Depth-of-Field Blur Effects for First-Person Navigation in Virtual Environments

Sébastien Hillaire ▪ *INRIA/France Télécom R&D*

Anatole Lécuyer ▪ *INRIA/Collège de France*

Rémi Cozot ▪ *University of Rennes 1/INRIA*

Géry Casiez ▪ *University of Lille 1/INRIA*

Depth-of-field blur effects are well-known depth cues in human vision. Computer graphics pipelines added DOF effects early to enhance imagery realism, but real-time VR applications haven't yet introduced visual blur effects. The authors describe new techniques to improve blur rendering and report experimental results from a prototype video game implementation.

In human vision, the depth of field (DoF) is the range of distances near the focus point where the eyes perceive the image as sharp. Objects behind and in front of the focus point are blurred. DoF and its associated blur effects are well-known classic depth cues in human vision.[1] Virtual images that lack a DoF blur can sometimes look "too perfect" and therefore synthetic. System designers therefore added DoF blur effects early to computer graphics pipelines.[2] Movies also use the classic blur effects of focal-distance changes to convey sensations or otherwise capture viewers' attention.

Real-time VR applications haven't yet introduced visual blur effects. We now have the programming capabilities and the processing power to compute them in real time. However, we don't know how such effects will influence user performance and subjective experience. We therefore need to

- develop new models of realistic visual blur effects for virtual environments (VE), taking interactivity and real-time constraints into account, and
- evaluate visual blur effects in terms of both VE user performance and subjective preferences.

Here, we describe a novel model of dynamic visual blur for first-person VE navigation. We also report results from an experiment to study how the blur effect influenced gamers' performance during a multiplayer first-person-shooter (FPS) session.

Visual Blur Effects for First-Person VE Navigation

We use a model for dynamic visual blur that combines DoF and peripheral blur effects. For landmark research and state-of-the-art implementations relative to these topics, see the sidebar, "Development and Related Work in Visual Blur Effects" (next page).

The DoF Blur Effect

This effect simulates visual blurring by blurring the pixels of objects in front of or behind the focus point. The focus point is associated with a focal distance (f_d)—the distance between the eyes (or camera) and the focus point.

The lens model. We use the classic lens model introduced by Michael Potmesil and Indranil Chakravarty.[2] In this model, the amount of blur—that is, the diameter of the circle of confusion ($DCoC_{dep}$) of a point projected on screen—is

Development and Related Work in Visual Blur Effects

Computer graphics researchers introduced visual blur simulation early to improve the photorealistic aspect of synthetic images. Michael Potmesil and Indranil Chakravarty first proposed simulating an optical lens to simulate depth-of-field (DoF) blur. Their algorithm uses the original sharp image, each pixel's depth, and a postprocessing step to compute the blur. The lens simulation provides the amount of each pixel's blur according to its depth. With lens simulation, an out-of-focus point becomes a disk or circle after the projection through the lens. The diameter of the resulting circle of confusion (CoC) corresponds to the amount of blur.[1]

After Potmesil and Chakravarty's pioneering study, most researchers used this lens model to compute the DoF blur.[2] However, Brian Barsky introduced the alternative concept of vision-realistic rendering, which uses all of an individual's optical-system characteristics.[3] This let Barsky accurately simulate the foveal image scanned from wavefront data of human subjects as measured by an aberrometry device.

The main problem of DoF blur algorithms is *color leaking* across depth discontinuities. This artifact blurs edges of in-focus objects that are in front of a blurred background. DoF algorithms compute the blur itself and avoid color leaking in different ways. In a survey of DoF algorithms, Joe Demers divides the different techniques into three main categories: scattering, gathering, and diffusion.[2] Gathering techniques (also called reverse-mapping techniques) use only the sharp image's pixels. For each of the final image's pixels, the algorithm gathers and blends source-image pixel colors that belong to the current pixel's CoC. Simple depth tests during the gathering step avoid color-leaking artifacts. This approach is easily implemented on current graphics hardware.[4]

Other blur effects can further enhance digital images' appearance. For instance, *peripheral blur* refers to the eye's coarser acuity from the fovea to the periphery.[5] Nelson Max and Douglass Lerner define a motion blur that simulates the images obtained from a digital camera.[6] This blur corresponds to the recording of objects that move rapidly. Indeed, integrating such images while the shutter is open generates a blur.

Julian Brooker and Paul Sharkey investigated the DoF blur effect using a stereoscopic display and an eye-tracking system to find a path in a 3D labyrinth.[7] However, their results show no evidence of performance improvement. They concluded that their application's very slow frame rate might have been the cause and suggested implementing and further evaluating real-time DoF blur effects in virtual environments.

Przemyslaw Rokita first suggested using visual blur effects in VR.[8] In the latest generation of video games, Epic Games' Unreal Engine 3 (www.epicgames.com) and Crytek's CryEngine 2 (www.crytek.com) propose temporary DoF blur together with motion blur. Techland's Chrome Engine (www.development.techland.pl) also introduces a dynamic DoF blur effect, but it remains limited to a "sniper mode" with only a few depth plans. All of these DoF blur effects suffer from leaking artifacts.

