ScreenMasker: An Open-source Gaze-contingent Screen Masking Environment

Pavel A. Orlov^{1,2} · Roman Bednarik¹

Published online: 15 August 2015
© Psychonomic Society, Inc. 2015

Abstract The moving-window paradigm, based on gaze-contingent technic, traditionally used in a studies of the visual perceptual span. There is a strong demand for new environments that could be employed by non-technical researchers. We have developed an easy-to-use tool with a graphical user interface (GUI) allowing both execution and control of visual gaze-contingency studies. This work describes ScreenMasker, an environment that allows create gaze-contingent textured displays used together with stimuli presentation software. ScreenMasker has an architecture that meets the requirements of low-latency real-time eye-movement experiments. It also provides a variety of settings and functions. Effective rendering times and performance are ensured by means of GPU processing under CUDA technology. Performance tests show ScreenMasker's latency to be 67–74 ms on a typical office computer, and high-end 144-Hz screen latencies of about 25–28 ms. ScreenMasker is an open-source system distributed under the GNU Lesser General Public License and is available at <https://github.com/PaulOrlov/ScreenMasker>.

Keywords Gaze-contingent tools · Experimental software · Open-source · Visual attention · Eye movements · Peripheral vision · CUDA · GPU

✉ Pavel A. Orlov
paul.a.orlov@gmail.com

Roman Bednarik
roman.bednarik@uef.fi

¹ School of Computing, University of Eastern Finland,
PO Box 111, FI-80101 Joensuu, Finland

² Department of Engineering Graphics and Design,
Peter the Great St. Petersburg Polytechnic University,
Polytechnicheskaya 29, St. Petersburg 195251, Russia

Introduction

Vision researchers use a plethora of software tools for experimental studies and data processing (Geisler & Perry, 2002; Duchowski & Çöltekin, 2007; Christie & Gianaros, 2013; Ringer et al., 2014). The available tools for eye-tracking experiments typically aim at solving two main problems: (1) executing tests with eye-tracking hardware and (2) processing and analysis of the experimental data obtained. In terms of licensing, this kind of software is often either proprietary or distributed as open-source: see Duchowski's survey of eye-tracking applications in this domain (Duchowski, 2002).

In 1975, McConkie and Rayner introduced the moving-window paradigm for the study of reading processes (McConkie & Rayner, 1975). Letters in the stimulus text were replaced by a symbol at locations corresponding to the direction of the subject's eye movements. The next step was made in 1979, when Rayner and Bertera published their work, which involved a visual mask moving in synchrony with the eye (Rayner & Bertera, 1979). After this advance, gaze-contingent displays were used in various studies (Turner, 1984; Longridge et al., 1989). Under the gaze-contingency paradigm, the visual scene is explored through a gaze-contingent window that can have different forms and shapes, while the remaining part of the screen is masked. The gaze-contingent displays allow researchers to study, inter alia, the role of peripheral information in visual search strategies (Loschky & McConkie, 2002; Perry & Geisler, 2002; Reingold et al., 2003; Geisler et al., 2006).

Reading is one of the most prominent research areas where the gaze-contingency and the moving-window paradigms have been extensively used (Rayner, 1998; Starr & Rayner, 2001; Rayner et al., 2009; Rayner et al., 2010).

Rayner and Bertera employed a mask-image to obliterate foveal vision and the near-parafoveal region during reading (Rayner & Bertera, 1979). This enabled them to study the reading process where fovea is not involved, that is, they emulated the experience of subjects with a retinal scotoma. Several experiments with an artificial central scotoma were also presented by Wensveen et al. (1995), who used a stimulus display with 1.5–8 degree letter size to discover that the letter size and presentation duration are optimized, and that the reading rates of subjects with a central scotoma are below the foveal rates (Wensveen et al., 1995). Bernard used a gaze-contingent visual display to simulate a visual central scotoma and found that reading speed was only slightly modulated by interline spacing (Bernard et al., 2007).

Gaze-contingent displays were also used to simulate a visual central scotoma and cataract (Fine and Rubin, 1999). In addition, gaze-contingent systems facilitated the research into the impact of simulated visual field defects on the heading task performance (Cornelissen & van den Dobbelen, 1999), on human walking in virtual environments (Apfelbaum et al., 2007), and on visual search (Cornelissen et al., 2005).

