Gaze-dependent Ambient Occlusion

Sebastian Janus*
Supervised by: Radosław Mantiuk†

Faculty of Computer Science
West-Pomeranian University of Technology
Szczecin / Poland

Abstract

Ambient Occlusion is a method of creating shades on the scene, due to occlusion. It is a good looking approximation of the light radiation, however it is very expensive method. It needs a large number of samples to get fair effects. In this article we propose a speed increase of the AO rendering, by using the eye tracker. Human cannot see high frequency details in parafoveal, and we can render this area with less accuracy. We decrease the number of AO samples with distance from the observer gaze point. The absence of AO shading in parafoveal is being rarely noticed and reducing the samples gives us considerable rendering speed boost.

Keywords: Ambient Occlusion, Eye tracking, Modern computer graphics

1 Introduction

Ambient Occlusion (AO) is a shading algorithm, which adds a reality to the rendered scene. It approximate how the given point is occluded by other objects (surfaces). However, it is fair complex and still it is hard to achieve a real time AO performance on the nowadays GPUs. In this work we provide a solution that speeds-up the AO rendering without significant decrease of the quality of the final image.

We present a concept of rendering ambient occlusion self-shadows affected by the information about the humans' viewing direction and its limited region of interest (ROI). Therefore, if we knew the point on which observer has focus, we could render this point surroundings with maximum precision and further regions with minor quality. We use the gaze-dependent Contrast Sensitivity Function to alter the influence of the AO factor and to model decrease of rendering quality.

During experimental evaluation we evaluate whether the humans are capable of seeing the difference of the rendering quality outside the ROI.

To achieve the interactive rendering, we base our AO implementation on nVidia OptiX[1] library, which operates on CUDA[2] for the supreme speed of the complicated calculations. It computes ambient occlusion factors with differential accuracy, depending on the location of human gaze point captured by the eye tracker.

In Section 2 we describe the Ambient Occlusion algorithm and discuss why the full Ambient Occlusion method has been chosen instead of cheaper approximated methods. Then in Section 3 we present our concept of the gaze-dependent ambient occlusion rendering. In Section 4 we describe the implementation. The results are discussed in Section 5, followed by conclusions and future work in Section 6.

2 Background

Ambient occlusion is a shading model which is used to increase scene realism in rendering systems based on the local illumination models. It requires much less computation in comparison to the full global illumination solutions, however it still needs demanding resources to achieve high quality renderings.

2.1 Ambient Occlusion

In Phong reflection model, diffuse and specular reflections are varying due to observer and lights position, but ambient reflections are constant. Having these assumptions we miss the self shadows of the rendered objects - which is a big lack of reality. Adding Ambient Occlusion [2] algorithm for computing the ambient reflections factor creates very convincing soft shadows, which combined with indirect lighting gives realistic results. The model does not look flat - what often is happening with indirect lighting and multiple light sources. The result looks similar to Global Illumination and it is possible to say that it simulates it. Good point is, it is of course less complex than full global illumination [1].

This method is an integration of visibility, computed from each pixel on the rendering screen. This integration is solved by Monte Carlo method, where we achieve the

*sebastian.janus@o2.pl
†rmantiuk@wi.zut.edu.pl

[1] See: http://www.nvidia.com/object/optix.html
[2] See: http://www.nvidia.com/object/cuda_home_new.html

result by casting very large number of samples - counted in hundreds. Sample rays are traced from every point into random directions on the hemisphere:

$$k_a = \frac{1}{\pi} \int_{\Omega} V_p(\vec{\omega})(N \cdot \vec{\omega}) d\omega, \quad (1)$$

where k_a denotes the occlusion factor. V stands for the binary visibility function from the certain point p, which returns positive values when ray does not intersect any geometry until reaching some given distance (for the purpose of ray tracing in closed scenes), and a negative value when traced ray hits any object. p and its normal vector N define the surrounding hemisphere Ω [5].

