Journal of Universal Computer Science, vol. 14, no. 19 (2008), 3085-3098
submitted: 23/7/08, accepted: 29/10/08, appeared: 1/11/08 © J.UCS

Gaze-based Interaction for Virtual Environments

Jorge Jimenez, Diego Gutierrez, Pedro Latorre
(Universidad de Zaragoza, Zaragoza, Spain
{jim,diegog,platorre}@unizar.es)

Abstract We present an alternative interface that allows users to perceive new sensations in virtual environments. Gaze-based interaction in virtual environments creates the feeling of controlling objects with the mind, arguably translating into a more intense immersion sensation. Additionally, it is also free of some of the most cumbersome aspects of interacting in virtual worlds. By incorporating a real-time physics engine, the sensation of moving something *real* is further accentuated.

We also describe various simple yet effective techniques that allow eyetracking devices to enhance the three-dimensional visualization capabilities of current displays. Some of these techniques have the additional advantage of freeing the mouse from most navigation tasks.

This work focuses on the study of existing techniques, a detailed description of the implemented interface and the evaluation (both objective and subjective) of the interface. Given that appropriate filtering of the data from the eye tracker used is a key aspect for the correct functioning of the interface, we will also discuss that aspect in depth.

Key Words: Eye tracking, virtual environments, human-computer interaction, video games, input device
Category: H.5.2, I.3.3

1 Introduction

The usual interface in nowadays computers, based in WIMP environments (Windows, Icons, Menus, Pointer) allows for a very limited sense of immersion. Recently, new devices have been introduced in the market, which aim at overcoming some of the current limitations of WIMP. A clear example is the successful wireless Nintendo Wii remote, capable of recognizing a limited subset of the players' movements, or *EyeToy*, a camera-based interaction device for video games which captures an approximation of the users' actions.

In this paper we explore the use of eye tracking devices creating a gaze-based control system that allows the user to interact in the virtual environment in a different way, allowing for a greater sense of immersion in virtual environments due to its ease of use.

1.1 State of the art

Previous work describes and analyzes user interfaces whose only input is the user's gaze [Jacob 1990, Salvucei and Anderson 2000, Ware and Mikaelian 1987]. In those systems, it is possible to select objects, move them or display information about them, within the context of a conventional user interface. Other studies [Sibert and Jacob 2000a] indicate that gaze-based interfaces can indeed be more efficient than traditional mouse- or pointer-based ones.

The work by Tanriverdi and Jacob [Tanriverdi and Jacob 2000b] introduces the selection of objects by sight in virtual environments, and it is compared with a selection mechanism based on a hand-tracking device. Although their results show that gaze-based selection is more efficient, posterior studies reveal that this efficiency depends also on the ray casting algorithm used [Cournia et al. 2003].

Smith and Graham studied the integration of gaze-based interfaces in already existing games of diverse nature [Smith and Graham 2006]. The results show that efficiency drops with respect to a more traditional mouse-based system. However, it is noted that the users' satisfaction increases with gaze-based devices when this is used for the control of avatars. This kind of interfaces in first-person games has also been analyzed by [Isokoski and Martin 2006]. The results confirm that efficiency drops even for this type of games, but it is pointed out that performance when using an eye-tracking device increases with training, and thus similarly-trained users in both types of interfaces should be used for a more fair comparison.

2 Resources used

The *Laboratorio Aragonés de Usabilidad* (Usability Laboratory of Aragón) is a research facility for the development, analysis and optimization of new human-computer interaction techniques. It has provided all the necessary resources for the implementation and evaluation of the work developed in this paper.

2.1 Hardware

To obtain the point in space observed by the user, it is necessary specific, dedicated hardware capable of detecting eye gaze in real time. We have used a T60 eye tracking system, developed by Tobii, which is made up of a conventional monitor with tracking sensors at the bottom. A typical use of this kind of devices is to gather statistics of user interfaces, allowing for an objective evaluation of such interfaces. Other uses of the technology include medicine or psychology. In this work we explore the use of eye-tracking devices as human-computer interfaces within virtual worlds.