Search and reading are not the only domains of visual attention that can be explored by means of the gaze-contingency methodology. For instance, recent research in computer programming focused on the behavior of the expert and the novice programmers reading a source code for comprehension (Busjahn et al., 2015). There is also research into the role of peripheral vision in programming (Bednarik & Orlov, 2012), which further necessitates new tools in the emerging domains. In these domains, there is a need to carry out gaze-contingency studies with interactive environments such as text-processors, simulators, games, and web browsers.

There are numerous ways to develop a gaze-contingent tool; capturing eye movements and altering the stimulus can, for example, be done with MATLAB (via psychtoolbox) or PYTHON (via PyGaze). These approaches require eye-tracking programming experience, knowledge of graphic card programming, and costly proprietary software (Brainard, 1997; Dalmaijer et al., 2013).

This paper describes ScreenMasker, software belonging to the first type of eye-tracking experimental tools. It allows customizable gaze-contingency experiments with a textured (masked) screen. ScreenMasker can be used in combination with SMI (and, possibly, other eye-tracking systems) and a GUI helper we developed. ScreenMasker makes gaze-contingent window studies easily available to non-technical users. Therefore, the software will help expand the community of researchers who can utilize the gaze-contingency paradigm.

In this paper, we present the architecture of ScreenMasker and list typical cases in which it can be employed.

We also present a systematic latency evaluation and discuss the tools limitations and prospects.

ScreenMasker

Architecture

ScreenMasker is a textured stimulus presentation tool used to study visual perception and eye movements. The software can be applied in a wide range of tasks, including studies of usability, visual search, perception span, and other properties of fixation zone and field of view. ScreenMasker enables non-technical users to study the properties of vision and attention in real, yet controlled tasks using both everyday and laboratory stimuli-display applications.

The software's architecture was designed to meet the requirements of real-time eye movement experiments. It provides a large number of features and does not require command-line literacy and scripting expertise. The components of ScreenMasker are presented in Fig. 1. The program runs on the stimulus PC and consists of two parts: the Renderer (ScreenMasker.exe) and the Launcher. The Renderer was developed using Microsoft Visual Studio and C++ language. This component contains neither graphical user interfaces nor control elements. The Renderer is set up by means of an options file that is automatically created by the Launcher and can be manually edited by the user (see "Appendix" for the options file details). We also developed a graphical tool launcher, which sets up the ScreenMasker system.

Raw eye-tracking data are stored in a separate text file. Though we developed our own interface for SMI trackers, the system's open architecture can accommodate third-party eye-tracking vendors. For example, the ETU driver can be used for interfacing with commonly available eye-tracking technology (Spakov, 2006).

The ScreenMasker application opens on top of a desktop in full-screen mode (see Fig. 2). It immediately begins

Fig. 1 The component diagram of the ScreenMasker environment

Fig. 2 Layers of ScreenMasker stimuli rendering system. *Layer A* - Desktop layer of the operating system; *Layer B* - Application level (experimental stimuli); *Layer C* - ScreenMasker texture with stencil

to render the screen in a gaze-contingent fashion. ScreenMasker's window does not have any decorations such as borders and buttons, and it acts as a top transparent layer that stays on top of all other user applications. Therefore, user interaction with other running programs is not hindered or altered in any way.

Internal Processing

The transparent layer contains a processing field for the masked pattern texturing. In this solution, we used the SMI

development kit for processing UPD packages with eye-tracking data from the IViewX station. The IViewX system works as a server and sends packages to the stimulus PC via LAN connection (see Fig. 1). We did not resort to fixation detection, but operated directly with the coordinates obtained.

Having obtained the X and Y coordinates, ScreenMasker employs the NVIDIA GPU processing at the pixel level. CUDA technology provides the GPU type of pixel calculation with a high level of parallel processing. The largest screen size that can be handled by ScreenMasker depends on the CUDA version of the graphics card. For the level before 2.0, the screen size is 8192×8192 pixels.