References

1. M. Potmesil and I. Chakravarty, "A Lens and Aperture Camera Model for Synthetic Image Generation," *Proc. Siggraph*, ACM Press, 1981, pp. 298–306.
2. J. Demers, "Depth of Field: A Survey of Techniques," *GPU Gems*, R. Fernando, ed., Addison-Wesley, 2004, pp. 375–390.
3. B.A. Barsky, "Vision-Realistic Rendering: Simulation of the Scanned Foveal Image from Wavefront Data of Human Subjects," *Proc. Symp. Applied Perception in Graphics and Visualization*, ACM Press, 2004, pp. 73–81.
4. G. Riguer, N. Tatarchuk, and J. Isidoro, "Real-Time Depth of Field Simulation," *ShaderX2: Shader Programming Tips and Tricks with DirectX 9*, Wolfgang Engel, eds., Wordware, 2003, pp. 529–556.
5. M. Sereno et al., "Borders of Multiple Visual Areas in Humans Revealed by Functional Magnetic Resonance," *Science*, vol. 268, no. 5212, 1995, pp. 889–893.
6. N. Max and D. Lerner, "A Two-and-a-Half-D Motion-Blur Algorithm," *Proc. Siggraph*, ACM Press, 1985, pp. 85–93.
7. J.P. Brooker and P.M. Sharkey, "Operator Performance Evaluation of Controlled Depth of Field in a Stereographically Displayed Virtual Environment," *Stereoscopic Displays and Virtual Reality Systems VIII*, Proc. SPIE, vol. 4297, 2001, pp. 408–417.
8. P. Rokita, "Generating Depth-of-Field Effects in Virtual Reality Applications," *IEEE Computer Graphics & Applications*, vol. 16, no. 2, 1996, pp. 18–21.

$$DCoC_{\mathrm{dof}} = \left| \frac{D \times f \times (f_d - z)}{f_d \times (z - f)} \right| \quad (1)$$

where D is the lens diameter, f is the lens focal length, f_d is the focal distance, and z is the point depth.

The autofocus zone. Eye-tracking systems offer an optimal way to determine the focal distance in real time.[3] However, such devices are expensive, complex, and unavailable for a mass market. In the absence of such a system, we chose a paradigm used in FPS games, where users employ a 2D mouse and keyboard to manipulate a virtual visor always located at the screen's center. In this approach, we can assume the user looks mainly at the part of the screen close to the visor. In fact, using an eye-tracking system, Alan Kenny and his colleagues found that more than 82 percent of the time, FPS video gamers indeed watched a central area corresponding to half the monitor's size.

We therefore introduce a notion called the *autofocus zone*—an area at a screen's center that the user is supposed to look at preferentially. This recalls digital-camera autofocus systems, which also aim to determine an appropriate focal distance when taking a picture. We can compute the depth of autofocus-zone pixels by using an auxiliary buffer. As in digital cameras, the function to compute the focal distance from the depths of all pixels in the autofocus zone could be minimum, maximum, or average. In FPS games, some objects in the environment are more important—for example, enemies or bonus objects. We therefore use a semantic weighting of pixel depths. The semantic weighting increases the pixel weights corresponding to objects (or targets) known to be important in the scene. We do so by adding a field to the initial virtual-object description that corresponds to the object's visual semantic weight. Each pixel's semantic weight ranges from $WS_{\min}$ to $WS_{\max}$.

Figure 1 illustrates the use of semantic weighting. In this example, the weight of the character in front is much higher than that of the background. Even if the character covers fewer pixels than the background (less than one-quarter of the area), the focus is systematically on the front character. In the work we describe here, the semantic weights remain constant. However, other applications could use dynamic weighting. For instance, we could simulate the habituation phenomenon by progressively decreasing each object's semantic value on the basis of how long the object remains on the screen.

In addition, we introduce a spatial weighting that slightly modifies the central pixels' weight. We use a Gaussian function that gives a weight of $WG_{\max}$ to the center and $WG_{\min}$ to the zone's borders. Finally, we compute the resulting focal distance f_d:

$$W(p) = WS(p) \times WG\big(d2AC(p)\big)$$
$$WD(p) = W(p) \times \text{depth}(p)$$
$$f_d = \frac{\sum_{p\in\text{zone}} WD(p)}{\sum_{p\in\text{zone}} W(p)} \qquad (2)$$

where $WS(p)$ is the semantic weight of pixel p, $WG(x)$ is the Gaussian spatial weight for distance x, and $d2AC(p)$ is the distance of p from the autofocus zone's center.

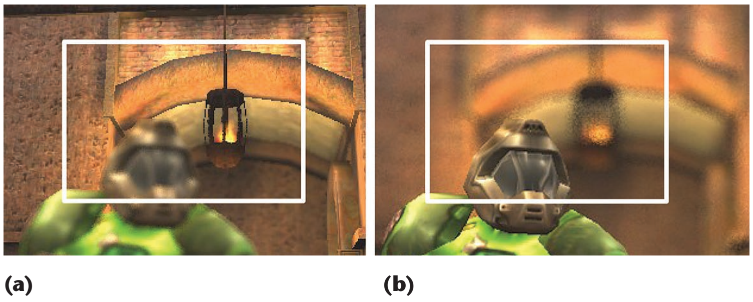
Figure 1. The depth-of-field blur when using a rectangular autofocus zone (the white rectangle): (a) Without semantic weighting, the focus is on the background. (b) With semantic weighting, the focus is automatically set on the important character. (Quake III Arena screenshot, courtesy of IdSoftware)

When we implemented the final blur model to study its influence on gamers' performance, we set $WG_{\min}$ to 0.7, $WG_{\max}$ to 1, $WS_{\min}$ to 0.004, and $WS_{\max}$ to 1.

