Eye-Based Interaction in Graphical Systems: Theory & Practice

Andrew Duchowski
Clemson University

Part II
Eye Tracking Systems

E The Eye Tracker

The measurement device most often used for measuring eye movements is commonly known as an eye tracker. In general, there are two types of eye movement monitoring techniques: those that measure the position of the eye relative to the head, and those that measure the orientation of the eye in space, or the "point of regard" [YS75]. The latter measurement is typically used when the concern is the identification of elements in a visual scene, e.g., in (graphical) interactive applications. Arguably, the most widely applied apparatus for measurement of the point of regard is the video-based corneal reflection eye tracker. In this section, most of the popular eye movement measurement techniques are briefly discussed first before covering video-based trackers in greater detail.

E.1 Brief Survey of Eye Tracking Techniques

There are four broad categories of eye movement measurement methodologies, involving the use or measurement of: electro-oculography (EOG), scleral contact lens/search coil, photo-oculography (POG) or video-oculography (VOG), and video-based combined pupil and corneal reflection.

Electro-oculography, or EOG, relies on (d. c. signal) recordings of the electric potential differences of the skin surrounding the ocular cavity. During the mid-seventies, this technique was the most widely applied eye movement method [YS75]. Today, arguably the most widely applied eye movement technique, primarily used for point of regard measurements, is the method based on corneal reflection.

The first method for objective eye measurements using corneal reflection was reported in 1901 [Rob68]. To improve accuracy, techniques using a contact lens were developed in the 1950's. Devices attached to the contact lens ranged from small mirrors to coils of wire. Measurement devices relying on physical contact with the eyeball generally provide very sensitive measurements. The obvious drawback of these devices is their invasive requirement of wearing the contact lens. So-called non-invasive (sometimes called remote) eye trackers typically rely on the measurement of visible features of the eye, e.g., the pupil, iris-sclera boundary, or a corneal reflection of a closely positioned, directed light source. These techniques often involve either manual or automatic (computer-based) analysis of video recordings of the movements of the eyes, either off-line or in real-time. The availability of fast image processing processing hardware has facilitated the development of real-time video-based point of regard turn-key systems.

E.1.1 Electro-Oculography (EOG)

Electro-oculography, or EOG, the most widely applied eye movement recording method some twenty years ago (and still used today), relies on measurement of the skin's electric potential differences, of electrodes placed around the eye. A picture of a subject earing the EOG apparatus is shown in Figure 25. The recorded potentials are in the range 15-200μV, with nominal sensitivities of order of 20$\mu V/$deg of eye movement.

Figure 25: Example of electro-oculography (EOG) eye movement measurement. From MetroVision [Met00].

This technique measures eye movements relative to head position, and so is not generally suitable for point of regard measurements unless head position is also measured (e.g., using a head tracker).

E.1.2 Scleral Contact Lens/Search Coil

One of the most precise eye movement measurement methods involves attaching a mechanical or optical reference object mounted on a contact lens which is then worn directly on the eye. Early such recordings (ca. 1898 [YS75]) used a plaster of paris ring attached directly to the cornea and through mechanical linkages to recording pens. This technique evolved to the use of a modern contact lens to which a mounting stalk is attached. The contact lens is necessarily large, extending over the cornea and sclera (the lens is subject to slippage if the lens only covers the cornea). Various mechanical or optical devices have been placed on the stalk attached to the lens: reflecting phosphors, line diagrams, and wire coils have been the most popular implements in magneto-optical configurations. The principle method employs a wire coil, which is then measured moving through an electromagnetic field.[4] A picture of the search coil embedded in a scleral contact lens and the electromagnetic field frame are shown in Figure 26. The manner of insertion of the contact lens is shown in Figure 27. Although the scleral search coil is the most precise eye movement measurement method (accurate to about 5-10 arc-seconds over a limited range of about 5° [YS75]), it is also the most intrusive method. Insertion of the lens requires care and practice. Wearing of the lens causes discomfort. This method also measures eye position relative to the head, and is not generally suitable for point of regard measurement.

[4] This is similar in principle to magnetic position/orientation trackers often employed in Virtual Reality applications, e.g., Ascension's Flock Of Birds (FOB) uses this type of method for tracking the position/orientation of the head.

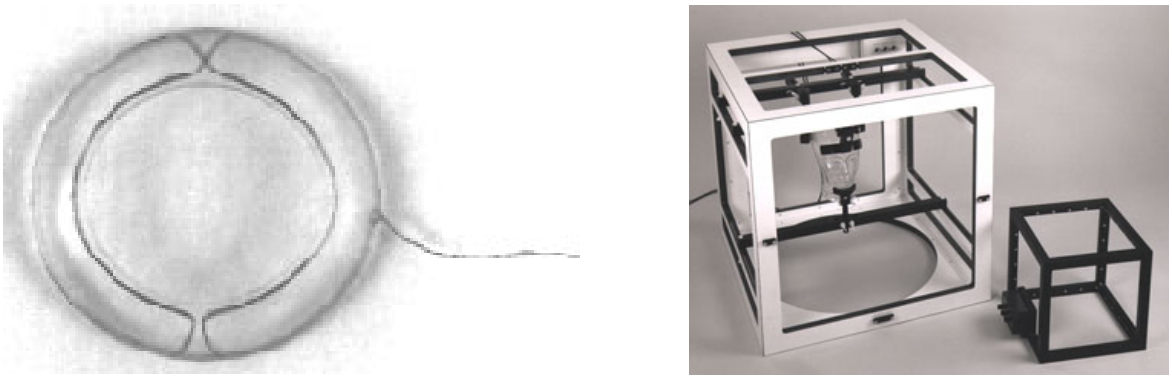


Figure 26: Example of search coil embedded in contact lens and electromagnetic field frames for search coil eye movement measurement. From Skalar Medical [Ska00].

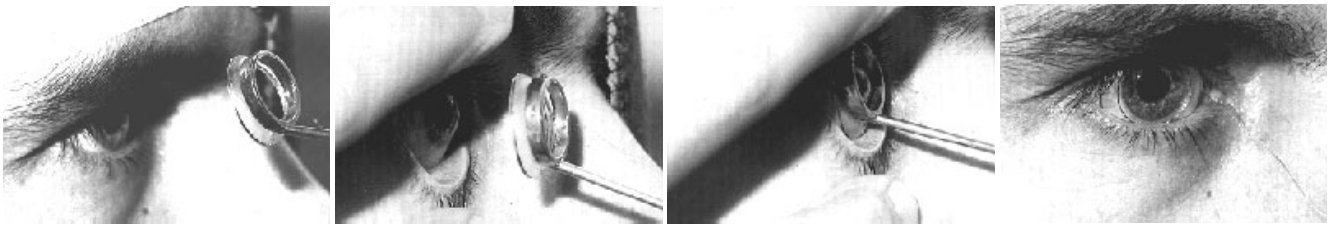
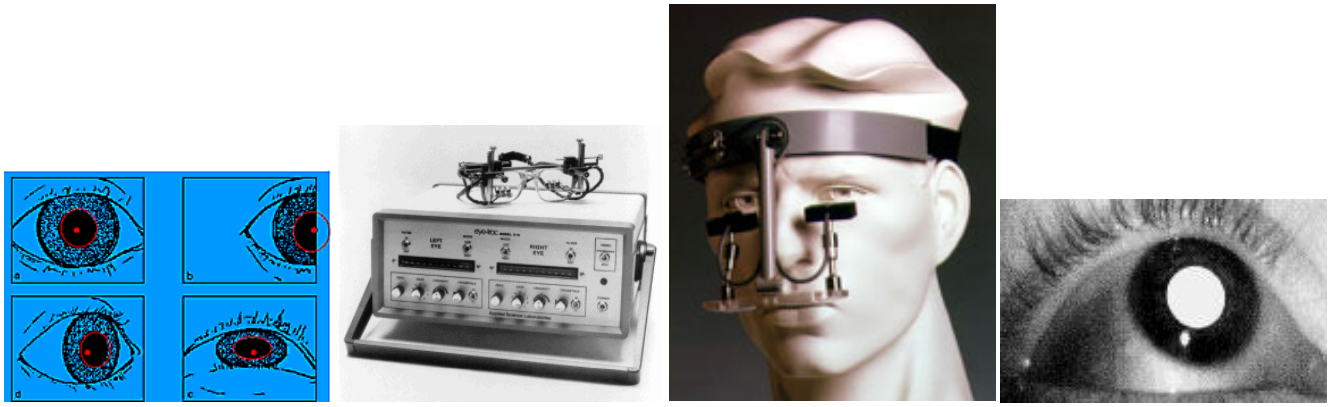


Figure 27: Example of scleral suction ring insertion for search coil eye movement measurement. From Skalar Medical [Ska00].

E.1.3 Photo-Oculography (POG) or Video-Oculography (VOG)

This category groups together a wide variety of eye movement recording techniques involving the measurement of distinguishable features of the eyes under rotation/translation, e.g., the apparent shape of the pupil, the position of the limbus (the iris-sclera boundary), and corneal reflections of a closely situated directed light source (often infra-read). While different in approach, these techniques are grouped here since they often do not provide point of regard measurement. Examples of apparatus and recorded images of the eye used in photo- or video-oculography and/or limbus tracking are shown in Figure 28. Measurement of ocular features provided by these measurement techniques may



(a) Example of apparent pupil size. From MetroVision [Met00].

(b) Example of infra-read limbus tracker apparatus. From ASL [ASL00].

(c) Another example of infra-read limbus tracker apparatus, as worn by subject. From Microguide [Mic00].

(d) Example of "bright pupil" (and corneal reflection) illuminated by infra-red light. From LC Technologies [LCT00].

Figure 28: Example of pupil, limbus, and infra-red (IR) reflection eye movement measurements.

or may not be made automatically, and may involve visual inspection of recorded eye movements (typically recorded on video tape). Visual assessment performed manually, e.g., stepping through a videotape frame-by-frame, can be extremely tedious and prone to error, and limited to the temporal sampling rate of the video device.