We used a standard CUDA implementation to ensure that ScreenMasker can run on typical office computers with an NVIDIA graphics card. The amount of GPU memory allocated by ScreenMasker in order to calculate CUDA depends on the screen size, stencils dimensions, and the size of the masked pattern.

User Interface for Launching User Studies

The Launcher is a customizable software interface designed to provide a user-friendly graphical access to the options file of the Renderer. In the Launcher GUI, the user selects the masking pattern, the stencil, and other parameters (see the use case below). For this work, we developed a JavaFX-based launcher with a graphical user interface as one of the many options to configure the system. The Launcher of ScreenMasker with the GUI is shown in Fig. 3.

Fig. 3 The main GUI elements of ScreenMasker launcher and the result—the texture on a screen. The masking pattern and a stencil are shown on the *left* and the result of that combination with black color for masked pattern is shown on the *right*

Use Case

To start working with ScreenMasker, the user selects the pattern to be used for contingent masking of the stimulus, the stencil, and the textured color. The researcher can perform an experimental study with any kind of custom stimuli display software. For example, ScreenMasker can be used jointly with a typical desktop application, a computer game, or other stimuli-generation software. ScreenMasker provides several options to change the default settings:

Masking Pattern This pattern is used by ScreenMasker to texture the entire display screen. It consists of white and black pixels. The former are transparent and to see the stimuli application behind ScreenMasker. The latter, in contrast, are visible and prevent the subject from seeing the underlying layer. The experimenter can choose any picture in the portable graymap format (PGM) and set the size of the source image. The interaction with the background application is not restricted in any way.

Figure 4 shows an MS Windows(R) application masked with ScreenMasker. The text shown is hard to read due to the selected masking. The experimenter can choose another pattern and control the visibility of the background. The window can also be masked entirely. In this case, ScreenMasker automatically positions the tiles of the selected pattern onto the whole screen. Different sizes of the picture can be used as a masked pattern, as exemplified in Fig. 6 (right).

Stencil The picture selected as *the stencil* is subtracted from the texture and placed in the gaze position.

ScreenMasker continuously executes a rendering algorithm that subtracts the contents of the stencil from the textured pixels. The values of the black pixels' color from the stencil correspond to the intensity of the opaqueness of

the mask. The opaqueness can range from zero (black color) to the maximum value (white). ScreenMasker subtracts the intensity from the texture.

For example, let us consider the stencil as shown in Fig. 3 (left). In this case, the processed image will appear as shown in Fig. 3 (right). The area will have no borders around the focal point and will contain gradient transparency from the center towards the border. During the experiment, the subject can see through the transparent pixels, while the stencil will follow his gaze.

Any picture in portable graymap format (PGM) can be selected as a stencil. Different sizes and shapes of the stencil can be used, as exemplified in Fig. 5.

Shift The stencil position can be adjusted in relation to the actual gaze point. It is achieved by user-defined *X* and *Y* positions. This option can be used for calibration with eye-tracking systems to level out the systematic errors of gaze measurements.

Color As we have mentioned, the black pixels of the pattern are visible. This option is used to change their color. For example, in Fig. 6 (left) white color used for the mask.

Eye Tracker UDP Server This setting indicates the port and IP address of the UDP server.

Eye Tracker Windows This setting indicates whether ScreenMasker will display additional eye-tracker windows.

These parameters being set, the researcher can start the masked screen and the Java Launcher will be closed automatically.

Fig. 4 Using ScreenMasker with a text editor. The text is visible only inside the stencil, which follows the gaze. The screen outside the stencil is masked, making the text details invisible

Fig. 5 *Left* ScreenMasker is used with a web browser and an eye image as the stencil. *Right* ScreenMasker is used with a cross image as the stencil

Performance Evaluation: Latency Tests

The performance of the rendering and gaze-contingency system was evaluated by means of a latency test (Saunders & Woods, 2014). The latency is the time lag between the movement of the input device and the change of the stencil position on the screen. In our study, we did not rely on the eye-loss event as offered by Saunders and Woods (2014). Instead, we examined real eye movements. To measure the latency of ScreenMasker, we employed the measurement method already presented in Orlov and Bednarik (2014). A high-speed GoPro camera was used to record eye movements and the corresponding movements of the stencil across the screen. The latency of ScreenMasker is the time lag between these two events.