GPU computation of focal distance. Computing Equation 2 on a CPU takes a long time, especially for a large autofocus zone. We therefore compute the focal distance using a general-purpose technique on a GPU (GPGPU).

To evaluate f_d using Equation 2, we must calculate computationally expensive sums, which can produce number overflow. So we replace the sum operation by the mean operation using equation 3:

$$\sum_{p\in\text{zone}} WD(p) = \overline{WD(p)} \times \text{card}(\text{zone})$$
$$\sum_{p\in\text{zone}} W(p) = \overline{W(p)} \times \text{card}(\text{zone}) \qquad (3)$$

First, we store the values of $WD(p)$ and $W(p)$ in a 2D texture that's the size of the autofocus zone. Then, we compute this texture's mean by recursively downsampling the texture by a factor of two until we reach the size of one texel, which finally contains the means $\overline{WD(p)}$ and $\overline{W(p)}$. In this algorithm, at each step and for each pixel, a fragment program computes the mean of the texture's four corresponding texels computed at the preceding step. Using this technique, we can finally accelerate the focal distance f_d computation using a GPU instead of a CPU.

Simulation of accommodation. Human eyes take a few milliseconds to accommodate a change in focus point. We simulate this accommodation phenomenon in our DoF blur effect by adding a temporal filter to the final focal-distance computation. After preliminary testing, we chose a low-pass filtering:

$$\bar{f}_d(n) = \left(f_d(n) + \frac{\tau}{Te}\bar{f}_d(n-1)\right)\frac{1}{1+\frac{\tau}{Te}} \qquad (4)$$

where τ equals $(\pi \times fc)/2$, fc is the cut-off frequency in Hertz, Te is the sampling period in seconds, $\bar{f}_d(n)$ is the filtered focal distance at frame n, and $f_d(n)$

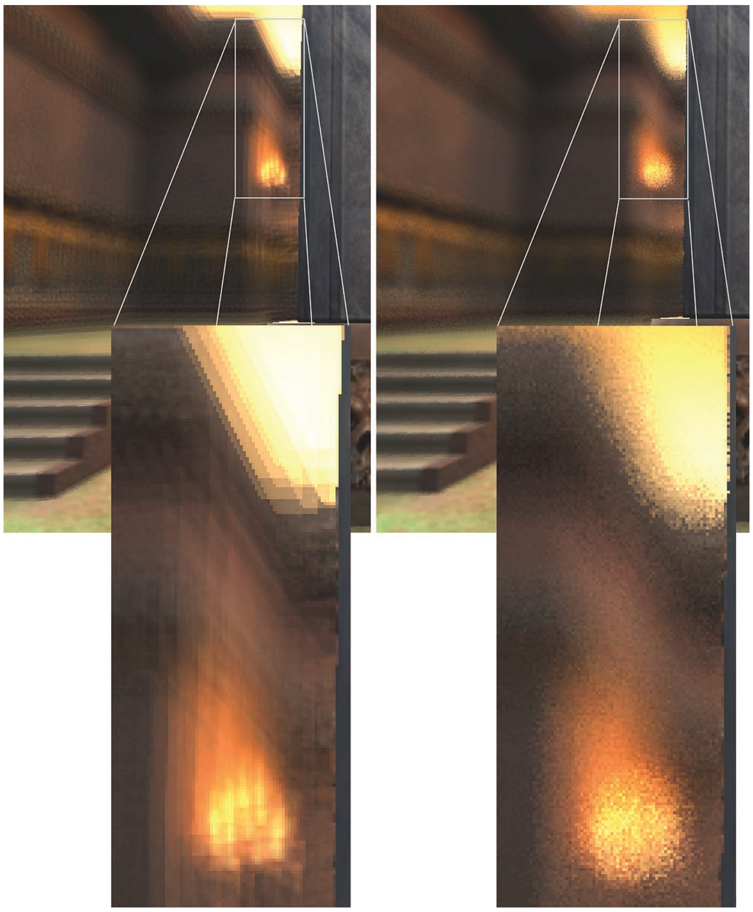

Figure 2. Canceling ghosting artifacts. On the left, the blur computation has generated a ghosting artifact. On the right, the ghosting artifact is replaced by high-frequency noise using per-pixel random rotation of the sampling kernel. (Quake III Arena screenshot, courtesy of IdSoftware)

is the focal distance given by the autofocus system (before filtering).

In our final implementation, we used $Te = 1/70$ s and $fc = 5$ Hz.

The Peripheral Blur Effect

We simulate the peripheral blur by decreasing an image's sharpness, when it's at the image scene's periphery. The effect progressively blurs the pixels located at a certain distance from the focus area's center.

