Computers & Graphics (2023)

Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

The *Salient360!* toolbox: handling gaze data in 3D made easy

Erwan David[a,*], Jesús Gutiérrez[b], Melissa Lè-Hoa Võ[a], Antoine Coutrot[c], Matthieu Perreira Da Silva[d], Patrick Le Callet[d]

[a]*Scene Grammar Lab, Department of Psychology, Goethe University, Frankfurt-am-Main, 60323, Germany*
[b]*Grupo de Tratamiento de Imágenes, Universidad Politécnica de Madrid, Madrid, 28040, Spain*
[c]*LIRIS, CNRS, University of Lyon, Lyon, 69622, France*
[d]*Nantes Université, École Centrale Nantes, CNRS, LS2N, UMR 6004, Nantes, 44000, France*

ARTICLE INFO

Article history:
Received September 9, 2023

Keywords: toolbox, gaze tracking, head tracking, 360 stimuli, processing, comparison, visualisation

ABSTRACT

Eye tracking has historically been a very popular tool. The data it records allow us to understand how people behave and to what they attend within our visual world; under this perspective, experiments, applications and use-cases are endless. It is therefore not surprising to witness a strong rise in the use of extended reality devices with embedded eye trackers in research. These devices allow for less obtrusive experimenting conditions, and a significantly much higher experimental control, when compared to traditional desktop testing. The use of eye tracking in eXtended Reality (XR) is increasing and so is the need for a toolbox enabling consensus about eye tracking methods in 3D. We present the *Salient360!* toolbox: it implements functions to identify saccades and fixations and output gaze features (e.g., saccade directions) to generate saliency maps, fixation maps, and scanpath data. It implements comparisons of gaze data with methods adapted to 3D. We plan continuous improvements of the toolbox as the community develops new tools and methods dedicated to 360 gaze tracking. We hope that this toolbox will spark discussions about the methodology of 3D gaze processing, facilitate running experiments, and improve studying gaze in 3D.

`https://github.com/David-Ef/salient360Toolbox`

1. Introduction

Gaze data is a very rich and complex signal [1, 2], it informs about where and how someone looked. Today, eye tracking is frequently utilised in many domains: scientific and engineering alike. Tracking eyes outside the lab with head-restraints used to be the norm, but modern devices allow unobtrusive recording of gaze in more naturalistic conditions: in the field or in the lab with extended reality devices. The recent popularity for the use of XR devices as an experimental tool is easily explained by the fact that they allow for near-perfect control of a virtual environment, coupled with unobtrusive measurement systems allowing participants to move and interact freely, all the while allowing for sufficiently high-quality tracking measures.

The quality of extended reality (XR) headsets has increased tremendously in the last decades. With the addition of embedded eye trackers, scientists have begun to rely on it more and more to study gaze and visual attention in immersive conditions closer to the natural world [3]. As these devices enter homes, more immersive and 360 contents (image, movies, video games) specific to this viewing paradigm are now being created. As a result, an understanding of how people look and behave in these environments [4, 5] is required to improve, for example, gaze prediction algorithms, which is essential to de-

*Corresponding author: Tel.: +49 (0)69 798 35409;
e-mail: `david@psych.uni-frankfurt.de` (Corresponding Author Name)

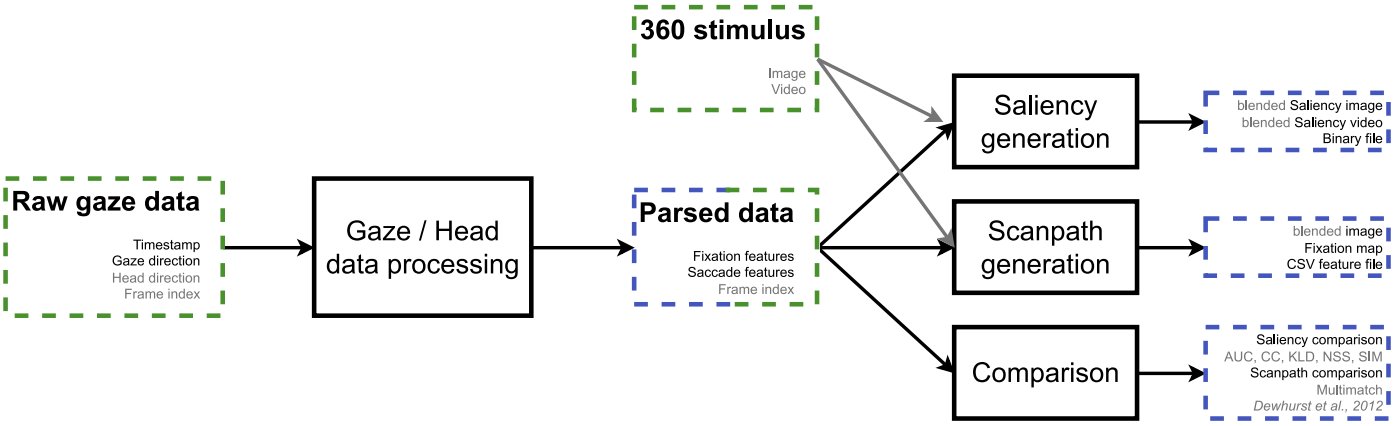

Fig. 1: Schematic representation of the functionalities of the toolbox, from input to be processed to generated outputs. Inputs appear in green dashed boxes, outputs show similarly in blue. Processing steps are embedded in boxes with solid black lines. Elements in grey are optional.

veloping compression algorithms adapted to XR. Consequently, there is a need to have robust and powerful tools to process gaze, eye, and head tracking in 3D.

There exist several eye tracking toolboxes meant to handle data obtained on standard computer screens. For example, eye tracker vendors often provide tools themselves, for example to identify saccades and fixations, and create saliency maps. Some of these toolboxes are dedicated to particular eye trackers (e.g., [6]). There are non-specialised toolboxes meant to be used in many circumstances [7, 8], while others are dedicated to particular analyses [9], applications [10] or experimental conditions [11].

When transitioning from on-screen to XR studies, it becomes clear that eye tracking toolboxes are not applicable. Moving from a screen as a 2D plane to a 3D world, one must now consider the movement of the head as part of the gaze. Therefore, eye rotation data are often referenced by "eye-in-head" and the combined head and eye rotation by "eye-in-space" [12, 13]. In addition to this nomenclature, in this paper we choose to employ eye and gaze, respectively. On top of this, gaze-parsing algorithms need to be modified (e.g., the Euclidean distance used to compute velocities is replaced by the angle between vectors; Eq 8). Many measures must be updated, which are used to process raw data, generate features of saccades and fixations, or saliency data. Saliency and scanpath comparison algorithms [14, 15] need modifying as well. The toolbox described in this article was created in response to this void in the community. Its functionalities related to saccade and fixation features are useful to experiments tracking head and eye, whereas those related to saliency maps are constrained to omnidirectional stimuli, like 360° pictures and videos. As more and more teams come to use gaze tracking in XR we believe it is of utmost importance to publish updated tools and start broader discussions about new/adapted tools and methods.