Apparatus

The PC was an Intel(R) Core(TM) 2 Duo CPU E8400 @ 3.00G Hz desktop computer with 3.25 GB of memory and the Microsoft Windows 7 operating system. The graphics

system contained an NVIDIA GeForce 8800GT graphics card and two displays:

1. DELL LCD flat screen (Flat Panel Monitor P2210, 22 inch, 47.5 × 29.8 cm) 1680 × 1050 px, 60 Hz.
2. High-performance monitor BENQ XL2411 with 24" screen size, 53 cm × 30 cm, 1920 × 1080 px with 144 Hz.

This hardware allowed us to compare the latencies of a typical office system with those of a high-end experimental screen. A GoPro HERO 3+ Black Edition camera 240 fps with 848 × 480 px screen size was used to capture the screen. Eye movements were registered with an SMI RED250.

The total latency of the system is calculated as ScreenMasker latency (with screen rendering process on) plus the eye trackers latency. For example, in the 60-Hz mode, the frame duration is about 16.7 ms, while in the 250-Hz mode, 4 ms. The latter figure is much less likely to affect the overall latency of the system.

Fig. 6 *Left* ScreenMasker over a source code editor background. The focus point is at the top left tab bar. White color was chosen for the patterns. *Right* ScreenMasker with a web browser and an oversized leaves-texture masking pattern

Theoretically, the 250-Hz eye tracker rate will provide, in the *worst case* of frequency synchronization, a latency of 26 ms: 4 ms tracker + 17 ms (60 Hz) screen + 5 ms GoPro. For the 144 Hz screen, the latency will be about 9 ms. These are the limits of frequency synchronization. To obtain the real latency, the following parameters are to be included in the calculation: (1) time for image processing required by the eye tracker to deal with the eye parameters (e.g., X and Y position and pupil size), in this particular case, it was iViewX software internal processing delay; (2) time for sending the UDP package to the stimuli workstation, and (3) time to process the UDP package and prepare the image for the screen.

ScreenMasker renders the stencil position on a sample gaze coordinates (X , Y) streamed by the eye tracker. To prevent sample-to-sample noise of the eye-tracker, we suggest that the eye tracker's data output filter should be used. All latency measurements of ScreenMasker's performance were done with the SMI Bilateral filter (Tomasi & Manduchi, 1998) recommended for Hi-Speed data.

Procedure

The eye movements are typically registered in the following way: a video-based tracker captures a video frame with the eye, detects the eye, and infers the gaze direction (Duchowski, 2007). The latency of an eye-tracking system (i.e., the lag between an eye movement and the detection frame) is determined by the speed of video frame capture and the processing time.

We conducted a series of tests to evaluate the effects of the eye-tracking sampling rate and the screen refresh rate on the latency limits of the screen rendering.

In the first test, we employed sampling frequencies of 60, 120, and 250 Hz; a screen with refresh rate of 60 Hz; and a fixed size 200×200 px of the stencil.

In the second test, we prepared three types of stencils with different sizes (200×200 px, 400×400 px, and 800×800 px) to estimate the effects of pixel processing on the latency. We used only the 250-Hz eye-tracking mode with the high-performance monitor BENQ XL2411(144 Hz).

In each of the trials, two icons were shown on the left and the right sides of the screen. These were standard 45×45 px desktop icons of text documents (real size on the screen 13×13 mm, visual angle 1.3°). The distance between icons was 23 cm in real size (21° visual angle). The background color was white. The subjects' eyes were approximately at 62 cm from the screen.

One male subject with normal vision attended to the evaluation. The eye-tracking calibration was performed at the beginning of the trial. The participant was instructed to repeatedly attend to one icon, and then to the other. In total, he performed 30 switches (shifts). The subject's eyes and the computer screen were recorded by the measurement system. For each of the sets, we collected 30 shifts on the GoPro camera. The camera was placed so that it could record the subjects' eyes and the screen at the same time.

Results

A summary of latency times (ms) samples with the 60-Hz screen is shown in Table 1.

