Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 207 (2022) 1641–1648

Procedia Computer Science

www.elsevier.com/locate/procedia

26th International Conference on Knowledge-Based and Intelligent Information & Engineering Systems (KES 2022)

3D Gaze in Virtual Reality: Vergence, Calibration, Event Detection

Andrew T. Duchowski[a], Krzysztof Krejtz[b], Matias Volonte[c], Chris J. Hughes[d], Marta Brescia-Zapata[e], Pilar Orero[e]

[a]*Clemson University, Clemson, SC, USA*
[b]*SWPS University of Social Sciences & Humanities, Warsaw, Poland*
[c]*Northeastern University, Boston, MA*
[d]*University of Salford, Manchester, UK*
[e]*Universitat Autònoma de Barcelona, Barcelona, Spain*

Abstract

Eye movement analysis in modern 3D rendering systems is reviewed and three new techniques are derived inspired by work developed in early Virtual Reality so-called 2.5D implementations, namely (a) gaze depth (i.e., vergence) estimation, (b) vergence calibration, and (c) real-time 3D event detection that considers eye- and head-coupling. The new 3D calibration shows excellent error reduction in terms of Mean Squared Error (MSE).

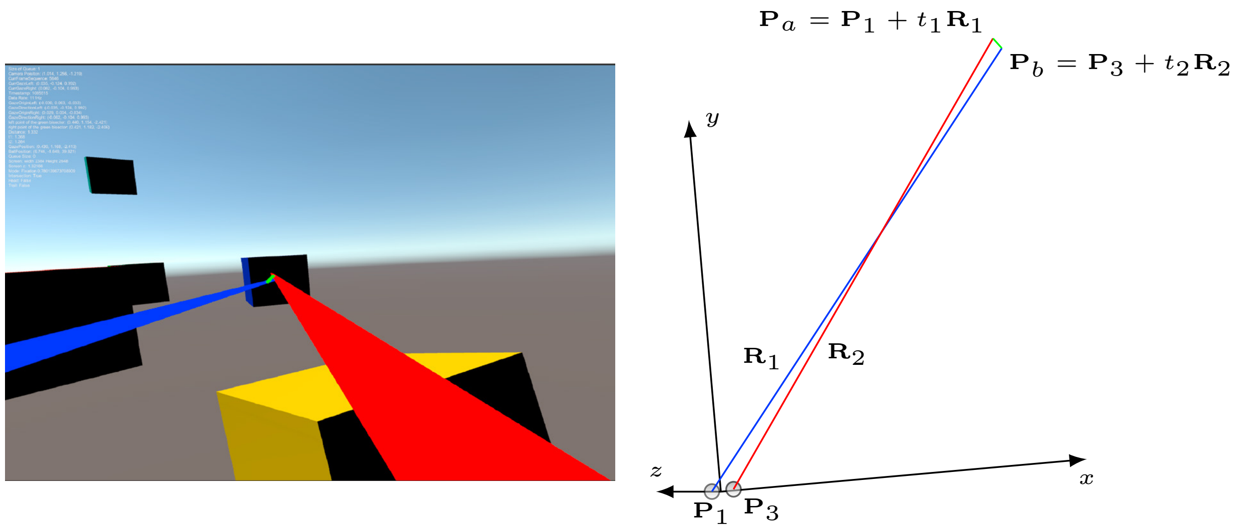
Keywords: virtual reality; eye tracking

1. Introduction & Background

In 2002, Duchowski et al. [2, 3] documented techniques for eye movement processing in Virtual Reality (VR) that estimated gaze rays in 3D obtained from the projection of the 2D left and right gaze points measured by the left and

* Andrew Duchowski
E-mail addresses: duchowski@clemson.edu (Andrew T. Duchowski)., kkrejtz@swps.edu.pl (Krzysztof Krejtz)., m.volont@northeastern.edu (Matias Volonte)., C.J.Hughes@salford.ac.uk (Chris J. Hughes)., Marta.Brescia@uab.cat (Marta Brescia-Zapata)., pilar.orero@uab.cat (Pilar Orero).

* Corresponding author. Tel.: +1-864-656-7677 ; fax: +0-000-000-0000.
E-mail address: duchowski@clemson.edu


$\mathbf{P}_a = \mathbf{P}_1 + t_1\mathbf{R}_1$
$\mathbf{P}_b = \mathbf{P}_3 + t_2\mathbf{R}_2$
y
$\mathbf{R}_1$
$\mathbf{R}_2$
z
x
$\mathbf{P}_1$
$\mathbf{P}_3$

(a) Unity 3D. (b) Vector geometry.

