Real-time Simulation and Visualization of Human Vision through Eyeglasses on the GPU

Matthias Nießner*
University of Erlangen-Nuremberg

Roman Sturm†
Rupp + Hubrach

Günther Greiner‡
University of Erlangen-Nuremberg

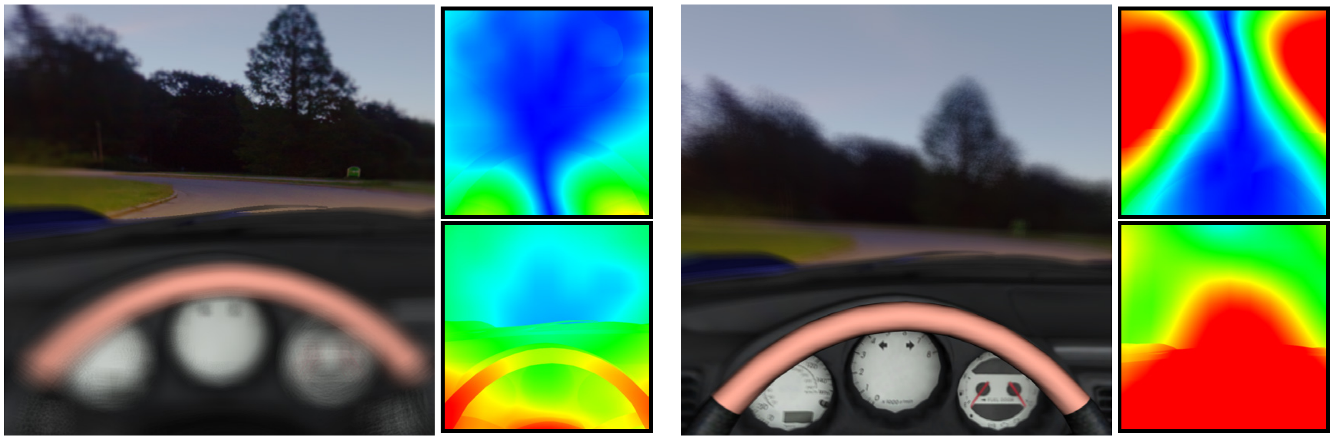
Figure 1: *Simulation results of a presbyopic eye (i.e., limited accommodation) using a progressive addition lens (B-spline/sphere): view through the far vision section (left) and near vision section (right). Defocus is computed approximately and rendertime is less than* 32 *ms per frame. The smaller images show effective astigmatism and refractive power of wavefronts at the virtual eye lens, respectively.*

Abstract

We present a novel approach that allows real-time simulation of human vision through eyeglasses. Our system supports glasses that are composed of a combination of spheric, toric and in particular of free-form surfaces. In order to obtain eye accommodation we perform wavefront tracing on the GPU. Defocus is achieved either by progressive distributed ray tracing of the eye lens (accurate) or by approximate blurring according to the obtained wavefront parameters. While the first variant is best suited guiding lens manufacturers during the design process of lenses, we consider the second approach ideal for giving customers a real-time impression of customized virtual spectacles in eye shops. Additionally, we visualize refractive power and effective astigmatism of incident wavefronts. That allows quality assessment of special purpose lenses such as reading or sport glasses in particular scene environments.

CR Categories: I.3.3 [Computer Graphics]: Three-Dimensional Graphics and Realism—Virtual Reality;

Keywords: simulation of human vision, real-time rendering

*e-mail: matthias.niessner@cs.fau.de
†e-mail: r.sturm@brillenglas.de
‡e-mail: guenther.greiner@cs.fau.de

1 Introduction

Simulating human vision through eyeglasses is important and beneficial for both spectacle lens manufactures and opticians. On the one hand, such a simulation allows verifying geometrical properties of eyeglasses during the lens design process without actually cutting physical glass. On the other hand, customers in eye shops can evaluate customized eyeglasses before giving the production order. In this paper we present a solution that serves both of these purposes. Thereby the main focus of this work lies on progressive addition lenses (**PALs**) due to their high production costs. The simulation process for those glass types is particularly challenging since they consist of relatively complex free-form (typically bicubic B-splines) surfaces. However, we also support standard eyeglass lenses composed only of spheric and toric surfaces.

