order of 10-30 fps) while tracking latencies are minimized (on the order of 10-30 ms for head and eye trackers).[2] Second, planar polygons (quadrilaterals) are particularly suitable for texture mapping. To provide a realistic appearance of the environment, in keeping with the desire of preserving geometric simplicity, images of the physical environment are used for texture maps.

3.2 Lighting and Flashlight Modeling

The SGI Onyx2 host provides real-time graphics rendering performance, while simultaneously processing tracking information sent to the host via the rs-232 serial connection.

[2]Display update and latency measures are only rough estimates at this point, we are working on obtaining more formal measurements.

To generate the environment, no specialized rendering algorithms are invoked beyond what is provided by the `OpenGL` graphics library Application Program Interface (API). Standard (1st-order) direct illumination is used to light the environment. Additionally, an `OpenGL` spotlight is used to provide the user with a "virtual flashlight". The flashlight effect is shown over a 2D polygon in Figure 6. The flashlight's position and orientation are obtained from the 6DOF electromagnetically tracked "flying mouse" from Ascension.

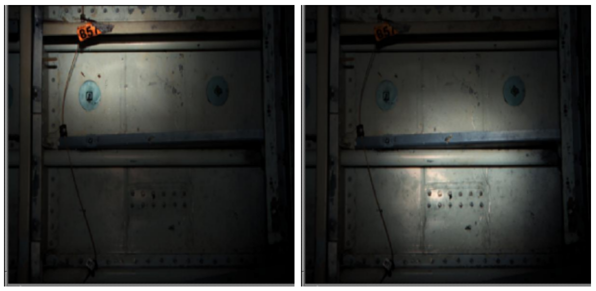


Figure 6: Virtual flashlight shown over a 2D polygon.

Because `OpenGL` relies on the Phong illumination model coupled with Gouraud shading to generate lighting effects, large polygons produce a coarse (blocky) flashlight beam. To correct this problem, the polygons were subdivided to smooth out the spotlight, producing a more aesthetically pleasing circular effect. The level of polygonal subdivision is user-adjustable.

3.3 Realistic Texture Maps

To texture map the simple 3D box-like environment, images of the physical environment were obtained. With permission from a commercial airline, images of an aircraft's cargo bay were taken while the aircraft was withheld from operation for inspection and servicing. Since the time to photograph the cargo bay interior was limited, no particular methodology or special equipment was used to obtain these images. Care was taken to attempt to photograph the environment in sections by translating a hand-held camera, however, inaccuracies in image alignment were inevitable.

In retrospect, a better approach would have been to measure the environment *a priori*, then calculate and record appropriate positions of the camera, and ensure proper placement of the camera on some stable rigging apparatus (e.g., a levelled tripod which could be fastened to the aircraft interior). This approach would be similar to a motion capture or an image-rendering session used in special effects production work. Of course, this method would require more time and care to carry out.

Although our approach was somewhat haphazard, image alignment was possible through careful labeling of photographs and a good deal of digital post-processing. In our case, we used The Gimp (the freeware analogue of PhotoShop) to orient and resize the images appropriately. The resulting environment, with the user-held "virtual flashlight" is shown in Figure 7.

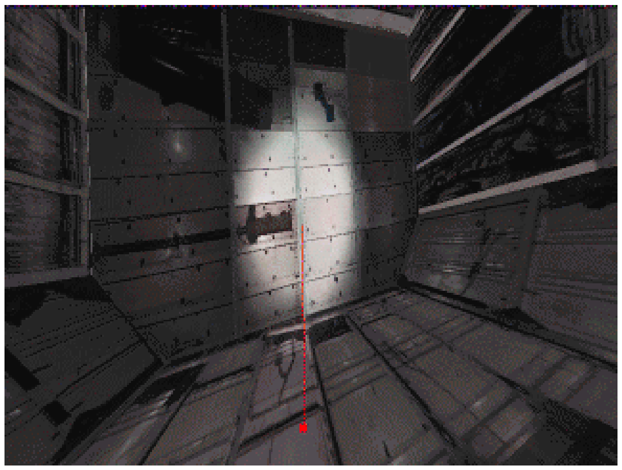


Figure 7: Virtual flashlight in virtual aircraft cargo bay environment.

