Eye Tracking in Virtual Environments: The Study of Possibilities and the Implementation of Gaze-point Dependent Depth of Field

Bartosz Bazyluk*
Supervised by: Anna Tomaszewska

Computer Graphics Group
West Pomeranian University of Technology, Szczecin

Abstract

In this paper we present the application of an eye tracker as an innovative real-time virtual environment interaction device, that enables a new level of control, and forms a base for many realistic sight-dependent visual effects. Current use and main stream of research regarding eye tracking technology aims at marketing and usability testing, as well as help for people who suffer from manual disabilities. The goal of our work is to spread interest in other uses of this advanced and impressive technology. Apart from the discussion about possible uses of eye tracking in the computer-generated worlds, we provide a complete case study of a depth of field post processing effect for real-time graphics, that relates on the user's gaze point.

Keywords: eye tracking, gaze point, depth of field, focus, virtual environment control, real-time visual effects.

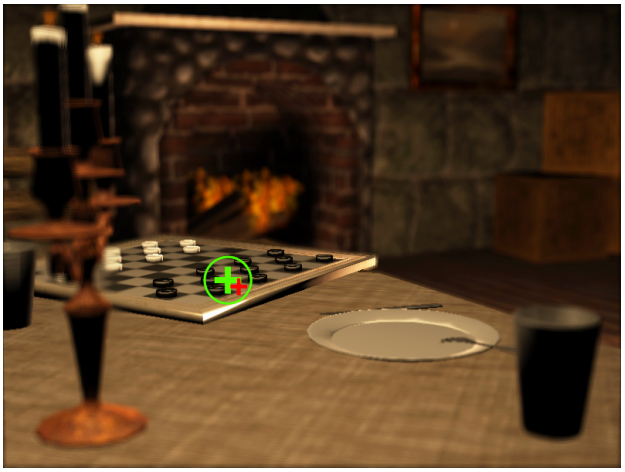


Figure 1: A virtual environment with depth of field dependent on user's gaze point.

1 Introduction

In the past few years a constant search for still more attractive and more innovative controllers used in human to virtual environment interaction can be observed. Examples of this trend that gained commercial success are the famous *Nintendo Wii Remote*, the multiple-input sensitive touch screens offered by Apple in its portable devices and by Microsoft in its *Surface* and *PlayStation Eye* from Sony. Each of these tries to take advantage of previously unused aspects of user's natural activity, improving his experience, and absorbing in an absolutely new and often much more engaging way [6].

Our goal is to encourage the discussion about interaction with artificial, computer-generated world that involves the use of an eye-tracker: a device that unleashes great possibilities that reside in the human visual system by constantly observing the user's eyes, and interpreting their movement to provide accurate information on the person's behaviour at a real time basis. The idea of such application of an eye tracker has emerged together with the recent advancement in eye tracking technology, which made it much easier to use and much more comfortable to utilise (see Section 2).

The first and most basic ideas regarding the use of eye tracking in video games, relied mainly on shifting the control of well-known features from other input devices to the eye tracker. These concepts include gaze-based virtual camera rotation and weapon aiming [10, 13]. Such applications however force the player to unnatural behaviour, requiring him to be constantly aware of where he looks and disallowing him to perform actions as simple as scanning the game status display, without invoking unwanted camera movement. The poor reception of these innovations is clearly shown in [13]. The other approach is driven by the idea of eye tracking as a technology broad in its capabilities to such an extent, that it should not just replace any of the well-known functionality. It rather should expand it, or open the door to a brand new, wide range of possibilities for enhancing the user's experience and creating previously impossible effects that simulate the natural world. Identifying ourselves with the latter belief, we took up the challenge of improving the well known depth of field effect, using an eye tracker.

*bartosz@bazyluk.net

Intending to present one of the common post-processing visual effects, we have built a demo application including the artificial depth of field phenomenon dependent on the user's gaze point (see Figure 1). It allowed us to test our assumptions. The development process led us to the problem of universal method for accessing data across different commercially available eye tracking platforms, which we have addressed by creating an interface library. It is meant to solve incompatibility issues and faciliate developing visual effects concerning user's eyes behaviour, by providing ready to use values rather than those uprocessed, supplied by the device itself.