The k_a factor is used in the Phong's reflection equation:

$$I_p = k_a i_a + \sum_{\text{m} \in \text{lights}} (k_d(\hat{L}_m \cdot \hat{N}) i_{m,d} + k_s(\hat{R}_m \cdot \hat{V})^{\alpha} i_{m,s}), \quad (2)$$

where i_d and i_s are defined as the intensities of the current light source, i_a is a constant value or a sum of ambient light of all light sources, k_d is a diffuse reflection constant (lambertian reflectance, depending on the angle between the direction $\hat{L}_m$ toward current light source and the surface normal vector $\hat{N}$), and k_s is a specular reflection constant with $\hat{R}_m$ being a light perfectly reflected ray.

The calculation of the ambient occlusion factor is depicted by the Algorithm 1.

Algorithm 1 Ambient occlusion algorithm

```
for i := 1 → screenWidth do
  i ← i + 1
  for j := 1 → screenHeight do
    j ← j + 1
    {Calculate ambient occlusion for every pixel}
    occlusion:=0;
    for k := 1 → aoRaysNumber do
      k ← k + 1
      {Cast rays in a random directions from the following pixel};
      occlusion+ = castAoRay(randomDirection)
      { castAoRay returns 1 if it hits something}
    end for
    occlusion = occlusion/aoRaysNumber
    { Calculate the ambient occlusion factor. }
  end for
end for
```

2.2 ScreenSpace Ambient Occlusion

The idea of the Screen-Space Ambient Occlusion (SSAO) was proposed by Crytek game studio in their Crysis game as the fast alternative of the standard ambient occlusion technique [4]. SSAO works in real time, although renders effects of lower quality. The idea lies on using Z-buffer data to compute visibility function for every pixel. It takes a pixel surrounding points and analysis their Z value [4].

However, this method has some important disadvantages, starting with self-occlusions. SSAO samples are taken from the inside of a sphere around each pixel - in non-occluded surfaces almost 50% comparisons return 'occluded' result. This causes haloing around the objects - where the self-occlusion effect disappears. These halos are visible around the boxes in Figure 1. There are various improvements to the original SSAO algorithms trying to fix that haloing, but they are not universal - they simply do not work for all cases.

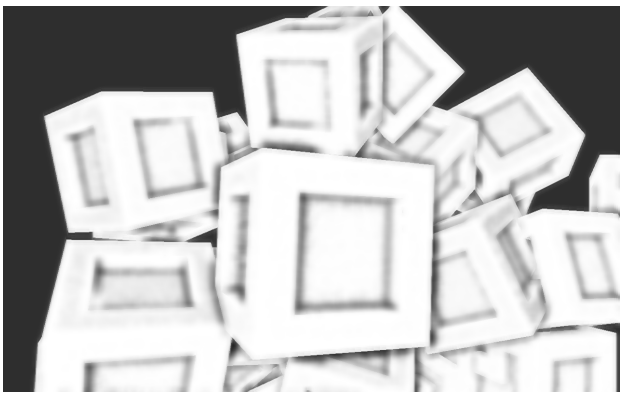

Figure 1: Inaccuracy of the SSAO algorithm may cause black and white halos on the screen [6].

This method is not precise because of the small samples number and the idea of the algorithm - it produces some noise [9]. Moreover, the only objects taken into the consideration are the visible ones. Everything outside the frustum is not interfering with rendered scene. Performance is also depending on the scene - closer objects will require larger radius of sampling.

All these cons made us to the decision for favouring the standard ambient occlusion technique. The main problem is that it requires a lot of computations - for every pixel one should trace hundreds of rays to make satisfactory results. For example for 1680x1050 image resolution and 500 AO factor test rays, the AO algorithm requires tracing of 882 million rays to render one frame. For the full HD (1920x1080) and increased quality to 1000 rays, the sum rises to 2,037 millions what makes the AO not possible to render in real time based on the contemporary graphics systems.

3 Gaze-dependent rendering of Ambient Occlusion

Gaze-dependent ambient occlusion is a solution of providing a full detailed ambient occlusion effect in a certain region of interest. The further from the gaze point, the less detailed ambient factor is rendered, saving the computing time and leaving observer with a feeling that the scene is fully detailed.

3.1 Rendering system

The main goal of this work is to track observers gaze, and to use information about the gaze direction to render parafoveal parts of the screen with less quality. The outline of the gaze-dependent ambient occlusion system is presented in Figure 2.