2.2 Software

The functional scheme of the system can be broken down into three parts: capture, update and display.

For capture, we have used the in-house software developed by Tobii, *Tobii Eyetracker SDK*, which consists on a series of libraries that allow for the acquisition of the data from the eye tracker. The update of the virtual world requires a real time physics engine to simulate the correct behavior of rigid solids under forces such as gravity or the user's input to the system. We have used the *Simple Physics Engine* [SPE] library, which lets us create realistic simulations through a simple programmer's user interface. Lastly, Direct3D has been used for real time display.

3 New interaction paradigms

The goal of the proposed system is to incorporate actions that let the user alter the virtual world. It is similar in spirit to the work by Jacob [Jacob 1990], although we extend it to the virtual reality realm. In this realm, previous works contemplate the observation of the worlds, but not their modification [Tanriverdi and Jacob 2000b], so the user can only select objects. Contrary to other similar cases [Smith and Graham 2006], we have designed a system from scratch, adapting the environment to the interface, and not the other way around. A key component that ties together the scene being displayed and the user's gaze is the built-in physics engine [Yeh et al. 2006]; this differentiating element helps the user believe that he is actually moving something *real*, as opposed to abstract object representations.

We call the system we present here *EyeMotion*, and it is made up of a fixed ground plane where a series of dice are thrown onto by gravity (Figure 1). The color of the objects change to green while observed by the user, so she knows over which object subsequent action will be taken. For this, there is a series of tools: if the user presses the right mouse button, the event triggers an action, which can be either grab, push or shoot the object (depending on the selected tool). Tools are selected by regular mouse interaction with icons on the screen menu. To change the point of view, we again rely on standard mouse-driven 3D navigation in order not to introduce too many new interaction paradigms.

Another feature in *EyeMotion* is the possibility of showing the point in the scene being observed by mimicking a laser pointer. We included this option given that knowing the output of the *eye tracker* proved to be very useful in cases where it was not particularly precise (a problem inherent to eye tracking technology that must be accounted for). Seeing the laser pointer, the user can make small adjustments based on its position.

3.1 Select

The problem of selecting an object in gaze-based system is known as the Midas touch problem. According to the Greek mythology, Midas would transform in

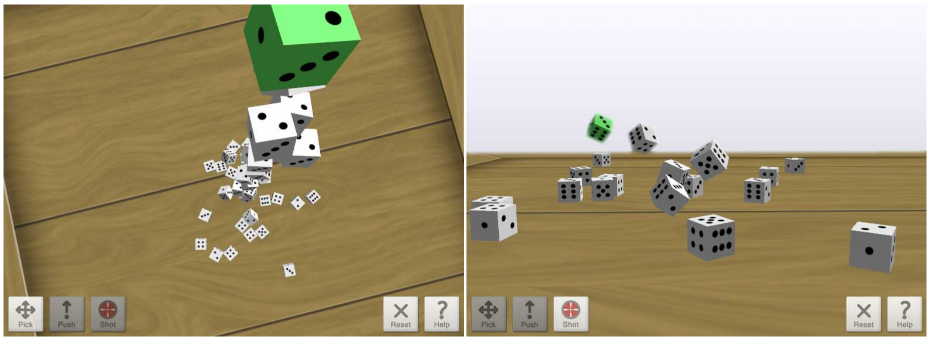

Figure 1: EyeMotion

gold everything he touched: very similarly, unless specific measures are taken, every object in the scene would be selected just by looking at it. To solve this problem, there are three different possibilities:

– Use blinks.

– Build some latency into the selection system (*dwell time* [Sibert and Jacob 2000a]), which activates selection only after staring at an object for a given period of time.

– Use hardware, such as buttons to begin/end actions.