An ANOVA test for three sets of modes (250, 120, and 60 Hz) showed no statistical significance: $F(2.86) = 1.127$, $p = .324$, effect size = .026. The standard deviation was greatest in the 60-Hz mode, while the minimum mean latency values were recorded at 250 Hz.

A summary of latency times (ms) samples for the 144-Hz screen is shown in Table 2.

An ANOVA test for three sets of modes (200×200 px, 400×400 px, 800×800 px) showed no statistical significance: $F(2.81) = .748$, $p = .502$, effect size = .015.

Discussion

The key characteristics of a window-contingent tool include the speed of response to the pointer or gaze movements and the update delay, that is, the latency of the contingent response. The visual signal takes about 75 ms alone to reach the brain areas related to motor control. Therefore, the preferable latency in a gaze-contingent system (the time between the onset of the eye movement and the stimulus

Table 1 The comparison of latency time on three frequency types (60 Hz screen)

Eye tracker frequency, Hz	Shapiro-Wilk normality test, p-value	Min	Mean	Max	Standard Deviation (SD)
60	.1825	30	67.67	100	20.79
120	.1028	28	73.55	98	15.6
250	.184	24	67.1	92	16.83

Table 2 The comparison of latency time on three stencil sizes and 250-Hz eye-tracker frequency (144-Hz screen)

Stencil size, px	Shapiro-Wilk normality test, p-value	Min	Mean	Max	Standard Deviation (SD)
200×200	.1415	8	25.62	64	14.12
400×400	.182	8	28.2	64	15.13
800×800	.11	8	27.15	60	15.32

drawn on the screen) should not exceed that value (Leigh & Zee, 1999; Loschky & Wolverton, 2007). This requirement is normally a major challenge for gaze-contingent systems.

Numerous studies in the gaze-contingency field focus on rendering methods (e.g., the level of detail), but do not report on the systems total latency (Ohshima et al., 1996; Luebke & Hallen, 2001; Murphy & Duchowski, 2001). Geisler and Perry (1999) use hardware acceleration with special software to build real-time foveating imaging systems. They report on the encoding and decoding of an 8-bit video at over 40 frames per second with a moderate degree of foveation.

Turner (1984) varied update delays between 130 and 280 ms to find that path following and target identification performance falls as update delays rise (Turner, 1984). For gaze-contingent multiresolution displays, it was found that update delays of 80 ms increase the speed of image detection in a major way (Loschky and Wolverton, 2007). Loschky and McConkie showed in an experiment that 5 ms is a no-delay baseline, 60–80 ms being the upper limit (Loschky & McConkie, 2005).

Available gaze-contingency studies rarely report on latency measurements. Aguilar and Castet (2011) carried out a gaze-contingent simulation of retinopathy. They used professional-grade hardware: Intel(R) Xeon(R) CPU - 3.23 Go RAM - NVIDIA Quadro FX580 (512 MB) and CRT 100-Hz color monitor. The rendering latency of their system was estimated to be about 30 ms, without the additional pixel processing required (Aguilar & Castet, 2011).

Dorr and Bex (2013) present their result in a peri-saccadic natural vision using gaze-contingent screens. They used a high-end workstation graphics board (NVIDIA Tesla C2070) with high-speed ViewSonic 3D VX2265wm 120 Hz. They report on the overall theoretical latency between the eye movement and its effect on the screen ranging at about 26 ms (Dorr & Bex, 2013). We also did not find direct latency measurements in two previous studies (Aguilar & Castet, 2011; Dorr & Bex, 2013) or in the study by Hayhoe et al. (2002).

Cornelissen et al. studied the influence of artificial scotomas on eye movements during a visual search and reported a latency of approximately 20–28 ms. The authors made an independent recording of eye movements and of

the photocell to measure the subsequent stimulus update-related changes in the light flux on the screen. However, the information on latency test procedure was not provided (Cornelissen et al., 2005).

We conducted a direct latency measurement of ScreenMasker using a high-resolution screen, a high-speed camera, and an established measurement procedure. We revealed that the mean latency of the system is about 67–74 ms on the 60-Hz screen and 25–28 ms on the 144-Hz screen. In both cases, these latencies fall under the 80-ms limit. Our subjects reported that the stencil was rendered faster than they saw the masked image, and none reported being able to perceive a temporal lagging. Therefore, ScreenMasker can safely be used in contemporary gaze-contingency studies. Within its limits, the tool can be useful in families of experiments where the latencies mentioned are relevant to the experiment design.