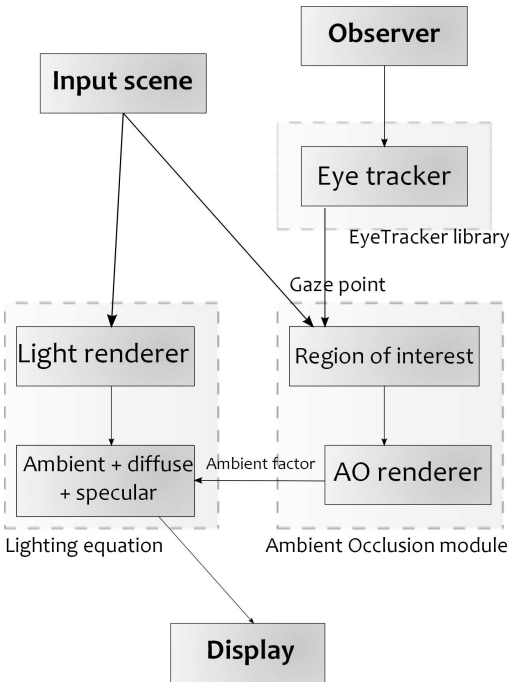

Figure 2: Gaze-dependent ambient occlusion rendering system.

The input data is a 3D scene defined by *.obj file. We have also an observer, whose eyes are tracked. From the eye tracker library we receive the actual gaze points, which are used to calculate the ROI shape and position. Then we have the main lighting renderer, calculating final colour from the Phong lighting equation, and AO renderer, calculating the ambient factor with use of the gaze-dependent ambient occlusion algorithm. Finally, we blend these two ambient factors depending on the distance from the centre of the ROI and we display it.

3.2 Region of interest sampling

Changing the rendering scene by modifying the ambient occlusion factor is rather subtle and it is treated as high frequency information. Therefore, we can use a gaze dependent Contrast Sensitivity Function (CSF) for a function describing how the human eye treats contrast changes in a given distance from a gaze point [15].

We will use drop-off of visual sensitivity across the visual field, for modelling the AO accuracy. We decrease ambient occlusion for the constant ambient factor in lighting equation while we increase the distance from the ROI

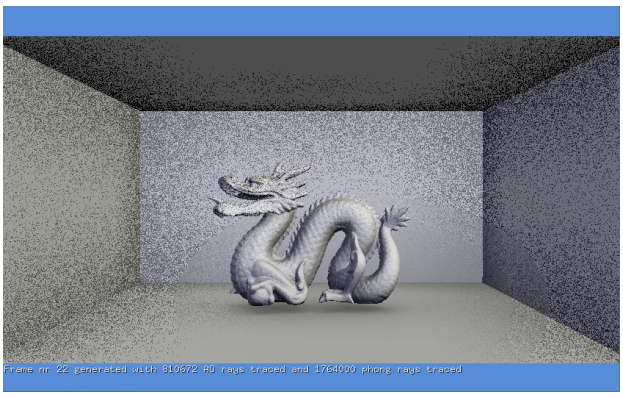

Figure 3: Gaze-dependent ambient occlusion without blending (visible noise).

centre. In the meantime, we decrease the number of the AO sampling rays. We do not need high accuracy, when the impact of this factor is getting more and more inconsiderable [3].

In addition, with CSF we avoid visible noise (see Figure 3), which is adverse for the observer. It is noticeable even in the parafoveal vision and it generates temporal aliasing. The noise is produced by the ambient occlusion when we use small samples number. In our work, that noisy result is hidden by blending with the normal lighting equation ambient factor.