We ruled out blinking since we believe is not a very natural way to activate tools. Plus, it can tire the user. Building latency would not adapt to all the different tools we wished to implement; more precisely, it would not be compatible with the push tool, since that tool would need a button to begin/end such action. Thus, we opted for the simplest solution, which is to use a button to make the selected tool active while it is being pushed, and inactive when released.

For selection, we use ray casting (Figure 2), which allows to detect the nearest object to the camera from any given point of view.

3.2 Grab

The grabbing tool allows the user to grab, drag and drop objects anywhere in the scene. The procedure consists on looking at the object one wishes to move, pushing the action button (after selecting the grabbing tool), looking at the new destination for the object and releasing the button. The physics engine will simulate the necessary rigid body dynamics to make the movement believable. The object can actually be placed anywhere in the scene, given that the view

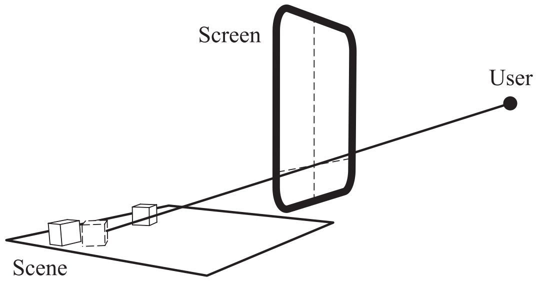

Figure 2: Object selection by ray casting

point can be changed while the object is being grabbed. To accomplish this task, the user would need to grab the object with the action button in order to move the object with the eyes, and then push the camera button to move the camera with the mouse. Both buttons should be pressed simultaneously until the user finishes the grab.

3.3 Push

The push tool is used to move objects without actually looking at them previously to select them. Once the tool is active, objects intersecting the gaze path are pushed out of the way according to the gaze direction. This tool allows for a basic but comfortable way of moving objects, since it does not require the previous selection.

In general, for the eyes to move smoothly throughout a scene, it is necessary that they follow a moving target [Jacob 1990]. Thus, the push tool as it is designed may seem not very appropriate in an eye tracker context; however, once an object starts moving it is possible to continue and steer its movement by gaze following. Additionally, it also possible to filter eye movement to smoothen the results, as we will describe in the following section.

3.4 Shoot

Our last interaction tool consists on shooting objects following gaze direction. To shoot an object, the user looks at the object and pushes the action button, after which the object is shot in the direction given by the normal to the plane of visualization.

4 Data filtering

To be able to clearly see an object, we need to move the eye so that said object falls in the foveal region. This is the central part of the retina, and thanks to its high concentration of color-sensitive photoreceptors (cones) it allows us to observe the region of interest with the maximum resolution and detail our visual system permits. The movements that allow the object to fall in the fovea are called saccades. After each saccade the eye is said to be in a fixation state, during which a series of involuntary movements (called micro saccades) alter the exact gaze direction around the point of interest, normally in an angle less than one degree [Jacob 1990].

This involuntary movements may cause gaze to fall outside the intended region of interest (object), without the observer being conscious about it. That may have the undesirable consequence that the system selects and deselects the observed object, making the interaction all the more difficult. To solve this problem, Jacob [Jacob 1990] proposes to detect only fixations, and to take into account only the gaze direction the moment a fixation was detected. The down side of this approach is that continuity in the eye cursor is lost, which may be confusing for the user.

In [Miniotas and Špakov 2004], a further step is proposed. After detecting a fixation within an object's area, this is selected and a time counter is initiated. Once selected, and while the time counter has not yet finished, small movements will not deselect it, even though gaze may fall outside the object. Each time that gaze falls back again on the object, the timer is reset, thus allowing for a more stable selection.