Our paper starts with a brief explanation of eye tracking technology and the way that eye trackers work. It is followed by the description of data that such devices provide and the visual effect, that we will try to enhance later on. We then provide a short survey of possible applications of eye tracking in virtual environments, and head towards the depth of field effect dependent on user's gaze point, the algorithm that we have chosen and finally the implementation.

2 Background

The eye tracking in its original meaning is a technique of gathering and providing the researcher with accurate real-time data concerning movement of subject's eyes and the point-of-regard, a point that can be simply described as the one that the person is looking at [14]. This valuable information was acquired in numerous ways within the past decades. Inaccuracy, constant researcher's assistance and often distracting invasiveness [3] limited the involvement of eye tracking only to scientific research, mainly in the fields of psychology and human reactions. The modern approach, however, lacks the former method's drawbacks and therefore may be successfully used in disciplines that require user's attention unaffected by any disrupting activities or those, where the user's immersion plays the main role – such as video gaming and exploring virtual environments.

Present day devices in vast majority apply the video-based combined pupil/corneal reflection method. The eye, or more commonly both subject's eyes, are exposed to direct, invisible infra-red light, what results in the appearance of so called Purkinje image, a reflection in the cornea, which is accompanied by illumination of the one's pupil (see Figure 2). Captured by a video camera sensitive to the infra-red spectrum, the relative movement of both the pupil and corneal reflection is measured, which allows to estimate accurately the gaze angle (commercial eye trackers achieve the accuracy of less than 0.5 degree [15, 17]). After having the device calibrated, which most often requires the subject to follow with his eyes several displayed points, it is possible to calculate the estimated screen-space gaze point coordinates on the fly [3].

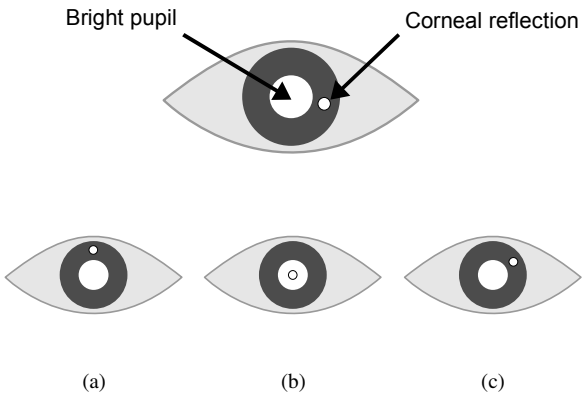


Figure 2: The pupil and corneal reflections as seen by an infra-red camera. Gaze point below the camera (a), in the centre (b) and down and to the right from the camera (c) [14].

2.1 Data provided by an eye tracker

Apart from what seems to be the most interesting, that is position in screen-space coordinates of subject's point-of-regard, eye trackers provide much more useful information that is worth bringing to attention. While calculating the gaze point, an eye tracker utilises many variables acquired from observation of person's eyes actions, that may be employed into calculating visual and behavioural effects. Tracking the movement of both eyes and their Purkinje-image-to-pupil relation, may lead to creation of a head tracking algorithm, and in consequence, the ability to implement extensive number of ways to control the player's avatar in a virtual environment. Also the diameter of subject's pupil, eye's distance from the device, and the camera image may be retrieved [15] depending on the actual device's capabilities.

What is offered by the device's API is not yet everything what may be acquired from an eye tracker. These basic data in connection with the time factor pose a great background to the calculation of more complex, derived values. Beside the most obvious and often necessary fixation times, we propose the introduction of a user's concentration factor concerning an amplitude of his eye movements in certain periods of time. The use of such a variable could involve for example the player's weapon's accuracy.

2.2 Depth of field in real-time graphics

Rendering a scene using artificial methods differs from capturing the real-world scene with a lens equipped camera in the lack of apparatus' physical dimensions. When using an actual device, the rays cast on the sensor are subject to refractions produced by physical properties of the lens. They appear in a photography as the phenomena named circles of confusion (*CoC*s), with diameter varying according to the sensor-lens and lens-object distance. As they blend together, the image appears blurred where the

objects are closer to the camera than it is focused, or when they are too far away. In the case of computer rendering, the image is created with an idealised, pinhole camera containing a virtual lens of zero size. Because of that the rays always cross at a single point, regardless of the distance. To achieve the effect of depth of field, special computation is needed [2].