The precision downfall is given by the contrast sensitivity function:

$$C_t(E,f) = C_t(0,f) * exp(kfE), \quad (3)$$

where C_t denotes contrast sensitivity for spatial frequency f at an eccentricity E, k determines how fast sensitivity drops off with eccentricity (the k value is ranged from 0.030 to 0.057). Based on the above equation, the cut-off spatial frequency f_c can be modelled as:

$$f_c = min(max\_display\_cpd, 43.1 * E_2/(E_2+E)), \quad (4)$$

with $E_2 = 3.118$, which is retinal eccentricity at which the spatial frequency cut-off drops to half its foveal maximum (from 43.1 cpd to 21.55 cpd, see details in [16]). In this equation we flatten the contrast sensitivity with $min(max\_display\_cpd, ...)$ operator to take into consideration the limited resolution of our display (see Section 5.1). That flattening is represented on the Figure 4 as a magenta line.

The plot of the contrast sensitivity function is presented on the Figure 4. As we can see, it is quickly decreasing and then staying around some low level. On the Figure 5 we can see a region of interest mask preview. The lighter areas mean that AO is computed with maximum precision, the darker areas mean lower precision. Blending factors of AO ambient and ambient from Phong lighting equation

have the same distribution - the lighter areas means higher weight of the AO ambient factor.

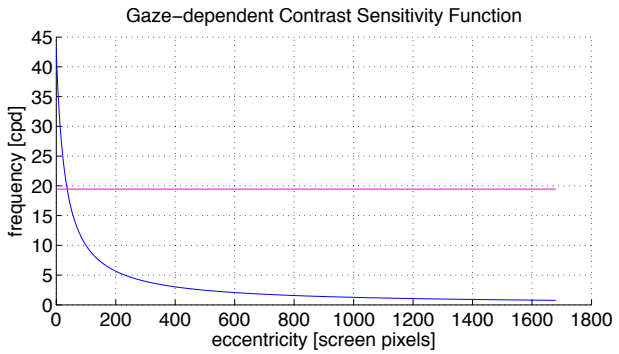

Figure 4: Gaze-dependent Contrast Sensitivity Function. The magenta line denotes threshold frequency of the display used in the experiments.

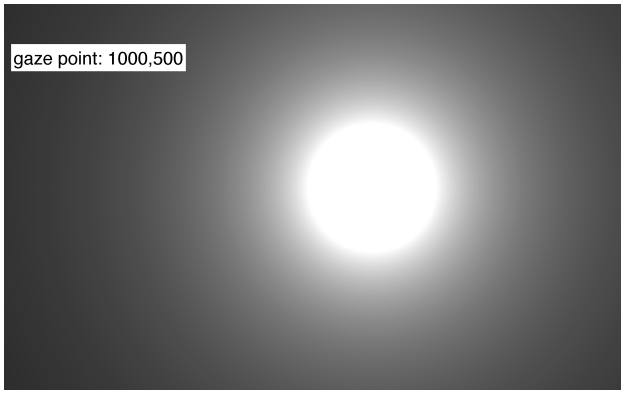

Figure 5: ROI based on Contrast Sensitivity Function preview.

3.3 Eye tracker fixations

Eye trackers capture two types of eye movement called *saccades* and *smooth pursuit*. A smooth pursuit is active when eyes track moving target and are capable of matching its velocity. A saccade represents a rapid eye movement used to reposition the fovea to a new location, which lasts from 10 ms to 100 ms [12]. The main goal of the gaze tracking is to capture a single location an observer intents to look at. This process is known as a *visual fixation*. A point of fixation can be estimated as a location where saccades remain stable in space and time [13]. We approximate this mechanism by averaging a number of raw gaze points. There are many fixation algorithms which are trying to provide a gaze point from these samples, however they are complicated and none of them had worked better for our solution, where we expect a stable results, than simple average of points [14].

Additional stabilisation of the gaze position is achieved by rendering delay. There are about 5-10 frames per second rendered, and we take gaze points array once before rendering new frame. It means that we are averaging gaze samples from about 100-200 ms period, which effects with a lot of samples and in the outcome the mean value is not affected by few different samples. In that case it is a fair advantage.

Human field of view for one eye is about 130° in horizontal plane and about 120° in vertical plane. What is more, the second eye is extending the horizontal plane to about 200°. The binocular field of view then is about 60° [10]. Assuming that, for human angle of view the ratio between horizontal and vertical planes is like $1.538 : 1$. That is why our region of interest shape would not be a simple circle, but a widened ellipse (to take advantage from the differing field sizes). This would not work if we will look with single eye, but for binocular gaze it will work perfect.