For our system, we opt to extend the approach in [Miniotas and Špakov 2004], by using a filter to stabilize the position of the cursor. Since the deviations caused by micro saccades can be described as a random walk, their mean value is zero. Thus, it make sense to measure the data to smooth the detected movement as follows: for each sample k, we obtain the weighted average with the $n-1$ previous samples:

$$p'_k = \sum_{i=1}^{min(k,n)} w_i p_{k-i+1} \tag{1}$$

We use a Gaussian as weighting function:

$$f(x) = e^{-\frac{x^2}{2\sigma^2}} \tag{2}$$

which must be normalized to preserve scale:

$$w_i = \frac{f(i)}{\sum_{i=1}^{n} f(i)} \tag{3}$$

As Figure 3, left, shows, the curve created by the weighting factors is made up of three distinct parts: acceleration, constant velocity and deceleration. Figure 3, right, shows the effect of filtering a series of positions with white noise using unit weights (arithmetic mean) and Gaussian-based weights. It can be seen how the

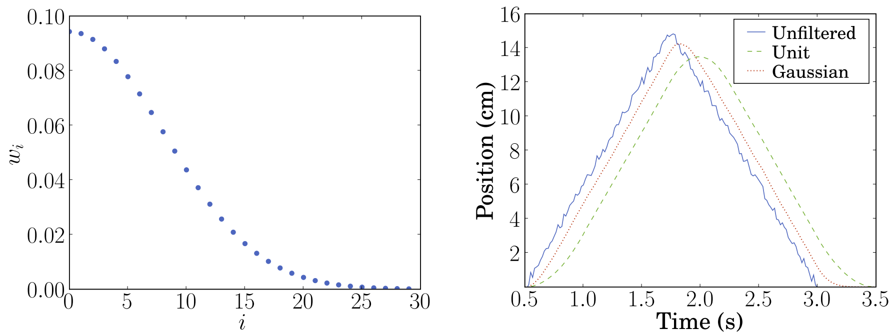

Figure 3: Left: weights used for obtaining the means ($\sigma^2 = 65$). Right: filtered positions using unit weights and Gaussian-based weights.

Gaussian transforms the positions by giving them a faster acceleration, and thus it introduces less delay between the original and the filtered positions. On the other hand, the deceleration from the Gaussian is smoother but faster that using unit weights.

By using the timer technique [Miniotas and Špakov 2004] we can stabilize the selection, thus allowing for a better interaction in those cases where the eye tracker precision does not suffice, given that it is not necessary to look *inside* an object to select it. The filtering presented in this section smoothens sudden eye movements, thus stabilizing the eye cursor. This has proved to be a simple yet powerful solution to achieve a satisfactory interaction, although more thoroughful tests may need to be performed.

5 Virtual window

Visualizing three-dimensional scenes using a monitor that can only show flat representations obviously diminishes the sensation of being manipulating a 3D world. Altering the view point with respect to the screen will not change the scene, creating the undesirable feeling of being looking at a static picture. In this section we present the use of the eye tracker as a means to enhance 3D depth using a standard monitor.

To obtain gaze direction, the eye tracker needs to previously locate the position of the eyes in 3D space. It is usually easy to query the eye tracker for those positions through the programming interface (being also the case for the eye tracker used in this work). Once obtained, we can thus position the observer with respect to the screen and make several corrections to the location of the virtual camera from which the world is rendered. Other devices could also be used for this purpose, as in [Kinoshita et al. 2006].

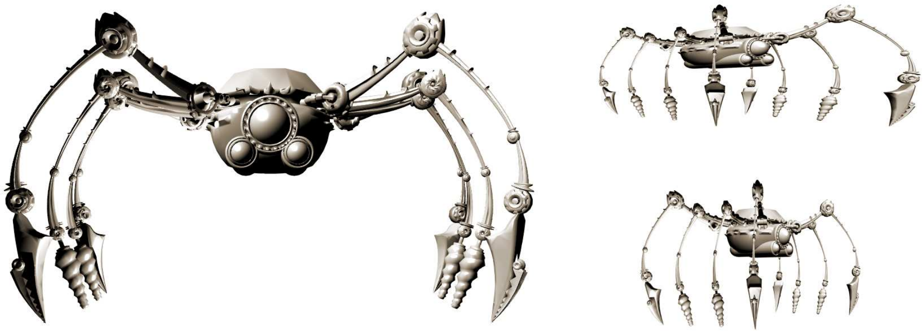

Figure 4: Left: model viewed from the front. Top right: model viewed from the right, as is displayed in the screen. Bottom right: distortion of the top right model is corrected if we look at it from the right side of the display.