Our simulation system allows a user in a virtual scene environment to see through an eyeglass and directly observe the simulated image in real-time. In addition, our approach allows visualizing resulting refractive power and effective astigmatism for a given eyeglass, scene environment and eye lens. This is particularly useful for special purpose lenses such as reading or sport glasses.

First, we determine the accommodation of the virtual eye model using wavefront tracing for each pixel of the result image. Thus, the obtained eye accommodation parameters correspond to a real human eye lens while scanning the field of view. Next, we compute defocus either accurately by progressive distributed ray tracing or approximately by filtering with respect to the traced wavefronts and the specific eye lens parameters. In contrast to previous methods we directly employ the original lens geometry (e.g., bicubic B-splines) and thus avoid artifacts caused by intermediate triangulation. Further, all operations such as wavefront and ray tracing are executed on the GPU in order to achieve real-time framerates even on mid-range hardware.

To sum up, we propose a novel system that

- simulates human vision through eyeglasses on the GPU
- supports all common types of spectacles
- uses the original analytic lens geometry
- visualizes resulting defocus and optical properties
- achieves real-time performance

2 Previous Work

Simulation of Human Vision: The first step of human vision simulation is to determine the eye accommodation. Therefore, Mostafawy et al. [1997] introduce the virtual eye that approximates its accommodation by taking only the distance of objects into account. A more elaborate approach to obtain the eye accommodation is wavefront tracing [Kneisly et al. 1964], [Stavroudis 1972], [Mitchell and Hanrahan 1992]. It provides for physically correct results and supports multi-lens systems. Loos et al. [1998] make use of this method within the context of PAL optimization. In addition, they perform distributed ray tracing [Cook et al. 1984] for depth of field effects and optical distortions. Our approach is similar, however, instead of an offline simulation process we focus on real-time image generation on modern GPUs. Kakimoto et al. [2007] precompute wavefront information (i.e., defocus) for every voxel of the scene; they name it a blur field. They displace vertices according to the blur field at runtime and obtain defocus by blending several renderings with distinct displacement seeds together. Their approach works within the rasterization pipeline since they use an environment map for refraction operations. This method can be extended by employing the concept of conoid tracing [Kakimoto et al. 2010] which accelerates the blur field generation. Compared to their approach we do not rely on coarsely discretized precomputed wavefront data. Instead we perform wavefront tracing at runtime and obtain accurate results by taking the analytic lens geometry (e.g., bicubic B-splines) into account.

Barsky [2004] also employs the concept of wavefronts. Wavefront data from human subjects is physically measured and used to render vision-realistic images.

GPU Ray Tracing: Ray tracing [Whitted 1980] is the key requirement for the physically-correct simulation of human vision. One application of ray tracing is the tracing of wavefronts since wavefronts are rays with additional payload such as principal curvatures and directions. Another is distributed ray tracing [Cook et al. 1984] that is the reference method for computing defocus. Recent development in graphics hardware makes GPUs attractive for ray tracing due to their computational capabilities. Aila and Laine [2009] demonstrate how to realize a ray tracer on modern GPUs. NVIDIA employs their method in its GPU ray tracing engine OptiX [Parker et al. 2010]. We use OptiX for our simulation system since it allows customized ray-object intersections. In addition, it provides for fast acceleration structure construction and efficient traversal on the GPU. With OptiX it is also feasible to define rays with customized payload (required for wavefront parameters).

3 Eyeglass Description

A real eyeglass consists of a homogeneous glass-like material with a particular *refraction index* and a specific *glass thickness*. The optical properties of eyeglasses are defined through a front and back lens surface. The side of the eyeglass is obtained by the *glass diameter* and the spectacle frame. However, for our simulation we ignore the frame since it does not contribute to the optical properties of the eyeglass. The front and back surface of the eyeglass are either **spheric**, **toric** or **free-form**. While a spheric surface is defined by a *radius* r (and the glass thickness), a toric surface is given by an *inner radius* r and an *outer radius* R. Typically, free-form lens surface are uniform bicubic B-splines defined by a regular control point grid. Progressive addition lenses (**PAL**) are composed of at least one free-form surface.

Our system directly incorporates the original geometric design that is used to manufacture the lens for subsequent simulation. That allows us to define eyeglasses for all common visual defects by selecting an appropriate front and back face for the lens:

- Hyperopia: spheric + spheric (positive/convex)
- Myopia: spheric + spheric (negative/concave)
- Astigmatism: toric + spheric
- Presbyopia: free-form + spheric/toric

Please note that bifocal lenses (e.g., lenses which are composed of surfaces that have two distinct refraction properties) can be also used to correct Presbyopia. However, nowadays bifocal lenses have been mostly replaced by PALs, which have superior optical properties. Due to their obsolescence we do not examine bifocal lenses explicitly.