2. What the *Salient360!* toolbox implements

Written originally for the *Salient360!* visual attention modelling challenge [16, 17, 18] our toolbox now covers three main applications: **processing** raw data, **generating** saliency and saccadic features, **comparing** saliency and scanpath data, and **visualising** raw and processed data. It supports processing raw eye and head rotations to produce eye-in-space data, which are processed further to identify fixation positions for saliency data generation, and more scanpath features (fixation and saccade features, Fig 2). Both type of generated data (saliency and scanpath features) have methods implemented to be compared.

The toolbox is written in Python 3 (Python Software Foundation, *https://www.python.org/*) with the help of the scientific computing toolbox (SciPi [19], with NumPy [20]) and statsmodels [21]. The OpenCV [22] and scikit-image [23] modules are used to manipulate saliency maps as images. Numba [24] is used to accelerate some processing steps (e.g., saliency map calculation), PyOpenGL and PyQt5 were needed to build the visualising part of the toolbox. A list of requirements and an installation scripts are provided in the repository's README file. Installation is made easier with the use of a Conda environment.

Although the main purpose of the toolbox is to handle gaze data as the combination of eye and head data, it also supports processing head data alone as well. In that particular case, the toolbox behaves as if the eyes were always perfectly still and centered in their respective orbit. Moreover, the toolbox also possesses special implementation variants dedicated to data obtained while visualising dynamic content (e.g., a 360 video). Keep in mind that, although we describe most of the toolbox as pertaining to gaze data obtained from viewing static stimuli, everything applies to head and dynamic data as well.

It is important to note that our toolbox, and the methods and algorithms implemented therein actually apply to any eye-in-space data, whether it be gathered using an XR device or from a mobile eye tracker used for testing in the field. The reason why we do not focus on the latter is because head tracking in these conditions is very arduous, and therefore often missing, allowing for eye-in-head processing alone. Our toolbox is suitable to process

2.1. Processing

The minimal input is, for each sample: a timestamp, an eye direction vector, and a head rotation value (Euler angles or

(a) Fixation map

(b) Saliency map

(c) Saliency data blended with stimulus

(d) Fixation positions over a stimulus (colour encodes for fixation index)

Fig. 2: Example outputs from our toolbox. a) a **fixation map** as a 2D matrix with fixation counts pixel-wise. b) a **saliency map** obtained by convolving a fixation map with a Gaussian kernel. c) a saliency map **blended** with an image to better identify salient regions. d) colour-coded points drawn at fixation locations to get a **scanpath image** showing time-course development (data from only one trial is shown here); lines between points can be added to help visualise transitions.

quaternion). When loading a raw or processed gaze file, the toolbox will try to identify the required variables amongst the file columns according to its header string and a set of allowed variable names (see Tab 1 for a full list). If the timestamp data is not in milliseconds it will be automatically adjusted (Algo 1). In cases where the eye data saved is not eye-in-head but eye-in-space (i.e., eye and head rotation data are already combined) the user still needs to provide head rotations for the toolbox to operate. For that purpose new columns for the identity quaternion should be added to the raw data file (x: 0, y: 0, z: 0, w: 1).

Algorithm 1 Method used to automatically adjust timestamp data to milliseconds

$\vec{t} \leftarrow$ Timestamp sample vector
$\Delta\vec{t} \leftarrow$ Time difference between consecutive timestamps
$log_{samplerate} \leftarrow \log_{10}(Mean(\Delta\vec{t}))$
$adjust \leftarrow 3 * Floor(log_{samplerate})$
if $adjust \neq 0$ **then**
    $\vec{t} \leftarrow \vec{t} * 10^{-adjust}$
end if

Raw data are processed to produce saccade and fixation features according to a set of allowed parameters. First, according to what **eye data** is available, one may choose which to use: left, right or combined. If combined eye data is not provided it will be computed as the average of left and right data. We recommend to **resample** head and eye data to have matching sampling rates, for example, using the Vive Pro Eye will result in head data sampled at 90Hz and eye tracking data at 120Hz. Head rotations are stored as quaternions, but if the head rotation data are provided as Euler angles, they will be converted:

$$Q = \begin{pmatrix} \sin\frac{E_x}{2}\cos\frac{E_y}{2}\cos\frac{E_z}{2} + \cos\frac{E_x}{2}\sin\frac{E_y}{2}\sin\frac{E_z}{2} \\ \cos\frac{E_x}{2}\sin\frac{E_y}{2}\cos\frac{E_z}{2} - \sin\frac{E_x}{2}\cos\frac{E_y}{2}\sin\frac{E_z}{2} \\ \cos\frac{E_x}{2}\cos\frac{E_y}{2}\sin\frac{E_z}{2} - \sin\frac{E_x}{2}\sin\frac{E_y}{2}\cos\frac{E_z}{2} \\ \cos\frac{E_x}{2}\cos\frac{E_y}{2}\cos\frac{E_z}{2} + \sin\frac{E_x}{2}\sin\frac{E_y}{2}\sin\frac{E_z}{2} \end{pmatrix}. \quad (1)$$

Where E_x, E_y and E_z are respectively pitch, yaw and roll. Q is a quaternion with components ordered X, Y, Z, W. We use the spherical quadrangle interpolation (SQUAD) method [25] to interpolate between quaternions; cubic interpolation is used for eye direction vectors.

It may be that eye data are provided as 2D positions on an XR device's left and right viewports; in that eventuality, eye data should be projected from viewport space to world space relative to the head. To do so, first, a projection matrix (Eq 2) is constructed from the characteristics of a (virtual) camera (Fov_y: vertical field of view, $Aspect$: the display's width to height pixel aspect ratio, $Near$ and Far: the near and far camera frustum planes distance).

$$Proj = \begin{pmatrix} \frac{1}{Aspect*\tan(Fov_y/2)} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan(Fov_y/2)} & 0 & 0 \\ 0 & 0 & \frac{Far+Near}{Far-Near} & 1 \\ 0 & 0 & -\frac{2FarNear}{Far-Near} & 0 \end{pmatrix} \quad (2)$$

Second, a simple view matrix is made, from the properties of a camera positioned at the origin ($Pos = (0,0,0)$), facing along the forward vector ($Target = (0,0,1)$), and using the up vector ($Up = (0,1,0)$) (Eq 3). With $F = (Target -$

Table 1: A file will be identified as a raw data file as long as it provides timestamp, either eye rotation (left, right or combined) and either head rotation (either Euler angle or quaternion) data. Non-alphabetic characters are removed before looking up names: *camera.quaternion.w* is parsed as *cameraquaternionw*. Custom column names should be added in functions *FindRawFeaturesByHeader* of file *helper.py*.