Automatic limbus tracking often involves the use of photodiodes mounted on spectacle frames (see Figure 28(b) and (c)), and almost always involves the use of invisible (usually infra-red) illumination (see Figure 28(d)). Several of these methods require the head to be fixed, e.g.,

either by using a head/chin rest, or a bite-bar [YS75].

E.1.4 Video-Based Combined Pupil and Corneal Reflection

While the above techniques are in general suitable for eye movement measurements, they do not often provide point of regard measurement. To provide this measurement, either the head must be fixed so that the eye's position relative to the head and point of regard coincide, or multiple ocular features must be measured in order to disambiguate head movement from eye rotation. Two such features are the corneal reflection (of a light source, usually infra-red) and the pupil center (see Figure 28(d)).

Using a combined feature tracking technique, video-based trackers utilize relatively inexpensive video cameras and image processing hardware to compute the point of regard in real-time. The apparatus may be table-mounted, as shown in Figure 29 or worn on the head, as shown

Figure 29: Example of table-mounted video-based eye tracker.

in Figure 30. The optics of both table-mounted or head-mounted systems are essentially identical, with the exception of size. These devices,

Figure 30: Example of head-mounted video-based eye tracker. From IOTA AB Eye Trace Systems [IOT00].

which are becoming increasingly available, are most suitable for use in (graphical) interactive systems.

The corneal reflection of the light source (typically infra-red) is measured relative to the location of the pupil center. Corneal reflections are known as the Purkinje reflections, or Purkinje images [Cra94]. Due to the construction of the eye, four Purkinje reflections are formed, as shown in Figure 31. Video-based eye trackers typically locate the first Purkinje image. With appropriate calibration procedures, these eye trackers are capable of measuring a viewer's point of regard (POR) on a suitably positioned (perpendicularly planar) surface on which calibration points are displayed.

Two points of reference on the eye are needed to separate eye movements from head movements. The positional difference between the pupil center and corneal reflection changes with pure eye rotation, but remains relatively constant with minor head movements. Approximate relative positions of pupil and first Purkinje reflections are graphically shown in Figure 32, as the left eye rotates to fixate 9 correspondingly

PR, Purkinje reflections: 1, reflection from front surface of the cornea; 2, reflection from rear surface of the cornea; 3, reflection from front surface of the lens; 4, reflection from rear surface of the lens—almost the same size and formed in the same plane as the first Purkinje image, but due to change in index of refraction at rear of lens, intensity is less than 1% of that of the first Purkinje image; IL, incoming light; A, aqueous humor; C, cornea; S, sclera; V, vitreous humor; I, iris; L, lens; CR, center of rotation; EA, eye axis; $a \approx 6$mm; $b \approx 12.5$mm; $c \approx 13.5$mm; $d \approx 24$mm; $r \approx 7.8$mm [Cra94].

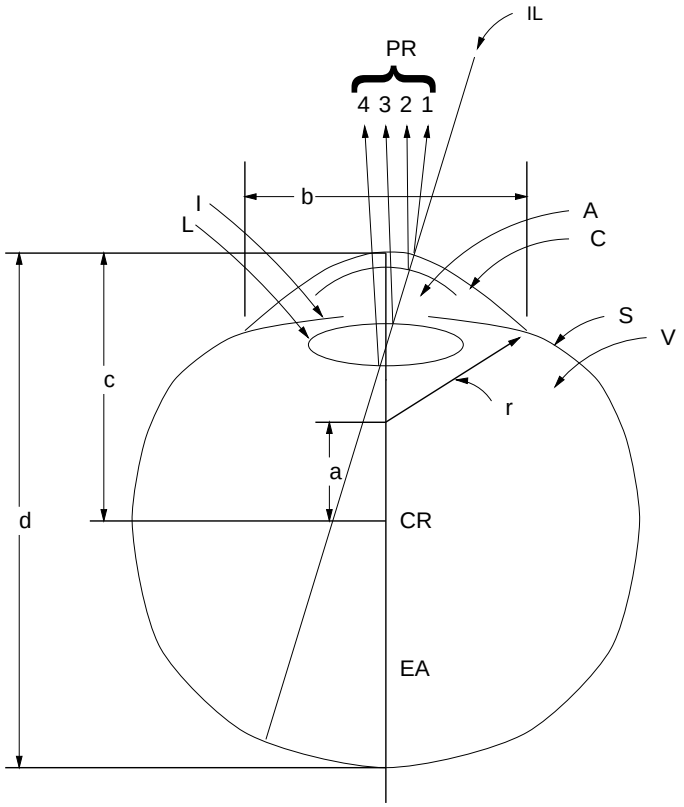


Figure 31: Purkinje images. Adapted from [Cra94, p.19 (Fig. 1)].

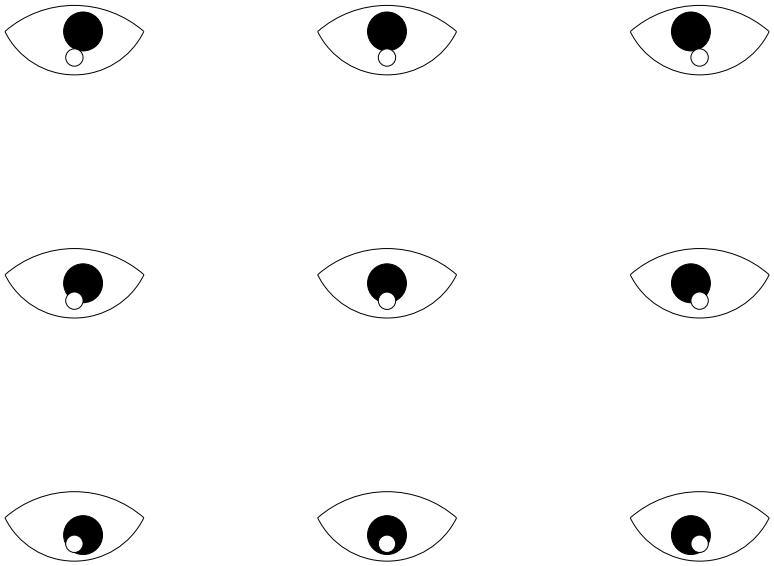


Figure 32: Relative positions of pupil and first Purkinje images as seen by the eye tracker's camera.

placed calibration points. The Purkinje reflection is shown as a small white circle in close proximity to the pupil, represented by a black circle. Since the infra-red light source is usually placed at some fixed position relative to the eye, the Purkinje image is relatively stable while the eyeball, and hence the pupil, rotates in its orbit. So-called generation-V eye trackers also measure the fourth Purkinje image [CS85]. By measuring the first and fourth Purkinje reflections, these dual-Purkinje image (DPI) eye trackers separate translational and rotational eye movements. Both reflections move together through exactly the same distance upon eye translation but the images move through different distances, thus changing their separation, upon eye rotation. This device is shown in Figure 33. Unfortunately, although the DPI eye tracker

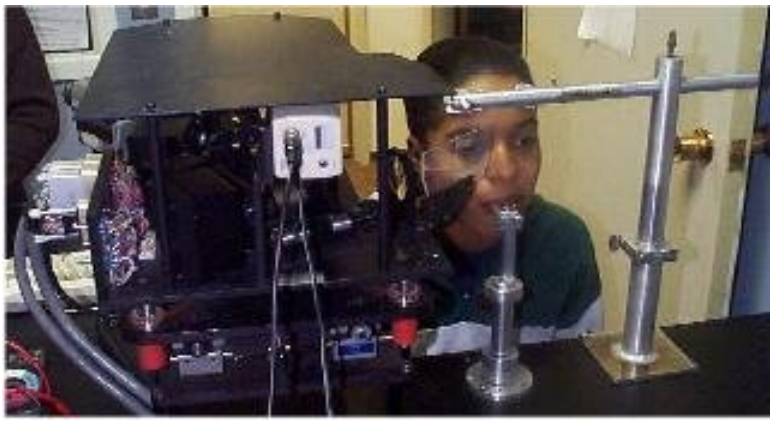


Figure 33: Dual-Purkinje eye tracker. From Fourward Optical Technologies [Fou00].

is quite precise, head stabilization may be required.

E.2 Summary and Further Reading

For a short review of early eye tracking methods, see [Rob68, II]. For another relatively more recent survey of eye tracking techniques, see [YS75]. Although Young and Sheena's survey article is somewhat dated by today's standards, it is an excellent introductory article on eye movement recording techniques, and is still widely referenced. An up-to-date survey of eye tracking devices does not appear to be available, although the number of (video-based) eye tracking companies seems to be growing. Instead, one of the best comprehensive lists of eye tracking manufacturers is available the on the Internet, the Eye Movement Equipment Database (EMED), URL: `<http://ibs.derby.ac.uk/emed/>`, maintained by David Wooding of the Institute of Behavioural Sciences, University of Derby, UK.

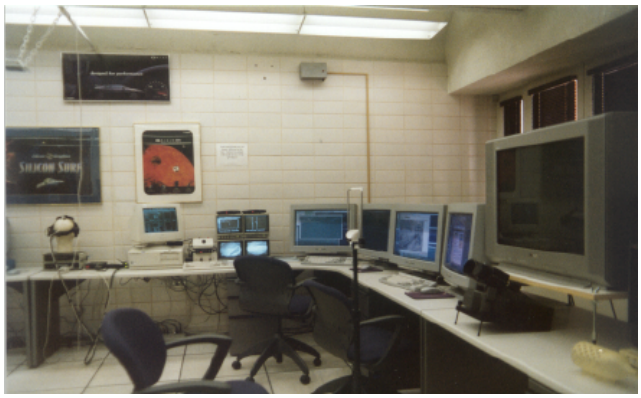
F Integration Issues and Requirements

Several types of eye trackers are available, ranging from scleral coils, EOG, to video-based corneal reflection eye trackers. While each has its advantages and drawbacks (e.g., accuracy vs. sampling rate), arguably for graphical or interactive applications the video-based corneal reflection trackers are the most practical devices. These devices work by capturing video images of the eye (illuminated by an infra-red light source), processing the video frames (at video frame rates) and outputting the eye's *x*- and *y*-coordinates relative to the screen being viewed. The *x*- and *y*-coordinates are typically either stored by the eye tracker itself, or can be sent to the graphics host via serial cable. The advantage of the video-based eye tracker over other devices is that it is relatively non-invasive, fairly accurate (to about 1° visual angle over a 30° viewing range), and, for the most part, not difficult to integrate with the graphics system. The video-based tracker's chief limitation is its sampling frequency, typically limited by the video frame rate, 60Hz. Hence, one can usually expect to receive eye movement samples at least every 16ms (typically a greater latency should be expected since the eye tracker needs time to process each video frame, and the graphics host needs to take time to update its display).

