Real-time Simulation of Human Vision using Temporal Compositing with CUDA on the GPU

Matthias Nießner, Nadine Kuhnert, Kai Selgrad, Marc Stamminger, Georg Michelson

Computer Graphics Group
University of Erlangen-Nuremberg
Cauerstraße 11
91058 Erlangen

matthias.niessner@cs.fau.de
nadine.kuhnert@medtech.stud.uni-erlangen.de
kai.selgrad@cs.fau.de
marc.stamminger@cs.fau.de
georg.michelson@uk-erlangen.de

Abstract: We present a novel approach that simulates human vision including visual defects such as glaucoma by temporal composition of human vision in real-time on the GPU. Therefore, we determine eye focus points every time step and adapt the lens accommodation of our virtual eye model accordingly. The focal distance is then used to determine bluriness of observed scene regions; i.e., we compute defocus for all visible pixels. In order to simulate the visual memory we introduce a *sharpness field* where we integrate defocus values temporally. That allows for memorizing sharply perceived scene points. For visualization, we ray trace the virtual scene environment while incorporating depth of field based on the sharpness field data. Thus, our algorithm facilitates the simulation of human vision mimicing the visual memory. We consider this to be particularly useful for illustration purposes for patients with visual defects such as glaucoma. In order to run our algorithm in real-time we employ massively parallel graphics hardware.

1 Introduction

Simulating human vision enables beneficial applications for eyeglass manufacturers, treating physicians, and patients with visual defects. Typically, the human eye is constantly in motion in order to gather visual information about the environment. While moving, the eye lens accommodates such that objects of interest appear sharp. That is, the eye lens adapts its shape and thus its refractive power in order to match the focal plane with those objects. Since the eye lens has only a single refractive state at each time step, objects not on the focal plane are unsharp. The human brain memorizes sharply perceived regions (obtained with potentially different refractive states) for some time, thus generating a visual impression without defocus [HB89].

In this work we simulate this process by employing an eye model that focuses on different areas of a virtual scene environment. Therefore, within a time step, we first determine potential focus points by rendering the scene from the current view point (using GPU ray tracing) and detecting features such as edges or corner points. We then set the eye lens accommodation (i.e., the refractive power) in order to define the focal plane. Next, we use the obtained focal plane to compute sharpness values for all observed scene regions; i.e., we compute the *circle of confusion* for all visible pixels. The bluriness, which is computed every time step, is then temporally integrated into a *sharpness field*. The sharpness field consists of sharpness values with corresponding time stamps for all scene locations. The last step of our approach is to visualize the scene environment under consideration of the sharpness field. While our approach facilitates the simulation of healthy eyes, a particularly useful application is the illustration of visual defects such as glaucoma [Ger08]. An overview of our algorithm is depicted in Figure 1.

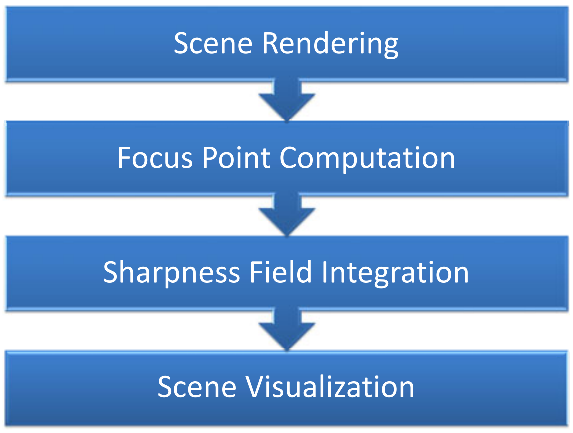

Figure 1: Algorithm overview.

For our simulation we employ NVIDIA's GPU ray tracing framework OptiX [PBD$^+$10]. OptiX provides an API for parallel GPU ray tracing that allows the customization of the ray tracing steps, for example acceleration structure construction, efficient ray traversal, shading and intersection computation. In addition, we use CUDA [Nvi11] to parallelize involved image processing operations whereas Optix's interoperability functions facilitate shared data access between the two GPU frameworks. In the end we are able to run our simulation approach in real-time on current graphics hardware.