5.1 Camera panning with adjusted frustum

Looking through a real window and craning sideways we see objects under a different perspective. By leveraging the fact that we know the user's position with respect to the screen, we can move the camera proportionally (or identically) to the user's movement. This creates the illusion of looking through a real window, as opposed to a flat screen (Figure 4). We have two different possibilities when moving the camera:

– To take into consideration the scale of the scene, size of the monitor and distance from the user to the screen to realistically alter the view point.

– To exaggerate or attenuate the camera's movement.

Choosing one over the other depends on factors like personal preferences or the specific goal of the application. In our case, since using the eye tracker to detect the head's position somewhat limits its range of movement, we have adjusted the factor between the real user's movement and the camera's, based on a value determined experimentally. The following simple equation will transform the input positions into the appropriate values:

$$\begin{aligned} x_s &= x * s \\ y_s &= y * s \end{aligned} \tag{4}$$

where x and y are the measured eye positions, normalized to the $-1..1$ range, and s is the desired scale factor.

Figure 5 describes the process required to achieve this effect. First, the original camera (marked in red) is translated to the correct position (marked in

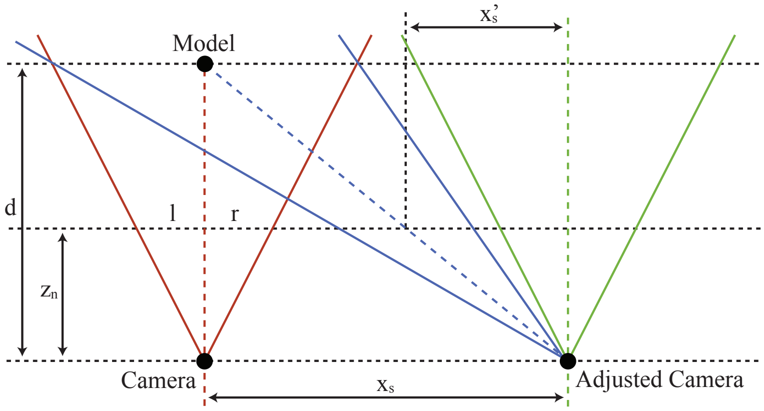

Figure 5: Top view of the required transformations to perform the camera correction. The order is red to green, then green to blue.

green), based on the position of user eyes. Then the view frustum is skewed with the purpose of keeping the focused model in the center of the projection. In order to accomplish those tasks we need to alter both the view and projection matrices.

Using a *look-at* type matrix, we can position the camera in the green position as follows:

$$\begin{aligned} eye &= (x_s, -y_s, -d) \\ at &= (x_s, -y_s, 0) \\ up &= (0, 1, 0) \end{aligned} \tag{5}$$

where x_s and y_s are the scaled eye positions and d is the distance from the camera to the focused object in the virtual world.

For the focused model to maintain its centered position after the camera translation, we need to move the frustum center by the amount pictured in Figure 5 as x'_s and the reciprocal measure y'_s. By using triangle similarity we can find its value as follows:

$$\begin{aligned} x'_s &= x_s * z_n / d \\ y'_s &= y_s * z_n / d \end{aligned} \tag{6}$$

where z_n define the distance to the near clipping plane, as depicted in Figure 5.

The skewed blue frustum can be then defined by the following equation:

$$(l', r', b', t') = (l - x'_s, r - x'_s, b + y'_s, t + y'_s) \tag{7}$$

where l, r, b and t defines the left, right, bottom and top clipping planes of the standard frustum that points forward.