Data		Accepted column names
Timestamp		*oculots*, *oculotimestamp*, *ocutimestamp*, *etts*, *ettimestamp*, *timestamp*, *ts*
Left gaze direction	X	*leftgazex*, *leftgazedirx*, *lgazex*, *xlgaze*, *lefteyedirectionx*, *leftgazedirectionx*
	Y	*leftgazey*, *leftgazediry*, *lgazey*, *ylgaze*, *lefteyedirectiony*, *leftgazedirectiony*
	Z	*leftgazez*, *leftgazedirz*, *lgazez*, *zlgaze*, *lefteyedirectionz*, *leftgazedirectionz*
Right gaze direction	X	*rightgazex*, *rightgazedirx*, *rgazex*, *xrgaze*, *righteyedirectionx*, *rightgazedirectionx*
	Y	*rightgazey*, *rightgazediry*, *rgazey*, *yrgaze*, *righteyedirectiony*, *rightgazedirectiony*
	Z	*rightgazez*, *rightgazedirz*, *rgazez*, *zrgaze*, *righteyedirectionz*, *rightgazedirectionz*
Combined gaze direction	X	*bingazex*, *bingazedirx*, *meangazedirx*, *lgazex*, *xlgaze*, *meangazedirectionx*, *meangazedirectionx*
	Y	*bingazey*, *bingazediry*, *meangazediry*, *lgazey*, *ylgaze*, *meangazedirectiony*, *meangazedirectiony*
	Z	*bingazez*, *bingazedirz*, *meangazedirz*, *lgazez*, *zlgaze*, *meangazedirectionz*, *meangazedirectionz*
Head quaternion	X	*xcam*, *camx*, *headx*, *xhead*, *camerarotationx*, *cameraquaternionx*
	Y	*ycam*, *camy*, *heady*, *yhead*, *camerarotationy*, *cameraquaterniony*
	Z	*zcam*, *camz*, *headz*, *zhead*, *camerarotationz*, *cameraquaternionz*
	W	*wcam*, *camw*, *headw*, *whead*, *camerarotationw*, *cameraquaternionw*
Head Euler rotation	Pitch	*pitch*, *campitch*, *pitchcam*, *pitchead*, *headpitch*
	Yaw	*yaw*, *camyaw*, *yawcam*, *yawhead*, *headyaw*
	Roll	*roll*, *camroll*, *rollcam*, *rollhead*, *headroll*
Sample validity	Left	*vall*, *lval*
	Right	*valr*, *rval*

$Pos)/\left\|Target - Pos\right\|$, $S = (Up \times F)/\left\|Up \times F\right\|$, and $U = F \times S$. Using the *OpenGL Mathematics* library, these two steps correspond to calls to `perspectiveLH_NO` and `lookAtLH` respectively.

$$View = \begin{pmatrix} S_x & U_x & F_x & 0 \\ S_y & U_y & F_y & 0 \\ S_z & U_z & F_z & 0 \\ -(S \cdot Pos) & -(U \cdot Pos) & -(F \cdot Pos) & 1 \end{pmatrix} \quad (3)$$

To finish, the inverse of the projection matrix multiplied by the view matrix is calculated ($VP^{-1} = (Proj \times View)^{-1}$) to project a 2D viewport position (normalised to [-1, 1]) to a 3D direction vector relative to the head position and rotation (Eq 4). The resulting vector should be normalised.

$$Pos^{world} = VP^{-1} \begin{pmatrix} Pos_x^{screen} \\ Pos_y^{screen} \\ 1 \\ 1 \end{pmatrix} \quad (4)$$

To **identify saccades and fixations** [26] we provide three methods:

- A velocity-based method – using a velocity threshold parameter (in °/s).
- A hidden Markov model method – model's parameters are trained on the velocity signals and hidden states come to represent samples of low (fixations) and high (saccades) velocities.
- A cluster-based method – DBSCAN [27] is fed sample positions and used to separate clusters of points (fixations) from noise (saccades).

Filter parameters to smooth the velocity signal are provided as well (Gaussian or Savitzky–Golay filters).

The following equations are used to convert between position representations on a sphere; 3D unit vector to 2D equirectangular projection (longitude, latitude):

$$fix^{equirect} = \begin{pmatrix} \arctan(fix_x, fix_y) \\ \arcsin(fix_z) \end{pmatrix}, \quad (5)$$

2D equirectangular projection to 3D unit vector:

$$fix = \begin{pmatrix} \sin fix_{lat}^{equirect} * \cos fix_{long}^{equirect} \\ \sin fix_{lat}^{equirect} * \sin fix_{long}^{equirect} \\ \cos fix_{lat}^{equirect} \end{pmatrix} \quad (6)$$

2D equirectangular projection to 2D Mercator projection

$$fix^{merc} = \begin{pmatrix} fix_{long}^{equirect} \\ \log(\tan(\frac{\pi}{4} + \frac{fix_{lat}^{equirect}}{2})) \end{pmatrix}, \quad (7)$$

were $fix_{long}^{equirect}$ is a longitude ($-\pi < fix_{long}^{equirect} < \pi$) and $fix_{lat}^{equirect}$ a latitude ($-\frac{\pi}{2} < fix_{lat}^{equirect} < \frac{\pi}{2}$).

Because our data samples are gaze points located on a sphere, the distance between two points is the angle between them, when manipulating vectors the angle between $\vec{u}$ and $\vec{v}$ is calculated thus:

$$Angle = \arccos(\vec{u} \cdot \vec{v}), \quad (8)$$

the orthodromic distance (great-circle distance) can also be used, though we chose to reduce the number of data conversions in our toolbox and work with vectors as much as possible.

Our toolbox allows head data to be processed by itself to produce a head trajectory. A sliding time-window is used (default

width = 90ms) to calculate the average position of the head a successive time-intervals. It should be considered as if the gaze were constantly centred in the visual field of view (forward vector of the head tracking data projected on a unit sphere). The result is a succession of head centroid positions making up a trajectory similarly to gaze positions do, as such the head trajectory data can be processed to obtained the same features as gaze (e.g., duration, amplitude; Fig 2).

The original use-case of the toolbox was to process data obtained from experiments implemented in the Unity game engine and SteamVR (now OpenXR). Therefore the coordinate convention used is that of Unity, i.e., left-handed (second component [Y] is the up axis and the third [Z] is the depth axis). In order to make sure that new data follow the same convention we recommend gathering eye and head tracking data in simple trials where you can verify that looking left and up result in the same directions in the toolbox.