Fig. 1: Captured data in Unity 3D (a) and corresponding vector geometry (b). Note that the gaze rays do *not* intersect where they appear to cross over. Rather, their intersection is defined as the midpoint of the shortest segment $\overline{\mathbf{P}_a\mathbf{P}_b}$ perpendicular to both rays (in green). Note that $\mathbf{P}_2$ and $\mathbf{P}_4$ are not shown as they are arbitrarily located along the rays $\mathbf{R}_1$ and $\mathbf{R}_2$, i.e., $\mathbf{P}_2 = \mathbf{P}_1 + t_1\mathbf{R}_1$ and $\mathbf{P}_4 = \mathbf{P}_3 + t_1\mathbf{R}_2$ for some arbitrary choice of t_1 and t_2, respectively.

right eye trackers embedded in a Helmet-Mounted Displays (HMD). Such 2.5D gaze ray computation was then used for VR object selection via gaze ray-object intersection, and later for gaze-contingent foveated rendering.

At the time, the main concern was estimation of a combined gaze ray, which Duchowski et al. [3] derived as a ray emanating from the central head position as a function of interpupillary distance. Today, this type of functionality is built in to eye-tracking VR systems such as the HTC Vive Pro Eye. Vive provides Eye and Facial Tracking Software Development Kits (SDKs) that give real-time access to eye gaze features through their `SRAnipal` code plugin. SRAnipal is meant to drive the eye rotation of an avatar, but it also provides functions to obtain run-time gaze ray information as well as gaze-object ray intersection. Although their ray intersection mechanism (`SRAnipal`'s `Focus()` function) yields depth of an object being looked at, computation of gaze depth in virtual reality free space is left to the developer.

The purpose of this paper is to provide details of such computation, based on prior art and what appears to be a novel instantiation of a ray-ray intersection derivation (as opposed to line-line intersection). C# implementation is provided (see Listing 1 below), suitable for inclusion in Unity 3D along with a novel 3D gaze calibration.

2. 3D Eye Tracking in Virtual Reality

There are at least two reasons for measuring gaze in Virtual Reality: determining gaze direction, e.g., for foveated rendering, and determining object and gaze ray intersection e.g., for object selection. Both of these for the most part ignore vergence, i.e., the measurement of gaze depth. However, this estimate can be useful for many reasons, e.g., for gaze-contingent depth-of-field rendering.

2.1. Gaze Depth Computation

Wibirama and Hamamoto [18] provide a derivation of gaze depth given the left and right gaze ray origins, $\mathbf{P}_1$ and $\mathbf{P}_3$ and two distant points along those gaze rays, $\mathbf{P}_2$ and $\mathbf{P}_4$ (see Fig. 1(b) where the points $\mathbf{P}_2$ and $\mathbf{P}_4$ are omitted as they are arbitrarily located along the rays $\mathbf{R}_1$ and $\mathbf{R}_2$.).

Two rays in 3 dimensions do not generally intersect at a point. They may be parallel (no intersections) or they may be coincident (infinite intersections) but most often only their projection onto a plane intersect. When they do not exactly intersect at a point they can be connected by a line segment, the shortest line segment is unique and is often considered to be their intersection in 3D.

According to Paul Bourke[1] the two endpoints of the unique line segment $\mathbf{P}_a$ and $\mathbf{P}_b$ are defined as

$$\mathbf{P}_a = \mathbf{P}_1 + t_1(\mathbf{P}_2 - \mathbf{P}_1) \tag{1}$$

$$\mathbf{P}_b = \mathbf{P}_3 + t_2(\mathbf{P}_4 - \mathbf{P}_3) \tag{2}$$

with the shortest line found by minimizing $|\mathbf{P}_b - \mathbf{P}_a|$:

$$\mathbf{P}_b - \mathbf{P}_a = \mathbf{P}_3 - \mathbf{P}_1 + t_2(\mathbf{P}_4 - \mathbf{P}_3) - t_1(\mathbf{P}_2 - \mathbf{P}_1) \tag{3}$$

Because $\overline{\mathbf{P}_a\mathbf{P}_b}$ is perpendicular to both lines $\overline{\mathbf{P}_1\mathbf{P}_2}$ and $\overline{\mathbf{P}_3\mathbf{P}_4}$ the dot products between them are zero:

$$(\mathbf{P}_b - \mathbf{P}_a) \cdot (\mathbf{P}_2 - \mathbf{P}_1) = 0 \tag{4}$$

$$(\mathbf{P}_b - \mathbf{P}_a) \cdot (\mathbf{P}_4 - \mathbf{P}_3) = 0 \tag{5}$$

Substituting (3) into (4) and (5) yields:

$$[\mathbf{P}_3 - \mathbf{P}_1 + t_2(\mathbf{P}_4 - \mathbf{P}_3) - t_1(\mathbf{P}_2 - \mathbf{P}_1)] \cdot (\mathbf{P}_2 - \mathbf{P}_1) = 0 \tag{6}$$

$$[\mathbf{P}_3 - \mathbf{P}_1 + t_2(\mathbf{P}_4 - \mathbf{P}_3) - t_1(\mathbf{P}_2 - \mathbf{P}_1)] \cdot (\mathbf{P}_4 - \mathbf{P}_3) = 0 \tag{7}$$

Wibirama and Hamamoto [18] note that (6) and (7) can be solved to produce t_1, t_2 which then can be used to find $\mathbf{P}_a$ and $\mathbf{P}_b$ and then the gaze depth is computed as the midpoint of this segment:

$$\mathbf{P}_m = \frac{\mathbf{P}_a + \mathbf{P}_b}{2} \tag{8}$$

Indeed, Bourke[1] provides the solution using explicit vector elements x, y, z. Several code examples are also provided that can directly be used in a system such as Unity.