Depth of field may be called a visual artefact as it limits the acuity of the scene. However, while talking about simulating such visual phenomena by the means of computer graphics, it is important to bring into focus the purpose of that effort. Indisputably the most commonly pursuited branch in artificial generation of images is their realism and effective real-world resemblance. This goal can be divided into three concepts: physical realism, functional realism and photorealism [7]. The latter touches the aspect of viewer being unable to distinguish the computer-rendered image from a photography, what must be achieved not only by accurately recreating the physically-correct light behaviour in the scene, but also by taking into account the limitations of the optics in devices used for image acquisition. Those renderings, which present results impossible to reproduce using conventional photographic techniques, will not fulfil that requirement.

The depth of field rendering is possible using many methods. The one that most closely follows the real-world phenomenon, apart from modelling the light rays behaviour, is an accumulation buffer technique. It involves rendering the whole scene many times to produce one frame, while moving the camera slightly to simulate the acquisition with a device possessing physical dimensions. Despite accurate results, this technique is highly ineffective for real-time graphics due to unacceptable computational overload. There are alternatives that contain simplifications, such as per object multi-layer rendering, or the most common, depth-buffer based methods. The latter owe their popularity to the easiness of obtaining the point's distance to camera, using values existent in the z-buffer. They can be divided into several groups regarding the way of using that information: from forward-mapped scattering methods that are hard to compute parallelly using modern programmable graphics hardware [2], through reverse-mapped gathering methods [5], up to innovative approaches using heat distribution simulation [1].

3 Depth of field with an eye tracker

Having the access to filtered screen-space coordinates of user's point-of-regard, we are able to use the same conventional methods for obtaining the knowledge of place, object, location that the user is actually focused on, as when analysing the mouse pointer coordinates. We present several ideas of utilising these valuable data.

- The ability of calculating the luminosity of an image fragment corresponding to the gaze point and its vicinity, together with high dynamic range rendering allows to alter the scene's exposition level dynamically, which may result in much more realistic tone mapping and accurate blooming, as well as true simulation of human eye's sensory adaptation to varying lighting conditions [11].

- Recording the recent user's scan path can be used to create dynamic, artificial after-images, emphasising the brightness of scene's elements.

- Altering the viewing frustum depending on the user's head position relative to the display screen, e.g. by slightly expanding the field of view when the intermediate distance decreases, creating the impression of looking through the window rather than watching a projected image.

- Also the simulation of vestibo-ocular reflex connected with camera swinging during player's movement may also be a subtle yet heavily immersive addition [9].

These examples of visual effects altering upon gaze shift bring the resulting scene closer to the state of what can be called interactive photorealism.

Worth mentioning is also a completely different application of eye tracking in complex scene rendering, taking into account the inacuity of human peripheral vision in favour of that involving the foveal disk region of retina. Human is in fact able to cover only 1-5 degrees of visual angle with clear, foveal vision, what may be visualised as only 3% of a 21" monitor screen viewed from the distance of 60 cm [3]. What is a human visual system's disadvantage, may be converted into an advantage in the terms of real-time computer graphics, by progressive reduction of rendering quality with the distance from user's gaze point to gain improvement in performance. The quality reduction is possible either in the domain of resolution, displayed scene's geometrical complexity [4], or precision of post-processing effects. This implementation of eye tracking is called the gaze-contingent display method.

In this paper, however, we wanted to present a relatively simple, yet spectacular effect of an artificial depth of field, which varies with the distance from camera of a virtual point that the user is looking at. Our choice emerged according to the fact, that besides improving the impression of scene's depth, an independent of user's eyes behaviour depth of field is often found distracting [8] as it blurs the screen areas that may catch the user's interest. The usefulness of knowledge about the user's gaze point in this field seems to be obvious.

The experiments with interactive, controlled by an eye tracker depth of field effect described in [9] involved the simple reverse-mapped z-buffer method, together with minor artefact correction and a technique of autofocusing, that helped translating the gaze point coordinates provided

by an eye tracker into actual focus distance. It though suffered from a drawback typical to classic depth-buffer approaches that calculate the blurriness with circle of confusion modelling. They are subject to heavily disturbing depth discontinuity problem, when an out-of-focus object occludes the in-focus background. It results in the blurred object's silhouette being clearly visible as a hard edge, which is highly unacceptable [2] (see Figure 4a).