2.2. *Saliency generation*

Equirectangular saliency maps can be generated on the basis of any positional data (on a sphere) by drawing and accumulating 2D Gaussian kernels (Eq 9) at their position. Traditionally, saliency maps are 2D matrices depicting gaze information on a flat plane, such as a desktop computer screen. In our particular case, gaze data is understood to be a set of points on a unit sphere, surrounding an observer's head. To represent this information visually and to be saved on disk, this is transformed using the equirectangular projection. Even though, the medium is again a 2D matrix, its cells are really positions on a sphere, therefore our saliency generation process must account for its circular characteristic. We generate saliency data by accumulating 2D Gaussian kernels at the location of data positions on an equirectangular map:

$$Gauss(x) = \exp(-\frac{\|x - Pos\|^2}{2\sigma^2}). \tag{9}$$

Where x is a 3D position back-projected from equirectangular to unit sphere coordinates, σ the spread of the Gaussian kernel, and Pos is another 3D position (e.g., a fixation position). We rely on the Euclidean distance here to calculate distances on the sphere instead of calculating the distance between unit vectors (Eq 8). This choice was made to make the process computationally lighter as this is a central operation, we judge it acceptable because distances are most often short so the impact is negligible.

The saliency generation process is optimised by defining a Gaussian window, so that only the relevant parts of an equirectangular map (saliency matrix) are updated. The Gaussian window's size is function of the Gaussian's σ and of the latitudinal distortion of the equirectangular projection, i.e., grows as a function of the distance to the equator (Fig 3). It is defined as centered on a point's position and extends latitudinally by a factor of $\sin(2.5\sigma)$ times the saliency matrix's height, and longitudinally as $(1 + \left|\tan(Pos_{lat} - \pi/2)\right|) * 1.5\sigma$ times the matrix's width, where Pos_{lat} is a latitudinal position in radians ($[0, \pi]$).

The Gaussian kernels drawn are isotropic on the sphere, but not on the equirectangular map due to the latitudinal distortion

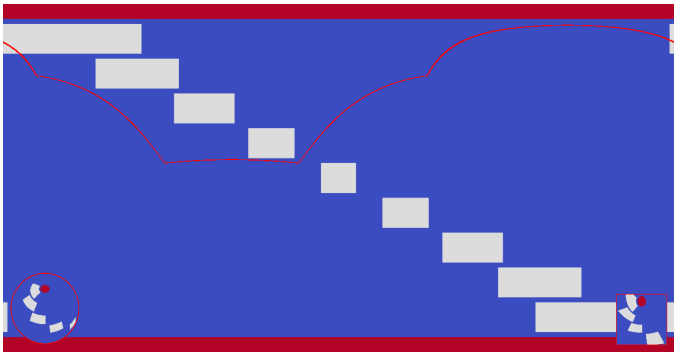

Fig. 3: Illustration of the Gaussian windows calculated when generating saliency data. Shown on the equirectangular projection (background image) the height of the window is a constant function of the Gaussian's σ, while the width changes with the latitudinal position of the point. The Gaussian window appears approximately square once projected in a viewport (bottom-right corner) or back-projected on the sphere (bottom-left corner).

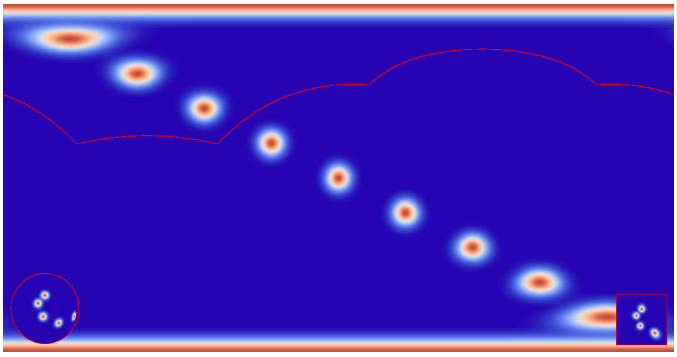

Fig. 4: Mock-up data showing a longitudinal and latitudinal linear progression plotted with 2D Gaussian kernels in order to demonstrate how the distortions obtained from an equirectangular projection increase as a function of the distance to the equator. The red lines delimit an artificial viewport's position, which projection appears in the bottom right. The equirectangular map as background is back-projected on the sphere in the bottom-left corner. It can be observed that the red bands covering both poles completely (top and bottom of the equirectangular map) appear as isotropic Gaussian kernels on the sphere.

resulting from the cylindrical projection (Fig 4). The default σ of the Gaussian kernel is set to 2 degrees, this value should be evaluated taking into consideration the precision of the eye tracking device used and the size of the para-fovea. Someone accustomed to saliency maps created from traditional screen presentation set-ups may note that an equirectangular saliency map appears quite sparse unless many sample points are provided. One has to keep in mind, that while a desktop display may represent approximately 30 to 40 degrees on both axes in a viewer's field of view, the equirectangular map represent content measuring 360 by 180 degrees.

The most common case for saliency data is in relation to fixation positions, generated to obtain information about where observers looked the most, this type of information can be compared between experimental conditions (see section 2.4). Saliency maps can also be obtained from raw data instead of fixations. The resulting maps will implicitly encode information about a fixation's duration: a fixation lasting longer will be made of more eye data samples than a shorter one, as such a long fixation will result in the accumulation of many more

Gaussian kernels drawn at the sample position, and thus will become more salient. Before producing saliency from raw data one should consider that the increase in gaze samples to draw will result in longer computation times.

Video saliency maps can be generated to support protocols showing dynamic stimuli, like videos. In that case, a frame index must be provided along with the raw data, saliency will then be computed frame-wise. A function is provided to output these saliency frames as images, an *ffmpeg* command [28] is generated automatically to splice images together and produce a saliency video. Blended saliency images and videos are produced if an image or a video is provided. The saliency data is added over the original stimulus with an opacity of 70% by default.

2.3. Scanpath generation

Scanpaths (i.e., fixation list) are saved as CSV-formatted files containing any of all 10 calculated features (Tab 2). Absolute and relative saccade angles are calculated on the Mercator projection because it is a conformal projection (conserves angles; Eq 7); Eq 8 is used in that space to obtain angles between 2D vectors. Eq 8 also appears when calculating any distance between points on the sphere, i.e., when calculating fixation dispersion and saccade amplitude. In addition to these features, we provide an index of the fixation/saccade, as well as the start and end timestamps of fixations in order to allow processing raw data samples on the basis of the fixation/saccade data segmentation. In the case of head trajectory data the exact same set of features will be calculated, considering time-window points as fixations, two such "fixations" make the start and end point of a "saccade".

A second type of scanpath generation is proposed in the form of images where the succession of fixation positions is represented as points on a 2D equirectangular map. A colour gradient is used to encode for fixation order, in addition, lines can be drawn between fixation points to emphasise the order and visualise saccades. The toolbox offers options to output this visual representation over a black background or over the original stimulus viewed when the data was recorded.