Those values could be passed to the OpenGL function `glFrustum` or DirectX `D3DXMatrixPerspectiveOffCenterLH` in order to build the projection matrix.

5.2 Camera panning with fixed frustum

In this case we move the camera proportionally to the user's movement, keeping the frustum fixed at its default form. We extend the illusion of being in front of a window, and allow the user to *peek out* through it while navigating the virtual environment. Since we have the position of both eyes, we can use the head's orientation to perform a matching camera roll. In a first-person game, this amounts to substituting the keys that usually control this type of movements for a more natural interface, thus enhancing the perception of immersion (Figure 6, left). The following equations define the necessary *look-at* matrix:

$$\begin{aligned} eye &= (x_s, -y_s, -d) \\ at &= (x_s, -y_s, 0) \\ up &= (sin(\phi), cos(\phi), 0) \end{aligned} \tag{8}$$

where ϕ is the angle between the x axis and the vector formed by the position of the left and right eyes.

5.3 Camera rotation

The limitation of the first technique is that, no matter how much the factor between the user's and the camera's movements is exaggerated, this will never be rotated more than 90 degrees, given that it moves on a plane parallel to the visualization plane. By changing the panning motion for a rotation, and modulating the angle based on the position of the user, we can visualize the world under any angle. This could be very useful in a 3D modeling package, since it would free the mouse from this task so that it could be used for modeling only. The rotating angle (radians) is given by:

$$\begin{aligned} \psi &= x * \pi \\ \theta &= y * \pi \end{aligned} \tag{9}$$

Notice how in this case it is more convenient to use rotation matrices instead of a *look-at* matrix, concatenating multiplications as follows:

$$M_{view} = M_\psi * M_\theta * M_t \tag{10}$$

where M_ψ denotes a rotation around the y axis of ψ radians, M_θ denotes a rotation around the x axis of θ radians and M_t is the translation matrix for the camera that places it at the proper distance.

6 Analysis

Six employees from the Usability Lab offered to take part in the tests. All of them were familiar with the eye tracker, and had used it previously in interface evaluation tasks by running the software provided by the vendor. However, knowledge of this software is independent of the system designed, and thus cannot be accounted for as previous knowledge specific to the tests. For calibration, they were asked to stay at approximately one meter from the eye tracker; five reference points were used. One subject was discarded due to calibration problems. The rest (four men and one woman) completed the tests satisfactorily.

6.1 Methodology

Participants had unlimited time to train in the use of the tools in both interface devices (mouse and eye tracker). The virtual window capabilities of the program were disabled during the tests; we believe that a more appropriate way of testing them would be within a real-world application (like a video game) which falls out of the scope of this paper. They were advised in how to undertake the proposed tasks. After training, each participant had a two-hour rest period before the tests started.

In each test, each participant completes three times each task, done on a special table with three numbered zones (Figure 6, right). The zones were designed to force the participants to navigate throughout the virtual environment to achieve the tasks. They are sized differently as well, to be able to measure actions which require different precision levels. They are placed at the corners so that the participant must try not to throw the dice outside the table. At the beginning of each task, the camera is placed in a default position, and 25 dice fall over the table. The tasks consist on:

- Grab: the participant must move 6 dice to the three zones, as many dice as the number signaling each zone (1,2,3).
- Push: the participant must push three dice to the red zone.
- Shoot: the participant must shoot all 25 dice out of the table.

6.2 Results and conclusions

We divide the results of our study in two categories:

- Objective measures, which allow us to compare the efficiency and training process between the mouse and the eye tracker interfaces (Figure 7).