4 GPU Ray Tracing and Intersection Tests

For computing both eye accommodation and defocus we need to perform ray tracing. Therefore, we use NVIDIA OptiX [2010] that gives us the flexibility to customize intersection tests for different primitives in a *ray intersection program*. Furthermore, acceleration structures such as kD-trees or BVHs are built according to a user-provided *bounding box program*.

4.1 Ray Primitive Intersections

Since we use triangle meshes to represent scene environments, we perform standard ray-triangle intersections in order to determine ray-scene hit points.

Spheric and toric eyeglass surfaces are handled by considering the respective analytic surface equation. In the case of a sphere we need to solve a quadratic equation that is obtained by putting the ray equation $\vec{r} = \vec{o} + t\vec{d}$ into the implicit form of the sphere $x^2 + y^2 + z^2 = r^2$. The torus is treated in the same way using its implicit form $(x^2 + y^2 + z^2 - r^2 - R^2)^2 + 4R^2(z^2 - r^2) = 0$. In order to find the roots of this degree 4 polynomial, we employ the iterative root finding algorithm proposed by Bairstow [1920]. Note that a lens surface only corresponds to a small section of the sphere or torus. We define the spheric or toric section using an additional cutoff parameter, respectively. In order to optimize primitive bounds we take the cutoff into account during acceleration structure construction; i.e., in the *bounding box program*.

Handling the free-form surfaces is less trivial since they consist of a larger and more complex data set. These lens surfaces are defined by a (potentially non-uniform) bicubic B-spline patch consisting of up to 100×100 control points and a corresponding knot vector. In order to ray trace such a surface efficiently we use the algorithm of Boehm [1980] to convert the spline into multiple Bézier patches in a preprocess. Thus, we obtain tight bounds (in the *bounding box program*) for each Bézier patch due to the convex hull property. These bounds are then used to construct a BVH comprising all patches obtained from the original B-spline of the lens surface. At runtime, if a ray hits a patch's bounding box, we determine an accurate hit point by taking the parametric patch representation into account. Therefore, we represent each ray as the intersection of two planes with normal vectors $\vec{N}_1$ and $\vec{N}_2$ being perpendicular to the ray direction (see [Martin et al. 2000], [Geimer and Abert 2005], [Abert et al. 2006]). In order to find the intersection point between the ray and the parametric surface patch $S(u,v)$, we determine the roots of

$$R(u,v) = \begin{pmatrix} \vec{N}_1 \cdot (S(u,v) - \vec{o}) \\ \vec{N}_2 \cdot (S(u,v) - \vec{o}) \end{pmatrix}$$

where $\vec{o}$ is the ray origin. Therefore, we employ the iterative Newton method using the domain center of the respective patch (i.e., $(u,v)^T = (0.5, 0.5)^T$) as a start value:

$$\begin{pmatrix} u_{n+1} \\ v_{n+1} \end{pmatrix} = \begin{pmatrix} u_n \\ v_n \end{pmatrix} - J(u_n, v_n)^{-1} \cdot R(u_n, v_n),$$

where J is the Jacobian of R, given by

$$J(u,v) = \begin{pmatrix} \vec{N}_1 \cdot S_u(u,v) & \vec{N}_1 \cdot S_v(u,v) \\ \vec{N}_2 \cdot S_u(u,v) & \vec{N}_2 \cdot S_v(u,v) \end{pmatrix}.$$

Due to numerical issues, previous methods often have problems when rays hit parametric surfaces in a tangential manner. Thus, they often require a large number of Newton iteration steps. Our approach does not suffer from these problems since rays typically hit eyeglass surfaces almost orthogonally. Hence, we require no more than 3 iteration steps in order to obtain accurate hit points.