2.4. Comparing

Saliency data comparison is achieved by comparing equirectangular saliency outputs (Sect 2.2) with established comparison metrics adapted to 360 stimuli. We made available the following metrics: AUC (Borji and Judd), CC, KLD, NSS, and SIM (see [29] and [15] for a review). Implementations of AUC (Borji and Judd), CC, NSS and SIM are originally by Chencan Qian [1]. We added a correction to CC, KLD, and SIM in the form of a weight vector applied to the saliency maps, in order to correct for the equirectangular distortions (latitudinal bias): to give less importance to points near the poles (using sine function). PyTorch [30] implementations of CC, KLD, NSS and SIM measures are provided in the interest of performance and compatibility, though the toolbox will use non-PyTorch implementations by default which are accelerated with Numba.

Scanpaths (time series of saccade/fixation features) are compared using the MultiMatch method [31]. This method does not rely on regions of interest, rather it tries to compare the shape of the scanpaths. It considers and reports several measures of scanpaths:

- *Direction* – Difference between saccade relative angles [5] (where fix_n^{merc} is a fixation position on a 2D Mercator sphere projection ;Eq 7):

$$\begin{aligned} \vec{sacc}_n &= fix_n^{merc} - fix_{n-1}^{merc} \\ \angle\vec{sacc}_n &= -\arctan(\vec{sacc}_{n-1} \times \vec{sacc}_n, \vec{sacc}_{n-1} \cdot \vec{sacc}_n) \\ \Delta Direction &= Abs(\angle\vec{sacc}_1 - \angle\vec{sacc}_2) \end{aligned} \quad (10)$$

- *Duration* – Difference between fixation durations (where fix_n^{dur} is the timestamp difference between the last and first gaze samples making up a fixation):

$$\Delta Duration = Abs(fix_1^{dur} - fix_2^{dur}) \quad (11)$$

- *Length* – Difference between saccade lengths (where fix_n is 3D unit vector for a fixation position on the unit sphere):

$$\begin{aligned} sacc_n^{ampl} &= \arccos(fix_n^a \cdot fix_n^b) \\ \Delta Length &= Abs(\left\|\vec{sacc}_1^{ampl}\right\| - \left\|\vec{sacc}_2^{ampl}\right\|) \end{aligned} \quad (12)$$

- *Position* – Angular distance between fixation positions on sphere (where fix_n is 3D unit vector for a position on the unit sphere):

$$\Delta Position = \arccos(fix_1 \cdot fix_2) \quad (13)$$

- *Shape* – Difference between saccade "shapes" (where $\vec{sacc}_n^{merc}$ is a 2D vector in Mercator space):

$$\Delta Shape = \vec{sacc}_1^{merc} - \vec{sacc}_2^{merc} \quad (14)$$

Note that the measures are normalised between 0 and 1, allowing to be interpreted as a percentage of dissimilarity and to be averaged together to produce a general single-value dissimilarity score. Thus, all measures are divided by pi, apart from the *duration* metric which is divided by the maximum duration observed in the two scanpaths compared.

When comparing **dynamic** saliency maps or scanpaths, data is compared over sequential windows of adjustable duration (). For every time window we calculate measures, then we average the results over the time windows.

[1] https://github.com/herrlich10/saliency, retrieved 2018.

Table 2: List of saccade and fixation features calculated by the toolbox, and available to be saved in CSV files.

Event	Name	Description
Fixation	Duration	Time difference between the last and first samples making up the fixation
	Position	Average gaze (or head position; centroid) as longitude and latitude, or as a unit vector
	Dispersion	Average distance of a fixation's samples to its centroid
	Peak velocity	Maximum velocity observed during the fixation
	Peak acceleration	Maximum acceleration observed during the fixation
Saccade	Amplitude	Angular distance between first and last saccade sample on the sphere
	Absolute angle	Angle between the saccade vector and the longitudinal axis
	Relative angle	Angle between two consecutive saccade vectors
	Peak velocity	Maximum velocity observed during the saccade
	Peak acceleration	Maximum acceleration observed during the saccade

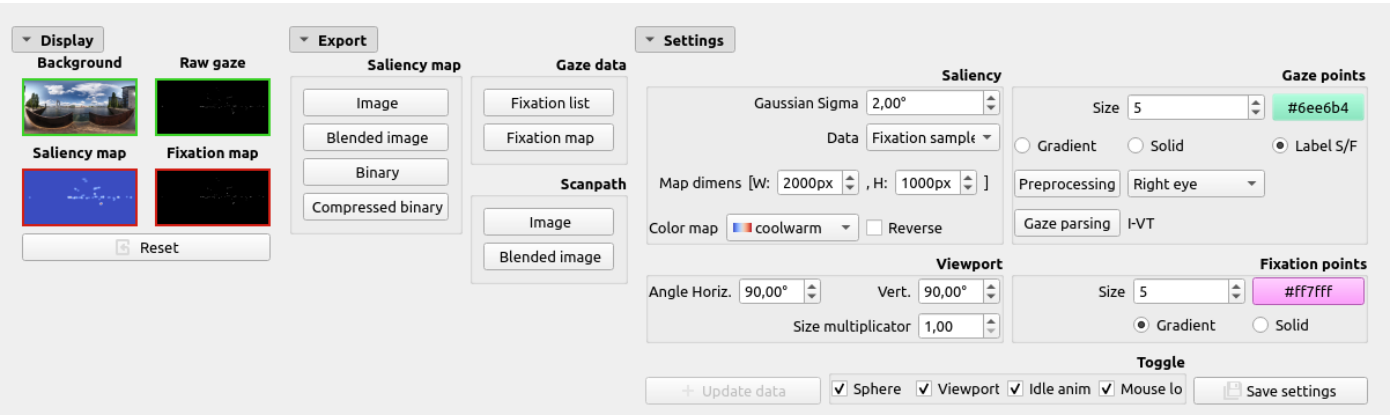

Fig. 5: Screenshot of the display options, outputs, and settings provided by the GUI.

2.5. *Visualising*

We created a graphical interface based on OpenGL and Qt5 to visualise gaze data. The following will launch it from the command line:

```
python -m Salient360Toolbox.visualise
```

We recommend the following uses:

- Assessing data quality, by estimating noise levels in raw data to determine if a better calibration procedures is required, for example;
- Confirming that the to-be-processed data is what is expected by the toolbox (e.g., up in the data is north for the toolbox);
- Experimenting with gaze-parsing parameters by plotting fixation points over a saliency maps calculated from raw data sample positions: a fixation is made of many samples at approximately the same position, thus its location will appear salient and it is easy to identify salient areas that are missing a fixation dot over them as a fixation missed by the gaze-parsing algorithm[2].

The GUI boasts a fair number of options and presets, one can overwrite the default settings by adding to the command line call (e.g., "`[...] --settings VP.mult=5`", will make the viewport's size five times the default), the list of available settings is provided by calling "`[...] --show-settings`". One can load settings from a file in this manner: "`[...] --load-settings settings.set`" (*settings.set* is provided as an example in the toolbox repository). Still in the terminal call, one can provide any number of paths to files or directories as free parameters. Paths leading or containing images or videos will be loaded and set as the equirectangular background image, while the toolbox will attempt to parse, process and display all CSV data files.