The peripheral blur is independent of the DoF blur. We added it as a supplementary visual effect, expecting that it might enhance the user's sensory experience. However, we also expected it to encourage users to look at the screen's center. Indeed, by slightly blurring an image's contour, we hoped to force the user to look through the visor—that is, inside the focus area. Equation 5 computes the amount of peripheral blur for each pixel:

$$DCoC_{\text{per}} = \left(\sqrt{\frac{1}{\mathbf{z} \times \mathbf{p}} - 1} \right)^{n} \quad (5)$$

where $\mathbf{z}$ is the look-at direction (for example, the camera's direction when the focus area is at the image's center) and $\mathbf{p}$ is the normalized direction of the pixel in the camera frame.

In our final implementation, we used a power n equal to 2.

The Final Blurred Image

After we compute both the peripheral and DoF blur amount, we can compute the final blurred image.

Computing the final blur amount. The total amount of blur for each pixel corresponds to the final diameter of its circle of confusion ($DCoC_{\text{fin}}$). Equation 6 shows the contributions of both the peripheral and DoF blurs:

$$DCoC_{\text{fin}} = D_{\max} \times \min(1, DCoC_{\text{dep}} + DCoC_{\text{per}}) \quad (6)$$

where $D_{\max}$ is the maximum amount of blur (the maximum diameter of the pixel's final CoC), $DCoC_{\text{dep}}$ is the normalized diameter of the DoF blur's CoC, and $DCoC_{\text{per}}$ is the normalized diameter of the peripheral blur's CoC. In our implementation, we set $D_{\max}$ to 11 pixels.

Rotating the blur sampling kernel. Our blur algorithm is based on a gathering blur technique.[4] This technique mixes the colors of 12 samples, which form the *sampling kernel*. To compute the blur, the color samples come from inside the CoC according to a Poisson-disk distribution. In this case, ghosting artifacts appear—that is, objects seem to duplicate (see Figure 2). To improve the blur rendering, we randomly rotate the sampling kernel per pixel instead of increasing the number of samples (and thus decreasing performance).

In the case of soft-edged shadow mapping, Yuri Uralsky proposed generating blur by using a set of different sampling kernels stored in a 3D texture.[5] In our case, instead of having many different kernels, we propose to always use the same kernel and to randomly rotate it for each pixel. As a result of this rotation, we replace the ghosting artifact with a high-frequency noise (see Figure 2), a visual cue that human eyes filter out efficiently.[5] In this way, the blur computation becomes also faster and uses less memory.

For each pixel, we just need information about the 2D rotation of angle α (see the algorithm in Figure 3). To construct a 2D rotation matrix, we use $\cos(\alpha)$ and $\sin(\alpha)$ values, which we precompute and store in a single low-resolution 2D texture repeated on the whole screen. Finally, we simply multiply each sample offset defined in the fragment program by this matrix. This method saves a lot of texture memory bandwidth because we need to read only one texel per pixel. Earlier methods required several readings in the 3D texture.[5]

Finally, we eliminate the color-leaking artifact by simple depth comparisons.[4] Figure 4 (next page) shows results from our successful implementation of the final blur algorithm in the open source engine of the Quake III Arena video game (www.idsoftware.com).

Implementation

Figure 5 (next page) shows the final software architecture for computing visual blur effects. From the graphic hardware, we get three raw components: the semantic weights, the depths, and the sharp image. Then, the autofocus algorithm uses the semantic weights, the depths, and the computed spatial weights as input to determine a focal distance. A low-pass filter then filters the focal distance (accommodation phenomenon). The DoF blur algorithm uses the lens model to compute the amount of DoF blur according to the filtered focal distance and the depths. At the same time, the peripheral blur algorithm first computes the distances between each pixel and the autofocus zone's center. Then it computes the amount of peripheral blur using Equation 5. The total amount of blur is computed using the amount of both the DoF and peripheral blurs. Finally, our algorithm computes the blurred image by applying the total amount of blur to the sharp image.

For application purposes, we implemented our blur effects in Quake III Arena's real-time 3D engine. Our code is open and available at www.irisa.fr/bunraku/eye. We used a desktop PC with a 2.8 GHz Intel PentiumD CPU, 1.0 Gbytes of RAM, and an ATI 1900 Series card with 512 Mbytes of video memory.