Integration of the eye tracker into a graphics system depends chiefly on proper delivery of the graphics video stream to the eye tracker and the subsequent reception of the eye tracker's calculated 2D eye coordinates. In the following description of system setup, a complete graphics system is described featuring two eye tracking systems: one a table-mounted, monocular eye tracker set underneath a standard NTSC television display, the other a binocular eye tracker fitted inside an HMD. Both displays are powered by the same graphics system. Such a laboratory has been set up within Clemson University's Virtual Reality Laboratory, and is shown in Figure 34. In Figure 34(b), the portion of the lab is shown, where, on the right, the TV display is installed with the monocular table-mounted eye tracker positioned below, and on the left, the HMD helmet is resting atop the d.c. electromagnetic head tracking units. The eye tracker unit (a PC) is situated between the HMD and the dual-head graphics display monitors to its right. Both HMD and TV (and graphics) displays are driven by the graphics host, which is out of view of the image (to the right and behind the camera). Figure 34(a) shows a close-up view of the eye tracker PC and the HMD. The four small TV monitors display the left and right scene images (what the user sees) and the left and right eye images (what the eye tracker sees).

The following system integration description is based on the particular hardware devices installed at Clemson's Virtual Reality Eye Tracking (VRET) laboratory, described here for reference. The primary rendering engine is a dual-rack, dual-pipe, SGI Onyx2® InfiniteReality™

(a) HMD-mounted binocular eye tracker.



(b) Table-mounted monocular eye tracker.

Figure 34: Virtual Reality Eye Tracking (VRET) lab at Clemson University.

system with 8 raster managers and 8 MIPS® R10000™ processors, each with 4Mb secondary cache.[5] It is equipped with 3Gb of main memory and 0.5Gb of texture memory. The eye tracker is from ISCAN and the system includes both binocular cameras mounted within a Virtual Research V8 (high resolution) Head Mounted Display (HMD) as well as a monocular camera mounted on a remote pan-tilt unit. Both sets of optics function identically, capturing video images of the eye and sending the video signal to the eye tracking PC for processing. The pan-tilt unit coupled with the remote table-mounted camera/light assembly is used to non-invasively track the eye in real-time as the subject's head moves. This allows limited head movement, but a chin-rest may optionally be used to restrict the position of the subject's head during experimentation to improve accuracy. 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 the user with directional motion control. The HMD is equipped with headphones for audio localization.

Although the following integration and installation guidelines are based on the equipment available at the Clemson VRET lab, the instructions should apply to practically any video-based corneal reflection eye tracker. Of primary importance to proper system integration are the following:

1. knowledge of the video format the eye tracker requires as input (e.g., NTSC or VGA);
2. knowledge of the data format the eye tracker generates as its output.

The first point is crucial to providing the proper image to the user as well as to the eye tracker. The eye tracker requires input of the scene signal so that it can overlay the calculated point of regard for display on the scene monitor which is viewed by the operator (experimenter). The second requirement is needed for proper interpretation of the point of regard by the host system. This is usually provided by the eye tracker manufacturer. The host system will need to be furnished with a device driver to read and interpret this information.

Secondary requirements for proper integration are the following:

1. the capability of the eye tracker to provide fine-grained cursor control over its calibration stimulus (a cross-hair or other symbol);
2. the capability of the eye tracker to transmit its operating mode to the host along with the eye x- and y point of regard information.

These two points are both related to proper alignment of the graphics display with the eye tracking scene image and calibration point. Essentially, both eye tracker capabilities are required to properly map between the graphics and eye tracking viewport coordinates. This alignment is carried out in two stages. First, the operator can use the eye tracker's own calibration stimulus to read out the extents of the eye tracker's displays. The eye tracker resolution may be specified over a 512×512 range, however, in practice it may be difficult to generate a graphics window which will exactly match the dimensions of the NTSC display. Differences on the order of 1 or 2 pixels will ruin proper alignment. Therefore, it is good practice to first display a blank graphics window, then use the eye tracker cursor to measure the extents of the window in the eye tracker's reference frame. Since this measurement should be as precise as possible, fine cursor control is needed. Second, the eye tracker operates in three primary modes: reset (inactive), calibration, and run. The graphics program must synchronize with these modes, so that a proper display can be generated in each mode:

- Reset: display nothing (black screen) or a single calibration point.
- Calibration: display a simple small stimulus for calibration (e.g., a small dot or circle) at the position of the eye tracker's calibration stimulus.
- Run: display the stimulus required over which eye movements are to be recorded.

It is of the utmost importance that proper alignment is achieved between the eye tracker's calibration stimulus points and those of the graphics' display. Without this alignment, the data returned by the eye tracker will be meaningless.

Proper alignment of the graphics and eye tracking reference frames is achieved through a simple linear mapping between eye tracking and graphics window coordinates. This is discussed in Section F.2, following notes on eye tracking system installation and wiring.

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

F.1 System Installation

The two primary installation considerations are wiring of the video and serial line cables between the graphics host and eye tracking systems. The connection of the serial cable is comparatively simple. Generation of the software driver for data interpretation is also straightforward, and is usually facilitated by the vendor's description of the data format and transmission rate. In contrast, what initially may pose the most problems is the connection of the video signal. It is imperative that the graphics host can generate a video signal in the format expected by both the graphics display (e.g., television set or HMD unit) and the eye tracker.

In the simplest case, if the host computer is capable of generating a video signal that is suitable for both the stimulus display and eye tracker, all that is then needed is a video splitter which will feed into both the stimulus display and eye tracker. For example, assume that the stimulus display is driven by an NTSC signal (e.g., a television), and the host computer is capable of generating a display signal in this format. (This is possible at the Clemson VRET lab since the SGI host can send a copy of whatever is on the graphics display via proper use of the `ircombine` command.) If the eye tracker can also be driven by an NTSC signal, then the installation is straightforward. If, however, the stimulus display is driven by VGA video (e.g., an HMD), but the eye tracker is driven by NTSC video, then the matter is somewhat more complicated. Consider the wiring diagram given in Figure 35. This schematic shows the dual display components, the HMD and TV, used for

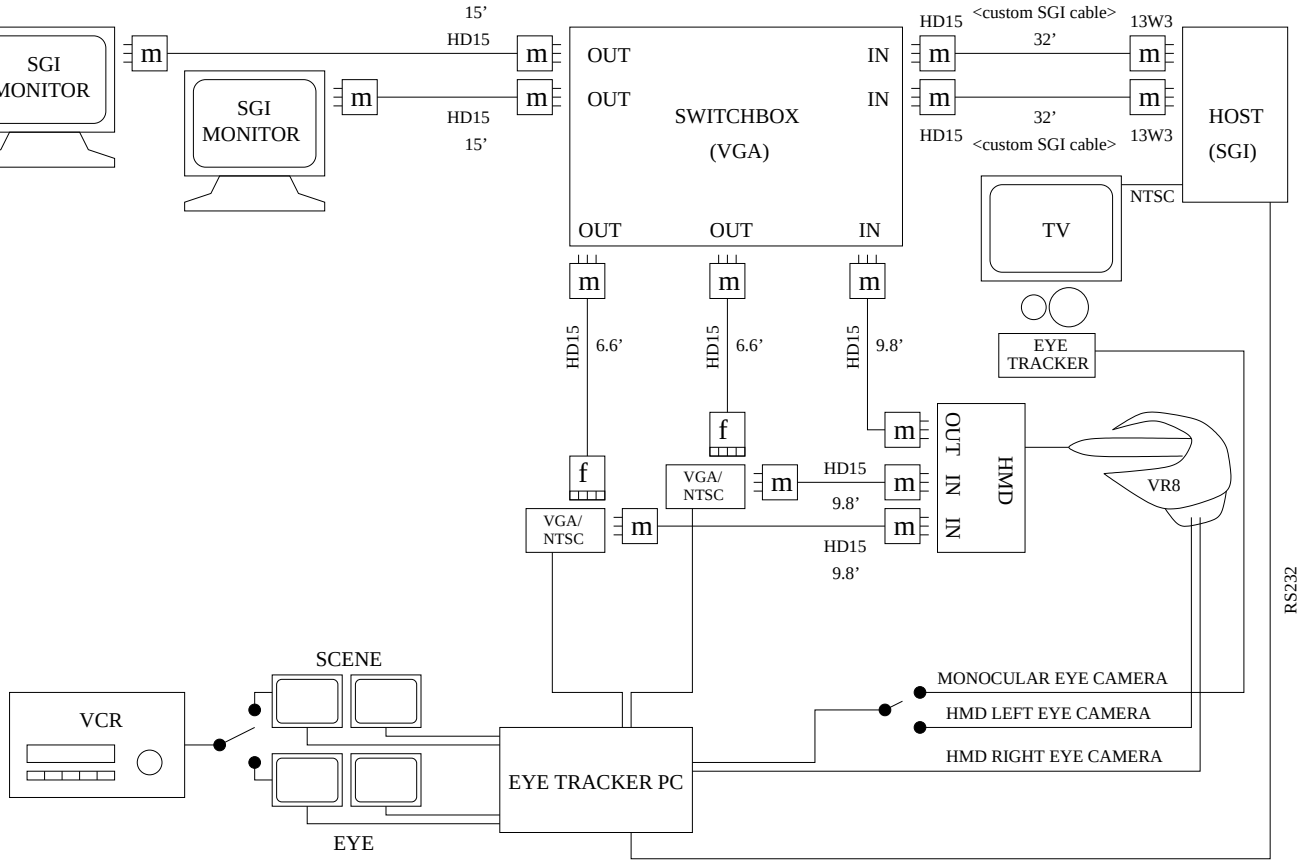


Figure 35: Video signal wiring of the VRET lab at Clemson University.

binocular eye tracking in a virtual reality environment, and monocular eye tracking over a 2D image or video display. The diagram features the necessary wiring of both left and right video channels from the host to the HMD and eye tracker, and a copy of the left video channel sent to the TV through the host's NTSC video output.