One of the key features of ray tracing the analytic eyeglass surfaces is that we are able to directly obtain first and second order derivatives. That allows us not only to compute surface normals (required for refraction), but also to determine principal curvatures and their directions using the first and second fundamental form (see [Do Carmo 1976]).

$$\mathrm{I} = \begin{pmatrix} S_u(u,v) \cdot S_u(u,v) & S_u(u,v) \cdot S_v(u,v) \\ S_u(u,v) \cdot S_v(u,v) & S_v(u,v) \cdot S_v(u,v) \end{pmatrix}$$

$$\mathrm{II} = \begin{pmatrix} S_{uu}(u,v) \cdot N(u,v) & S_{uv}(u,v) \cdot N(u,v) \\ S_{uv}(u,v) \cdot N(u,v) & S_{vv}(u,v) \cdot N(u,v) \end{pmatrix}$$

The principal curvatures κ_1^S and κ_2^S of the surface are given by $\kappa_{1,2}^S = H \pm \sqrt{(H^2 - K)}$, where $K = \frac{\det(\mathrm{II})}{\det(\mathrm{I})}$ and $H = 0.5 \cdot \mathrm{trace}(\mathrm{I}^{-1}\mathrm{II})$ are Gaussian and mean curvature, respectively. The corresponding directions of principal curvature are the columns of $\mathrm{II} - \kappa_i \cdot \mathrm{I}$. More precisely, these are their coefficients with respect to the tangent vectors $\mathrm{span}\{S_u(u,v), S_v(u,v)\} \in \mathbb{R}^3$. We require both principal curvatures and directions in order to compute the accommodation of the eye lens (see Section 5).

Please note that once a hit point on a lens surface is determined, a refraction ray is constructed according to the surface normal and refraction index. Since OptiX supports recursive ray tracing, secondary rays can be directly cast in the *closest hit program* (this program is executed automatically once a hit point is found).

4.2 Scene Graph

The scene graph features we require to represent virtual environments are provided by OptiX [Parker et al. 2010]. It is crucial for our system to move around freely in a scene at runtime and render eyeglasses in front of the camera independently at the same time. Therefore, eyeglass and scene geometry are represented as two scene graph nodes that are defined as *geometry groups* in OptiX. Each of them has a separate acceleration structure that includes all respective child nodes. While the scene geometry node is directly linked to the root node, root and eyeglass node are linked together with a transform node that allows operations to be applied on the eyeglass exclusively. On top of the scene graph we build a global acceleration contained by the root. It is updated every frame due to the dynamic transform operation of the eyeglass. The eyeglass node itself has two child nodes containing the front and back lens surface (those can be either spheric, toric, or free-form). Within OptiX these are realized as *geometry instances* and thus directly bound to the underlying geometry. Since a scene typically consists of multiple objects, the scene environment node has a corresponding number of children (those are also geometry instances). In order achieve to real-time performance we consider scene objects to be static. Thus, we build a highly-efficient kD-tree including all scene geometry (but not eyeglass lens surfaces). This is shared by all scene objects without the need of costly dynamic updates. In the end, the design of our scene graph allows eyeglass (corresponds to eye movement) and the scene geometry (corresponds to head or user motion) to be moved independently at runtime. An overview of our scene graph is shown in Figure 2.

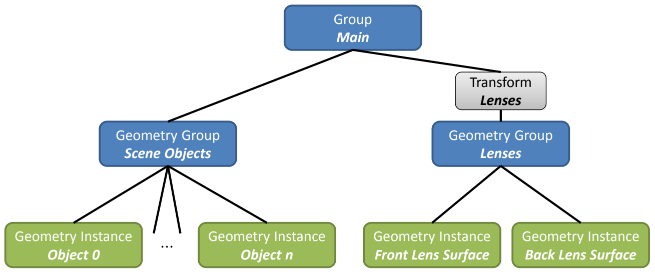

Figure 2: *Scene graph of our rendering system realized in OptiX: the lens transform is applied at runtime and thus allows dynamic transformation of the eyeglasses; i.e., an user is allowed to move its head, eyes or eyeglasses arbitrarily.*

5 Eye Accommodation Using Wavefront Tracing

In reality, if the eye focuses on a particular object, the eye lens accommodates accordingly. In order to simulate this effect, we compute the accommodation of the virtual eye lens accordingly. Therefore, we employ wavefront tracing [1972] similar as proposed by Loos et al. [1998]. A wavefront along a ray can be described by its normal $\vec{n}$, that is equivalent to the direction $\vec{d}$ of the corresponding ray, the principal curvatures κ_1, κ_2 and the principal directions $\vec{e}_1$, $\vec{e}_2$. In the following sections we describe wavefront transformations and wavefront tracing.