The problem with the above derivation is that it relies on two (distant) points along both gaze rays. Wibirama and Hamamoto [18] use a gaze tracking system where the gaze rays intersect a 2D screen, and so they provide two natural gaze ray intersections that can be used as $\mathbf{P}_2$ and $\mathbf{P}_4$. In VR, however, there is no such obvious plane to use, making choice of points $\mathbf{P}_2$ and $\mathbf{P}_4$ arbitrary.

What is needed instead is derivation of the segment $\overline{\mathbf{P}_a\mathbf{P}_b}$ using rays $\mathbf{R}_2$ and $\mathbf{R}_4$ directly, without the need for estimation of either of $\mathbf{P}_2$ or $\mathbf{P}_4$, i.e., $\mathbf{R}_2 \neq (\mathbf{P}_2 - \mathbf{P}_1)$ and $\mathbf{R}_4 \neq (\mathbf{P}_4 - \mathbf{P}_3)$ necessarily since neither of $\mathbf{P}_2$ nor $\mathbf{P}_4$ need be specified. Rather, in Unity, the `SRAnipal` SDK simply provides both $\mathbf{R}_2$ and $\mathbf{R}_4$ via two `GetGazeRay()` calls (see below). The ray-based derivation is similar to the system described by Abbott and Faisal [1], who omit explicit derivation of the vector intersection. Our derivation makes this explicit and is as follows.

Equations (6) and (7) are rewritten with gaze rays $\mathbf{R}_2$ or $\mathbf{R}_4$:

$$[(\mathbf{P}_3 - \mathbf{P}_1) + t_2\mathbf{R}_2 - t_1\mathbf{R}_4] \cdot \mathbf{R}_2 = 0 \tag{9}$$

$$[(\mathbf{P}_3 - \mathbf{P}_1) + t_2\mathbf{R}_2 - t_1\mathbf{R}_4] \cdot \mathbf{R}_4 = 0 \tag{10}$$

This derivation is easier to follow than Bourke's, especially when expressed in vector form, so long as one keeps track of whether the resultant expression elements are themselves vector or scalar quantities, remembering that the dot product yields a scalar. Expanding (9) and (10) yields

$$(\mathbf{P}_3 - \mathbf{P}_1) \cdot \mathbf{R}_2 + t_2(\mathbf{R}_2 \cdot \mathbf{R}_2) - t_1(\mathbf{R}_4 \cdot \mathbf{R}_2) = 0 \tag{11}$$

$$(\mathbf{P}_3 - \mathbf{P}_1) \cdot \mathbf{R}_4 + t_2(\mathbf{R}_2 \cdot \mathbf{R}_4) - t_1(\mathbf{R}_4 \cdot \mathbf{R}_4) = 0 \tag{12}$$

Rewriting (11) and solving for t_1 gives

$$t_1 = \frac{(\mathbf{P}_3 - \mathbf{P}_1) \cdot \mathbf{R}_2 + t_2(\mathbf{R}_2 \cdot \mathbf{R}_2)}{(\mathbf{R}_4 \cdot \mathbf{R}_2)} \tag{13}$$

[1] `http://paulbourke.net/geometry/pointlineplane/`

```
public bool CalculateVectorVectorIntersection(
        Vector3 P1, Vector3 P3,
        Vector3 R2, Vector3 R4,
        out float t1, out float t2) {

  Vector3 p13 = P1 - P3;

  t1 = t2 = 0.0f;

  // dot products
  float r4dotr4 = Vector3.Dot(R4,R4);
  float r2dotr2 = Vector3.Dot(R2,R2);
  float r2dotr4 = Vector3.Dot(R2,R4);

  // check denominator (R2 · R4)^2 − (R2 · R2)(R2 · R4)
  float denom = Mathf.Pow(r2dotr4, 2) - (r2dotr2 * r4dotr4);

  if (r2dotr4 < Mathf.Epsilon || Math.Abs(denom) < Mathf.Epsilon)
    return false;

  t2 = ((Vector3.Dot(p13,R2) * r4dotr4) -
        (Vector3.Dot(p13,R4) * r2dotr4)) / denom;
  t1 = (Vector3.Dot(p13,R2) + t2 * r2dotr2) / (r2dotr4);

  return true;
```

Listing. 1: C# implementation.