To sum up, the contributions of our approach are

- simulation of human vision under temporal aspects,
- real-time execution on current graphics hardware,
- illustration of visual defects such as glaucoma.

2 Previous Work

Simulation of Human Vision Traditional approaches that simulate human vision consider a single time step and visualize simulated results based on a selected eye model. The first step in such a simulation is to determine the eye lens accommodation. To this end, Mostafawy et al. [MKL97] introduce the *virtual eye* that determines its accommodation based on the distance of observed objects. We use a similar idea since we determine the refractive power of the eye lens based on the view distance of specific features (see Section 3). The disadvantage of this approach is the inability to (correctly) deal with multi-lens systems such as eyeglasses. Wavefront tracing [K$^+$64, Sta72] is used to avoid this limitation and has been applied for the optimization of the design of eyeglasses [LSS98]. In addition, wavefront tracing is an integral part of several approaches that simulate human vision [Bar04], [KTMN07], [KTN10], [NSG12]. While these approaches focus on the lens system of the human eye, they do not address temporal aspects of human vision.

Visual Defects Most common visual defects are related to the refractive properties of the eye lens (*ametropia*) [BP07]. Common cases are *hyperopia* where the eye is too short, *myopia* where the eye is too long, or *astigmatism* where the eye lens creates two focal lines instead of a single focal plane. In our work we particularly focus on *glaucoma* [Ger08]. This disease is mostly caused by increased intraocular pressure that, if unnoticed, damages the optical nerve. As a result the visual field is significantly affected. Patients suffering from the disease typically require more time to perceive their environment. Our simulation resembles this effect as depicted in Section 3.

GPU Ray Tracing Ray tracing [Whi80] is a widely used rendering approach in computer graphics. Since ray tracing resembles light paths on a physical basis, it is essential for the simulation of human vision. For instance, distributed ray tracing [CPC84] is typically used to facilitate depth of field effects. In the recent years ray tracing approaches for GPUs have been developed due to the advancement of such hardware architectures. Aila and Laine [AL09] propose such an approach that exploits the computational capabilities of modern GPUs. NVIDIA uses this method in their ray tracing engine OptiX [PBD$^+$10]. We use OptiX for our simulation approach since it provides fast acceleration structure construction and efficient ray traversal on the GPU.

3 Scene Rendering and Focus Computation

The first step of our simulation is to render the virtual scene environment in order to obtain color and distance values for each observed scene point. Therefore, we ray cast the scene using a pinhole camera model for rendering. To achieve real-time framerates for this step, we use NVIDIA's ray tracing engine OptiX [PBD$^+$10] that runs on the GPU. Since we consider the scene environment to be static, we gain best performance employing a kD-tree for acceleration. Hence, for each frame we obtain a color and depth map depending on the current (virtual) camera setup. Next, we use that data to determine potential focus points. Focus points are considered to be features of the color image; i.e., points that are most likely of visual interest such as edges or corners [Hub88]. Therefore, we perform corner detection on the previously obtained color data by using the FAST feature detector [RD05]. We then select a single feature (per time step) that we consider to be the most likely focus point and set the focal plane according to the depth value of the chosen feature. At every subsequent time step we use a different feature from the obtained feature list. In reality, the typical timeframe between two features (for a real eye) is 10 to 100 ms [Duc07].

Initially, we select the a relatively centric feature based on a bivariate Gaussian. Then, our approach tries to select features that are nearby, however, considering a minimal distance threshold with respect to previously selected features. Thus, we are able to resemble the motion of a real eye (*saccades*). Note that if the eye position changes, the feature list is re-computed for the current view. Figure 2 shows an example scene with features detected by the FAST detector and the resulting estimated eye movement.