The tool can also be incorporated into the experimental toolkit of applied practitioners. The possible tasks include, inter alia, HCI and programming cognition research, optimization of graphical user interfaces in general, and problem-solving and program comprehension in particular. The studies that require even higher performance that exceeds the minimal latencies need to consider the given limitations.

Studies with ScreenMasker can be carried out on standard graphics card hardware configurations that support the first CUDA level without other special video cards or displays. We employed a high-resolution screen, but others can use more powerful hardware, with smaller screen sizes and resolutions. This will allow decreasing the effects of latency factors even further.

ScreenMasker diverges from the moving-window paradigm in the reading processes (McConkie and Rayner, 1975). One of the differences is that the mask is applied to the whole screen. This means that it covers the entire surrounding field, and not just the letters out of focus, as implemented in the moving-mask or the moving-window paradigms (McConkie & Rayner, 1975; Henderson & Ferreira, 1990; Starr & Rayner, 2001).

The ScreenMasker environment can be controlled by an iterative script, and thus is fully customizable on a trial-to-trial basis. The script can also be adjusted so as to modify

the parameters in each trial. The current version of ScreenMasker acts as an overlay and cannot manipulate the screen content on a trial-to-trial basis. That is why we recommend that ScreenMasker be employed with stimuli-presentation software such as Ogama (Vosskühler et al., 2008) or Experiment Center by SMI. Alternatively, ScreenMasker (open-source license) can be built into another program on a trial-to-trial basis with different eye trackers.

Dissemination of ScreenMasker

ScreenMasker is published as open-source software under the GNU Lesser General Public License (Version 2). By editing the source code of ScreenMasker, the software can be extended to virtually any needs that may be encountered during the development of experimental tasks and studies in the field of gaze-contingency paradigm.

Thanks to liberal licensing, fellow researchers can (and are) strongly contributing their own ideas to the ScreenMasker main release in order to optimize the tool for different fields of research. The source code can be downloaded from <https://github.com/PaulOrlov/ScreenMasker> and the use case video is available at <http://vimeo.com/85146929>.

Appendix

An example of options.ini file:

```
[Base]

stencilPicturePath = D:\\soft\\img\\stencil.pgm
maskedPicturePath = D:\\soft\\img\\Mask.pgm
color = 1A3399
shiftX = 32
shiftY = 31
serverIP = 192.168.0.2
serverPort = 4444
listenUDPIP = 192.168.0.3
listenUDPPort = 5555
isEyeWindowShow = true
isEyePositionWindowShow = false
isCalibrate = true
```

Explanation:

The field «stencilPicturePath» is the full path of the «stencil» image file on your hard disk. This image will be rendered at the gaze' position on the screen.

The field «maskedPicturePath» is the full path of the «masked» image file on your hard disk. This image will be used as a tile for screen's texture.

«color» Hexadecimal representation of the color code to be replaced (without # symbol).

«shiftX» Stencil that will be shown at X pixels from the gaze position.

«shiftY» Stencil that will be shown at Y pixels from the gaze position.

«serverIP» Current IP.

«serverPort» Current UDP port address.

«listenUDPIP» IViewX IP.

«listenUDPPort» IViewX port.

«isEyeWindowShow» If this flag is true, the window with the subject's eyes will be rendered.

«isEyePositionWindowShow» If this flag is true, the window with the subject's eye locations will be rendered.

«isCalibrate» If this flag is true, the calibration will be started at the beginning.