4 Eye Tracking

Interest in gaze-contingent interface techniques has endured since early implementations of eye-slaved flight simulators and has since permeated several disciplines including human-computer interfaces, teleoperator environments, and visual communication modalities [7, 9, 6]. Recent applications of an eye tracker in VR have shown promise in the use of the device as a component of multi-modal interface systems. For example, Jacob has used an eye tracker as a selection device in VR [10], and Danforth et al. use an eye tracker as an indicator of gaze in a gaze-contingent multi-resolution terrain navigation environment [1]. Although these are examples of eye trackers used as *interactive* devices, in the present application, the eye tracker serves a *diagnostic* purpose. That is, no changes in the display occur due to the location of the user's gaze, rather, the user's eye movements are simply (and unobtrusively) recorded in real-time for post-immersion analysis.

In our system, a dedicated PC calculates the point of regard in real-time (60Hz) from left and right video images of the user's pupils and infra-red corneal reflections (first Purkinje images). Figure 8 shows a user wearing the eye tracking HMD. Eye images captured by the cameras can be seen in two video monitors near the lower right of the figure. In this section, the technical issues of system integration and operation are presented (for a technical description of system installation and wiring, see [2]).

4.1 Eye Tracker Integration

In designing the visual inspection simulator, a critical concern is the mapping of eye tracker coordinates to the appli-

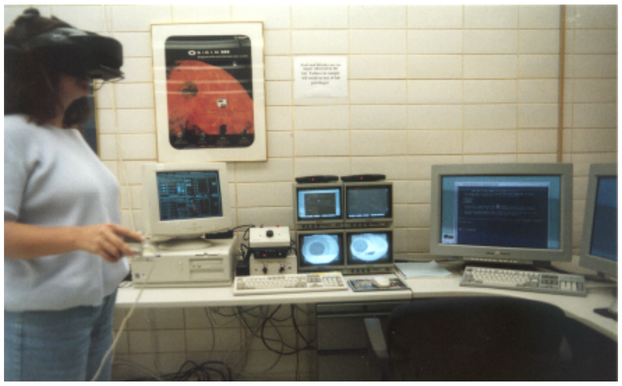


Figure 8: Eye tracking HMD.

cation program's reference frame. The eye tracker calculates the viewer's point of regard relative to the eye tracker's screen reference frame, e.g., a 512×512 pixel plane, perpendicular to the optical axis. That is, for each eye, the eye tracker returns a sample coordinate pair of *x*- and *y*-coordinates of the point of regard at each sampling cycle (e.g., once every ∼16ms for a 60Hz device). This coordinate pair must be mapped to the extents of the application program's viewing window. The VR application must also, in the same update cycle, map the coordinates of the head's position and orientation tracking device.

4.2 Eye Tracker Coordinate Mapping

The eye movement data obtained from the tracker must be mapped to a range appropriate for the VR application. Specifically, the 2D eye tracker data, expressed in eye tracker screen coordinates, must be mapped to the 2D dimensions of the near viewing frustum. The 3D viewing frustum employed in the perspective viewing transformation is defined by the parameters `left, right, bottom, top, near, far`.[3] Figure 9 shows the dimensions of the eye tracker screen (left) and the dimensions of the viewing frustum (right). To convert the eye tracker coordinates (x',y')

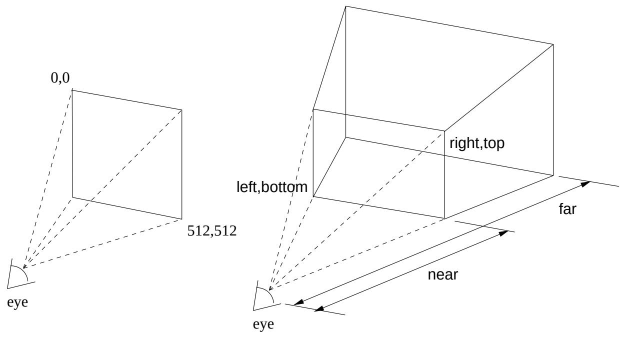


Figure 9: Eye tracker to 3D viewing frustum screen coordinate mapping.

[3]For example, as used in the `OpenGL` function call `glFrustum()`.

to graphics coordinates (x,y) a linear interpolation mapping is used:

$$x = \texttt{left} + \frac{x'(\texttt{right} - \texttt{left})}{512} \tag{1}$$

$$y = \texttt{bottom} + \frac{(512 - y')(\texttt{top} - \texttt{bottom})}{512} \tag{2}$$

Since the eye tracker origin is at the top-left of the screen and the viewing frustum's origin is at the bottom-left (a common discrepancy between imaging and graphics applications), the term $(512 - y')$ in Equation (2) handles the necessary *y*-coordinate mirror transformation.