This brought up our deliberations according to the perception of gaze-point dependent depth of field. It is worth noticing, that such way of control results in fully-blurred areas rarely being visible to the user, as they will instantly become clear when gaze is shifted upon them. It is then more important than when creating a non-gaze contingent depth of field effect, to take care of visual artefacts occurring near the objects' edges, as it is where the results of the depth of field will be observed most often. Such artefacts include both depth discontinuity problem and intensity leaking. These observations led us to the search for our own algorithm modification.

3.1 Enhancement of the basic approach

Our approach is an extension to the basic reverse-mapped z-buffer technique with the gathering blurring method based on Poisson disk samples, and involves implementing the extrapolation of *CoC* values for objects closer to the camera than the focus plane. Similarly as in many other methods, we base on a thin-lens camera model and derive from it the circle of confusion diameter equation [5]:

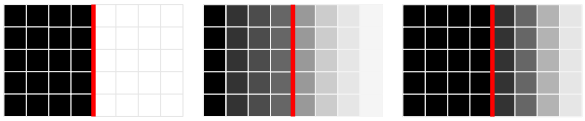
$$CoC = a \cdot \left|\frac{f}{d_0 - f}\right| \cdot \left|1 - \frac{d_0}{d_p}\right| \quad (1)$$

Where a is the aperture, f is the focal length of the lens, d_0 is the focus distance and d_p is the source point's distance to camera. The aperture and focal length need to be selected empirically, as their values should suit the scene's dimensions, the camera's angle of view and the desired intensity of resulting effect. The focal length can be set to increase in very short focus distances, to extend the depth of field for extremely small distance values while maintaining the effect's visibility in longer distances.

To address the depth discontinuity problem described in the previous section, we have surveyed existing solutions. The most basic and popular at the same time [2], involves blurring the *CoC* values represented as a texture, which makes the object edges semi-transparent, as they are mixed with background in the proportion of 50% to 50% (because of blurred *CoC*'s gradual distribution; see Figure 3b). We however decided to follow [5] to calculate extrapolated *CoC* value, using both original CoC_o and blurred CoC_b textures, with the formula:

$$CoC = 2 \cdot max(CoC_o, CoC_b) - CoC_o \quad (2)$$

In effect, we blur the occluding object outside its silhouette, which results in nice, soft edges. The downside is



(a) (b) (c)

Figure 3: Extrapolation of *CoC* values using original (a) and blurred (b) *CoC* texture comparison to produce the result (c) *CoC* texture. Centre line is the object's silhouette, and darker pixels represent the area with higher *CoC* radius.

the fact of background as well being blurred in the occluding object's vicinity (see Figure 4c). This problem may be addressed with multi-layer rendering or per-pixel layers [12], but it would involve multiple scene rendering what we tried to avoid, as this common problem seems to be far less intrusive than the former hard edges.

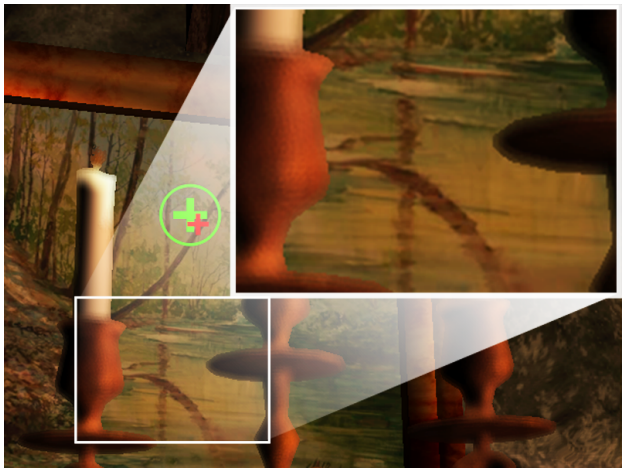
Intending to test our implementation with a modern, accurate eye tracker, we have decided to rely on the gaze point data processed in the same way, as for other uses. We have intentionally abandoned creating an algorithm for autofocusing that was proposed in [9], in favour of averaging performed in the interface library.

4 Implementation

The development of an application that uses the data provided by an eye tracker can be divided into two separate layers: one, that governs the computer-device communication, and the other which is responsible for the data utilisation itself.