4 Implementation

In this section we provide a description of our implementation of the gaze-dependent AO system, proceeded by a short introduction to the OptiX library. Our application is based on the AO sample from the OptiX SDK package. We extended this sample to the gaze-dependent technique supported by the eye tracker library.

4.1 OptiX

Our software is implemented using OptiX engine [11]. It is developed by nVidia and it is described as programmable ray tracing framework, which can be used to rapidly build ray tracing applications. Computing speed of application based on that engine gives fast results across nVidia GPUs with conventional C or C++ programming.

OptiX is helpful in detecting collisions, calculating sound volume, radiation research, and other rendering purposes. Developers can write their own single ray programs. These programs are divided into few categories: closest hit, any hit, intersection, selection, ray generation, miss and exception programs. Using an ensemble of these programs gives us a rendering algorithm. Shading language is based on C/C++ for CUDA, with all its features like pointers, templates, and overloading. One is encouraged to use object model as well.

4.2 Ambient occlusion based on OptiX

Ambient occlusion based on OptiX is divided into a host program (written in C++), and a few GPU shaders programs:

- Ray generation program - responsible for proper generation of the rays from the viewer towards the scene.

Whenever the camera has moved, different rays are generated.

- Closest hit program - responsible for calculating lighting radiance. This is main shader used for tracing rays from the viewer - it looks for intersections with scene objects, and if it finds any it starts computing the colour basing on the Phong lighting equation and our gaze-dependent AO.
- Any hit program - responsible for calculating any hit occlusion. It is used for tracing rays from the intersection points (pointed by the rays which are calculating radiance). We can call them secondary rays, which are cast in a big number to obtain the ratio for hit/miss rays - it gives us the ambient occlusion factor.
- Miss program - responsible for determining the background colour - it is used when our primary rays do not hit anything.
- Exception program - responsible for exceptions - used in case we get incorrect value of lighting.

Apart from the host program which is responsible for all the input/output, communication with user and setting (or changing) the parameters of the GPU programs, the main code is in the closest hit program. There we are calculating the number of the ambient occlusion rays to be cast, there we cast these rays, and there we finally calculate the colour of each pixel.

As it was said before our application is based on a ambient occlusion sample. The main differences instead of main host program (another scene, controls and so on) are in the closest hit shader. We had to provide ROI computation based on contrast sensitivity function, which gives us varying ambient occlusion rays number. Because of that information which pixel are we computing at the moment was significant. Then we had to implement blending the AO ambient factor and Phong lighting equation ambient factor, and there comes gaze-dependent ambient occlusion.

4.3 EyeTracker library

We use the ETlib library for the purpose of tracking observers' gaze direction. This software is responsible for managing communication between the eye tracker software (which runs on another computer) and our application. The eye tracking session starts with the calibration - observer looks at on given point on the screen. The process is finished after registration of 5 points. Precise calibration is extremely important because it affects further accuracy of captured data [7, 8].

Using ETlib we can receive last gaze point (which is not stable - unfortunately, human eye is moving many times in a very short period) or a set of gaze points since the last request. We use the second approach, and with an array of points we make an average point - then we proceed this point as a variable which will be used then by the GPU programs.

5 Experimental evaluation

Our objective is to present, that we can minimise the ambient occlusion sampling in parafoveal. What is more it gives us a performance boost. We present a images for different ROI position, and analyse rendering times.

5.1 Stimuli and hardware setup

The scene presents the fixed Stanford Dragon Model[3], enclosed in the 5 walls box. The scene consist of 50,008 vertices and 100,005 faces, and gives us good performance test object. We render this scene with full frame ambient occlusion to obtain an ideal image, or we render it with use of ROI (controlled by eye tracker) using gaze-dependent AO.

In our experiment, we used the SMI RED250 eye tracker, which gives us refresh rate 250 Hz and accuracy 0.5°. The computer is equipped with 2.8 GHz Intel i7 930 CPU with 8 GB of RAM, Windows 7 64bit OS, and a GPU nVidia GeForce 480 GTI 512MB - one of the fastest nVidia graphic cards. The hardware setup is presented in Figure 6.