Table 1 (next page) shows the performance (frame rate) of our video game application under different conditions: with and without the blur effect, and with and without accelerated GPU computation for several screen resolutions and two autofocus zone ratios.

```
input : Texture texSharp contains the sharp image
input : Texture texCoCD contains pixels depth and circle of confusion size
input : Texture texRot contains per pixel rotation parameters
input : Current texel coordinates texCoord
input : Current texel coordinates texCoordRot of the rotation texture
input : Samples offsets samplesOffsets [12]
input : Maximum circle of confusion diameter cocMaxD
input : Pixel final color Out
texRd (t,c) : read the texture t at the coordinate c

//initialization
float pixelCoc = texRd (texCoCD, texCoord).r;
float pixelDepth = texRd (texCoCD, texCoord).g;
vec3 colorSum = texRd (texSharp, texCoord);
float cocSize = cocMaxD × pixelCoc;
float totalContrib = 1.0;

//creation of the rotation matrix from cos(α) and sin(α)
rMat = createRMat (texRd (texRot, texCoordRot));

for i=0 to 12 do
    //rotation of the current offset
    vec2 offset = samplesOffsets [i] · rMat;
    //texture coordinates of the current sample
    vec2 splCoord = texCoord + cocSize × offset;

    //sampling
    vec3 splColor = texRd (texSharp, splCoord);
    float splCoc = texRd (texCoCD, splCoord).r;
    float splDepth = texRd (texCoCD, splCoord).g;

    //avoid color leaking artifact using depth comparison
    float sampleContrib = splCoc;
    if (splDepth > pixelDepth) then
        sampleContrib = 1.0;
    end

    //sum of all contributions
    colorSum += splCoc × sampleContrib;
    totalContrib += sampleContrib;
end
//output final color
Out = colorSum ÷ totalContrib;
```

Figure 3. Pseudocode of the fragment program that computes peripheral and DoF blur. The algorithm randomly rotates the blur sampling kernel per pixel using a precomputed texture to store rotation information.

Focus-Point Measurement and Analysis

We conducted a preliminary experiment to measure the focus point of participants during first-person navigation in a VE. Each of six participants faced various first-person navigation situations. The experiment's main objective was to analyze the distance between the user's focus point and the screen's center so that we could better set the autofocus zone's size.

Procedure

We recorded the focus point of six males with a mean age of 24.2 (standard deviation = 2.8) during several FPS game sessions using an eye-tracking system. All participants were healthy and had normal or corrected vision. We used the ASL 6000 eye-tracker system with head-mounted optics and a chin rest to maintain the participants' heads at the same position. We used a 5:4 monitor with a resolution of 1,280 × 1,024, positioned 50 cm in front of the participant.

The experiment consisted of playing Quake III Arena on the game's Q3DM7 map with no blur

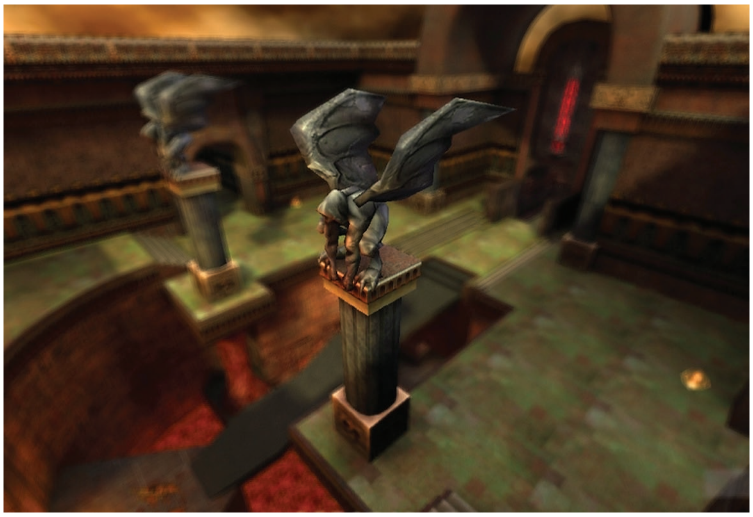

Figure 4. A Quake III Arena frame with blur effects. The frame shows the final effect of the algorithm implementation. (Quake III Arena screenshot, courtesy of IdSoftware)

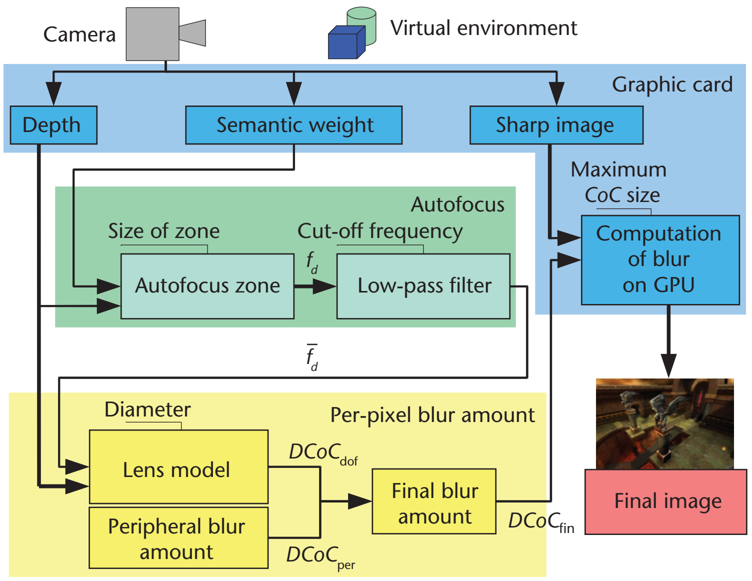

Figure 5. The software architecture for the blur effects system. The system computes the final image's blur from a camera image in a virtual environment.

Table 1. Frame rate of video game application with and without blur effects for various configurations.