5.1 Wavefront Spread in Homogeneous Media

A wavefront passing through homogeneous media has constant principal directions; thus $\vec{e}_1$ and $\vec{e}_2$ remain unchanged. However, its principal curvatures κ_1, κ_2 are transformed according to the ray length t:

$$\kappa_i' = \frac{\kappa_i}{1 - t \cdot \kappa_i}$$

Since this transformation only depends on the ray parameter t, we directly transform the wavefront in the respective *ray intersection program*.

5.2 Wavefront Transformation at Optical Boundaries

In order to transform a wavefront W at an optical boundary, we determine the normal $\vec{N}^s$, principal curvatures κ_1^s, κ_2^s as well as the principal directions $\vec{e}^s_1$, $\vec{e}^s_2$ of the transition point of the lens surface S. Since we use analytic lens surfaces, we obtain these parameters using differential geometry formulae (see Section 3).

At optical boundaries wavefronts are transformed according to the two corresponding indices of refraction η_1 and η_2. Then, the normal $\vec{N}'$ of the transformed wavefront W' is a linear combination $\vec{N}' = \mu\vec{N} + \gamma\vec{N}^s$ with $\vec{N}$ being the normal before the refraction and $\vec{N}^s$ the corresponding surface normal. With Snell's law we obtain $\mu = \frac{\eta_1}{\eta_2}$ and $\gamma = -\mu\cos\phi + \cos\phi'$ with ϕ being the angle of incidence and ϕ' being the angle of refraction. Next, we choose a common unit tangent vector $\vec{\xi}$ of the wavefront W and the surface S as well as another unit tangent vector $\vec{\eta}$ of S that is orthogonal to $\vec{\xi}$. Then we compute the angle θ between $\vec{\xi}$ and the first principal direction $\vec{e}_1$ on W, i.e.,

$$\vec{\xi} = \cos\theta\vec{e}_1 - \sin\theta\vec{e}_2, \quad \vec{\eta} = \sin\theta\vec{e}_1 + \cos\theta\vec{e}_2.$$

The curvatures κ_ξ and κ_η on W in direction $\vec{\xi}$ and $\vec{\eta}$ are then given by Euler's formula [1972]:

$$\begin{aligned} \kappa_\xi &= \kappa_1\cos^2\theta + \kappa_2\sin^2\theta \\ \kappa_\eta &= \kappa_1\sin^2\theta + \kappa_2\cos^2\theta \\ \kappa_{\xi\eta} &= (\kappa_1 - \kappa_2)\cos\theta\sin\theta \end{aligned}$$

We compute the directional curvatures κ^s_ξ, κ^s_η and $\kappa^s_{\xi\eta}$ on S analogously. Now, the curvatures κ_ξ and κ^s_ξ share the same direction (this also applies to κ_η and κ^s_η). This allows us to compute the curvatures for these directions after refraction:

$$\begin{aligned} \kappa'_\xi &= \mu\kappa_\xi + \gamma\kappa^s_\xi \\ \kappa'_\eta &= \mu\kappa_\eta\frac{\cos^2\phi}{\cos^2\phi'} + \gamma\kappa^s_\eta\frac{1}{\cos^2\phi'} \\ \kappa'_{\xi\eta} &= \mu\kappa_{\xi\eta}\frac{\cos\phi}{\cos\phi'} + \gamma\kappa^s_{\xi\eta}\frac{1}{\cos\phi'} \end{aligned}$$

In order to represent the wavefront after refraction we invert the Euler formula to obtain the transformed principal curvatures κ'_1 and κ'_2:

$$\begin{aligned} \theta' &= \tfrac{1}{2}\tan^{-1}\frac{2\kappa'_{\xi\eta}}{\kappa'_\xi - \kappa'_\eta} \\ \kappa'_1 &= \tfrac{1}{2}(\kappa'_\xi + \kappa'_\eta + \frac{\kappa'_\xi - \kappa'_\eta}{\cos 2\theta'}) \\ \kappa'_2 &= \tfrac{1}{2}(\kappa'_\xi + \kappa'_\eta - \frac{\kappa'_\xi - \kappa'_\eta}{\cos 2\theta'}) \end{aligned}$$

With θ' we determine $\vec{e}'_1$ and $\vec{e}'_2$ of W' accordingly.