The HMD is driven by a horizontal-synch (h-sync) VGA signal. A switchbox is used (seen in Figure 34(a) just above the eye tracker keyboard) to switch the VGA display from either the dual-head graphics monitors or the HMD. The HMD video control box diverts a copy of the left video channel through an active pass-through splitter back through the switchbox to the left graphics display monitor. The switchbox effectively "steals" the signal meant for the graphics displays and sends it to the HMD. The left switch on the switchbox has two settings: monitor or HMD. The right switch on on the switchbox has three settings: monitor, HMD, or both. If the right switch is set to monitor, no signal is sent to the HMD, effectively providing a biocular display in the HMD (instead of a binocular, or stereoscopic display). If the right switch is set to HMD, the graphics display blanks out since the HMD does not provide an active pass-through of the right video channel. If the right switch is set to both, the right video channel is simply split between the HMD and the monitor, resulting in a binocular display in both the HMD and on the monitors. This last setting provides no amplification of the signal and hence both the right LCD in the HMD and the right graphics monitor displays appear dim. This is mostly used for testing purposes.

The entire video circuit between the graphics host, the switchbox, and the HMD is VGA video. The eye tracker, however, operates on NTSC. This is the reason for the two VGA/NTSC converters which are inserted into the video path. These converters output an NTSC signal to the eye tracker and also provide active pass-throughs for the VGA signal so that, when in operation, the VGA signal appears undisrupted. The eye tracker then processes the video signals of the scene and outputs the signal to the scene monitors, with its own overlayed signal containing the calculated point of regard (represented by a cross-hair cursor). These two small displays show the operator what is in the user's field of view as well as as what s/he is looking at.

The eye images, in general, do not pose a complication since this video signal is exclusively processed by the eye tracker. In the case of the eye tracker at Clemson's VRET lab, both the binocular HMD eye cameras and the table-mounted monocular camera are NTSC and the signals feed directly into the eye tracker.

F.1.1 Notes and lessons learned from the installation at Clemson

The eye tracker at Clemson contains hardware to process dual left and right eye and scene images. It can be switched to operate in monocular mode, for which it requires just the left eye and scene images. In this case, a simple video switch is used to switch the signal between the eye image generated by the left camera in the HMD and the camera on the table-mounted unit.

The SGI display monitors used at Clemson can be driven by either VGA video, or the default R, G, B video delivered by 13W3 video cables. To drive the HMD, VGA video was required, connected by HD15 cables. To connect the video devices properly, special 13W3-HD15 cables were needed. Although this seems trivial, custom-built cables were required. These cables are not cheap, and take a day or two to build and deliver. If timing and finances are a consideration, planning of the system down to the proper cabling is a must!

A problem that was particularly difficult to troubleshoot during the Clemson installation was the diagnosis of the format of the VGA signal emitted by the SGI host computer. Initially, before the eye tracker was installed, the HMD was tested for proper display. The output of the switchbox was used directly to drive the HMD. Everything functioned properly. However, after inserting it into the video circuit, the eye tracker would not work. It was later found that the problem lay in the VGA/NTSC converters: these converters expect the more common VGA signal which uses a timing signal synchronized to the horizontal video field (h-sync signal; the horizontal and vertical sync signals are found on pins 13 and 14 of the VGA HD15 cable). The SGI host computer by default emits a sync-on-green VGA signal leaving pins 13 and 14 devoid of a timing signal. The VR HMD contains circuitry which will read either h-sync or sync-on-green VGA video and so functions quietly given either signal. The fault, as it turns out, was in an improper initial wiring of the switchbox. The switchbox was initially missing connections for pins 13 and 14 since these are not needed by a sync-on-green VGA signal. Once this was realized, the entire cable installation had to be disassembled and the switchbox had to be rewired. With the switchbox rewired, the custom 13W3 cables had to be inspected to confirm that these components carried signals over pins 13 and 14. Finally, a new display configuration had to be created (using SGI's `ircombine` command) to drive the entire circuit with the horizontal sync signal instead of the default sync-on-green. The moral here is: be sure of the video format required by all components, down to the specific signals transmitted on individual cable pins!

F.2 Application Program Requirements

In designing a graphical eye tracking application, the most important requirement is mapping of eye tracker coordinates to the appropriate application program's reference frame. The eye tracker calculates the viewer's point of regard (POR) 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 POR at each sampling cycle (e.g., once every $\sim$16ms for a 60Hz device). This coordinate pair must be mapped to the extents of the application program's viewing window.

If a binocular eye tracker is used, two coordinate sample pairs are returned during each sampling cycle, x_l, y_l for the left POR and x_r, y_r for the right eye. A VR application must, in the same update cycle, also map the coordinates of the head's position and orientation tracking device (e.g., typically a 6DOF tracker is used).

The following sections discuss the mapping of eye tracker screen coordinates to application program coordinates for both monocular and binocular applications. In the monocular case, a typical 2D image-viewing application is expected, and the coordinates are mapped to the 2D (orthogonal) viewport coordinates accordingly (the viewport coordinates are expected to match the dimensions of the image being displayed). In the binocular (VR) case, the eye tracker coordinates are mapped to the dimensions of the near viewing plane of the viewing frustum. For both calculations, a method of measuring the application's viewport dimensions in the eye tracker's reference frame is described, where the eye tracker's own (fine-resolution) cursor control is used to obtain the measurements. This information should be sufficient for most 2D image viewing applications.

For VR applications, subsequent sections describe the required mapping of the head position/orientation tracking device. While this section should generalize to most kinds of 6DOF tracking devices, the discussion in some cases is specific to the Ascension Flock Of Birds d.c. electromagnetic 6DOF tracker. This section discusses how to obtain the head-centric view and vectors from the matrix returned by the tracker, and also explains the transformation of an arbitrary vector using the obtained transformation matrix.[6] This latter derivation is used to transform the gaze vector to head-centric coordinates, which is initially obtained from, and relative to, the binocular eye tracker's left and right POR measurement.

Finally, a derivation of the calculation of the gaze vector in 3D is given, and a method is suggested for calculating the three-dimensional gaze point in VR.

[6] Equivalently, the rotation quaternion be used, if this data is available from the head tracker.

F.2.1 Mapping Eye Tracker Screen Coordinates to Application-Specific Coordinates

When working with the eye tracker, the data obtained from the tracker must be mapped to a range appropriate to the given application. If working in VR, the 2D eye tracker data, expressed in eye tracker screen coordinates, must be mapped to the 2D dimensions of the near viewing frustum. If working with images, the 2D eye tracker data must be mapped to the 2D display image coordinates.

In general, if $x' \in [a,b]$ needs to be mapped to the range $[c,d]$, we have:

$$x = c + \frac{(x'-a)(d-c)}{(b-a)} \tag{5}$$

This is a linear mapping between two (one-dimensional) coordinate systems (or lines in this case). Equation (5) has a straightforward interpretation:

1. The value x' is translated (shifted) to its origin, by subtracting a.
2. The value $(x'-a)$ is then normalized by dividing through by the range $(b-a)$.
3. The normalized value $(x'-a)/(b-a)$ is then scaled to the new range $(d-c)$.
4. Finally, the new value is translated (shifted) to its proper relative location in the new range by adding c.

Mapping Eye Tracker Screen Coordinates to the 3D Viewing Frustum The 3D viewing frustum employed in the perspective viewing transformations is defined by the parameters `left, right, bottom, top, near, far`, e.g., as used in the `OpenGL` function call `glFrustum()`. Figure 36 shows the dimensions of the eye tracker screen (left) and the dimensions of the viewing frustum (right). Note that the eye tracker origin is the top-left of the screen while the viewing frustum's origin is bottom-left (this is a common discrepancy

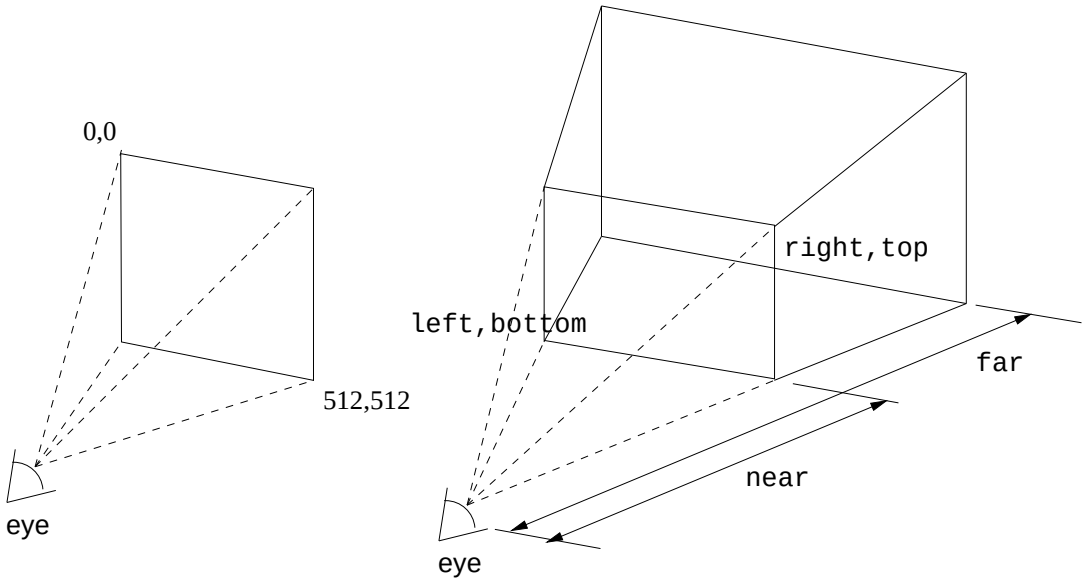


Figure 36: Eye tracker to VR mapping.

between imaging and graphics). To convert the eye tracker coordinates (x',y') to graphics coordinates (x,y), using Equation (5), we have:

$$x = \texttt{left} + \frac{x'(\texttt{right}-\texttt{left})}{512} \tag{6}$$

$$y = \texttt{bottom} + \frac{(512-y')(\texttt{top}-\texttt{bottom})}{512} \tag{7}$$

Note that the term $(512-y')$ in Equation (7) handles the y-coordinate mirror transformation so that the top-left origin of the eye tracker screen is converted to the bottom-left of the viewing frustum.