The above coordinate mapping assumes that the eye tracker coordinates are in the range $[0, 511]$. In reality, the usable, or effective, coordinates will be dependent on: (a) the size of the application window, and (b) the position of the application window. Proper mapping between eye tracker and application coordinates is achieved through the measurement of the application window's extents in the eye tracker's reference frame. This is accomplished by using the eye tracker's own fine cursor movement and cursor location readout.

To obtain the extents of the application window in the eye tracker's reference frame, the application window's corners are measured with the eye tracker's cursor. These window extents are then used in the linear mapping equation. Figure 10 illustrates an example of a 600×450 application window as it would appear on the eye tracker scene monitor. Based on the measurements shown in Figure 10, the linear

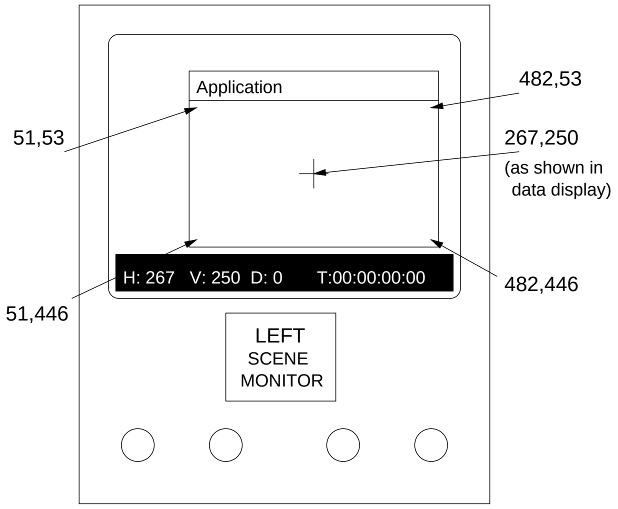


Figure 10: Mapping measurement example.

coordinate mapping is:

$$x = \frac{x' - 51}{(482 - 51 + 1)}(600) \tag{3}$$

$$y = 449 - \frac{y' - 53}{(446 - 53 + 1)}(450) \tag{4}$$

The central point on the eye tracker display is $(267, 250)$.

Note that y is subtracted from 449 to take care of the image/graphics vertical origin flip.

4.3 Gaze Vector Calculation

The calculation of the point of regard in three-space depends on only the relative positions of the two eyes in the horizontal axis. The parameters of interest here are the three-dimensional virtual coordinates, (x_g, y_g, z_g), which can be determined from traditional stereo geometry calculations. Figure 11 illustrates the basic binocular geometry. Helmet tracking determines both helmet position and the (orthogonal) directional and up vectors, which determine viewer-local coordinates shown in the diagram. The helmet position is the origin, the helmet directional vector is the optical (viewer-local z) axis, and the helmet up vector is the viewer-local y axis.

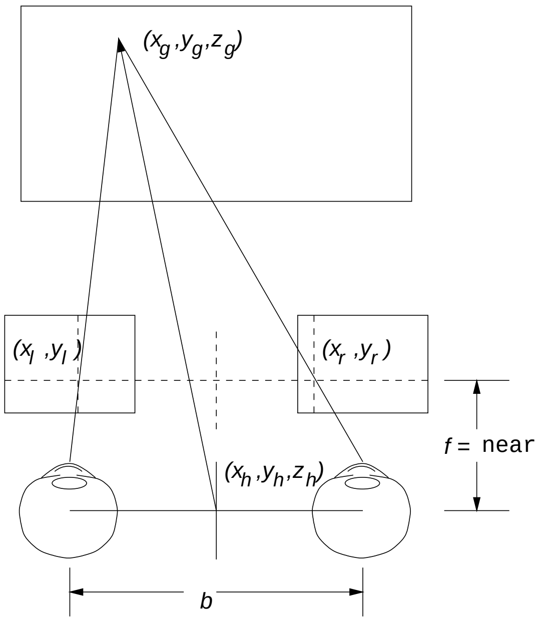


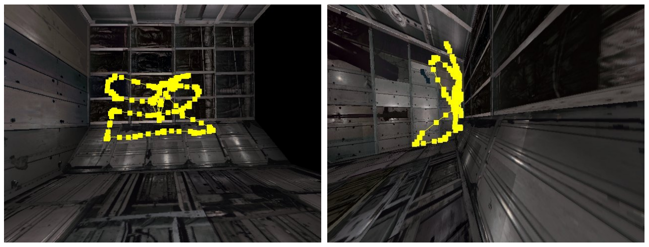
Figure 11: Basic binocular geometry.