5.3 Wavefront Tracing

For a single point on the retina (i.e., a pixel), wavefront tracing involves two steps. First, we trace a ray from the center of the respective pixel through the center of the pupil (i.e., eye lens) using the thin lens model [Hecht and Zajak 2002]. This ray will be refracted once at the back surface and once at the front surface of the eyeglass lens until a scene object is hit (see Section 4.1). Second, once a scene hitpoint is determined, a spheric wavefront is initiated at that point and backtraced along the ray.

Since the wavefront is spheric when it is initiated, $\vec{e}_1$, $\vec{e}_2$ are arbitrary. However, we ensure that $\vec{e}_1 \cdot \vec{e}_2 = \vec{e}_1 \cdot \vec{n} = \vec{e}_2 \cdot \vec{n} = 0$. When the wavefront hits the front surface of the eyeglass the principal curvatures are given by the wavefront sphere radius; thus $\kappa_1 = \kappa_2 = \frac{1}{t}$. At that point the wavefront is transformed using the transition rules for wavefronts at optical boundaries (see Section 5.2). In our simulation system, that uses OptiX, this is applied in the *closest hit program* of the lens surface. The transformed wavefront is then traced through the eyeglass until the back surface is hit. Thereby, the wavefront sustains another transformation according to its spread in homogenous media (see Section 5.1). When the wavefront hits the back surface of the eyeglass the wavefront is transformed in the same way as for the front surface in the respective *closest hit program.* After that the wavefront is further traced and passes the eye lens where it is not transformed due to the thin lens property, and eventually hits the retina. However, in order to compensate for the spread in homogeneous media, we apply the respective lens transform. An overview of the wavefront route and the corresponding transform operations is shown in Figure 3.

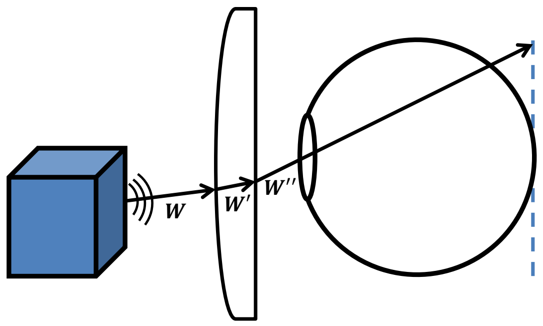

Figure 3: *Wavefront tracing for an eye-eyeglass-object setup: once the ray-object hitpoint is determined, a spheric wavefront is initiated at the intersection point; then the wavefront is backtraced along the ray. Thereby, wavefront parameters are updated according to the wavefront transformations for homogeneous media and optical boundaries.*

5.4 Eye Model and Accommodation

Once we obtain the parameters of the wavefront that hits the eye lens we need to employ an appropriate eye model in order to determine its accommodation. We assume a spherical eye with diameter l and a lens aperture size of a. Typically l is about $25mm$ and a is about $3.5mm$, however, we can easily adjust these parameters corresponding to a certain user and/or a particular lighting condition that affects a. They eye lens itself is characterized by two refractive powers $Q_{\min,1}$ and $Q_{\min,2}$ (corresponding to the respective principal directions of the lens) in relaxed state (i.e., no accommodation) and the orientation of the corresponding focal lines l_1 and l_2. Please note that for a non-astigmatic eye $Q_{\min,1} = Q_{\min,2}$ and thus the two focal planes f_1 and f_2 are identical. In addition, the eye can accommodate by ΔQ to achieve an overall refraction Q:

$$Q = \frac{Q_{\min,1} + Q_{\min,2}}{2} + \Delta Q$$

For given refractive powers $Q_{\min,1}$ and $Q_{\min,2}$ we employ the mean curvature transformation by a thin lens with refractive power P (see [Loos et al. 1998])

$$\frac{\kappa'_1 + \kappa'_2}{2} = \frac{\kappa_1 + \kappa_2}{2} + P$$

we derive ΔQ based on the incident wavefront W'' with curvatures κ''_1 and κ''_2 at the eye lens

$$\Delta Q = \frac{1}{l} - \frac{\kappa''_1 + \kappa''_2}{2} - \frac{Q_{\min,1} + Q_{\min,2}}{2}$$

Further, we clamp ΔQ according to the maximum eye accommodation $\Delta_{\max}$ of a particular user: $\Delta Q = \max(\min(\Delta Q, \Delta_{\max}), 0)$.

If ΔQ was not in the interval of $[0, \Delta_{\max}]$ before, the specific eye lens was not capable of accommodating properly causing a visual defect.