Figure 6: Apparatus used during evaluation and experiments. The RED250 eye tracker is located under the display screen.

The lab display is 22 inch Dell, with the 1680x1050 resolution (60 Hz). It is measuring 47.5 cm wide and 30 cm high, what gives 43° horizontal, and 28° vertical. With the screen resolution 1680x1050 that gives 40 pixels for one degree, which is about 20 cpd. That is our maximum,

[3] http://www.mrbluesummers.com/3572/downloads/stanford-dragon-model

which is marked as *max_display_cpd* in Equation 4 and as a magenta line in Figure 4.

Eye tracker is connected to the remote computer. We launch it using remote desktop, and the ETlib (See subsection 4.3) gives as the array of gaze points since last call.

5.2 Results

Quality of rendering

An example rendering with the full frame ambient occlusion is presented in the Figure 7. We consider the full frame rendering with 400 AO rays per pixel as image in all quality comparisons.

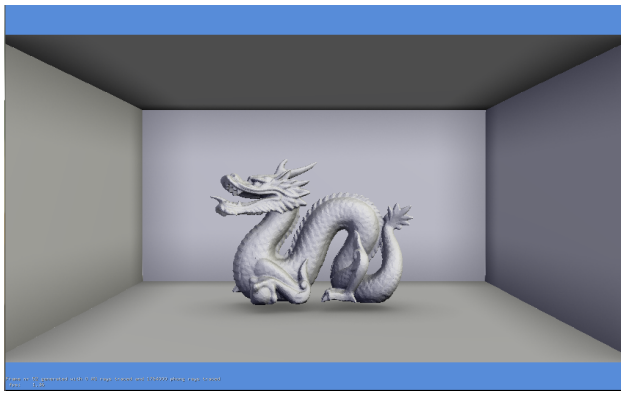

Figure 7: Ideal, reference image with full frame ambient occlusion

In Figure 8 one can see images rendered with gaze-dependent ambient occlusion for various locations of the ROI. Please, notice that shading caused by the AO factor is stronger in the centre of the ROI and weakens with the distance.

On the upper image in Figure 8 there is visible AO effect in the upper corner of the box, and there is no shadow below the dragon model. On the image where the position of the ROI is on the dragon these shadow is visible very well, and there are little shadow on the wall behind the dragon. There are no shadows on the higher corners at all.

During the pilot study with the eye tracker, we assessed the quality of the AO shadows as a very good in comparison to the reference image. The contrast sensitivity function was doing well, and smaller shades in a greater distance from a centre of ROI were rarely noticeable.

Rendering time

To compare timings for the full frame AO and the gaze-dependent AO we measure the speed of rendering for three different camera settings (called as Camera 1, 2, and 3, see Figure 9).

We achieved 1.26 fps, 0.96 fps, and 0.62 fps for Camera 1, 2, and 3 respectively for the resolution of 840x525

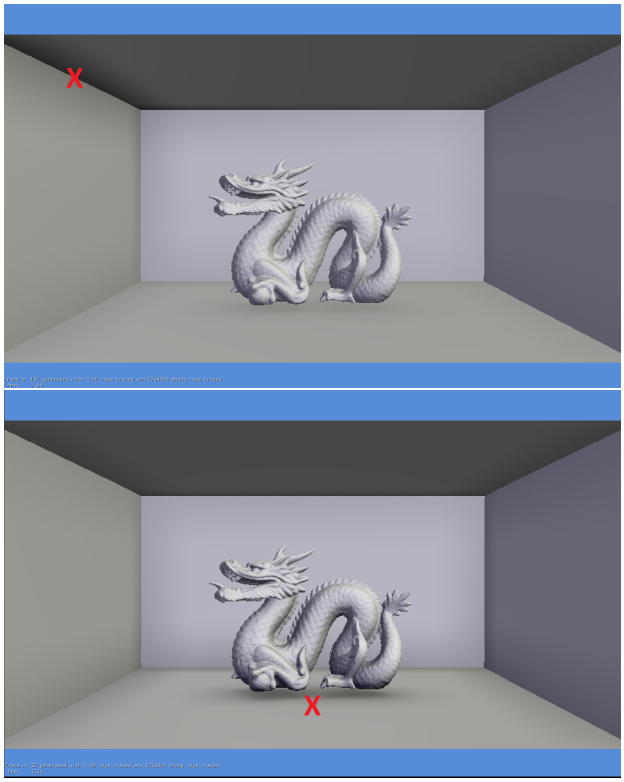

Figure 8: Image rendered with gaze-dependent ambient occlusion. The red X points the centre of the ROI.

pixels and 400 AO rays (the resolution was reduced for performance reasons).