Given instantaneous, eye tracked, viewer-local coordinates (x_l, y_l) and (x_r, y_r) in the left and right image planes (mapped from eye tracker screen coordinates to the near view plane), and head-tracked head position coordinates (x_h, y_h, z_h), the viewer-local coordinates of the gaze point, (x_g, y_g, z_g), are determined by the relations:

$$x_g = (1-s)x_h + s(x_l + x_r)/2 \quad (5)$$
$$y_g = (1-s)y_h + s(y_l + y_r)/2 \quad (6)$$
$$z_g = (1-s)z_h + sf \quad (7)$$

where $s = b/(x_l - x_r + b)$, b is the disparity distance between the left and right eye centers, and f is the distance to the near viewing plane along the viewer-local z axis.

Figure 12 shows a user's 3D gaze points in the aircraft cargo bay environment. Gaze point coordinates based on vergence calculations given by Equations (5)–(7) are presently closely correlated with the user's head location.[4] Figure 12(b) shows the collection of gaze points from a side viewpoint. With respect to depth, the gaze points do not precisely fall on the polygonal surfaces of the environment. To compare the visual search performance of experts to novices in a task-specific environment such as the aircraft cargo bay, recorded scanpath information should convey the location of gaze points in the same relative reference frame as the visual search targets. Specifically, we wish to compare these gaze points to the set of stored locations of artificially generated defects located at known coordinates in the cargo bay texture maps. An example of defect location in an environment texture map is shown in Figure 13.[5] We are therefore interested in recording the points of intersection of the user's gaze and the environment walls (polygons).



(a) Front view. (b) Side view.

Figure 12: 3D gaze points in cargo bay virtual environment.

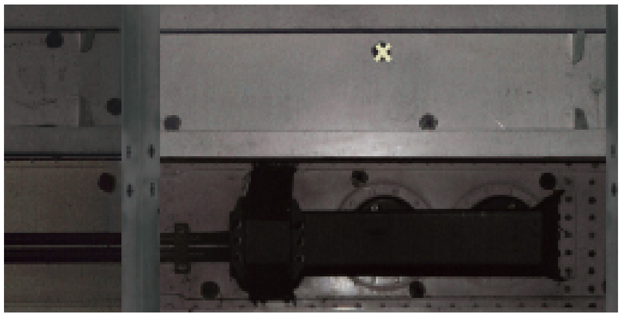


Figure 13: Example of location of artificially generated defect in an environment texture map.

To calculate the the gaze/polygon intersection, the gaze point is expressed parametrically as a point on a ray with origin (x_h, y_h, z_h), the helmet position, with the ray emanating along a vector scaled by parameter s. That is, rewriting Equations (5), (6), and (7), we have:

$$x_g = x_h + s\left(\frac{x_l + x_r}{2} - x_h\right)$$

[4]We are working towards the specification of focal and disparity parameters to give us clearer depth information of the gaze point, dissociating the head position from the point of gaze.

[5]The simple target symbol in Figure 13 is currently only used for testing purposes; eventually artificial defects will be made more realistic.

$$\begin{aligned} y_g &= y_h + s\left(\frac{y_l + y_r}{2} - y_h\right) \\ z_g &= z_h + s(f - z_h) \end{aligned}$$

or, in vector notation,

$$\mathbf{g} = \mathbf{h} + s\mathbf{v} \tag{8}$$

where **h** is the head position, **v** is the central view vector and s is the scale parameter as defined previously. The view vector **v** is obtained by subtracting the helmet position from the midpoint of the eye tracked x-coordinate and focal distance to the near view plane, i.e.,

$$\begin{aligned} \mathbf{v} &= \begin{bmatrix} (x_l + x_r)/2 \\ (y_l + y_r)/2 \\ f \end{bmatrix} - \begin{bmatrix} x_h \\ y_h \\ z_h \end{bmatrix} \\ &= \mathbf{m} - \mathbf{h} \end{aligned} \tag{9}$$

where **m** denotes the left and right eye coordinate midpoint.[6] To align the view vector to the current head orientation, the vector **m** must first be transformed to the proper (instantaneous) head orientation. This is done by first normalizing **m** and then multiplying it by the orientation matrix returned by the head tracker.[7]

Given the three-dimensional gaze vector, **v**, specified by Equation (9), Equation (8) gives the coordinates of the gaze point parametrically along a ray originating at the head position (x_h, y_h, z_h). The depth of the three-dimensional gaze point in world coordinates is valid only if $s > 0$.