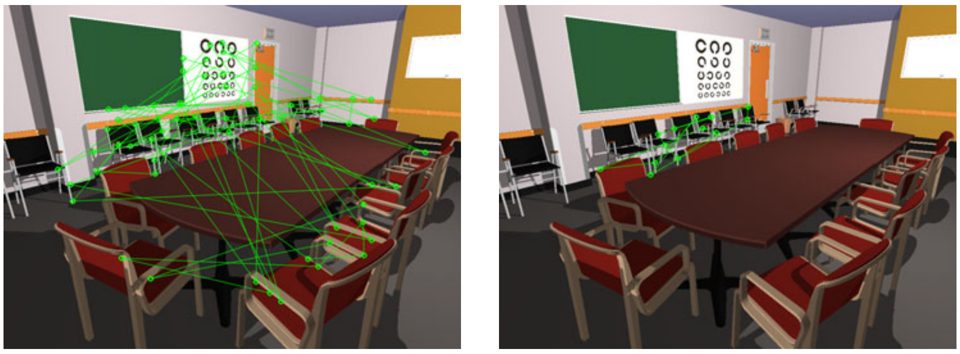

Figure 2: Simulated eye movements of a patient with (right) and without glaucoma (left). The focused points (green circles) are computed using the FAST feature detector.

Once we determined a certain feature in the current view, we set the focal plane P for the current frame accordingly. We compute a circle of confusion (**CoC**) for every pixel based on this focal plane P and the respective pixel's depth value D. Hence we obtain the circle of confusion for each pixel following Demers [Dem04]

$$CoC = \left| A \frac{F(P-D)}{D(P-F)} \right|$$

where A is the aperture size (i.e., pupil radius, which is about 3.5mm), I is the image plane (i.e., distance from eye lens to retina, about 25mm) and F is the focal point computed by

$$\frac{1}{F} = \frac{1}{P} + \frac{1}{I}.$$

Please note that for each frame the focal plane P is fixed (depending on the chosen feature), however, each pixel has different depth values D.

Glaucoma To incorporate the visual glaucoma defect in our simulation, we adapt both the feature selection and CoC computation. Depending on the progress of the disease we limit the trajectory of focus points over time. Thus, we adapt the probability distribution function for feature selection accordingly. We might also skip feature selection for a frame and reuse the feature of the previous frame. In that case we do not need to update the sharpness field (see Section 4). In addition, we enlarge the CoC of all pixels with respect to the screen space distance of the selected feature. While healthy eyes also suffer from this CoC-penalty, the magnitude is significantly larger for glaucomatous eyes [BP07].

4 Sharpness Field

In Section 3 we determine the CoC for every pixel of the current frame. We now introduce the *sharpness field*, that is a uniform voxel grid with cells corresponding to specific locations in space. Every cell defines a CoC and a time stamp value defining the bluriness within the cell. For every subsequent frame, we now integrate the newly obtained CoC values for every pixel. Therefore, we backproject pixel coordinates (depth value and pixel index) into world space to determine corresponding voxel cells in the sharpness field. If the CoC value of a pixel is smaller than the CoC stored in the sharpness field at the obtained location, we update the voxel with the CoC of the pixel and the current time stamp.

Further, we *starve* values in the sharpness field depending on the time stamp. Hence, if a voxel is not updated for a certain amount of time, we increase the CoC accordingly. In our examples we achieved good results by increasing the CoC by about $1-2\%$ every time step.

The sharpness field is initialized with infinitely large CoCs, meaning that the region has not been observed and never been in focus. In addition, we occasionally reset the sharpness field to its initial state mimicing eye blinks.

5 Visualization

For the visualization of the scene environment, we use the color data obtained by the scene rendering pass (see Section 3) and the updated sharpness field (see Section 4). For every pixel in the color image we then determine the corresponding cell in the sharpness field by backprojecting the pixel coordinates into world space. That allows us to retrieve the bluriness for a pixel. The CoC is then used to compute the depth of field for each pixel. Therefore, we distribute filter taps within the CoC and weight them according to a bivariate Gaussian distribution. This is similar to the approach of Riguer et al. [RTI03], however, in contrast to their method, our Gaussian distributions have different standard deviations for every pixel. Thus, we cannot pre-compute filter taps and need to evaluate the Gaussian for each sample on-the-fly while setting the standard deviation according to the CoC diameter of the respective pixel. Please note that the Gaussian cannot be separated in our case since the corresponding kernel is different for each pixel.