References

- Aguilar, C., & Castet, E. (2011). Gaze-contingent simulation of retinopathy: Some potential pitfalls and remedies. *Vision Research*, 51(9), 997–1012.
- Apfelbaum, H., Pelah, A., & Peli, E. (2007). Heading assessment by tunnel vision? Patients and control subjects standing or walking in a virtual reality environment. *ACM Transactions on Applied Perception (TAP)*, 4(1), 8.
- Bednarik, R., & Orlov, P. A. (2012). Investigating the role of programmers peripheral vision: A gaze-contingent tool and an experiment proposal. In *Psychology of programming interest group PPIG 2012*.
- Bernard, J.-B., Anne-Catherine, S., & Eric, C. (2007). Page mode reading with simulated scotomas: A modest effect of interline spacing on reading speed. *Vision Research*, 47(28), 3447–3459.
- Brainard, D. H. (1997). The psychophysics toolbox. *Spatial Vision*, 10(4), 433–436.
- Busjahn, T., Bednarik, R., Begel, A., Crosby, M., Peterson, J., Schulte, C., Sharif, B., & Tamm, S. (2015). Eye movements in code reading: relaxing the linear order. In *23rd IEEE international conference on program comprehension*.
- Christie, I. C., & Gianaros, P. J. (2013). PhysioScripts: An extensible, open-source platform for the processing of physiological data. *Behavior Research Methods*, 45(1), 125–31.
- Cornelissen, F. W., Bruin, K. J., & Kooijman, A. C. (2005). The influence of artificial scotomas on eye movements during visual search. *Optometry and Vision Science: Official Publication Of the American Academy of Optometry*, 82(1), 27–35.
- Cornelissen, F. W., & van den Dobbelaars, J. J. (1999). Heading detection with simulated visual field defects. *Visual Impairment Research*, 1(2), 71–84.
- Dalmajer, E. S., Mathôt, S., & Van der Stigchel, S. (2013). PyGaze: an open-source, cross-platform toolbox for minimal-effort programming of eyetracking experiments. *Behavior Research Methods*, 1–16.
- Dorr, M., & Bex, P. J. (2013). Peri-saccadic natural vision. *The Journal of Neuroscience*, 33(3), 1211–1217.
- Duchowski, A. (2007). *Eye tracking methodology*. London: Springer Science+Business Media.