	Frame rate (Hz)				
		With blur			
		Zone ratio: 0.25		**Zone ratio: 0.5**	
Frame resolution (pixels)	**Without blur**	**CPU**	**GPU**	**CPU**	**GPU**
800 × 600	313	80.0	97.5	55.3	91.5
1,024 × 768	308	65.2	81.7	50.1	78.8
1,280 × 1,024	305	48.8	62.2	35.5	61.3

effects. Four successive sessions recorded four different conditions:

- *Navigation*. Each participant navigates freely and alone in the map (no enemies) for 3 minutes.
- *1Enemy*. Each participant fights against one enemy for 5 minutes.
- *4Enemies*. Each participant fights against four enemies for 5 minutes.
- *4EnemiesBonus*. Each participant fights against four enemies for 5 minutes and can pick up a life bonus.

At each frame, and during each session, we recorded the 2D position of the participant's focus point on screen and whether he or she shoots (a Boolean value).

Focus-Point Experimental Results

Figure 6 illustrates the percentage of time spent looking inside a rectangular zone centered on the screen as a function of the zone's size—that is, the zone/screen-size ratio. The zone/screen-size ratio corresponds to the ratio between the focus zone's height and the height of the full image on the screen. This ratio is equal to 1 if the zone covers the entire image and 0 if the zone is equal to one pixel. The ratio is 0.5 if the focus zone's area corresponds to one-quarter of the image's area.

As expected, all participants looked close to the screen's center (the mean curve). When firing (the mean-shooting curve), the participant's focus point naturally comes even closer to the FPS's central visor. As a result, when the task involves more enemies and thus more shooting (the 4Enemies curve), the participants are looking much closer to the center. In other words, the more enemies on the map, the more participants focus on the screen's center. For the 1Enemy session, participants probably alternated between navigation and fighting, so the 1Enemy curve is logically a mix between the Navigation curve and the 4Enemies curve.

On average, when the zone-screen ratio is 0.5 (that is, the zone's height corresponds to half the screen), the participants look inside the zone 93 percent of the time. This result is consistent with previous findings of Kenny and his colleagues, who obtained a score of 82 percent.[6] The difference could be the result of more visual information displayed at the periphery of the screen during Kenny's experiment.

Developers can use the curves in Figure 6 to set the autofocus zone's size. For instance, to ensure that the autofocus zone will capture 75 percent of the user's gaze when fighting against four FPS enemies, the zone-screen ratio must be 0.25. This translates to a rectangle of 320 × 256 pixels.

Blur-Effects Evaluation

We conducted a second experiment to study how our visual blur affects the performance and subjective preference of FPS gamers. The DoF blur might annoy gamers because it blurs the displayed image to some extent. Alternatively, the effect might make the VE look more realistic, thus increasing users' feelings of immersion and, as a result, their enjoyment of the navigation and game experience.

Experimental Apparatus

We implemented peripheral and DoF blurs with automatic focal-distance computation and accommodation simulation in Quake III Arena's 3D engine. We activated both the semantic and spatial weights, setting the enemies' semantic weights to remain constant at WS_{max} and all other semantic weights to remain constant at WS_{min}. (The other numerical values we used appear in earlier sections.)

We used the Q3DM7 Quake III Arena map but removed all life packs, special bonuses, and weapons. We connected six PCs on a local network with identical graphic cards, monitors, and resolutions. There was no perceivable lag. All participants had an infrared mouse and a stereo headset for audio feedback. For the experiment's purpose, the video cards' wait-for-vertical-synchronization feature remained constant at 60 frames per second.

We set the zone-screen ratio at 0.25—that is, the focus zone's area covered one-eighth of the full image. On the basis of the previous experiment's results (the 4Enemies curve in Figure 6), this value ensured that participants look at least 75 percent of the time in the autofocus zone.

Procedure

The task consisted of playing an FPS game in death-match mode (each player fights against all the other players). We instructed participants to be as precise as possible when shooting while using as little ammunition as possible. To reduce the variability across subjects, all players had only one weapon (a submachine gun) with unlimited ammunition. We increased the amount of life to 200 points (from the usual 100 points) to make the fights last longer.

Participants included 28 males and two females with a mean age of 24.0 (standard deviation = 2.2). Of the 30 participants, 30 percent assessed themselves as expert gamers, 37 percent as intermediate, and 33 percent as beginners.

We used a repeated-measures-within-subjects design. The independent variable was the visual effect (Veffect) with two levels: no blur effect and blur effects. The experiment lasted 1 hour including breaks. We divided participants randomly into five groups of six subjects each. Then we divided the experiment into two parts. For each part of the experiment, we counterbalanced the presentation order among the participants and between the groups.