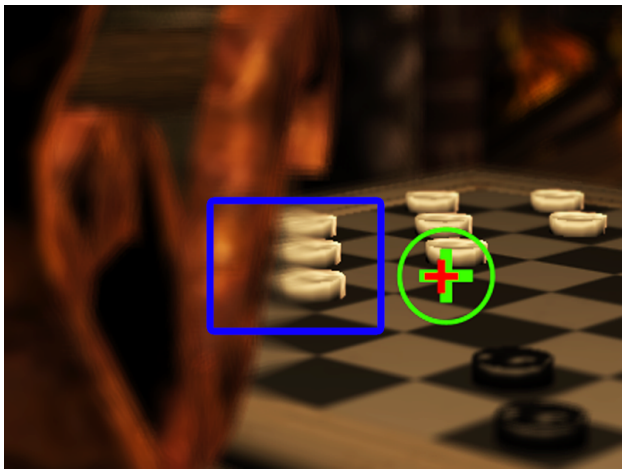
4.1 Communication interface

Bearing in mind the diversity of programming interfaces across the available commercial eye-tracking platforms, it is necessary to introduce a solution which will provide standardised output usable in end applications. Our proposition is a universal library, utilising the concept of adaptor-based architecture (see Figure 5). Offering the ease of extending compatibility to new devices, exposing both primitive and derived data in a uniform, platform-independent manner, as well as offering device capability tests, the library may become a major step in popularising the idea of adopting eye-tracking into the field of virtual reality and gaming. At the time this article is being published, the adaptor for SensoMotoric Instruments (SMI) API is under development, and two adaptors useful for application development and debugging are completed: first simulating the eye tracker with a mouse, and second that provides an artificial gaze path at random or constant basis. The latter two were used in the development process of our demo. Further tests and research will involve the SMI RED250 eye tracker. The library is due to be released publicly when completed.



(a)

(b)



(c)

Figure 4: The depth discontinuity problem (a) and the solution (b) with its downside (c).

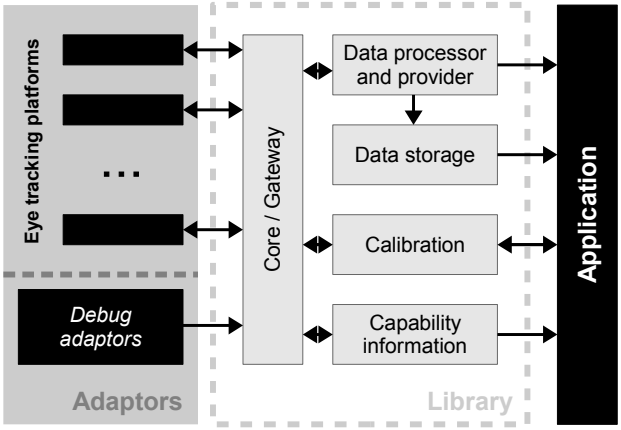


Figure 5: The proposed library architecture.

4.2 Demo application

To test the method, we have prepared a first person perspective demonstration program written in *C#* language, using the most recent *OpenGL 3.2* together with *GLSL 1.50* in its core profile, and utilising the *.Net* based *OpenTK 1.0 beta-2* library. The existing scene is intended to be a testbed for future studies regarding eye tracking applications in both visual effects and interactivity areas. It depicts a fantasy-world interior of a magician's house, which may be easily suited to demonstrate new techniques.

The eye tracker communication layer, utilising its own thread, is the library mentioned in Section 4.1. During each frame generation period, it is queried for both raw and filtered screen-space coordinates of the user's current gaze point. Despite the fact that only the filtered values are necessary for the depth-of-field calculation algorithm, both of them are used for drawing optional on-screen feedback.

The original scene is rendered to a buffer, using the *Frame Buffer Object* with two textures attached: one for colour output, and the other for depth values storage. Then the camera distance from the point equal to filtered gaze-point coordinates is acquired, and assumed to be the focus distance. In the next pass, *CoC* values are calculated with Equation 1 from the depth values in regard to the focus distance, and they are stored in a texture: the nearer-than and farther-than the focus plane values separately. The „near" *CoC* texture is downsampled to 1/8th of its size in each dimension, in order to obtain the blurred *CoC* values used for final computation. In the last pass, blur is applied to the colour texture from the first pass, according to the *CoC* value. To obtain the final *CoC* value which determines the actual blurring radius, the „near" *CoC* value is calculated using Equation 2 and then compared with the corresponding value from the „far" texture. The higher value is used. Finally, the image is rendered on the screen (for the complete diagram see Figure 6).