If typical dimensions of the near plane of the graphics viewing frustum are 640×480, with the origin at $(0,0)$, then Equations (6) and (7) reduce to:

$$x = \frac{x'(640)}{512} = x'(1.3) \tag{8}$$

$$y = \frac{(512-y')(480)}{512} = (512-y')(.9375) \tag{9}$$

Mapping Eye Tracker Screen Coordinates to the 2D Image Coordinates The linear mapping between eye tracker screen coordinates and 2D image coordinates is handled similarly. If the image dimensions are 640×480, then Equations (8) and (9) are used without change. Note that an image with dimensions 600×450 appears to fit the display of the TV in the Clemson VRET lab better.[7] In this case, Equations (8) and (9) become:

$$x = \frac{x'(600)}{512} = x'(1.171875) \tag{10}$$

$$y = \frac{(512-y')(450)}{512} = (512-y')(.87890625) \tag{11}$$

Measuring Eye Tracker Screen Coordinate Extents for Coordinate Mapping The above coordinate mapping procedures assume 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. For example, if an image display program runs wherein an image is displayed in a 600×450 window, and this window is positioned arbitrarily on the graphics console, then the ISCAN coordinates of interest are restricted only to the area covered by the application window, as seen on the small ISCAN black and white scene monitors. Since the window is not likely to cover the entire NTSC display, as seen in the monitors, only a restricted range of ISCAN coordinates will be of relevance to the eye tracking experiment.

The trick to the calculation of the proper mapping between ISCAN and application coordinates is the measurement of the application window's extents, in the ISCAN reference frame. This is accomplished by using the eye tracker itself. One of the eye tracker's most useful features, specifically for this purpose, is its ability to move its crosshair cursor. The software allows fine movement (pixel by pixel), or coarse movement (in jumps of 10 pixels). Furthermore, the ISCAN data screen at the bottom of the black/white monitor displays the coordinates of the cursor, relative to ISCAN's reference frame.

To obtain the extents of the application window in ISCAN's reference frame, simply move the cursor to the corners of the application window. Use these coordinates in the above mapping formulae. The following measurement "recipe" should provide an almost exact mapping, and only needs to be performed once. Assuming the application window's dimensions are fixed, the mapping obtained from this procedure can be hardcoded into the application. Here are the steps:

1. Position your display window so that it covers the display fully, e.g., in the case of the image stimulus, the window should just cover the entire TV display.
2. Use ISCAN's cursor positioning utility to measure the viewing area's extents (use ISCAN's function keys to toggle between course and fine cursor movement).
3. Calculate the mapping.
4. In Reset mode, position ISCAN cursor in the middle of the display.

An important consequence of the above is that you can always position the application window in the same place, provided that the program displays the calibration point obtained from the eye tracker mapped to local image coordinates. When the application starts up (and the eye tracker is on and in Reset mode), simply position the application so that the central calibration points (displayed when the tracker is in Reset mode) line up.

As an example, consider a 600×450 image display application. Once the window is positioned so that it fully covers the TV display, it may appear slightly off-center on the ISCAN black/white monitor, as sketched in Figure 37. In this example, assume that if the ISCAN cursor is moved to the corners of the drawable application area, the measurements are as shown in Figure 37. Based on those measurements, the mapping is:

$$x = \frac{x'-51}{(482-51+1)}(600) \tag{12}$$

$$y = 449-\frac{y'-53}{(446-53+1)}(450) \tag{13}$$

The central point on the ISCAN display is $(267,250)$. Note that y is subtracted from 449 to take care of the image/graphics vertical origin flip.

F.2.2 Mapping Flock Of Birds Tracker Coordinates to Virtual World Coordinates

In Virtual Reality, the position and orientation of the head is typically delivered by a real-time head tracker. In our case, we have a Flock Of Birds (FOB) d. c. electromagnetic tracker from Ascension. The tracker reports 6 Degree Of Freedom (6DOF) information regarding sensor position and orientation. The latter is given in terms of Euler angles. Euler angles determine the orientation of a vector in three-space by specifying the required rotations of the origin's coordinate axes. These angles are known by several names, but in essence each describes a particular rotation angle about one of the principal axes. Common names describing Euler angles are given in Table 3 and the geometry is sketched in Figure 38, where roll, pitch, and yaw angles are represented by R, E, and A, respectively. Each of these rotations is represented

[7] A few of the pixels of the image do not fit on the TV display, possibly due to the NTSC flicker-free filter used to encode the SGI console video signal.

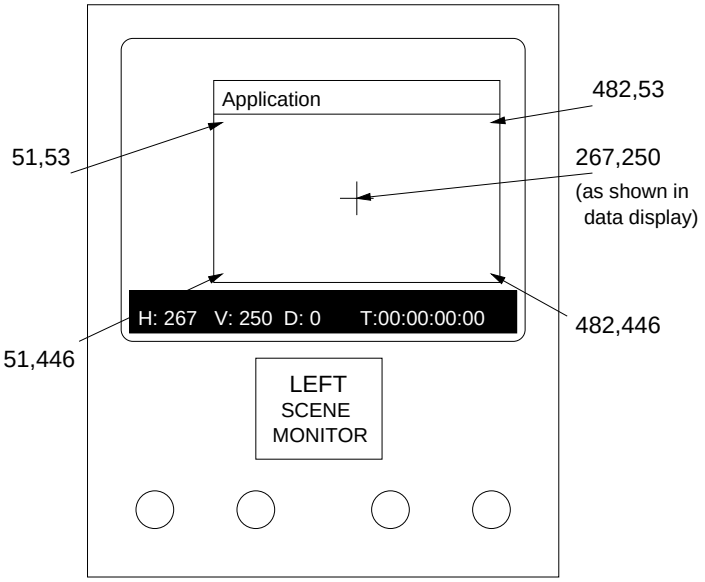


Figure 37: Example mapping measurement.

rot y	rot x	rot x
yaw	pitch	roll
azimuth	elevation	roll
longitude	latitude	roll

Table 3: Euler angles.

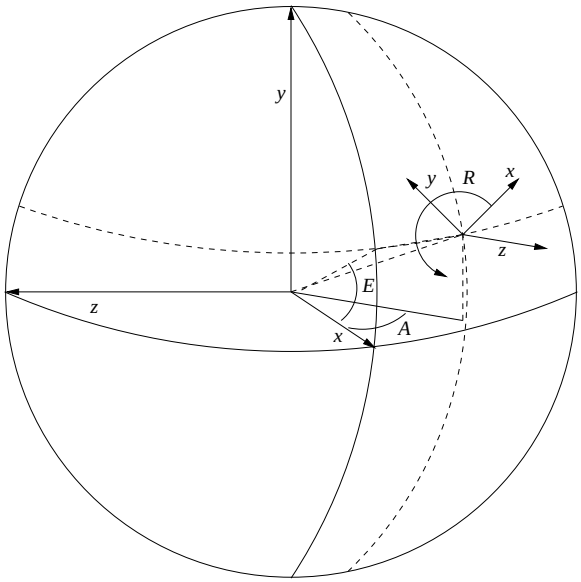


Figure 38: Euler angles.

by the familiar homogeneous rotation matrices:

$$\mathbf{R}_z = \begin{bmatrix} \cos R & \sin R & 0 & 0 \\ -\sin R & \cos R & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \mathbf{R}_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos E & \sin E & 0 \\ 0 & -\sin E & \cos E & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \mathbf{R}_y = \begin{bmatrix} \cos A & 0 & -\sin A & 0 \\ 0 & 1 & 0 & 0 \\ \sin A & 0 & \cos A & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{14}$$

The composite 4×4 matrix, containing all of the above transformations rolled into one, is:

$$\mathbf{M} = \mathbf{R}_z\mathbf{R}_x\mathbf{R}_y = \begin{bmatrix} \cos R\cos A + \sin R\sin E\sin A & \sin R\cos E & -\cos R\sin A + \sin R\sin E\cos A & 0 \\ -\sin R\cos A + \cos R\sin E\sin A & \cos R\cos E & \sin R\sin A + \cos R\sin E\cos A & 0 \\ \cos E\sin A & -\sin E & \cos E\cos A & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{15}$$

The FOB delivers a similar 4×4 matrix, **F**,

$$\mathbf{F} = \begin{bmatrix} \cos E\cos A & \cos E\sin A & -\sin E & 0 \\ -\cos R\sin A + \sin R\sin E\cos A & \cos R\cos A + \sin R\sin E\sin A & \sin R\cos E & 0 \\ \sin R\sin A + \cos R\sin E\cos A & -\sin R\cos A + \cos R\sin E\sin A & \cos R\cos E & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{16}$$

where the matrix elements are slightly rearranged, such that $\mathbf{M}[i,j] = \mathbf{F}[(i+1)\%3][(j+1)\%3]$, e.g., row 1 of matrix **M** is now row 2 in matrix **F**, row 2 is now row 3, and row 3 is now row 1. Columns are interchanged similarly, where column 1 of matrix **M** is now column 2 in matrix **F**, column 2 is now column 3, and column 3 is now column 1. This "off-by-one" shift present in the Bird matrix may be due to the non-C style indexing which starts at 1 instead of 0.