The graphical interface allows to visually assess the quality of the data, to experiment with parameters related to the processing and generating functions (Fig 5). It implements exporting functionalities to output saliency maps (it does not support dynamic stimuli), scanpath as coloured points on the original stimulus or fixation lists with the features of your choice. The GUI offers an equirectangular view window on which gaze data appears as points (raw gaze sample or fixation position) over an image, a greyscale saliency map or an image blending the stimulus with the saliency data (Fig 6). This view also presents a spheres on which the equirectangular data is back-projected (lower-right), along with a viewport (lower-right) approximating what would be perceived in a XR headset. The viewport's boundary appears on the equirectangular map as a deformed

[2]Here are the steps to achieve this, 1) drop a raw data file over the GUI, 2) in "*display*" toggle "saliency map" and "fixation map", 3) in "*Settings*" set "Data" to "Raw gaze sample", 4) in the same section click on the "*Update data*" button.

Table 3: A file will be identified as a fixation list file as long as either equirectangular or unit sphere positions are provided. Non-alphabetic characters are removed before looking up column names to simplify the process: *x.gaze* is parsed as *xgaze*. Custom column names should be added in functions *FindFixlistFeaturesByHeader* of file *helper.py*.

Data		Accepted column names
Equirectangular position	Longitude	*long*, *longitude*, *longgaze*
	Latitude	*lat*, *latitude*, *latgaze*
Unit sphere position	X	*x*, *xsph*, *xgaze*
	Y	*y*, *ysph*, *ygaze*
	Z	*z*, *zsph*, *zgaze*
Timestamp		*time*, *starttimestamp*
		timestamp, *timestart*
Duration		*dur*, *duration*
Index		*idx*, *index*, *i*

square with a red border.

If a CSV file is dropped over the equirectangular view, the toolbox will check if it contains raw gaze data or if it is a fixation list containing a series of fixation positions by checking the file's column names (Tab 1 for raw file and Tab 3 for fixation list files). The content will then be processed to produce saliency maps and (raw and fixation list) scanpath data. If the Control key is pressed while dropping a text file, its data will be added to what was currently on screen instead of replacing it. Additionally, if a directory is dropped onto the GUI, the toolbox will attempt to use all files within (be them images, videos or text files). Internally, a list of path to files is stored and each file is reloaded to be processed anew if relevant settings are modified (e.g., gaze-parsing parameters).

3. Usages

One can use the toolbox one of three ways:

- Scripting interface (Python)
- Command line interface (CLI)
- Graphical user interface (GUI)

The **scripting interface** allows the most control of the toolbox, we recommend it for processing entire databases. We wrote a "helper" module (*helper.py*) to simplify accessing the most used procedures. For example, getting features of saccades and fixations from raw data requires parsing a file, processing raw data, labelling samples as fixation or saccades, then calculating features such as fixation duration or saccade amplitude. This is streamlined as the `getFixationList` function which takes as input a path to a CSV file along with parameters related to all the steps involved (e.g., resampling, gaze-parsing). The function will deduce the variables it needs from the CSV file's header. Similarly, the `getSaliencyMap` functions takes as input a fixation list (part of the output from `getFixationList`) and returns a saliency maps, along the way it handles static and dynamic saliency generation, and caching the raw saliency data.

Below is an example of reading, processing, and generating outputs:

```
# Tracking can be HE (Head+Eye) or H (Head alone)
tracking = "HE"
# Targeted eye
eye = "R"
# Resampling rate
resample = 120
# Filter settings
filterSettings = {"name": "savgol", "params": {"win": 9,
↪  "poly": 2}}
# Gaze parsing settings
parsingSettings = {"name": "I-VT", "params":
↪  {"threshold": 120}}
# Dimensions of output images (Height, Width)
dim = [500, 1000]
# Path to CSV file containing raw gaze data
path_raw_file = "/PATH/TO/FILE.csv"
# Path to image stimulus (or video)
path_stim = "/PATH/TO/IMAGE.png"
# Path to write outputs in
path_out = "./"
savename = "example"

from Salient360Toolbox import helper

# Get processed raw data and list of fix/sacc features
gaze_data, fix_list = helper.loadRawData(path_raw_file,
      # If gaze tracking, which eye to extract
      eye=eye,
      # Gaze or Head tracking
      tracking=tracking,
      # Resampling at a different sample rate?
      resample=resample,
      # Filtering algo and parameters if any is selected
      filter=filterSettings,
      # Fixation identifier algo and its parameters
      parser=parsingSettings)

# Generate saliency map from loaded data
sal_map = helper.getSaliencyMap(fix_list[:, [2,3,4,
↪  0,1]], dim,
      # Name of binary saliency file created for caching
      ↪  purposes
      name=savename,
      # If a binary file exists at this location we load
      ↪  the saliency data from it, unless force_generate
      ↪  is True. Saliency will be saved if caching is
      ↪  True
      path_save=path_out,
      # Sigma of the 2D Gaussian drawn at the location of
      ↪  fixations
      gauss_sigma=2,
      # Asks to return saliency data rather than a path to
      ↪  a saliency data file if it exists
      force_return_data=True,
      # Generate data instead of reading from pre-existing
      ↪  file
      force_generate=False,
      # Will save saliency to binary file to fast load at
      ↪  a later time
      caching=True)

# Get a fixation map (2d matrix with number of fixations
↪  observed at each pixel location)
fix_map = helper.getFixationMap(fix_list[:, :2], dim)

from Salient360Toolbox.generation import saliency as
↪  sal_generate
from Salient360Toolbox.generation import scanpath as
↪  scanp_generate

sal_image = sal_generate.toImage(sal_map,
↪  cmap="coolwarm")
```