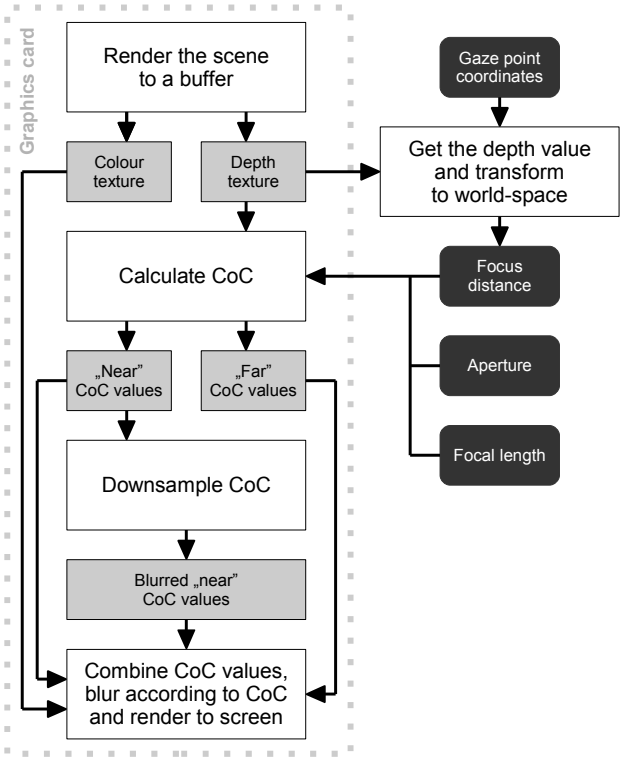


Figure 6: The depth of field rendering algorithm.

5 Results

Developing the demo application enabled us to test our assumptions in practice. The introduction of a consistent interface library led to the possibility of concentrating on the utilisation of provided data itself, rather than processing them. Separating the eye tracker communication from our presentation layer thread resulted in achieving the integration without any noticeable decrease in computing and rendering performance.

The depth of field algorithm that we have chosen to implement delivered convincing simulation of this photographic phenomenon, and offered an improved look of out-of-focus objects occluding the in-focus surface than in the method used in [9]. The implementation did not require us to introduce any changes into the scene rendering process.

The overall outcome brought a new level of interactivity to the artificially generated scene, which proved the very positive results of other studies on using eye tracking in virtual environments [9]. The screens taken from our demonstrative application may be seen in Figure 7.

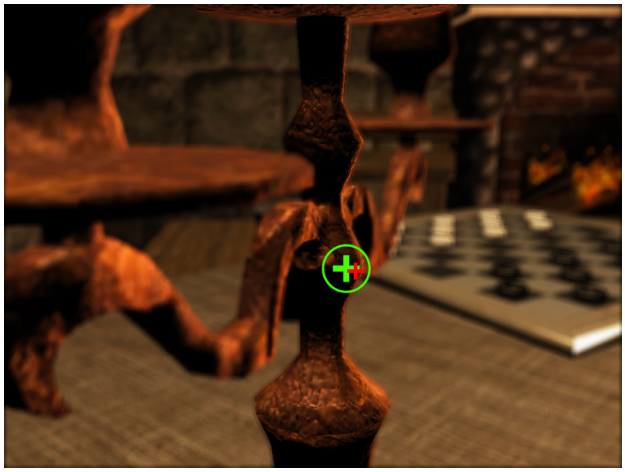


Figure 7: Example screens from our demo, containing on-screen feedback. Green crosshair represents the filtered, while red cross is the raw gaze point.

6 Conclusions and future work

With our work on the demonstration we have proved that modern commercial eye trackers may be successfully used to provide data necessary for rendering advanced, gaze-

aware visual effects in real time. The depth of field algorithm we have used may be improved by addressing the visual artefacts it still produces, like intensity leaking or background blurring in the out-of-focus object's vicinity. However, our approach already emerged to deliver satisfactionary and realistic results.