The same images was rendered using eye tracker and the gaze-dependent AO rendering technique. The rendering speed depends on location of the ROI. There are more computations (e.g. intersection tests) in regions of the scene with more triangles so AO rendering time increases. However, the overall rendering time using the gaze-dependent technique is shorter in comparison with the full screen AO, even by 276% in the best case (see Table 1).

Generally, the results show significant performance increase in a regions of a small triangles number (box walls all around the screen). Worse outcome is when we take the hardest to compute environment - middle of the screen (the biggest samples number) and the dragon object (high level of the triangles - ray tracer has harder work). However, even in that worse case, we get a noticeable rendering speed-up.

5.3 Discussion

Using gaze-dependent ambient occlusion can increase rendering speed without a noticeable quality loss. Better GPU

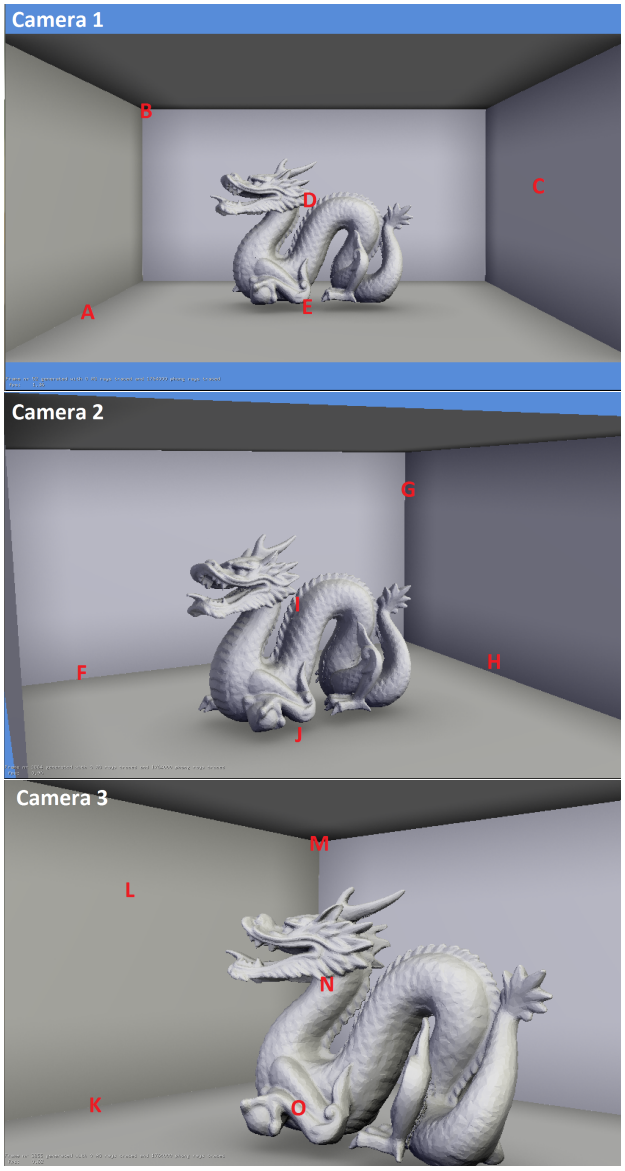

Figure 9: Locations of the ROIs depicted as the red letters.

would be useful, both for rendering smooth ambient occlusion and for performing study concerning rating a visibility of AO in the parafoveal area. The faster reaction time would also improve the gaze-dependent approach in context of fast eyes movement.