Substituting (13) into (12) gives:

$$\frac{(\mathbf{P}_3 - \mathbf{P}_1) \cdot \mathbf{R}_4 + t_2(\mathbf{R}_2 \cdot \mathbf{R}_4)}{(\mathbf{R}_4 \cdot \mathbf{R}_4)} = \frac{(\mathbf{P}_3 - \mathbf{P}_1) \cdot \mathbf{R}_2 + t_2(\mathbf{R}_2 \cdot \mathbf{R}_2)}{(\mathbf{R}_4 \cdot \mathbf{R}_2)} \tag{14}$$

Re-arranging and solving for t_2 yields:

$$t_2 = \frac{(\mathbf{P}_3 - \mathbf{P}_1) \cdot \mathbf{R}_2(\mathbf{R}_4 \cdot \mathbf{R}_4) - (\mathbf{P}_3 - \mathbf{P}_1) \cdot \mathbf{R}_4(\mathbf{R}_4 \cdot \mathbf{R}_2)}{(\mathbf{R}_2 \cdot \mathbf{R}_4)^2 - (\mathbf{R}_2 \cdot \mathbf{R}_2)(\mathbf{R}_2 \cdot \mathbf{R}_4)} \tag{15}$$

If either $|(\mathbf{R}_2 \cdot \mathbf{R}_4)| < \epsilon$ or $|(\mathbf{R}_2 \cdot \mathbf{R}_4)^2 - (\mathbf{R}_2 \cdot \mathbf{R}_2)(\mathbf{R}_2 \cdot \mathbf{R}_4)| < \epsilon$ then the rays do not intersect. If either $t_1 < 0$ or $t_2 < 0$ then the intersection is behind the gaze ray origins, i.e., when gaze diverges. The line segment endpoints are obtained as in (1) and (2), namely $\mathbf{P}_a = \mathbf{P}_1 + t_1\mathbf{R}_2$ and $\mathbf{P}_b = \mathbf{P}_3 + t_2\mathbf{R}_4$ and gaze depth is the segment bisector given in (8).

Example implementation in C# suitable for inclusion in Unity is given in Listing 1. Note that the `SRanipal` plugin for Unity provides the `SRanipal_Eye_v2.GetGazeRay()` function that returns the gaze ray origin and direction of either of the left or right eyes (or combination thereof).

2.2. Vergence Calibration

Gaze estimation in Virtual Reality, at least in the HTC Vive Pro Eye, depends on calibration of the eye tracker based on gaze localization on a 2D plane, e.g., as performed in `SteamVR` before rendering of a 3D environment. Thus gaze depth estimation as derived above yields raw gaze data in 3D and is uncalibrated.

Wang et al. [15, 16] described a method of 3D gaze depth calibration using a continuous form of calibration using a simple sphere traversing along a Lissajous-knot path where the sphere's position $S(t) = (x(t), y(t), z(t))$ over time t in seconds, is specified by the vector equation

$$\mathbf{S}(t) = \mathbf{\Lambda} \cos(2\pi\mathbf{\Gamma}(t) + \mathbf{\Phi}) \tag{16}$$

with component amplitudes $\mathbf{\Lambda} = (9, 5, 20)$ in mm, frequencies $\mathbf{\Gamma} = (0.101, 0.127, 0.032)$ in Hz, and phase angles $\mathbf{\Phi} = (0, -90, 57)$ in degrees. Calibration can take a short amount of time, e.g., 10 seconds, which, at a sampling rate of 120 Hz, yields 1,200 gaze position (and sphere position) samples.

Calibration relies on Lagrange's method of least squares, or the multivariate multiple regression model [8, 4], to minimize errors due to systematic shift (from $x=0$, $y=0$ and $z=0$) and scale.

In two dimensions, e.g., when calibrating *x*- and *y*-coordinates on a 2D plane, often perpendicular to the viewer as on a computer monitor, calibration points (s_{ix}, s_{iy}) are defined by their screen coordinates. A second-order polynomial is often used to handle quadratic distortions such as pin-cushion or barrel effects [12]:

$$s_{ix} = a_0 + a_1 x_i + a_2 y_i + a_3 x_i y_i + a_4 x_i^2 + a_5 y_i^2 \tag{17}$$
$$s_{iy} = b_0 + b_1 x_i + b_2 y_i + b_3 x_i y_i + b_4 x_i^2 + b_5 y_i^2 \tag{18}$$

where parameters a_0–a_5 and b_0–b_5 are the unknowns. Equations (17) and (18) can be reformulated into matrix form:

$$\begin{bmatrix} s_{ix} & s_{iy} \end{bmatrix} = \begin{bmatrix} 1 & x_i & y_i & x_i y_i & x_i^2 & y_i^2 \end{bmatrix} \begin{bmatrix} a_0 & a_1 & a_2 & a_3 & a_4 & a_5 \\ b_0 & b_1 & b_2 & b_3 & b_4 & b_5 \end{bmatrix}^T$$

For each time sample *i*, (s_{ix}, s_{iy}) denotes the known coordinate of the calibration point, and (x_i, y_i) the corresponding gaze point. With a_i and b_i the unknown coefficients, a second order model is used to minimize error for *x* and *y*:

$$\mathbf{S} = \mathbf{X}\hat{\mathbf{B}} \quad \text{with} \quad \hat{\mathbf{B}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{S}$$

where $\mathbf{X}^T\mathbf{X}$ is the symmetric covariance matrix and the estimate of $\hat{\mathbf{B}}$ is the result of Lagrange's method of least squares. Wibirama and Hamamoto [18] used a quadratic model such as this followed by Singular Value Decomposition to correct for the *z*-coordinate.