- Duchowski, A. T. (2002). A breadth-first survey of eye-tracking applications. *Behavior Research Methods, Instruments, & Computers*, 34(4), 455–470.
- Duchowski, A. T., & Çöltekin, A. (2007). Foveated gaze-contingent displays for peripheral LOD management, 3D visualization, and stereo imaging. *ACM Transactions on Multimedia Computing, Communications, and Applications*, 3(4), 1–18.
- Fine, E. M., & Rubin, G. S. (1999). The effects of simulated cataract on reading with normal vision and simulated central scotoma. *Vision Research*, 39(25), 4274–4285.
- Geisler, W. S., & Perry, J. S. (1999). Variable-resolution displays for visual communication and simulation. *SID Symposium Digest of Technical Papers*, 30(1), 420–423.
- Geisler, W. S., & Perry, J. S. (2002). Real-time simulation of arbitrary visual fields. In *ETRA '02 Proceedings of the 2002 symposium on Eye tracking research & applications* (pp. 83–87).
- Geisler, W. S., Perry, J. S., & Najemnik, J. (2006). Visual search: the role of peripheral information measured using gaze-contingent displays. *Journal of Vision*, 6, 858–873.
- Hayhoe, M. M., Ballard, D. H., Triesch, J., Shinoda, H., Aivar, P., & Sullivan, B. (2002). Vision in natural and virtual environments. In *Proceedings of the symposium on eye tracking research & applications - ETRA '02* (p. 7).
- Henderson, J. M., & Ferreira, F. (1990). Effects of foveal processing difficulty on the perceptual span in reading: Implications for attention and eye movement control. *Journal of Experimental Psychology. Learning, Memory, and Cognition*, 16(3), 417–429.
- Leigh, R. J., & Zee, D. S. (1999). *The neurology of eye movements* Vol. 90. New York: Oxford University Press.
- Longridge, T., Fernie, A., Williams, T., Thomas, M., & Wetzel, P. (1989). Design of an eye slaved area of interest system for the simulator complexity testbed. In *The interservice/industry training, simulation & education conference (IITSEC)*, Vol. 1989, NTSA.
- Loschky, L. C., & McConkie, G. W. (2002). Investigating spatial vision and dynamic attentional selection using a gaze-contingent multiresolutional display. *Journal of Experimental Psychology: Applied*, 8(2), 99.
- Loschky, L. C., & McConkie, G. W. (2005). How late can you update? Detecting blur and transients in gaze-contingent multi-resolutional displays. In *Proceedings of the Human Factors and Ergonomics Society 49th annual meeting* (pp. 1527–1530).
- Loschky, L. C., & Wolverton, G. S. (2007). How late can you update gaze-contingent multiresolutional displays without detection? *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMCCAP)*, 3(4), 25–35.
- Luebke, D., & Hallen, B. (2001). Perceptually driven simplification for interactive rendering. In Gortler, S., & Myszkowski, K. (Eds.) *Rendering techniques 2001, EuroGraphics* (pp. 223–234). Vienna: Springer.
- McConkie, G. W., & Rayner, K. (1975). The span of the effective stimulus during a fixation in reading. *Perception & Psychophysics*, 17(6), 578–586.
- Murphy, H., & Duchowski, A. T. (2001). Gaze-contingent level of detail rendering. In *EuroGraphics 2001*.
- Ohshima, T., Yamamoto, H., & Tamura, H. (1996). Gaze-directed adaptive rendering for interacting with virtual space. In *Virtual reality annual international symposium, 1996., Proceedings of the IEEE 1996* (pp. 103–110). IEEE.
- Orlov, P., & Bednarik, R. (2014). Low-cost latency measurement system for eye-mouse software. In *Proceedings of the 8th Nordic Conference on Human-Computer Interaction Fun, Fast, Foundational - NordiCHI '14* (pp. 1085–1088).
- Perry, J. S., & Geisler, W. S. (2002). Gaze-contingent real-time simulation of arbitrary visual fields. In *Proceedings of SPIE 2002, San Jose, CA: Human Vision and Electronic Imaging* (pp. 512–471–5380).
- Rayner, K. (1998). Eye movements in reading and information processing: 20 years of research. *Psychological Bulletin*, 124(3), 372–422.
- Rayner, K., & Bertera, J. H. (1979). Reading without a fovea. *Science*, 206(4417), 468–469.
- Rayner, K., Castelano, M. S., & Yang, J. (2009). Eye movements and the perceptual span in older and younger readers. *Psychology and Aging*, 24(3), 755–60.
- Rayner, K., Slaterry, T. J., & Bélanger, N. N. (2010). Eye movements, the perceptual span, and reading speed. *Psychonomic Bulletin & Review*, 17(6), 834–9.
- Reingold, E. M., Loschky, L. C., McConkie, G. W., & Stampe, D. M. (2003). Gaze-contingent multiresolutional displays: An integrative review. *Human Factors: The Journal of the Human Factors and Ergonomics Society*, 45(416), 307–328.
- Ringer, R. V., Gaspar, J. G., Crowell, J., Loschky, L. C., & Kramer, A. F. (2014). Creating a new dynamic measure of the useful field of view using gaze-contingent displays. In *ETRA 2014* (pp. 59–66).
- Saunders, D. R., & Woods, R. L. (2014). Direct measurement of the system latency of gaze-contingent displays. *Behavior Research Methods*, 46(2), 439–447.
- Spakov, O. (2006). Eye-Tracking Universal Driver (ETUDriver).
- Starr, M. S., & Rayner, K. (2001). Eye movements during reading: Some current controversies. *Trends in Cognitive Sciences*, 5(4), 156–163.
- Tomasi, C., & Manduchi, R. (1998). Bilateral filtering for gray and color images. In *Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271)*.
- Turner, J. A. (1984). Evaluation of an eye-slaved area-of-interest display for tactical combat simulation. In *The 6th interservice/industry training equipment conference and exhibition* (pp. 75–86).
- Vosskuhler, A., Nordmeier, V., Kuchinke, L., & Jacobs, A. M. (2008). OGAMA (Open Gaze and Mouse Analyzer): Open-source software designed to analyze eye and mouse movements in slideshow study designs. *Behavior Research Methods*, 40(4), 1150–1162.
- Wensveen, J. M., Bedell, H. E., & Loshin, D. S. (1995). Reading rates with artificial central scotomata with and without spatial remapping of print. *Optometry & Vision Science*, 72(2), 100–114.