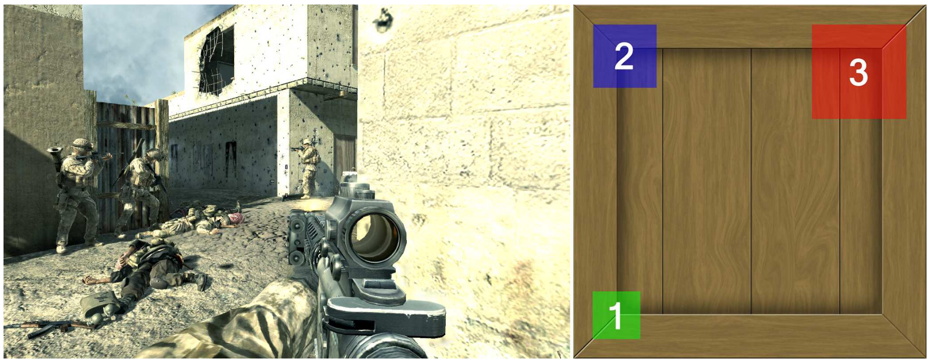

Figure 6: Left: screenshot of Call of Duty 4 that exemplify the possible usage of the eye tracked camera panning in a video game. Right: numbered table for the tasks.

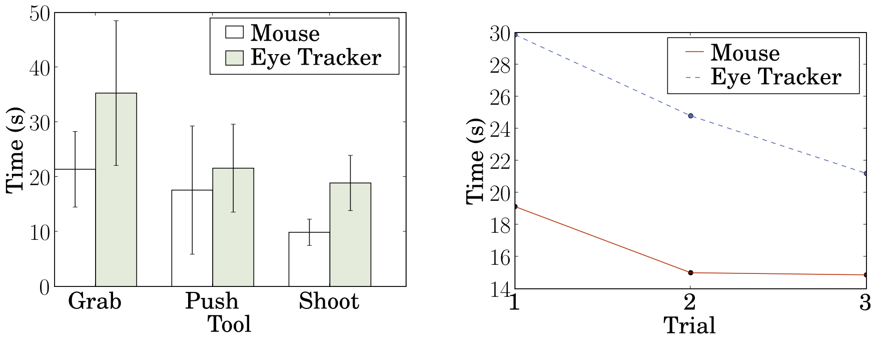

Figure 7: Results of the objective analysis

Table 1: Results of the subjective analysis per tool. ET (*eye tracker*). M (mouse)

	Grab (%)		Push (%)		Shoot (%)	
Question	ET	M	ET	M	ET	M
Which method did you enjoy the most?	100	0	40	60	100	0
Which method seemed easier to learn?	20	80	0	100	20	80
Which method seemed easier to use?	60	40	20	80	60	40
Which method better enhances immersion?	100	0	100	0	80	20

Table 2: Results of the subjective general analysis. ET (*eye tracker*). M (mouse)

Question	ET (%)	M (%)
Did you feel fatigue using it?	80	20
Do you think it is precise enough?	40	80
Do you think it would be useful in VR environments?	100	100

– Subjective measures, which inform us of the users' satisfaction for both interfaces for each tool (Table 1) and overall (Table 2).

The objective results suggest that eye tracking technology is still not precise enough to allow a user to select distant objects: the user is forced to track the virtual camera to get closer to it, negatively affecting the performance of gaze-based interaction. In fact, for some users the error could be as large as one centimeter, hampering selection even at close distances. As Figure 7, left, shows, the grab and push tools have significantly bigger standard deviation than the shot tool. In the case of the first tool, the reasons seems to be related with the intensive viewport manipulations required by the test –which were performed differently by the users–, and in the latter tool, with the fact that it was the first time the users cope with this way of interaction, which lead to very different adaptations to the tool.

On the other hand, training is sensibly better performed using the eye tracker, taking into account the number of tries versus execution time. This can be due to the fact that the participants were accustomed to using the mouse in their every day work, which somehow counts as previous training. This is obviously not the case for eye trackers.