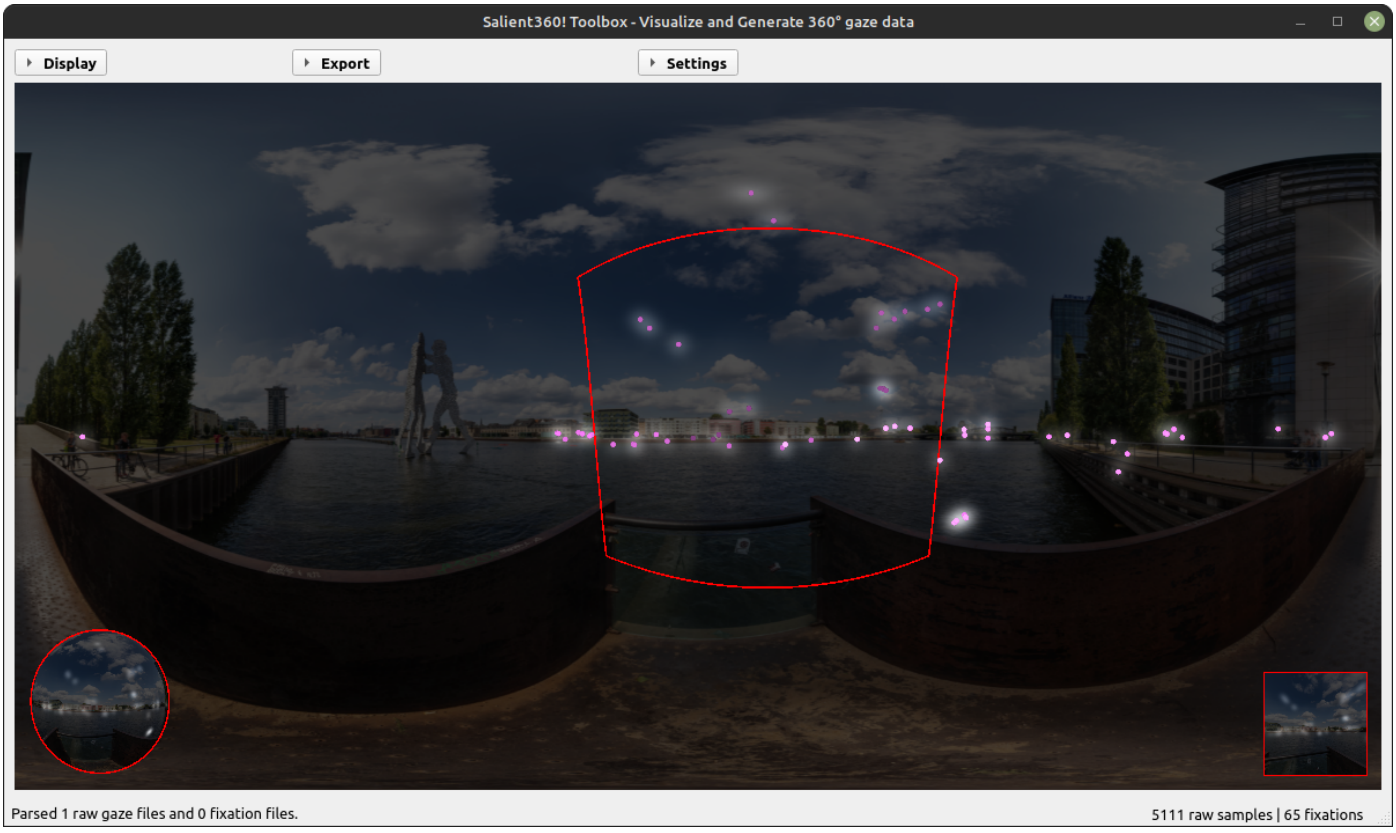

Fig. 6: Screenshot of the equirectangular rendering in the GUI with a sphere (lower-left) and a viewport (lower-right) showing other projections of the data.

```
# (fig 2.a) Save fixation map as a gray scale image
fix_map_img = sal_generate.toImage(fix_map,
↪  cmap="binary", reverse=True)
sal_generate.saveImage(fix_map_img, path_out + savename +
↪  "_fixmap")

# (fig 2.b) Save saliency map as greyscale image
sal_generate.saveImage(sal_map / sal_map.max() * 255,
↪  path_out + savename + "_salmap")

# Save saliency map with as colour map
sal_generate.saveImage(sal_image[:,:, [2,1,0]], path_out
↪  + savename + "_csalmap")

# (fig 2.c) Save saliency map blended with stimulus
sal_generate.saveImage(sal_map, path_out + savename +
↪  "_bsalmap", blend=path_stim)

# (fig 2.d) Save stimulus with fixation points drawn over
↪  it
scanp_generate.toImage(fix_list[:, :2], dim, path_out +
↪  savename + "_bscanpath", blend=path_stim)

# Save scanpath data (fixation and saccade features) to
↪  file
scanp_generate.toFile(fix_list, path_out + savename +
↪  "_fixation.csv",
        # Save all features
        saveArr=np.arange(fix_list.shape[1]), mode="w")
```

The **CLI** gives access to the toolbox through a terminal or invoked via another process. It exposes all functionalities of the toolbox, as well as some of the simplicity of the GUI (see below) by sharing batch processing functionalities (via *helper.py* functions), based on passing directory as an argument and loading all text file within.

The **GUI** is the most limited of the three solutions because it does not allow batch processing of files in order to automatically produce outputs for a database. Nevertheless, as described previously visually assessing data is an important step in many experimental and data science processes. The GUI's state is updated via its options and settings (Fig 5) and by drag-and-dropping a file onto the graphical interface. When registering a file drop event the toolbox will attempt to identify the file type. If it is an image it will be loaded in memory and will be set as the background image (replacing the current one if necessary). As for a video file, OpenCV's video capture module is used to extract a frame. In the case of a text file (CSV or otherwise) the toolbox will parse the first line as a header listing the file's column names (separated by commas) in order to identify it as a raw data file or a fixation list file (see Sect 2.1 and Sect 2.5 for more information on this process).

4. Conclusion

The advent of extended reality devices with embedded eye trackers allows, on one hand, for interactive and omnidirectional viewing conditions with unrestricted movements, on the other hand for easy tracking of head and eye rotations to study eye-in-space behaviour in immersive and controlled conditions.

As a response, scientific and industrial communities have enthusiastically been improving and using XR devices more and more. Today we share with the community the *Salient360!* toolbox: a complete set of tools dedicated to handling eye-in-space data. We provide implementations that cover processing, comparing, generating, and visualising gaze data. The toolbox's functionalities also support processing head movements alone, and supports dynamic stimuli (e.g., video saliency output). We project to continuously improve the toolbox as method standards evolve, in particular we plan the following additions in the short term: new choices of gaze-parsing algorithms (saccade and fixation identification) dedicated to 3D and omnidirectional data (e.g., [32, 33]), the addition of estimated vergence distance data ([34]), better support for dynamic stimuli with the identification of smooth pursuits [32]. We hope that our work will be emancipating for groups that do not wish to develop the skills and tools needed to analyse 3D gaze data. With the release of this toolbox we anticipate discussions about best practices and methods that will certainly lead to improvements and consensus within the communities relying on 3D head and eye tracking.

Acknowledgments

This work was supported by RFI Atlanstic2020, the SFB/TRR 26 135 project C7 to Melissa L.-H. Võ and the Hessisches Ministerium fuür Wissenschaft und Kunst (HMWK; project 'The Adaptive Mind').

References

[1] Liversedge, SP, Findlay, JM. Saccadic eye movements and cognition. Trends in cognitive sciences 2000;4(1):6–14.

[2] Coutrot, A, Hsiao, JH, Chan, AB. Scanpath modeling and classification with hidden markov models. Behavior research methods 2018;50(1):362–379.

[3] Clay, V, König, P, Koenig, S. Eye tracking in virtual reality. Journal of eye movement research 2019;12(1).