For the time of writing this article, we did not use the newest graphic card (nVidia GeForce GTX 580), which has about 20% faster memory bandwidth, texel rate and pixel rate. Probably, using two GPUs connected with SLI[4], would give us better times. With such hardware improvements, we would be closer to the smooth real time rendering and the results would be even better.

[4]SLI stands for Scalable Link Interface, which is the name of technology allowing to link two or more GPUs to perform parallel processing of a computer graphic for single output.

Table 1: Rendering speeds for gaze-dependent AO. Speed-up is $((FPS - OriginalFPS)/OriginalFPS) * 100\%$, where $OriginalFPS$ means the rendering speed of full frame rendering.

Camera	Gaze-point	Rendering speed	Speed-up
Setting 1	A	4.10 fps	201%
	B	4.21 fps	210%
	C	4.06 fps	199%
	D	1.68 fps	24%
	E	2.11 fps	55%
Setting 2	F	2.36 fps	146%
	G	2.65 fps	176%
	H	2.19 fps	128%
	I	1.30 fps	35%
	J	1.40 fps	46%
Setting 3	K	1.82 fps	194%
	L	2.33 fps	276%
	M	2.24 fps	261%
	N	1.00 fps	61%
	O	1.17 fps	89%

6 Conclusions and future work

Summarising, we are capable of having significant rendering speed increase with the Ambient Occlusion shading. Furthermore, the result of rendering worse shaded image in the parafoveal is being skipped by the human eye.

The impression of the gaze-dependent image is received as a bit worse, but mainly because of not perfect eye tracking. Sometimes, the ROI is quickly moving which could be uncomfortable for the viewer, especially when suddenly the gaze point escapes from the real gaze point for one animation frame. However, there is a need to perform a experiments on a frequent group of people to know in which way should we improve our approach. Anyhow, with improvement of the fixation algorithm we could achieve better results. Also, with the improvement of the hardware we will have smoother animations.

References

[1] P. Berto. Occlusion tutorial. 2007.

[2] M. Bunnell. *GPU Gems 2, Chapter 14, Dynamic Ambient Occlusion and Indirect Lighting*. Addison Wesley, 2005.

[3] J. Yang E. Peli and R. B. Goldstein. *Image invariance with changes in size: the role of peripheral contrast thesholds*. JOSA A, Vol.8, Issue 11.

[4] M. Mittring14 Crytek GmbH. Finding next gen cryengine 2. 2007.

[5] S. Hill. Hardware accelerating art production. 2004.

[6] M. Lagergren. Ssao. 2009.

[7] R. Mantiuk, M. Kowalik, A. Nowosielski, and B. Bazyluk. Do-it-yourself eye tracker: Low-cost pupil-based eye tracker for computer graphics applications. *Lecture Notes in Computer Science (Proc. of MMM 2012)*, 7131:115–125, 2012.

[8] R. Mantiuk, A. Tomaszewska, and B. Bazyluk. Gaze-dependent depth-of-field effect rendering in virtual environments. *Lecture Notes in Computer Science (Proc. of SGDA 2011)*, 6944:1–12, 2011.

[9] J. M. Mendez. A simple and practical approach to ssao. 2010.

[10] M. H. Nizankowska. *Podstawy okulistyki*. VOLUMED, 1992.

[11] S. Parker. Interactive ray tracing with the nvidia optix engine, 2009.

[12] D. A. Robinson. The mechanics of human saccadic eye movement. *Journal of Physiology*, 174:245–264, 1964.

[13] D. D. Salvucci and J. H. Goldberg. Identifying fixations and saccades in eye-tracking protocols. In *Proceedings of the 2000 symposium on Eye tracking research & applications (ETRA)*, pages 71–78, New York, 2000.

[14] Frederick Shic, Brian Scassellati, and Katarzyna Chawarska. The incomplete fixation measure. In *Proceedings of the 2008 symposium on Eye tracking research & applications*, ETRA '08, pages 111–114, New York, NY, USA, 2008. ACM.

[15] P. Wenderoth. The contrast sensitivity function.

[16] Qi X. Makous W. Yang, J. *Zero frequency masking and a model of contrast sensitivity.* Vision Research.