Obtaining the Transformed View Vector To calculate the transformed view vector, $\mathbf{v}'$, assume initial view vector is $\mathbf{v} = (0,0,-1)$ (looking down *z*-axis), and apply the composite transformation (in homogeneous coordinates):[8]

$$\mathbf{vM} = \begin{bmatrix} 0 \\ 0 \\ -1 \\ 1 \end{bmatrix}^T \begin{bmatrix} \cos R\cos A + \sin R\sin E\sin A & \sin R\cos E & -\cos R\sin A + \sin R\sin E\cos A & 0 \\ -\sin R\cos A + \cos R\sin E\sin A & \cos R\cos E & \sin R\sin A + \cos R\sin E\cos A & 0 \\ \cos E\sin A & -\sin E & \cos E\cos A & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{17}$$

$$= \begin{bmatrix} -\cos E\sin A & \sin E & \cos E\cos A & 1 \end{bmatrix} \tag{18}$$

which is simply the third row of **M**, negated. By inspection of the Bird matrix **F**, given by Equation (16), the view vector is simply obtained by selecting appropriate entries in the matrix, i.e.,

$$\mathbf{v}' = \begin{bmatrix} -\mathbf{F}[0,1] & -\mathbf{F}[0,2] & \mathbf{F}[0,0] & 1 \end{bmatrix} \tag{19}$$

Obtaining the Transformed Up Vector The transformed up vector is obtained in a similar manner to the transformed view vector. To calculate the transformed up vector, $\mathbf{u}'$, assume the initial up vector is $\mathbf{u} = (0,1,0)$ (looking up *y*-axis), and apply the composite transformation (in homogeneous coordinates):

$$\mathbf{uM} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}^T \begin{bmatrix} \cos R\cos A + \sin R\sin E\sin A & \sin R\cos E & -\cos R\sin A + \sin R\sin E\cos A & 0 \\ -\sin R\cos A + \cos R\sin E\sin A & \cos R\cos E & \sin R\sin A + \cos R\sin E\cos A & 0 \\ \cos E\sin A & -\sin E & \cos E\cos A & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{20}$$

$$= \begin{bmatrix} -\sin R\cos A + \cos R\sin E\sin A & \cos R\cos E & \sin R\sin A + \cos R\sin E\cos A & 1 \end{bmatrix} \tag{21}$$

By inspection of the Bird matrix **F**, given by Equation (16), the transformed up vector is simply obtained by selecting appropriate entries in the matrix, i.e.,

$$\mathbf{u}' = \begin{bmatrix} \mathbf{F}[2,1] & \mathbf{F}[2,2] & \mathbf{F}[2,0] & 1 \end{bmatrix} \tag{22}$$

Because of our setup in the lab, the *z*-axis is on the opposite side of the Bird transmitter (behind the Bird emblem on the transmitter). For this reason, the *z*-component of the up vector is negated, i.e.,

$$\mathbf{u}' = \begin{bmatrix} \mathbf{F}[2,1] & \mathbf{F}[2,2] & -\mathbf{F}[2,0] & 1 \end{bmatrix} \tag{23}$$

Note that the negation of the *z*-component of the transformed view vector does not make a difference since the term is a product of cosines.

[8] Recall trigonometric identities: $\sin(-\theta) = -\sin(\theta)$ and $\cos(-\theta) = \cos(\theta)$.

Transforming an Arbitrary Vector To transform an arbitrary vector, an operation similar to the transformations of the up and view vectors is performed. To calculate the transformed arbitrary vector, $\mathbf{w} = \begin{bmatrix} x & y & z & 1 \end{bmatrix}$, apply the composite transformation by multiplying by the transformation matrix **M** (in homogeneous coordinates):

$$\mathbf{wM} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}^T \begin{bmatrix} \cos R\cos A + \sin R\sin E\sin A & \sin R\cos E & -\cos R\sin A + \sin R\sin E\cos A & 0 \\ -\sin R\cos A + \cos R\sin E\sin A & \cos R\cos E & \sin R\sin A + \cos R\sin E\cos A & 0 \\ \cos E\sin A & -\sin E & \cos E\cos A & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{24}$$

$$= \begin{bmatrix} x(\cos R\cos A + \sin R\sin E\sin A) + y(-\sin R\cos A + \cos R\sin E\sin A) + z(\cos E\sin A) \\ x(\sin R\cos E) + y(\cos R\cos E) + z(-\sin E) \\ x(-\cos R\sin A + \sin R\sin E\cos A) + y(\sin R\sin A + \cos R\sin E\cos A) + z(\cos E\cos A) \\ 1 \end{bmatrix}^T \tag{25}$$

To use the Bird matrix, there is unfortunately no simple way to select the appropriate matrix elements without hardcoding Equation (25). Probably the best bet would be to undo the "off-by-one" shift present in the Bird matrix. On the other hand, hardcoding the solution may be the fastest method. This rather inelegant, but luckily localized, operation looks like this:

$$\mathbf{w}' = \begin{bmatrix} x\mathbf{F}[1,1] + y\mathbf{F}[2,1] + z\mathbf{F}[0,1] \\ x\mathbf{F}[1,2] + y\mathbf{F}[2,2] + z\mathbf{F}[0,2] \\ x\mathbf{F}[1,0] + y\mathbf{F}[2,0] + z\mathbf{F}[0,0] \\ 1 \end{bmatrix}^T \tag{26}$$

Furthermore, as in the up vector transformation, it appears that the negation of the z component may also be necessary. If so, the above equation will need to be rewritten as:

$$\mathbf{w}' = \begin{bmatrix} x\mathbf{F}[1,1] + y\mathbf{F}[2,1] + z\mathbf{F}[0,1] \\ x\mathbf{F}[1,2] + y\mathbf{F}[2,2] + z\mathbf{F}[0,2] \\ -(x\mathbf{F}[1,0] + y\mathbf{F}[2,0] + z\mathbf{F}[0,0]) \\ 1 \end{bmatrix}^T \tag{27}$$

F.2.3 3D Gaze Point Calculation

The calculation of the gaze point in three-space depends only on the relative positions of the two eyes in the horizontal axis. The parameters of interest here are the three-dimensional virtual coordinates of the gaze point, (x_g, y_g, z_g), which can be determined from traditional stereo geometry calculations. Figure 39 illustrates the basic binocular geometry. Helmet tracking determines helmet position and orthogonal directional and up vectors, which determine viewer-local coordinates shown in the diagram. The helmet position is the origin, (x_h, y_h, z_h), the helmet directional vector is the optical (viewer-local z) axis, and the helmet up vector is the viewer-local y-axis.

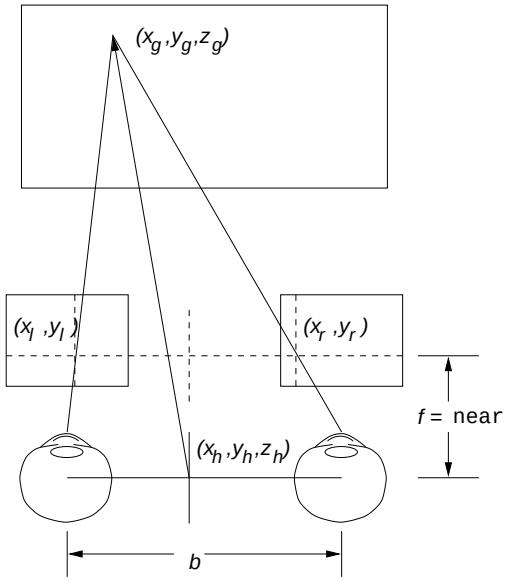


Figure 39: Basic binocular geometry.

Given instantaneous, eye tracked, viewer-local coordinates (*mapped from eye tracker screen coordinates to the near view plane coordinates*), (x_l, y_l) and (x_r, y_r) in the left and right view planes, at focal distance f along the viewer-local z-axis, we can determine viewer-local coordinates

of the gaze point, (x_g, y_g, z_g) by deriving the stereo equations parametrically. First, express both the left and right view lines in terms of the linear parameter s. These lines originate at the eye centers $(x_h - b/2, y_h, z_h)$ and $(x_h + b/2, y_h, z_h)$ and pass through (x_l, y_l, f) and (x_r, y_r, f), respectively. The left view line is (in vector form):

$$\left[\begin{array}{ccc} (1-s)(x_h - b/2) + sx_l & (1-s)y_h + sy_l & (1-s)z_h + sf \end{array} \right] \tag{28}$$

and the right view line is (in vector form):

$$\left[\begin{array}{ccc} (1-s)(x_h + b/2) + sx_r & (1-s)y_h + sy_r & (1-s)z_h + sf \end{array} \right] \tag{29}$$

where b is the disparity distance between the left and right eye centers, and f is the distance to the near viewing plane. To find the central view line originating at the local center (x_h, y_h, z_h), calculate the intersection of the left and right view lines by solving for s using the x-coordinates of both lines, as given in Equations (28) and (29):

$$\begin{aligned} (1-s)(x_h - b/2) + sx_l &= (1-s)(x_h + b/2) + sx_r \\ (x_h - b/2) - s(x_h - b/2) + sx_l &= (x_h + b/2) - s(x_h + b/2) + sx_r \\ s(x_l - x_r + b) &= b \\ s &= \frac{b}{x_l - x_r + b} \end{aligned} \tag{30}$$

The interpolant s is then used in the parametric equation of the central view line, to give the gaze gaze point at the intersection of both view lines:

$$\begin{aligned} x_g &= (1-s)x_h + s((x_l + x_r)/2) \\ y_g &= (1-s)y_h + s((y_l + y_r)/2) \\ z_g &= (1-s)z_h + sf \end{aligned}$$

giving:

$$x_g = \left(1 - \frac{b}{x_l - x_r + b}\right) x_h + \left(\frac{b}{x_l - x_r + b}\right) \left(\frac{x_l + x_r}{2}\right) \tag{31}$$

$$y_g = \left(1 - \frac{b}{x_l - x_r + b}\right) y_h + \left(\frac{b}{x_l - x_r + b}\right) \left(\frac{y_l + y_r}{2}\right) \tag{32}$$

$$z_g = \left(1 - \frac{b}{x_l - x_r + b}\right) z_h + \left(\frac{b}{x_l - x_r + b}\right) f \tag{33}$$

Eye positions (at viewer local coordinates $(x_h - b/2, y_h, z_h)$ and $(x_h + b/2, y_h, z_h)$), the gaze point, and an up vector orthogonal to the plane of the three points then determine the view volume appropriate for display to each eye screen.