Wang et al. [16] showed that for calibrating gaze depth in a desktop eye-tracking stereo system, a linear system can be used instead, constructed for the *z*-coordinates $\mathbf{S_z} = \mathbf{Zc}$, where $\mathbf{S_z}$ is the vector of known values for *z*, $\mathbf{Z}$ is the matrix formed from measured values of *z*, and $\mathbf{c} = [c_0, c_1]^T$ is the vector of unknown coefficients:

$$\begin{bmatrix} s_{iz} \end{bmatrix} = \begin{bmatrix} 1 & z_i \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \end{bmatrix}$$

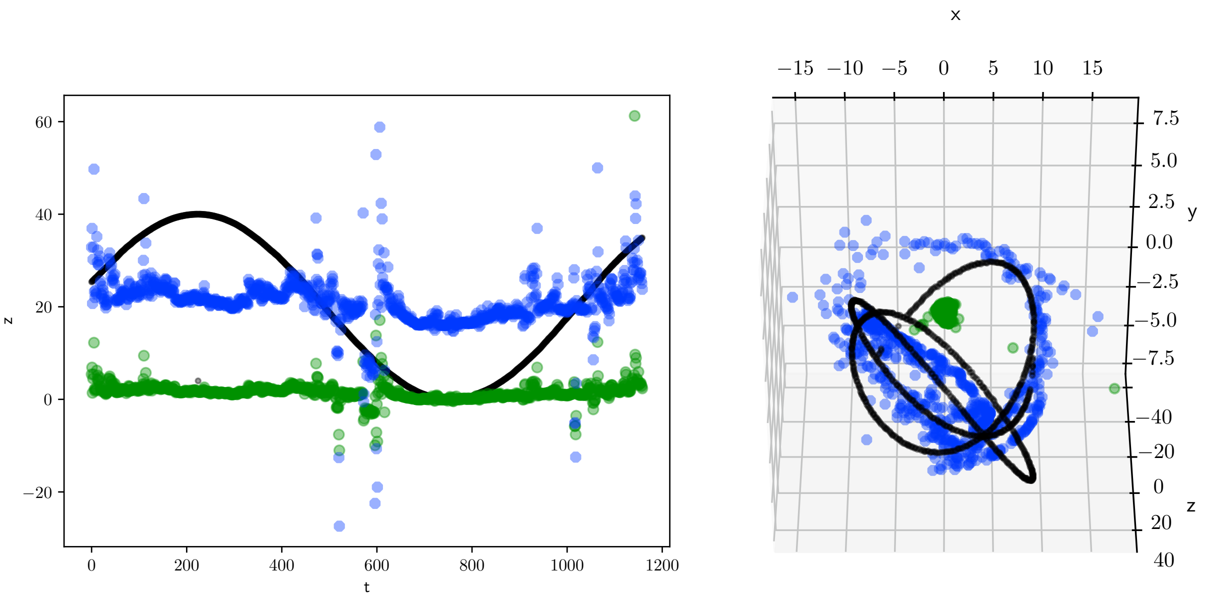
In the case of desktop stereo with a remote 2D eye tracker, both the *x*- and *y*-coordinates are calibrated in 2D, and are used to yield the 3D gaze vectors, and so there is no need to re-calibrate. Instead, only the solution for *z* is sought, where the solution to this over constrained system, yielding the least mean squared error, is $\mathbf{c} = (\mathbf{Z}^T\mathbf{Z})^{-1}\mathbf{Z}^T\mathbf{S_z}$. The resulting coefficient vector ($\mathbf{c}$) is saved and used in real-time during eye tracking to compute the calibrated gaze depth

$$(\hat{x}, \hat{y}, \hat{z}) = (x, y, c_0 + c_1 z) \tag{19}$$

from the measured gaze vectors and gaze intersection point (x, y, z). Collecting 3D data in VR exposes the inadequacy of this so-called 2.5D solution, as seen in Fig. 2(a). Although the calibration *z*-coordinate fluctuates sinusoidally as per the *z*-coordinate of the Lissajous curve in (16), gaze depth appears linear, before and after transformation. Transformed gaze depth is properly aligned with the apparent mean of the Lissajous curve *z*-coordinate, but the sinusoidal undulations are not faithfully reproduced due to the linear nature of the transform given by (19). The approach may have been sufficient for the 2.5D stereoscopic setup used by Wang et al. [16], but a more complete solution is needed for 3D gaze.