	Conference Room	Crytek Sponza
Ray Tracing	7.5ms	11.7ms
Feature Detection	9.8ms	6.8ms
Sharpness Field Update	0.4ms	0.5ms
Image Blur	11.8ms	10.4ms
Total	**29.5ms**	**29.4ms**

Table 1: Total per frame timing measurements for our test scenes. We also break out the separate steps (implemented as distinct GPGPU kernels) of our pipeline.

We would like to point out that this filtering approach is an approximation of physically-based distributed ray tracing [CPC84]. However, the approximate variant provides similar quality while computational cost is significantly lower. In the end we are able to assemble the final image where pixels are sharp that are in-focus in the current frame as well as those that have been in focus before and are now being memorized.

6 Results

Our implementation runs on Windows 8 and uses NVIDIA OptiX 3.0.0 for ray tracing which is based on CUDA 5.0. We have tested our simulation on an Intel Core i7 processor with 2.8 GHz and an NVIDIA GTX Titan graphics card. As test scenes we used the *Conference Room* ($\approx$ 282K triangles) and the *Crytek Sponza* ($\approx$ 263K triangles) where performance measurements account for all run-time overhead except scene shadowing.

Figure 2 shows the results of our feature detection including corresponding eye movements retrieval. We compare healthy with glaucomatous vision where the latter receives less and locally clustered (i.e., eye movement paths are shorter) features.

Figures 3 and 4 depict the result of our simulation after specific time steps. Here within each time step we obtain a single feature and update the sharpness field accordingly. In addition, the scene environment is visualized under consideration of the sharpness field. Note how sharp regions grow as more features are being integrated into the sharpness field. We show the simulation after 1, 4, 20 and 69 time steps/features (first image sequence) and 1, 4, 20, and 82 (second sequence), respectively.

In addition, we provide performance numbers for our test scenes in Table 1 and break out timings for different steps of our algorithm. Even when accounting for dynamic eye movment, which requires the execution of feature detection every frame, we are able to run our approach in real-time above 30Hz.

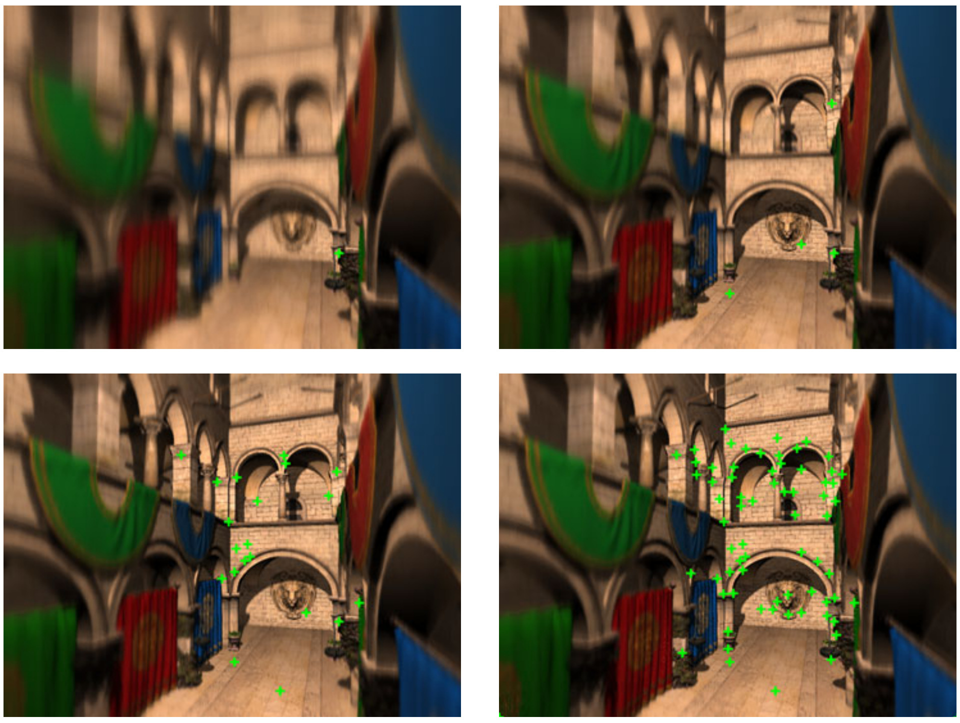
Figure 3: Simulation after 1, 4, 20, and 69 features, respectively. Each additional feature increases the sharply perceived region.