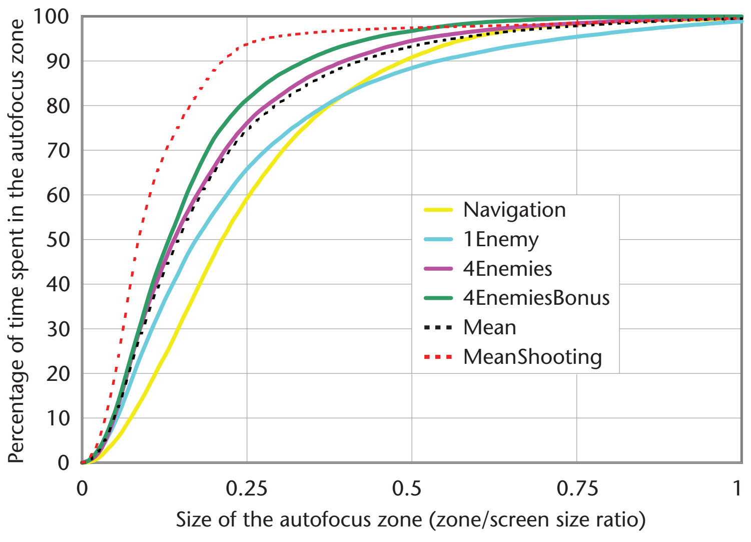

Figure 6. First-person shooter sessions. The curves show the percentage of time spent looking inside a centered zone as a function of the zone's size for four game situations.

The first part consisted of a 4-minute training session with or without blur, followed by the real experiment with a first session of 5 minutes in the same conditions as the training session and a second 5-minute session with the reverse condition. Participants filled out a subjective questionnaire after these two sessions.

The second part consisted of a performance test including six sessions of 5 minutes each. For each session, three participants played with blur and the other three played without blur. The blur condition was automatically swapped for all players when running a new session. At the end, participants filled out a general-appreciation questionnaire and a personal form to indicate their preferences and skill level.

Blur-Effects Experimental Results

Table 2 displays participants' performance with and without the blur effects. The dependent variables were the number of enemies that the player killed (Frags), the number of times the player was killed (Deaths), the number of the player's shots (Total Shots), and the player's precision (Precision). Precision is a percentage value computed as the ratio of the number of successful shots to the total number of shots.

Performance. Repeated-measures analysis of variance (Anova) showed that the presentation order of the visual blur effects and groups had no significant effect on or interaction with the dependent variables, indicating that a within-subjects design was appropriate. Repeated-measures Anova found a significant main effect for Veffect on Frags ($F_{1,29}$ = 8.1, p = 0.008), Deaths ($F_{1,29}$ = 5.7, p = 0.023), and Precision ($F_{1,29}$ = 17.3, p < 0.0001). These results show that

Table 2. Mean and standard deviation (SD) for each dependent variable of the performance test, with and without the blur effects.

	No blur effect		With blur effect	
	Mean	SD	Mean	SD
Frags	15.4	6.6	14	6.6
Deaths	14.4	3.6	15.2	3.4
Total Shots	1,011.9	271.4	983.4	262.4
Precision (%)	27.1	6.4	25.2	6.2

Precision decreases significantly from 27.1 to 25.2 percent with blur effects while Frags decreases from 15.4 to 14 and Deaths increases from 14.4 to 15.2.

Interestingly, when we treated the game-experience level reported by each participant as a between-subject factor and the Veffect as a within-subject factor, Anova showed a significant interaction between the experience level and Veffect ($F_{2,27} = 4.02$, $p = 0.03$) on Precision. This shows that Precision decreases for expert gamers when visual effects are enabled (from 31.2 to 28.7 percent) and for intermediate gamers (from 26.1 to 23.3 percent), whereas the precision for beginners remains around 24 percent.

Questionnaire and user feedback. After the two first sessions (with and without blur effects), 21 of the participants noticed a difference between the sessions, and 20 of them could explain that a blur effect had been applied to the rendering. The final general-appreciation questionnaire showed no significant trend concerning a potential appreciation or dislike of the blur effects. Indeed, participant opinions were balanced concerning increased realism (11 preferred playing with the blur, 13 preferred playing without it, and six had no preference), gameplay fun (9 with, 10 without, 11 no preference), perception of depth and distances in the VE (10 with, 14 without, 6 no preference), and feeling of presence (11 with, 10 without, 9 no preference).

The participants who preferred the game with blur effects could be very enthusiastic: “an improvement in realism, especially during volte-face,” “without blur, I felt more visible, I had to hide more,” “funny,” “it focuses my attention,” “surprising, striking,” “much more realistic,” “I have a better precision,” and “higher immersion.” Many participants thus seemed ready to activate the blur effects for gameplay.

The participants who preferred the game without the blur effects generally found the blur disturbing and tiring (producing a headache, for example). Five participants said they thought the blurring effect caused their fatigue. Some participants, especially some expert gamers, described the blur as a “discomfort.” Some participants perceived it as “too strong.” Furthermore, the blur annoyed these people when they were exploring the image and looking for targets on the screen.

Most participants (83 percent, 25/30) said the use of blur didn’t modify their gaming strategy.

Discussion. Before the experiment, we had formulated two hypotheses:

1. Visual blur degrades participants’ performance because it blurs the displayed image to some extent.
2. Visual blur improves the participants’ subjective preference because it provides an additional visual effect that potentially increases the scene’s realism or gameplay fun.

Our results corroborate Hypothesis 1. Players were indeed less accurate during shooting. Frags decreased by about 7 percent. The number of deaths increased slightly, but this difference is less relevant because it was less than one Frag.

The increase in Deaths with blur effects, together with the decrease in Frags and lower precision, could indicate that blur effects make the game harder. However, this result is significant only for more expert gamers who are used to playing the game with low visual quality and all special effects disabled. These players have invested a lot of strategy and tactics in the original game, and any change reduces that investment’s advantage. This stresses the importance of considering both learning and resistance to change when designing visual rendering and gameplay for video games.