In 3D, neither 2D quadratic solution for *x*- and *y*-coordinates (17-18) nor 1D linear solution for the *z*-coordinate (19) is sufficient. In 3D, a quadratic system for solving all three *x*-, *y*-, and *z*-coordinates simultaneously is needed:

$$s_{ix} = a_0 + a_1 x_i + a_2 y_i + a_3 z_i + a_4 x_i y_i + a_5 x_i z_i + a_6 y_i z_i + a_7 x_i y_i z_i + a_8 x_i^2 + a_9 y_i^2 + a_{10} z_i^2 \tag{20}$$
$$s_{iy} = b_0 + b_1 x_i + b_2 y_i + b_3 z_i + b_4 x_i y_i + b_5 x_i z_i + b_6 y_i z_i + b_7 x_i y_i z_i + b_8 x_i^2 + b_9 y_i^2 + b_{10} z_i^2 \tag{21}$$
$$s_{iz} = c_0 + c_1 x_i + c_2 y_i + c_3 z_i + c_4 x_i y_i + c_5 x_i z_i + c_6 y_i z_i + c_7 x_i y_i z_i + c_8 x_i^2 + c_9 y_i^2 + c_{10} z_i^2 \tag{22}$$



(a) 1D calibration.

(b) 3D calibration.

Fig. 2: Calibration visualization. The black scatter plot shows the position of the calibration target traversing the Lissajous curve path. In (a) only the z-coordinate is visible. 10.94 for transformed z-coordinate (in Unity meters). The green scatter plot is the raw 3D gaze position. The blue scatter plot shows the transformed gaze coordinates. Mean squared error for raw 3D coordinates is 25.93 and 6.05 for transformed 3D coordinate (in Unity meters).

or in matrix form $\mathbf{S} = \mathbf{X}\hat{\mathbf{B}}$ as before, where

$$\mathbf{S} = \begin{bmatrix} s_{ix} & s_{iy} & s_{iz} \end{bmatrix}$$

$$\mathbf{X} = \begin{bmatrix} 1 & x_i & y_i & z_i & x_i y_i & x_i z_i & y_i z_i & x_i y_i z_i & x_i^2 & y_i^2 & z_i^2 \end{bmatrix}$$

and

$$\hat{\mathbf{B}}^T = \begin{bmatrix} a_0 & a_1 & a_2 & a_3 & a_4 & a_5 & a_6 & a_7 & a_8 & a_9 & a_{10} \\ b_0 & b_1 & b_2 & b_3 & b_4 & b_5 & b_6 & b_7 & b_8 & b_9 & b_{10} \\ c_0 & c_1 & c_2 & c_3 & c_4 & c_5 & c_6 & c_7 & c_8 & c_9 & c_{10} \end{bmatrix}$$

Results of the transformation are shown in Fig. 2(b) which show much better adherence of the transformed 3D gaze to the Lissajous curve then when uncalibrated. Mean squared error between gaze point and target is clearly reduced following 3D calibration. The advantage of this method is that it provides a direct solution (via least-squares minimiation), obviating the need for more exotic solutions, such as neural networks suggested by Li et al. [9] or visual-intertial simultanesou localization and mapping, i.e., SLAM-based localization given by Wang et al. [14].

2.3. Real-time 3D Event Detection

Defining the gaze point $\mathbf{P}_i = (\hat{x}, \hat{y}, \hat{z})$ as in (19), the next step is to derive an *event detection* [6] algorithm suitable for implementation in virtual reality to distinguish fixations from saccades, as well as smooth pursuits and Vestibulo-Ocular Reflex (VOR) movements, if possible. VOR movements (and depth estimation) can be useful in disambiguating selection of occluded targets at depth [11]. Llanes-Jurado et al. [10] note that fixation detection in head-mounted displays, where head movement is unrestricted, is still an open issue. Their dispersion-based approach was based on one given by Duchowski et al. [3], who used the gaze point computed as the intersection of the gaze ray and objects in the scene, which, for example, is possible to retrieve via the `Focus()` call within the `SRAnipal` SDK.

Table 1: Event detection w.r.t. coupled head and eye motion.

$\dot{\mathbf{\Omega}}$	$\dot{\mathbf{\Theta}}$	Event
$< T_h$	$< T_e$	both still $\Longrightarrow$ *fixation*
$< T_h$	$> T_e$	head still, eye motion $\Longrightarrow$ *saccade*
$> T_h$	$< T_e$	head motion, eyes still $\Longrightarrow$ *VOR*
$> T_h$	$> T_e$	both in motion, eye motion $\Longrightarrow$ *pursuit* or *VOR*, depending on congruence

Duchowski et al. [3] and Llanes-Jurado et al. [10] defined an estimate of instantaneous visual angle at each gaze sample θ_i, obtained from the dot product of successive gaze points and averaged head position:

$$\theta_i = \cos^{-1}\left(\frac{\rho_i \cdot \rho_{i-1}}{\|\rho_i\|\|\rho_{i-1}\|}\right), \quad \text{where} \quad \rho_i = \mathbf{P}_i - \overline{\mathbf{h}} \tag{23}$$

and $\overline{\mathbf{h}} = (\mathbf{h}_i + \mathbf{h}_{i-1})/2$ is the (averaged) head position. With this definition, ρ_i is simply the gaze ray originating at the head position. Note also that either the gaze intersection point or the gaze point, as defined in (19) and at calibrated gaze depth $\hat{z}$ located in *free (virtual) space*, can be used in (23) to compute the visual angle.