4.4 Gaze Point Calculation

The formulation of the gaze direction given by Equation (8) can be used for testing virtual gaze/polygon intersection coordinates via traditional ray/polygon intersection calculations commonly used in ray tracing [3]. The gaze/polygon intersection point is found on the closest polygon to the viewer intersecting the gaze ray, assuming all polygons are opaque. This polygon is found by testing all polygons in the scene for intersection with the gaze ray. To find the intersection point **g** of the gaze ray with the closest polygon, a new interpolant t is obtained by calculating the gaze ray intersections with all scene polygons. All such intersections are examined for which $t > 0$.[8] The interpolant t is obtained by substituting the gaze ray equation into the polygon's plane equation (in vector notation):

$$t = \frac{-(\mathbf{N} \cdot \mathbf{h} + D)}{\mathbf{N} \cdot \mathbf{v}} \tag{10}$$

[6]Note that since the vertical eye tracked coordinates y_l and y_r are expected to be equal (since gaze coordinates are assumed to be epipolar), the vertical coordinate of the central view vector defined by $(y_l + y_r)/2$ is somewhat extraneous; either y_l or y_r would do for the calculation of the gaze vector. However, since eye tracker data is also expected to be noisy, this averaging of the vertical coordinates enforces the epipolar assumption.

[7]Equivalently, head orientation in the form of a quaternion, as returned by the head tracker, may be used for the same purpose.

[8]If $t < 0$, the polygon may intersect the gaze ray, but behind the viewer.

where **N** is the negated polygon normal and D is the height parameter of the polygon's plane equation. The geometry of this calculation is depicted in Figure 14. The calculation

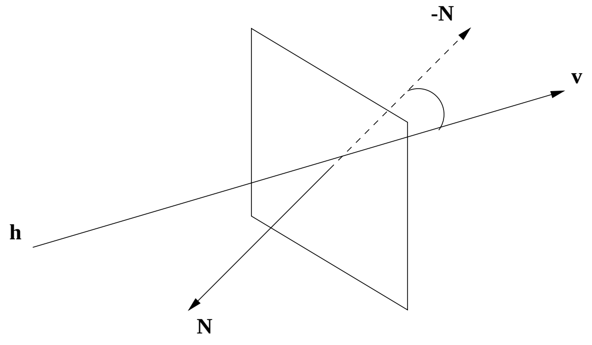


Figure 14: Ray/plane geometry.

of the ray/plane intersection may be speeded up by evaluating the denominator of Equation (10) first. The intersection algorithm is given in Figure 15. Note that the ray/polygon

```
$v_d = \mathbf{N} \cdot \mathbf{v}$;                                  // denominator

if($v_d < 0$) {
  $v_o = -(\mathbf{N} \cdot \mathbf{h} + D)$;                    // numerator
  $t = v_o / v_d$;
}
```

Figure 15: Ray/polygon intersection.

intersection algorithm only returns the intersection point of the ray and the infinite plane defined by the polygon's face normal. Because the normal defines a plane of infinite extent, the point **g** must be tested against all of the polygon's edges to establish whether the point lies inside the polygon. This is an instance of a solution to the well-known "point-in-polygon" problem. The geometry of this algorithm is shown in Figure 16. If the point **g** is bounded by the perpendicular

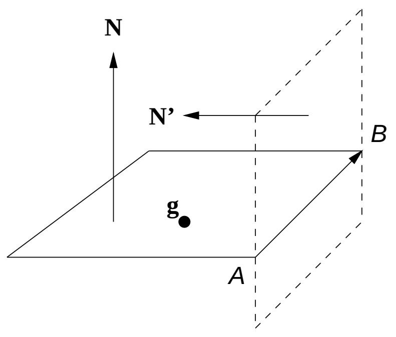


Figure 16: Point-in-polygon problem.

planes defined by the polygon's edges, then **g** lies within the polygon, otherwise it lies on the plane defined by the face normal **N**, but outside the polygonal region. The resulting algorithm generates a scanpath constrained to lie on polygonal regions within the virtual environment. Such a scanpath is shown in Figure 17. Provided the number of polygons is