Our future efforts will be aimed at experimenting with other types of visual effects as well, together with eye-based environment controlling and gaze-contingent rendering performance optimisation. Our demo is planned to include a multi-display feature, that will allow observation of actions performed by the user in real time, displaying visible overlays providing statistical data regarding his eyes' current and recent behaviour, without distracting the subject. The ability to save such data for future interpretation is also projected.

Being aware that eye tracking is consequently gaining interest in the entertainment and portable computers sector, it seems correct to assume that during the few upcoming years we should encounter the introduction of such devices to the consumer market. The rumours of Apple trying to implement an eye tracker in the recently unveiled iPad resulted in discussions about the idea of an eye-controlled operating system, and brought into light the patent for gaze vector navigation that the company has been pending [16]. This makes the search for possible eye tracking use in popular, consumer applications very up-to-date and encourages studies in this yet largely unexplored field.

Acknowledgments

We would like to thank Karolina Lubiszewska for her work on 3D models which we have used in our demo.

References

[1] Marcelo Bertalmio, Pere Fort, and Daniel Sánchez-Crespo. Real-time, accurate depth of field using anisotropic diffusion and programmable graphics cards. In *3D Data Processing, Visualization, and Transmission, 2nd International Symposium*, pages 767–773, Washington, 2004. IEEE.

[2] Joe Demers. Depth of field: A survey of techniques. In Randima Fernando, editor, *GPU Gems*. NVIDIA Corporation, 2004.

[3] Andrew T. Duchowski. *Eye Tracking Methodology: Theory and Practice 2nd Edition*. Springer, London, 2007.

[4] Andrew T. Duchowski, Nathan Cournia, and Hunter Murphy. Gaze-contingent displays: Review and current trends. In *Adaptive Displays Conference, 2004*, Los Angeles, 2004.

[5] Jr. Earl Hammon. Practical post-process depth of field. In Hubert Nguyen, editor, *GPU Gems 3*. NVIDIA Corporation, 2008.

[6] Ross Eldridge and Heiko Rudolph. Stereo vision for unrestricted human-computer interaction. In Asim Bhatti, editor, *Stereo Vision*. InTech, Vienna, 2008.

[7] James A. Ferwerda. Three varieties of realism in computer graphics. In *SPIE Human Vision and Electronic Imaging 2003*, Bellingham, 2003. SPIE.

[8] Sebastien Hillaire, Anatole Lecuyer, Remi Cozot, and Gery Casiez. Depth-of-field blur effects for first-person navigation in virtual environments. In *IEEE Computer Graphics and Applications*, pages 47–55. IEEE, Los Alamitos, 2008.

[9] Sebastien Hillaire, Anatole Lecuyer, Remi Cozot, and Gery Casiez. Using an eye-tracking system to improve camera motions and depth-of-field blur effects in virtual environments. In *IEEE Virtual Reality Conference 2008*, pages 47–51. IEEE, 2008.

[10] Erika Jönsson. If looks could kill – an evaluation of eye tracking in computer games. Master's thesis, Royal Institute of Technology, Stockholm, 2005.

[11] Grzegorz Krawczyk, Karol Myszkowski, and Hans-Peter Seidel. Perceptual effects in real-time tone mapping. In *Spring Conference on Computer Graphics, 2005*, 2005.

[12] Sungkil Lee, Gerard Jounghyun Kim, and Seungmoon Choi. Real-time depth-of-field rendering using point splatting on per-pixel layers. *In Computer Graphics Forum*, Vol. 27 Issue 7:1955–1962, 2008.

[13] J. Leyba and J. Malcolm. Eye tracking as an aiming device in a computer game. Technical report, Clemson University, Clemson.

[14] Alex Poole and Linden J. Ball. Eye tracking in human-computer interaction and usability research: Current status and future prospects. In C. Ghaoui, editor, *Encyclopedia of Human-Computer Interaction*. Idea Group, Inc., Pennsylvania, 2005.

[15] SensoMotoric Instruments GmbH. *RED250 Technical Specification*, 2009.

[16] Chris Stevens. *Is Apple about to open a can of eye-tracking?*, January 2010. http://recombu.com/news/a_M11321.html.

[17] Tobii Technology AB. *Tobii T/X series Eye Trackers. Product Description*, 2.0 edition, 2009.