However, Duchowski et al. [3] originally used this because their system at the time was not as completely integrated as modern eye-tracked VR systems are, case in point the HTC Vive Pro Eye. Using such a modern system, one can easily obtain the head position as well as head and gaze directions, without needing to explicitly re-compute the gaze direction vector. Moreover, using (23) ignores head *movement* since it only uses head position.

Taking advantage of modern eye-tracking code plugins, the gaze ray $\mathbf{R} = (\mathbf{R}_1 + \mathbf{R}_4)/2$, bisector of the two gaze rays $\mathbf{R}_1$ and $\mathbf{R}_4$, easily computed or available via `SRAnipal` call `GetGazeRay()`, can be used directly to obtain the instantaneous visual angle $\mathbf{\Theta}_i$, at each gaze sample, from the dot product of successive gaze rays:

$$\mathbf{\Theta}_i = \cos^{-1}(\mathbf{R}_i \cdot \mathbf{R}_{i-1}) \tag{24}$$

assuming the gaze ray $\mathbf{R}$ is normalized. The head angle can be obtained similarly,

$$\mathbf{\Omega}_i = \cos^{-1}(\mathbf{H}_i \cdot \mathbf{H}_{i-1}) \tag{25}$$

assuming normalized head direction $\mathbf{H}$. Given both directions, a velocity-based approach can be derived with head and gaze direction velocities as follows:

$$\dot{\mathbf{\Theta}}_i = \frac{1}{\Delta t}\sum_{j=0}^{n} \Theta_{i-j}\mathbf{g}_j, \quad \dot{\mathbf{\Omega}}_i = \frac{1}{\Delta t}\sum_{j=0}^{n} \Omega_{i-j}\mathbf{g}_j, \quad j \in [0, n] \tag{26}$$

where $\dot{\mathbf{\Theta}}$ and $\dot{\mathbf{\Omega}}$ are gaze and head velocities, respectively, $\mathbf{g}$ is a Finite Impulse Response (FIR) high-pass differentiation filter, n is the filter length, and Δt is the sampling period over the length of the filter. Although Duchowski et al. [3] suggested that filters as short as 2-tap filters could be used for this purpose, e.g., one with with normalized coefficients $-1/\sqrt{2}, 1/\sqrt{2}$, these are known to be noisy. Instead, a filter derived by Savitzky and Golay [13] offers additional smoothing at minimal computational expense. A 7-tap filter incurs convolution of only 7 recent samples over a period of $\Delta t = 7 \times (1/120) = 56$ ms.

Given both gaze and head velocities, $\dot{\mathbf{\Theta}}$ and $\dot{\mathbf{\Omega}}$, it is now possible to consider the coupling between eye and head movements [5, 7]. Because the head is relatively much slower than the eyes [17], an event detection algorithm can be proposed based on two thresholds of head and eye rotation, T_h and T_e, respectively, as given in Table 1. Thresholds can be determined empirically, with initial setting potentially guided by known quantities. For example, saccades are roughly bounded by 250 deg/s. Fixations can potentially be identified with velocity less than 3 deg/s, while smooth pursuits are expected to range from 3 to 100 deg/s. Validation of this proposed event detection algorithm is left as future work.

3. Conclusion

Given the proliferation of affordable eye-tracking Virtual and Augmented Reality (VR, AR) equipment, it seemed timely to review three straightforward techniques for gaze estimation in what is collectively referred to as eXtended

Reality (XR). Namely, three methods were reviewed: (a) gaze depth (i.e., vergence) estimation in free space (as opposed to gaze ray-object intersection), (b) vergence calibration using continuous rather than discrete sampling, and (c) proposed real-time 3D event detection, including fixation, saccadic, smooth pursuit, and vestibulo-ocular response eye movement.

Acknowledgments

This work is supported in part by the U.S. National Science Foundation (grant IIS-1748380 and grant CAREER IIS-2045523), and Polskie Ministerstwo Nauki i Szkolnictwa Wyższego (Regional Initiative of Excellence, grant 012-RID-2018/19). Any opinions, findings and conclusions or recommendations expressed in this material are the author(s) and do not necessarily reflect those of the sponsors.

References

[1] Abbott, W., Faisal, A., 2012. Ultra-low-cost 3D gaze estimation: An intuitive high information throughput compliment to direct brain-machine interfaces. Journal of Neural Engineering 9, 11. doi:10.1088/1741-2560/9/4/046016.