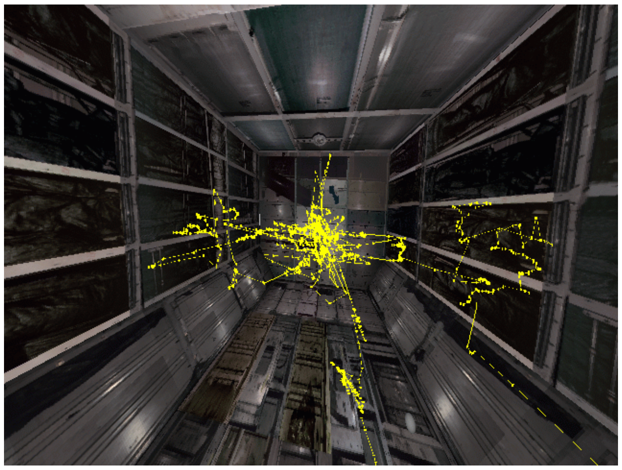


Figure 17: 3D scanpath in aircraft cargo bay virtual environment.

sufficiently small, the algorithm executes in real-time.

5 Conclusions & Future Work

We have described an operational platform for real-time recording of eye movements in Virtual Reality. The platform is based on high-end graphics engines and an electromagnetically tracked, binocular helmet equipped with infra-red eye tracking capability. Rendering techniques are relatively simple, relying only on standard (`OpenGL`) graphics library calls. Tracking routines deliver helmet position and orientation in real-time, which are used directly to provide updated images to the HMD.

User gaze direction is tracked in real-time, along with calculated gaze/polygon intersections. We are in the process of analyzing recorded gaze intersection points for comparison with stored locations of artificially generated defects in the inspection environment.

Controlled studies of human performance in the virtual cargo bay inspection environment are forthcoming.

6 Acknowledgements

This work was supported in part by Clemson University Innovation award (Project #1-20-1906-51-4087), NASA Ames grant (Task #NCC 2-1114), and NSF CAREER award (Grant IIS-9984278).

References

[1] DANFORTH, R., DUCHOWSKI, A., GEIST, R., AND MCALILEY, E. A Platform for Gaze-Contingent Virtual Environments. In *Smart Graphics (Papers from the 2000 AAAI Spring Symposium, Technical Report SS-00-04)* (Menlo Park, CA, 2000), AAAI, pp. 66–70.

[2] DUCHOWSKI, A. T., AND VERTEGAAL, R. *Course 05: Eye-Based Interaction in Graphical Systems: Theory & Practice*. ACM SIGGRAPH, New York, NY, July 2000. SIGGRAPH 2000 Course Notes.

[3] GLASSNER, A. S., Ed. *An Introduction to Ray Tracing*. Academic Press, San Diego, CA, 1989.

[4] GRAMOPADHYE, A., BHAGWAT, S., KIMBLER, D., AND GREENSTEIN, J. The Use of Advanced Technology for Visual Inspection Training. *Applied Ergonomics 29*, 5 (1998), 361–375.

[5] GRAMOPADHYE, A. K., MELLOY, B., CHEN, S., AND BINGHAM, J. Use of Computer Based Training for Aircraft Inpsectors: Findings and Recommendations. In *Proceedings of the HFES/IEA Annual Meeting* (San Diego, CA, August 2000).

[6] HELD, R., AND DURLACH, N. Telepresence, time delay and adaptation. In *Pictorial Communication in Virtual and Real Environments*, S. R. Ellis, M. Kaiser, and A. J. Grunwald, Eds. Taylor & Francis, Ltd., London, 1993, pp. 232–246.

[7] JACOB, R. J. What You Look at is What You Get: Eye Movement-Based Interaction Techniques. In *Human Factors in Computing Systems: CHI '90 Conference Proceedings* (1990), ACM Press, pp. 11–18.

[8] OHSHIMA, T., YAMAMOTO, H., AND TAMURA, H. Gaze-Directed Adaptive Rendering for Interacting with Virtual Space. In *Proceedings of VRAIS'96* (March 30–April 3 1996), IEEE, pp. 103–110.

[9] STARKER, I., AND BOLT, R. A. A Gaze-Responsive Self-Disclosing Display. In *Human Factors in Computing Systems: CHI '90 Conference Proceedings* (1990), ACM Press, pp. 3–9.

[10] TANRIVERDI, V., AND JACOB, R. J. K. Interacting with Eye Movements in Virtual Environments. In *Human Factors in Computing Systems: CHI 2000 Conference Proceedings* (2000), ACM Press, pp. 265–272.