7 Conclusion and Future Work

We have presented a novel approach that simulates human vision under temporal aspects. Our algorithm facilitates the simulation of both healthy eyes and eyes with visual defects. It runs in real-time by employing modern graphics hardware and uses NVIDIA OptiX (based on CUDA).

In the future we would like to extend our approach by relying on eye tracker data in order to resemble eye motion more accurately. That would replace our current focus point detection as presented in Section 3. In addition, we would like to incorporate eyeglasses and ametropic visual defects into our simulation. Therefore, we plan to determine eye lens accommodation by using a wavefront-tracing-based algorithm; e.g., [NSG12]. In fact, the goal towards handling multi-lens systems led us to the design choice of using ray tracing rather than (potentially faster) rasterization for scene rendering.

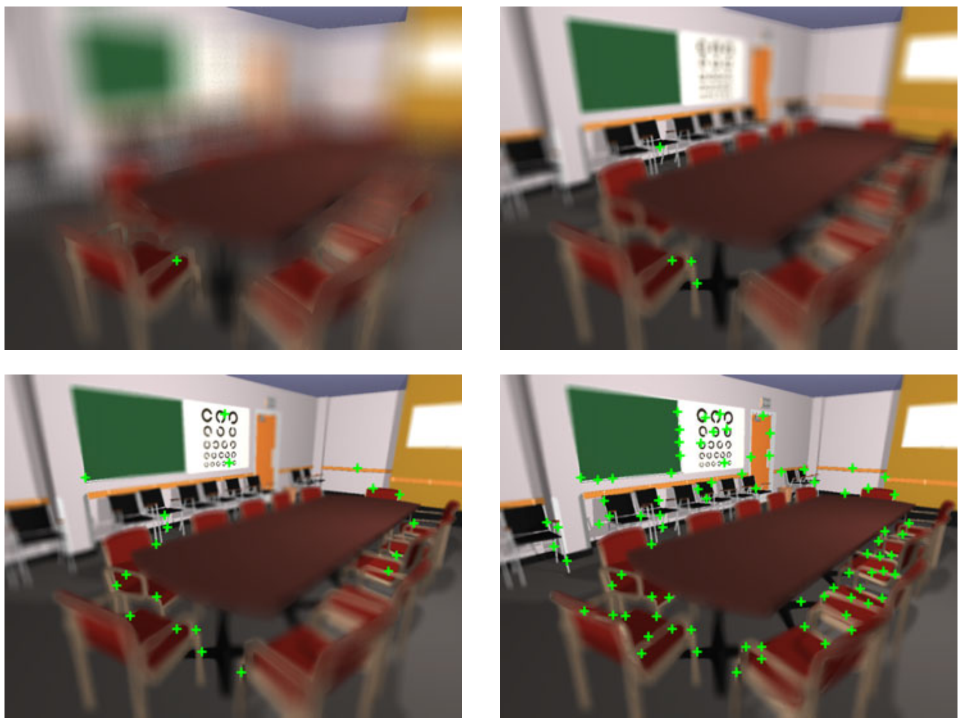

Figure 4: Simulation after 1, 4, 20, and 82 features, respectively. Each additional feature increases the sharply perceived region.