[2] Duchowski, A., Medlin, E., Cournia, N., Gramopadhye, A., Melloy, B., Nair, S., 2002a. 3D Eye Movement Analysis for VR Visual Inspection Training, in: ETRA '02: Proceedings of the 2004 symposium on Eye tracking research & applications, ACM, New Orleans, LA. pp. 103–110,155.

[3] Duchowski, A.T., Medlin, E., Cournia, N., Gramopadhye, A., Nair, S., Vorah, J., Melloy, B., 2002b. 3D Eye Movement Analysis. Behavior Research Methods, Instruments, Computers (BRMIC) 34, 573–591.

[4] Finn, J.D., 1974. A General Model for Multivariate Analysis. Holt, Rinehart and Winston, Inc., New York, NY.

[5] Guitton, D., Munoz, D.P., Galiana, H.L., 1990. Gaze Control in the Cat: Studies and Modeling of the Coupling Between Orienting Eye and Head Movements in Different Behavioral Tasks. Journal of Neurophysiology 64, 509—531. URL: https://doi.org/10.1152/jn.1990.64.2.509, doi:10.1152/jn.1990.64.2.509.

[6] Holmqvist, K., Nyström, M., Andersson, R., Dewhurst, R., Jarodzka, H., van de Weijer, J., 2011. Eye Tracking: A Comprehensive Guide to Methods and Measures. Oxford University Press, Oxford, UK.

[7] Kothari, R., Yang, Z., Kanan, C., Bailey, R., Pelz, J.B., Diaz, G.J., 2020. Gaze-in-wild: A dataset for studying eye and head coordination in everyday activities. Scientific Reports 10. URL: https://doi.org/10.1038/s41598-020-59251-5, doi:10.1038/s41598-020-59251-5.

[8] Lancaster, P., Šalkauskas, K., 1986. Curve and Surface Fitting: An Introduction. Academic Press, San Diego, CA.

[9] Li, S., Zhang, X., Webb, J.D., 2017. 3-D-Gaze-Based Robotic Grasping Through Mimicking Human Visuomotor Function for People With Motion Impairments. IEEE Transactions on Biomedical Engineering 64, 2824–2835. doi:10.1109/TBME.2017.2677902.

[10] Llanes-Jurado, J., Marín-Morales, J., Guixeres, J., Alcañiz, M., 2020. Development and Calibration of an Eye-Tracking Fixation Identification Algorithm for Immersive Virtual Reality. Sensors 20. URL: https://www.mdpi.com/1424-8220/20/17/4956, doi:10.3390/s20174956.

[11] Mardanbegi, D., Langlotz, T., Gellersen, H., 2019. Resolving Target Ambiguity in 3D Gaze Interaction through VOR Depth Estimation. Association for Computing Machinery, New York, NY. p. 1–12. URL: https://doi.org/10.1145/3290605.3300842, doi:10.1145/3290605.3300842.

[12] Morimoto, C.H., Mimica, M.R.M., 2005. Eye Gaze Tracking Techniques for Interactive Applications. Computer Vision and Image Understanding 98, 4–24.

[13] Savitzky, A., Golay, M.J.E., 1964. Smoothing and differentiation of data by simplified least squares procedures. Analytical Chemistry 36, 1627–1639. URL: http://pubs.acs.org/doi/abs/10.1021/ac60214a047.

[14] Wang, H., Pi, J., Qin, T., Shen, S., Shi, B.E., 2018. SLAM-Based Localization of 3D Gaze Using a Mobile Eye Tracker, in: Proceedings of the 2018 ACM Symposium on Eye Tracking Research & Applications, Association for Computing Machinery, New York, NY. URL: https://doi.org/10.1145/3204493.3204584, doi:10.1145/3204493.3204584.

[15] Wang, R., Pelfrey, B., Duchowski, A.T., House, D.H., 2012. Online Gaze Disparity via Binocular Eye Tracking on Stereoscopic Displays, in: Second Joint 3DIM/3DPVT Conference: 3D Imaging, Modeling, Processing, Visualization & Transmission (3DimPVT 2012), IEEE, Zurich, Switzerland.

[16] Wang, R.I., Pelfrey, B., Duchowski, A.T., House, D.H., 2013. Online 3D Gaze Localization on Stereoscopic Displays. Transactions on Applied Perception .

[17] Watson, B., Walker, N., Hodges, L.F., 1997. Managing Level of Detail through Head-Tracked Peripheral Degradation: A Model and Resulting Design Principles, in: Virtual Reality Software & Technology: Proceedings of the VRST'97, ACM. pp. 59–63.

[18] Wibirama, S., Hamamoto, K., 2013. 3D Gaze Tracking System for NVidia 3D Vision, in: Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society, IEEE Engineering in Medicine and Biology Society. doi:10.1109/EMBC.2013.6610220.