Regarding Hypothesis 2, half the players who gave an opinion preferred the presence of blur in terms of fun, presence, and VE realism. The same number of participants disliked the effects. Although these results only partially support Hypothesis 2, they seem sufficient for recommending blur effects in some game design cases.

Several participants’ comments suggested that they didn’t constantly look inside the focus area at the screen’s center. This suggests at least two gameplay phases. In one phase, the player uses his or her visor when shooting at enemies. In this case, our model’s computation of focal distance was always well adapted to the user’s gaze. In the other phase, the player explores the image to locate enemies. During this phase, the computed focal distance might not correspond to the users’ actual attention. For some of them, this situation generated both discomfort and fatigue. An autofocus zone with a zone-screen ratio of 0.25 receives 75 percent of a user’s gaze (see Figure 6). This seems sufficient for many players who liked the blur effect but not for others. The appreciation seems to depend strongly on players' gaming strategy.

Because some participants complained that the blur was too strong, even attributing fatigue to it, developers must carefully tune the blur intensity. Applications might also let users change the amount of blur.

To our knowledge, developers for the new generation of games simply compute the focal distance using the depth of the pixel located at the screen's center. This straightforward technique will likely increase users' discomfort and degrade their subjective preferences. We encourage developers to compute the blur effect the way we propose it and give users the options to adjust the blur strength and to enable or disable the blur effect.

This study focused on FPS video games, which provide a challenging scenario for testing our blur effects. However, visual blur effects could also be valuable in other applications and contexts. For example, a game designer could use them to focus or distract the player. In applications such as architectural design and project reviews, the DoF blur could be used to improve the perception of depths and distances in the virtual world. Indeed, it could be more adapted to navigations in which the user is less stressed and less forced to explore the entire image to find enemies. Further work is now necessary to investigate the future applications and uses of visual blur effects.

Future work might also involve evaluating how the DoF blur effect influences cybersickness and depth perception in the VE. Finally, it would be interesting to further evaluate this effect when computed with an accurate eye-tracking system.[3]

Acknowledgments

We thank the anonymous reviewers for their comments. We also thank all the participants in our experiments for their kindness and patience. The French National Agency of Research supported this research through the grant ANR05 RNTL01715.

References

1. D. Atchinson and G. Smith, *Optics of the Human Eye*, Elsevier, 2000.
2. M. Potmesil and I. Chakravarty, "A Lens and Aperture Camera Model for Synthetic Image Generation," *Proc. Siggraph*, ACM Press, 1981, pp. 298–306.
3. S. Hillaire et al., "Using an Eye-Tracking System to Improve Camera Motions and Depth-of-Field Blur Effects in Virtual Environments," *Proc. IEEE Virtual Reality Conf.* (VR 08), IEEE Press, 2008, pp. 47–50.
4. G. Riguer, N. Tatarchuk, and J. Isidoro, "Real-Time Depth of Field Simulation," *ShaderX2: Shader Programming Tips and Tricks with DirectX 9*, Wolfgang Engel, ed., Wordware, 2003, pp. 529–556.
5. Y. Uralsky, "Efficient Soft-Edged Shadows Using Pixel Shader Branching," *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation*, M. Pharr, ed., Addison-Wesley, 2005, pp. 269–282.
6. A. Kenny et al., "A Preliminary Investigation into Eye Gaze Data in a First Person Shooter Game," *Proc. European Conf. Modelling and Simulation*, ECMS, 2005, pp. 146–152.

Sébastien Hillaire *is a PhD student in computer science in the Bunraku research team at the French National Research Institute for Computer Science and Control (INRIA) and France Télécom R&D. His research interests include VR, 3D interaction, and computer graphics using graphics hardware and visual-attention models. Hillaire received his master's in computer science from the University of Rennes 1. Contact him at sebastien.hillaire@irisa.fr.*

Anatole Lécuyer *is a senior researcher in the Bunraku research team at the French National Research Institute for Computer Science and Control (INRIA) and Collège de France's Laboratory of Physiology of Perception and Action. His research interests include VR, 3D interaction, haptic interaction, and brain-computer interfaces. Lécuyer received his PhD in computer science from the University of Paris XI. He's an associate editor of* ACM Transactions on Applied Perception. *Contact him at anatole.lecuyer@irisa.fr.*

Rémi Cozot *is an assistant professor at the University of Rennes 1 and a member of the Bunraku research team at the French National Research Institute for Computer Science and Control (INRIA). His research interests include enhancement of real-time rendering using human vision features and color appearance models. Cozot received his PhD in computer science from the University of Rennes 1. Contact him at remi.cozot@irisa.fr.*

Géry Casiez *is an assistant professor in the University of Lille 1's computer science department and a member of the Alcove research team at the French National Research Institute for Computer Science and Control (INRIA). His research interests include 2D and 3D interaction, haptic interaction, and the empirical evaluation of user interfaces, including associated metrics and predictive models of human performance. Casiez received his PhD in computer science from the University of Lille 1. Contact him at gery.casiez@lifl.fr.*