References

[AL09] Timo Aila and Samuli Laine. Understanding the Efficiency of Ray Traversal on GPUs. In *Proc. High-Performance Graphics 2009*, pages 145–149, 2009.

[Bar04] B.A. Barsky. Vision-realistic rendering: simulation of the scanned foveal image from wavefront data of human subjects. In *Proceedings of the 1st Symposium on Applied perception in graphics and visualization*, pages 73–81. ACM, 2004.

[BP07] R.N. Braun and W. Pschyrembel. *Pschyrembel Klinisches Wörterbuch*. Number Bd. 1. De Gruyter, 2007.

[CPC84] R.L. Cook, T. Porter, and L. Carpenter. Distributed ray tracing. In *ACM SIGGRAPH Computer Graphics*, volume 18, pages 137–145. ACM, 1984.

[Dem04] J. Demers. Depth of field: A survey of techniques. *GPU Gems*, 1:375–390, 2004.

[Duc07] Andrew T. Duchowski. *Eye Tracking Methodology: Theory and Practice*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

[Ger08] Ronald D. Gerste. Ophthalmologie: Wie das Normaldruckglaukom entsteht. *Deutsches Ärzteblatt International*, 105(38):A–1960–, 2008.

[HB89] Glyn W. Humphreys and Vicki Bruce. *Visual Cognition: Computational, experimental and neuropsychological Perspectives*. Lawrence Erlbaum Associates Ltd., 1989.

[Hub88] David H. Hubel. *Eye, Brain and Vision*. W H Freeman & Co, NewYork, 1988.

[K$^+$64] J.A. Kneisly et al. Local curvature of wavefronts in an optical system. *Journal of the Optical Society of America (1917-1983)*, 54:229, 1964.

[KTMN07] M. Kakimoto, T. Tatsukawa, Y. Mukai, and T. Nishita. Interactive simulation of the human eye depth of field and its correction by spectacle lenses. In *Computer Graphics Forum*, volume 26, pages 627–636. Wiley Online Library, 2007.

[KTN10] M. Kakimoto, T. Tatsukawa, and T. Nishita. An Eyeglass Simulator Using Conoid Tracing. In *Computer Graphics Forum*, volume 29, pages 2427–2437. Wiley Online Library, 2010.

[LSS98] J. Loos, P. Slusallek, and H.P. Seidel. Using wavefront tracing for the visualization and optimization of progressive lenses. In *Computer Graphics Forum*, volume 17, pages 255–266. Citeseer, 1998.

[MKL97] S. Mostafawy, O. Kermani, and H. Lubatschowski. Virtual eye: retinal image visualization of the human eye. *Computer Graphics and Applications, IEEE*, 17(1):8–12, 1997.

[NSG12] M. Nießner, R. Sturm, and G. Greiner. Real-time simulation and visualization of human vision through eyeglasses on the GPU. In *Proceedings of the 11th ACM SIGGRAPH International Conference on Virtual-Reality Continuum and its Applications in Industry*, pages 195–202. ACM, 2012.

[Nvi11] C. Nvidia. NVIDIA CUDA programming guide, 2011.

[PBD$^+$10] S.G. Parker, J. Bigler, A. Dietrich, H. Friedrich, J. Hoberock, D. Luebke, D. McAllister, M. McGuire, K. Morley, A. Robison, et al. Optix: A general purpose ray tracing engine. *ACM Transactions on Graphics (TOG)*, 29(4):66, 2010.

[RD05] E. Rosten and T. Drummond. Fusing points and lines for high performance tracking. In *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, volume 2, pages 1508–1515. IEEE, 2005.

[RTI03] G. Riguer, N. Tatarchuk, and J. Isidoro. Real-time depth of field simulation. *ShaderX2: Shader Programming Tips and Tricks with DirectX*, 9:529–556, 2003.

[Sta72] O.N. Stavroudis. The optics of rays, wavefronts, and caustics. Technical report, DTIC Document, 1972.

[Whi80] T. Whitted. An improved illumination model for shaded display. *Communications of the ACM*, 23(6):343–349, 1980.