The gaze point, as defined above, is given by the addition of a scaled offset to the view vector originally defined by the helmet position and central view line in virtual world coordinates.[9] The gaze point can be 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 (31), (32), and (33), we have:

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

or, in vector notation,

$$\mathbf{g} = \mathbf{h} + s\mathbf{v} \tag{34}$$

where $\mathbf{h}$ is the head position, $\mathbf{v}$ is the central view vector and s is the scale parameter as defined in Equation (30). Note that the view vector used here is not related to the view vector given by the head tracker. It should be noted that the view vector $\mathbf{v}$ 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.,

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

where $\mathbf{m}$ denotes the left and right eye coordinate midpoint. To transform the vector $\mathbf{v}$ to the proper (instantaneous) head orientation, this vector should be normalized, then multiplied by the orientation matrix returned by the head tracker (see Section F.2.2 in general and Section F.2.2 in particular). This new vector, call it $\mathbf{m}'$, should be substituted for $\mathbf{m}$ above to define $\mathbf{v}$ for use in Equation (34), i.e.,

$$\mathbf{g} = \mathbf{h} + s(\mathbf{m}' - \mathbf{h}) \tag{35}$$

[9] Note that 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.

Parametric Ray Representation of Gaze Direction Equation (35) gives the coordinates of the gaze point through a parametric representation (e.g., a point along a line) such that the depth of the three-dimensional point of regard (POR) in world coordinates is valid only if $s > 0$. Given the gaze point, $\mathbf{g}$ and the location of the helmet, $\mathbf{h}$, we can obtain just the three-dimensional gaze vector, ν which specifies the direction of gaze (but not the actual fixation point). This direction vector is given by:

$$\begin{aligned} \nu &= \mathbf{g} - \mathbf{h} && (36)\\ &= (\mathbf{h} + s\mathbf{v}) - \mathbf{h} && (37)\\ &= s\mathbf{v} && (38)\\ &= \left(\frac{b}{x_l - x_r + b}\right) \begin{bmatrix} (x_l + x_r)/2 - x_h \\ (y_l + y_r)/2 - y_h \\ (f - z_h) \end{bmatrix} && (39) \end{aligned}$$

where $\mathbf{v}$ is defined as either $\mathbf{m} - \mathbf{h}$ as before, or as $\mathbf{m}' - \mathbf{h}$ as in Equation (35). Given the helmet position $\mathbf{h}$ and the gaze direction ν, we can express the gaze direction via a parametric representation of a ray using a linear interpolant t:

$$\text{gaze}(t) \quad = \quad h + t\nu, \quad t > 0, \tag{40}$$

where h is the ray's origin (a point; the helmet position), and ν is the ray direction (the gaze vector). (Note that adding h to $t\nu$ results in the original expression of the gaze point, $\mathbf{g}$ given by Equation (34), provided $t = 1$.) The formulation of the gaze direction given by Equation (40) can then be used for testing virtual fixation coordinates via traditional ray/polygon intersection calculations commonly used in ray tracing.

F.2.4 Virtual Fixation Coordinates

In 3D eye tracking studies, we are often interested in knowing the location of one's gaze, or more importantly one's fixation, relative to some feature in the scene. In VR applications, we'd like to calculate the fixation location in the virtual world and thus identify the object of interest. The identification of the object of interest can be accomplished following traditional ray/polygon intersection calculations, as employed in ray tracing [Gla89].

The fixated object of interest is the one closest to the viewer which intersects the gaze ray. This object is found by testing all polygons in the scene for intersection with the gaze ray. The polygon closest to the viewer is then assumed to be the one fixated by the viewer (assuming all polygons in the scene are opaque).

Ray/Plane Intersection The calculation of an intersection between a ray and all polygons in the scene is usually obtained via a parametric representation of the ray, e.g.,

$$\text{ray}(t) \quad = \quad r_o + t\mathbf{r_d} \tag{41}$$

where r_o defines the ray's origin (a point), and $\mathbf{r_d}$ defines the ray direction (a vector). Note the similarity between Equations (41) and (40)—there, h is the head position, and ν is the gaze direction. To find the intersection of the ray with a polygon, calculate the interpolant value where the ray intersects each polygon, and examine all the intersections where $t > 0$. If $t < 0$, the object may intersect the ray, but behind the viewer.

Recall the plane equation $Ax + By + Cz + D = 0$, where $A^2 + B^2 + C^2 = 1$, i.e., A, B, and C define the plane normal. To obtain the ray/polygon intersection, substitute Equation (41) into the plane equation:

$$A(x_o + tx_d) + B(x_o + ty_d) + C(z_o + tz_d) + D = 0 \tag{42}$$

and solve for t:

$$t = -\frac{Ax_o + Bx_o + Cz_o + D}{Ax_d + By_d + Cz_d} \tag{43}$$

or, in vector notation:

$$t = \frac{-(\mathbf{N} \cdot r_o + D)}{\mathbf{N} \cdot \mathbf{r_d}} \tag{44}$$

A few observations of the above simplify the implementation:

(1) Here $\mathbf{N}$, the face normal, is really $-\mathbf{N}$, since what we're doing is calculating the angle between the ray and face normal. To get the angle, we need both ray and normal to be pointing in the same relative direction. This situation is depicted in Figure 40.
(2) In Equation (44), the denominator will cause problems in the implementation should it evaluate to 0. However, if the denominator is 0, i.e., if $\mathbf{N} \cdot \mathbf{r_d} = 0$, then the cosine between the vectors is 0, which means that the angle angle between the two vectors is 90° which means the ray and plane are parallel and don't intersect. Thus, to avoid dividing by zero, and to speed up the computation, evaluate the denominator first. If it is sufficiently close to zero, don't evaluate the intersection further, we know the ray and polygon will not intersect.
(3) Point (2) above can be further exploited by noting that if the dot product is greater than 0, then the surface is hidden to the viewer.

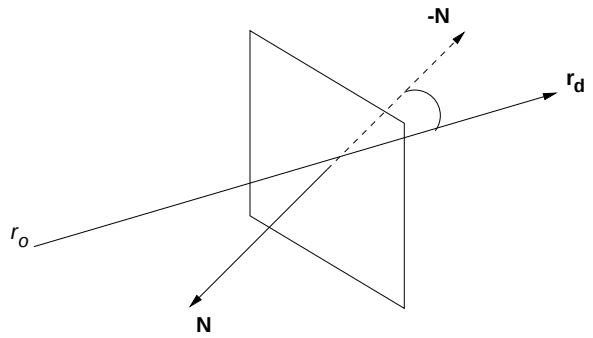


Figure 40: Ray/plane geometry.

```
v_d = N · r_d;                                  // denominator

if(v_d < 0) {
  v_o = −(N · r_o + D);                         // numerator
  t = v_o / v_d;
}
```

Figure 41: Ray/polygon intersection.

The first part of the intersection algorithm follows from the above and is given in Figure 41.

In the algorithm, the intersection parameter t defines the point of intersection along the ray at the plane defined by the normal $\mathbf{N}$. That is, if $t > 0$, then the point of intersection, p, is given by:

$$p = r_o + t\mathbf{r_d} \tag{45}$$

Note that the above only gives 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, we need to test the point p to see if it lies within the confines of the polygonal region, which is defined by the polygon's edges. This is essentially the "point-in-polygon" problem.

Point-In-Polygon Problem To test whether a point p lies inside a polygon (defined by its plane equation which specifies a plane of finite extent), we need to test the point against all edges of the polygon. To do this, the following algorithm is used:

For each edge:

1. Get the plane perpendicular to the face normal $\mathbf{N}$, which passes through the edge's two vertices A an B. The perpendicular face normal, $\mathbf{N}'$, is obtained by calculating the cross product of the original face normal with the edge, i.e.,

   $$\mathbf{N}' = \mathbf{N} \times (B - A)$$

   where the face vertices A and B are specified in counter-clockwise order. This is shown if Figure 42.

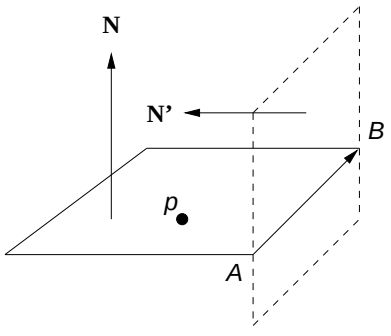


Figure 42: Point-In-Polygon geometry.

2. Get the perpendicular plane's equation by calculating D using either of A or B as the point on the plane (e.g., plug in either A or B into the plane equation, then solve for D).

3. Test point p to see if it lies "above" or "below" the perpendicular plane. This is done by plugging p into the perpendicular plane's plane equation and testing the result. If the result is greater than 0, then p is "above" the plane.
4. If p is "above" all perpendicular planes, as defined by successive pairs of vertices, then the point is "boxed in" by all the polygon's edges, and so must lie inside the polygon as originally defined by it face normal **N**.

F.3 System Calibration and Usage

Currently, most video-based eye trackers require calibration. This is usually a sequence of simple stimuli displayed sequentially at far extents of the viewing region. The eye tracker calculates the point of regard by measuring the relative observed position of the pupil and corneal reflection at these locations, and then (probably) interpolates the POR value at intermediate eye positions. The individual stimuli used for this purpose are simple white dots or cross-hairs on a black background.

If the application program is responsible for displaying the visual stimulus, i.e., the calibration dots and then later the test imagery, it is imperative that: (a) the program knows when to display the calibration dots or the test imagery, and (b) the calibration dots are aligned as precisely as possible so that the program's calibration dot is displayed at the location where the eye tracker expects it to be. The latter condition is satisfied partly by appropriate mapping of coordinates between application program window (viewport) coordinates and the eye tracker screen coordinates, and the initial placement of the application program window on the display screen (e.g., the window is moved to the appropriate position on the display device such as the TV or within the HMD). The former condition is met by obtaining a status word from the eye tracker itself. It is therefore imperative that the eye tracker provide this valuable piece of data along with the POR value(s).