Subjectively, the eye tracker interface seems to be more enjoyable than its mouse counterpart. As for the sense of immersion, the eye tracker performed significantly better, thus confirming the results presented by Smith and Graham [Smith and Graham 2006]. The push tool received the most negative evaluation, mainly because it is tricky to perform correctly, requiring a tough learning before it can be used properly. However, once learned, it is the tool that performed best when used with the eye tracker in comparation with its performance when using the mouse. This can be explained by taking into account that it is the only tool that is not usually required by common operating system actions, and hence there is no previous training in its usage.

We believe that the use of gaze-based interfaces in physically-based applications could help handicapped people interact with objects in virtual environments. Possible applications range from entertainment to even rehabilitation.

As future work, precision seems to be the biggest issue. This could be tackled from two different angles: either improving tracking hardware, or developing smart algorithms capable of deciding what the user is really looking at.

Acknowledgements

This work has ben funded by projects UZ2007-TEC06 (Universidad de Zaragoza) and TIN2007-63025 (Ministerio de Educación y Ciencia). Jorge Jimenez has been funded by a research grant from the Instituto de Investigación en Ingeniería de Aragón. The authors would like to thank the Laboratorio de Usabilidad and the participants who took part in the tests, specially Elena Lafuente and David Eguizábal. We thank John Michael Guerrero for providing the texture used in the virtual table.

References

[Cournia et al. 2003] Cournia, N., Smith, J. D., Duchowski, A. T: "Gaze vs. hand based pointing in virtual environments"; CHI Extended Abstracts, ACM, Ft. Lauderdale (2003), 772-773.

[Engbert and Kliegl 2004] Engbert, R., Kliegl, R: "Microsaccades keep the eyes balance during fixation"; Psychological Science, 15, 6 (2004), 431-436.

[Isokoski and Martin 2006] Isokoski, P., Martin, B: "Eye tracker input in first person shooter games"; Proceedings of COGAIN 2006: Gazing into the Future, Communication by Gaze Interaction (2006), 76-79.

[Jacob 1990] Jacob, R. J. K: "What you look at is what you get: Eye movement based interaction techniques"; Proceedings of the SIGCHI Conference on Human Factors in Computing Systems: Empowering people, ACM, Seattle (1990), 11-18.

[Kinoshita et al. 2006] Kinoshita, K., Ma, Y., Lao, S., Kawade, M: "A Fast and Robust 3D Head Pose and Gaze Estimation System"; Proceedings of the 8th international conference on Multimodal interfaces, ACM, Banff (2006), 137-138.

[Miniotas and Špakov 2004] Miniotas, D., Špakov, O: "An algorithm to counteract eye jitter in gaze-controlled interfaces"; Information Technology and Control, Kaunas (2004), 64-67.

[Salvucei and Anderson 2000] Salvucei, D. D., Anderson, J. R: "Intelligent gaze-added interfaces"; Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, ACM, The Hague (2000), 273-280.

[Sibert and Jacob 2000a] Sibert, L. E., Jacob, R. J. K: "Evaluation of eye gaze interaction"; Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, ACM, The Hague (2000), 281-288.

[Smith and Graham 2006] Smith, J. D., Graham, T. C: "Use of eye movements for video game control"; Advances in Computer Entertainment Technology, ACM, Hollywood (2006).

[SPE] Simple Physics Engine, `http://www.spehome.com`.

[Tanriverdi and Jacob 2000b] Tanriverdi, V., Jacob, R. J. K: "Interacting with eye movements in virtual environments"; Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, ACM, The Hague (2000), 265-272.

[Ware and Mikaelian 1987] Ware, C., Mikaelian, H. H: "An evaluation of an eye tracker as a device for computer input"; Proceedings of the SIGCHI Conference on Human Factors in Computing Systems and Graphics Interfaces, ACM, Toronto (1987), 183-188.

[Yeh et al. 2006] Yeh, T. Y., Faloutsos, P., Reinman, G: "Enabling real-time physics simulation in future interactive entertainment"; Proceedings of the 2006 ACM SIGGRAPH symposium on Videogames, ACM, Boston (2006), 71-81.