[4] Sitzmann, V, Serrano, A, Pavel, A, Agrawala, M, Gutierrez, D, Masia, B, et al. Saliency in vr: How do people explore virtual environments? IEEE transactions on visualization and computer graphics 2018;24(4):1633–1642.

[5] David, EJ, Lebranchu, P, Da Silva, MP, Le Callet, P. What are the visuo-motor tendencies of omnidirectional scene free-viewing in virtual reality? Journal of Vision 2022;22(4):12–12.

[6] Cornelissen, FW, Peters, EM, Palmer, J. The eyelink toolbox: eye tracking with matlab and the psychophysics toolbox. Behavior Research Methods, Instruments, & Computers 2002;34(4):613–617.

[7] Krassanakis, V, Filippakopoulou, V, Nakos, B. Eyemmv toolbox: An eye movement post-analysis tool based on a two-step spatial dispersion threshold for fixation identification. Journal of Eye Movement Research 2014;7(1).

[8] Andreu-Perez, J, Solnais, C, Sriskandarajah, K. Ealab (eye activity lab): a matlab toolbox for variable extraction, multivariate analysis and classification of eye-movement data. Neuroinformatics 2016;14(1):51–67.

[9] Moacdieh, NM, Sarter, NB. Eye tracking metrics: A toolbox for assessing the effects of clutter on attention allocation. In: Proceedings of the Human Factors and Ergonomics Society annual meeting; vol. 56. SAGE Publications Sage CA: Los Angeles, CA; 2012, p. 1366–1370.

[10] Cercenelli, L, Tiberi, G, Corazza, I, Giannaccare, G, Fresina, M, Marcelli, E. Saclab: A toolbox for saccade analysis to increase usability of eye tracking systems in clinical ophthalmology practice. Computers in Biology and Medicine 2017;80:45–55.

[11] Marighetto, P, Coutrot, A, Riche, N, Guyader, N, Mancas, M, Gosselin, B, et al. Audio-visual attention: Eye-tracking dataset and analysis toolbox. In: 2017 IEEE International Conference on Image Processing (ICIP). IEEE; 2017, p. 1802–1806.

[12] Larsson, L, Schwaller, A, Nyström, M, Stridh, M. Head movement compensation and multi-modal event detection in eye-tracking data for unconstrained head movements. Journal of neuroscience methods 2016;274:13–26.

[13] Lappi, O. Eye movements in the wild: Oculomotor control, gaze behavior & frames of reference. Neuroscience & Biobehavioral Reviews 2016;69:49–68.

[14] Le Meur, O, Baccino, T. Methods for comparing scanpaths and saliency maps: strengths and weaknesses. Behavior research methods 2013;45(1):251–266.

[15] Bylinskii, Z, Judd, T, Oliva, A, Torralba, A, Durand, F. What do different evaluation metrics tell us about saliency models? arXiv preprint arXiv:160403605 2016;.

[16] Gutiérrez, J, David, E, Rai, Y, Le Callet, P. Toolbox and dataset for the development of saliency and scanpath models for omnidirectional/360 still images. Signal Processing: Image Communication 2018;69:35–42.

[17] Gutiérrez, J, David, EJ, Coutrot, A, Perreira Da Silva, M, Le Callet, P. Introducing un salient360! benchmark: A platform for evaluating visual attention models for 360° contents. In: 2018 Tenth International Conference on Quality of Multimedia Experience (QoMEX). IEEE; 2018, p. 1–3.

[18] David, E, Gutiérrez, J, Coutrot, A, Perreira Da Silva, M, Callet, PL. A dataset of head and eye movements for 360° videos. In: Proceedings of the 9th ACM Multimedia Systems Conference. ACM; 2018, p. 432–437.

[19] Virtanen, P, Gommers, R, Oliphant, TE, Haberland, M, Reddy, T, Cournapeau, D, et al. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. Nature Methods 2020;17:261–272. doi:10.1038/s41592-019-0686-2.

[20] Harris, CR, Millman, KJ, van der Walt, SJ, Gommers, R, Virtanen, P, Cournapeau, D, et al. Array programming with NumPy. Nature 2020;585(7825):357–362. URL: https://doi.org/10.1038/s41586-020-2649-2. doi:10.1038/s41586-020-2649-2.

[21] Seabold, S, Perktold, J. statsmodels: Econometric and statistical modeling with python. In: 9th Python in Science Conference. 2010,.

[22] Bradski, G. The opencv library. Dr Dobb's Journal of Software Tools 2000;.

[23] van der Walt, S, Schönberger, JL, Nunez-Iglesias, J, Boulogne, F, Warner, JD, Yager, N, et al. scikit-image: image processing in Python. PeerJ 2014;2:e453. URL: https://doi.org/10.7717/peerj.453. doi:10.7717/peerj.453.

[24] Lam, SK, Pitrou, A, Seibert, S. Numba: A llvm-based python jit compiler. In: Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC. 2015, p. 1–6.

[25] Shoemake, K. Animating rotation with quaternion curves. In: Proceedings of the 12th annual conference on Computer graphics and interactive techniques. 1985, p. 245–254.

[26] Salvucci, DD, Goldberg, JH. Identifying fixations and saccades in eye-tracking protocols. In: Proceedings of the 2000 symposium on Eye tracking research & applications. ACM; 2000, p. 71–78.

[27] Ester, M, Kriegel, HP, Sander, J, Xu, X, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In: Kdd; vol. 96. 1996, p. 226–231.

[28] Tomar, S. Converting video formats with ffmpeg. Linux Journal 2006;2006(146):10.

[29] Kümmerer, M, Wallis, TS, Bethge, M. Information-theoretic model comparison unifies saliency metrics. Proceedings of the National Academy of Sciences 2015;112(52):16054–16059.

[30] Paszke, A, Gross, S, Chintala, S, Chanan, G, Yang, E, DeVito, Z, et al. Automatic differentiation in pytorch 2017;.

[31] Dewhurst, R, Nyström, M, Jarodzka, H, Foulsham, T, Johansson, R, Holmqvist, K. It depends on how you look at it: Scanpath comparison in multiple dimensions with multimatch, a vector-based approach. Behavior research methods 2012;44(4):1079–1100.

[32] Agtzidis, I, Startsev, M, Dorr, M. 360-degree video gaze behaviour: A ground-truth data set and a classification algorithm for eye movements. In: Proceedings of the 27th ACM International Conference on Multimedia. 2019, p. 1007–1015.

[33] Llanes-Jurado, J, Marín-Morales, J, Guixeres, J, Alcañiz, M. Develop-

ment and calibration of an eye-tracking fixation identification algorithm for immersive virtual reality. Sensors 2020;20(17):4956.

[34] Duchowski, AT, Krejtz, K, Volonte, M, Hughes, CJ, Brescia-Zapata, M, Orero, P. 3d gaze in virtual reality: vergence, calibration, event detection. Procedia Computer Science 2022;207:1641–1648.