Assuming the eye tracker provides its current state (along with the current eye coordinates x and y), the pseudocode for the usual graphics drawing routine, modified to draw the calibration points at the appropriate time, is shown in Figure 43. This routine is sensitive to the three

```
sceneProjection();                                  // projection (e.g., ortho or perspective)

sceneMatrix();                                      // model (object) transformation(s)

switch(eye tracker state) {
  case RUN:
    if(displayStimulus) {
      displayStimulus();                            // show image, VR, etc.
    }
    break;
  case RESET:
  case CALIBRATE:
    drawCalibrationDot(x-5,y-5,x+5,y+5);            // show calibration dot
    break;
}

swapbuffers();                                      // swap buffers
```

Figure 43: Usual graphics draw/expose routine augmented with mode-sensitive eye tracking code.

possible modes of the eye tracker: RUN, RESET, and CALIBRATE. A typical double-buffered display update routine, for systems without eye trackers, would normally just be expected to set up the scene projection and object transformation matrices, then display the scene (stimulus) and swap buffers. In the modified routine, what is mostly needed is a way of drawing the calibration stimulus when needed.

Notice that in the pseudocode of Figure 43 the calibration stimulus is displayed in both RESET and CALIBRATE states. This facilitates the initial alignment of the application window with the eye tracker's initial positioning of its cross-hairs. The default position is usually the center of the eye tracker screen. Provided the coordinate mapping routine is in place in the main loop of the program, the application program should correspondingly display its calibration dot at the center of its application window. The operator then simply positions the application window so that both points align in the eye tracker's scene monitor.

Notice also in Figure 43 that the stimulus scene (the image or VR environment) is only displayed if a display stimulus condition is satisfied. This condition may be set outside the drawing routine (e.g., in the main program loop) depending on some timing criterion. For example, if the experimental trial requires that an image be displayed for only 5 seconds, a timer in the main loop can be used to control the duration of the displayed stimulus by setting the display condition just after calibration has completed (state change from calibration to run). The timer then unsets the condition after the required duration has expired.

For VR applications, the draw routine may be preceded by viewport calls which determine which display, left or right, is drawn to. If the view is shifted horizontally, a binocular display is presented, otherwise, a biocular display is seen by the user. In VR, the calibration stimulus may require an orthographic projection call so that the 2D calibration points are not perturbed by a perspective transformation.

The main loop of the program is responsible for:

- reading the data from the eye tracker (and the head tracker if one is being used),

- mapping the eye tracker coordinates (and head tracker coordinates if one is being used),
- starting/stopping timers if the experimental conditions call for a specific stimulus duration period, and
- either storing or acting on the 2D (or 3D) gaze coordinates, if the application is diagnostic or gaze-contingent in nature, respectively.

This general algorithm is shown as pseudocode in Figure 44. In this instance of the main loop, a timer of length DURATION is used to

```
while(true) {

  getEyeTrackerData(x,y);                                    // read serial port

  mapEyeTrackerData(x,y);                                    // map coordinates

  switch(eye tracker state) {
    case RUN:
      if(!starting) {
        starting = true;
        startTimer();
        displayStimulus = true;
        redraw();                                            // redraw scene event
      }
      if(checkTimer() > DURATION) {
        displayStimulus = false;
        redraw();                                            // redraw scene event
      }else {
        storeData(x,y);
      }
      break;
    case RESET:
    case CALIBRATE:
      starting = false;
      redraw();                                              // redraw scene event
      break;
  }
}
```

Figure 44: Main loop (2D imaging application).

control the duration of stimulus display.

A similar loop is needed for a VR application. There are a few additional requirements dealing with the head position/orientation tracker and calculation of the three-dimensional gaze vector. The pseudocode for the main loop of a VR application is shown in Figure 45. The main loop routine for the VR application mainly differs in the calculation of the gaze vector, which is dependent on the stereoscopic geometry of the binocular system. The only other main difference is the data storage requirements. Instead of just the 2D point of regard, now the locations of both eyes (or the gaze vector) and/or the head need to be recorded. For gaze-contingent applications, instead of data storage, the gaze vector may be used directly to manipulate the scene or the objects within.

Once the application program has been developed, the system is ready for use. The general manner of usage requires the following steps, to be performed by the operator after starting the program:

1. Move the application window to align it with the eye tracker's central calibration dot.
2. Adjust the eye tracker's pupil and corneal reflection threshold controls.
3. Calibrate the eye tracker.
4. Reset the eye tracker and run (program records the data).
5. Save recorded data.
6. Optionally calibrate again.

The final calibration step may be repeated to judge the amount of instrument slippage during the experimental trial [Duc97].

F.4 Data Collection and Analysis

Data collection is straightforward. For 2D imaging applications, the point of regard can be recorded, along with the timestamp of each sample. Figure 46 shows a pseudocode sample data structure suitable for recording the point of regard. Because the number of samples may be rather large, depending on sampling rate and duration of viewing trial, a linked list may be used to dynamically record the data. In Figure 46, the data type `queue` denotes a linked list pointer.

```
while(true) {
  getHeadTrackerData(eye,dir,upv);                                  // read serial port
  getEyeTrackerData(x_l,y_l,x_r,y_r);                               // read serial port
  mapEyeTrackerData(x_l,y_l,x_r,y_r);                               // map coordinates
  s = b/(x_l − x_r + b)                          // calculate linear gaze interpolant parameter s
  h = [eye_x, eye_y, eye_z]                                         // set head position
  v = [(x_l + x_r)/2 − x_h, (y_l + y_r)/2 − y_h, f − z_h]           // calculate central view vector
  transformVectorToHeadTrackerReferenceFrame(v);                    // multiply v by FOB matrix
  g = h + sv                                                        // calculate gaze point
  switch(eye tracker state) {
    case RUN:
      if(!starting) {
        starting = true;
        startTimer();
        displayStimulus = true;
        redraw();                                                   // redraw scene event
      }
      if(checkTimer() > DURATION) {
        displayStimulus = false;
        redraw();                                                   // redraw scene event
      }else {
        storeData(x_l,y_l,x_r,y_r);
      }
      break;
    case RESET:
    case CALIBRATE:
      starting = false;
      redraw();                                                     // redraw scene event
      break;
  }
}
```

Figure 45: Main loop (2D imaging application).

```
typedef struct {
  queue link;                                                       // linked list node
  int x,y;                                                          // POR data
  double t;                                                         // timestamp (usually in ms)
} PORnode;
```

Figure 46: 2D imaging point of regard data structure.

For 3D VR applications, the simplest data structure to record the calculated gaze point is similar to the 2D structure, with the addition of the third z component. Data captured, i.e., the three-dimensional gaze point, can then be displayed for review of the session statically in the VR environment in which it was recorded, as shown in Figure 47. In Figure 47 consecutive gaze points have been joined by straight line segments

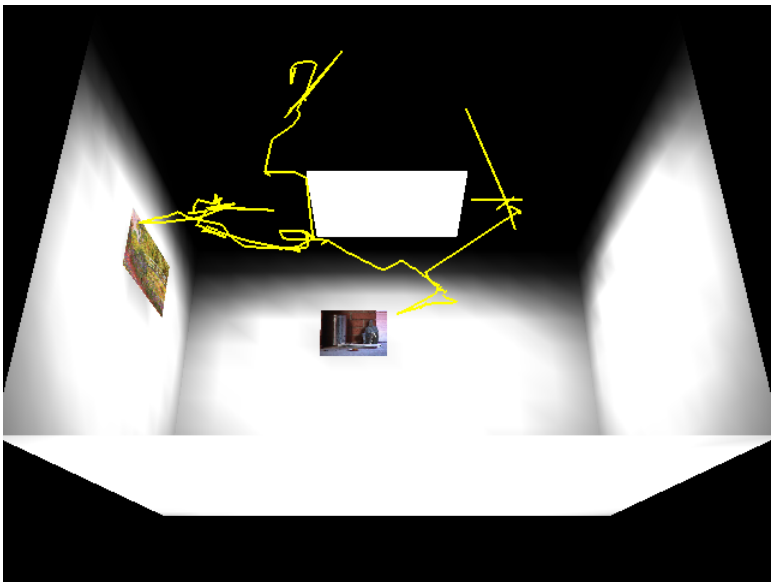


Figure 47: Example of three-dimensional gaze point captured in VR. Courtesy of Tom Auchter and Jeremy Barron.

to display a three-dimensional scanpath. Note that in this visualization it may be easy to misinterpret the scanpath data with the position of the head. To disambiguate the two, head position (and tilt angle) may also be stored/displayed.

There is one fairly important aspect of data storage which is worth mentioning, namely, proper labeling of the data. Because eye movement experiments tend to generate voluminous amounts of data, it is imperative to properly label data files. Files should encode the number of the subject viewing the imagery, the imagery itself, the trial number, and the state of the data (raw, or processed). One of the easiest methods for encoding this information is in the data filename itself (at least on file systems which allow long filenames).

Analysis of stored data follows the algorithms described in Section D.6. In general, there are two important considerations: (a) elimination of noise and (b) identification of fixations. Noise may be present in the signal due to eye blinks, or other causes for the eye tracker's loss of proper imaging of the eye. These types of gross noise artifacts can generally be eliminated by knowing the device characteristics, e.g., the eye tracker outputs $(0, 0)$ for eye blinks or other temporary missed readings. Currently there are two general methods for identification of fixations: the position-variance strategy, or the velocity-detection method. A third alternative may be a hybrid approach which compares the result of both algorithms for agreement.

F.5 Summary

Due to technological advancements, today's eye trackers are cheaper, faster, more accurate, and easier to use than ever before. These factors have recently revived interest in this more than 30-year-old technology. The importance of eye tracking technology cannot be overlooked. By measuring the dynamic movements of the eyes, the eye tracker supplies tangible clues to the perceptual processes effecting vision, our most important sense. In its diagnostic role, the eye tracker provides objective and quantitative evidence of the user's visual and attentional processes. As an interface modality, the eye tracker serves as a powerful input device which can be utilized by a host of visually-mediated applications.