6 Defocus Computation and Visualization

In this section we show how to visualize the results of wavefront tracing (see Section 5). Further, we demonstrate how to use the wavefront parameters in order to compute defocus for a particular eye. This is achieved either by distributed ray tracing or approximate blurring.

6.1 Visualizing Power of Refraction and Effective Astigmatism

In order to give a meaningful statement about the visual quality of spectacles we visualize power of refraction P and effective astigmatism A of the eyeglass (see Figure 4). However, it is even more important knowing these parameters for the wavefronts incident at the eye lens for a particular scene environment. That is ideal suited for validating the properties of glasses designed for specific purposes; e.g., sport glasses, reading glasses etc.. Wavefront tracing allows us to directly determine both power of refraction P and effective astigmatism A given by the wavefront W'' at the eye lens. Those are typically measured in *Diopter* (m^{-1}) and are given by the wavefront's principal curvatures κ_1'' and κ_2'':

$$\begin{aligned} P = &\quad \frac{\kappa_1''+\kappa_2''}{2} \\ A = &\quad \kappa_1'' - \kappa_2'' \end{aligned}$$

Since we trace a wavefront for every point on the retina (i.e., every pixel), we visualize P and A as a 2D image, respectively. Therefore, we employ the *HSV* color model and map these scalar to the *hue* and set *saturation* as well as *value* to 1. An example of this visualization is shown in Figure 5.

6.2 Defocus Computation using Distributed Ray Tracing

In Section 5.4 we determine the eye accommodation ΔQ by considering the wavefronts W'' at the eye lens. It is important to note that we obtain a different ΔQ for every pixel on the screen. That corresponds to scanning the field of view and accommodation for every discrete viewing point (i.e., pixel).

Now, we use this information in order to compute defocus. Therefore, we employ distributed ray tracing [Cook et al. 1984] to simulate the eye lens. We trace multiple sample rays per pixel to determine the final color of an image point. In order to maintain interactivity, we perform progressive distributed ray tracing: only a single sample ray is traced per frame and pixel; samples are distributed temporarily over multiple frames.

Non-astigmatic eye: In the case of an non-astigmatic eye, the focal plane f is given by the lens diameter l and the power of refraction of the eye Q (cf. Section 5.4): $f = \frac{2}{\kappa_1''+\kappa_2''}$. Virtual object points $\vec{x}$ are then obtained by rays through the center of the eye lens due to the thin lens model. Then we distribute samples $\vec{s}_i$ on the eye lens with respect to its aperture size a and construct corresponding sample rays $\vec{r}_i = \vec{s}_i + \lambda(\vec{x} - \vec{s}_i)$. The final color is then given by the average color of all sample rays $\vec{r}_i$. If all rays hit the same scene point, the result is sharp; otherwise it is blurred.

Astigmatic eye: In the case of an astigmatic eye, the lens is specified by two distinct (minimum) powers of refraction $Q_{\min,1}$ and $Q_{\min,2}$ and the orientation of the focal lines l_1 and l_2. Based on ΔQ this leads to two separate focal planes f_1 and f_2 and two virtual object points $\vec{x}_1$ and $\vec{x}_2$. The two points $\vec{x}_1$ and $\vec{x}_2$ then define the focal lines l_1 and l_2 according to the given orientation. We then distribute samples $\vec{s}_i$ on the eye lens with respect to its aperture size a. Now, sample rays are constructed, such that they have a $\vec{s}_i$ as origin and intersect both focal lines l_1 and l_2. The result of all sample rays determines the final color of a pixel. Please note that the astigmatic case generalizes the non-astigmatic case; that is given if $Q_{\min,1} = Q_{\min,2}$ and thus $f_1 = f_2$.

6.3 Defocus Computation using Approximate Blurring

In order to achieve real-time framerates we employ an approximate depth of field algorithm [Riguer et al. 2003] (a survey of some real-time depth of field techniques is provided in [Demers 2004]). Therefore, we add a color parameter to the wavefront. This allows obtaining wavefront and color information by tracing a single ray per pixel. At the eye lens we transform the wavefront (as shown in Section 5.2) into the space of the principal directions of the eye lens using Euler's curvature formula [1972]. The principal directions of the eye lens are given by the orientation of its focal lines l_1 and l_2. This provides for curvatures κ_{l_1} and κ_{l_2} which define the radii r_1 and r_2 of the *ellipse of confusion*. Finally, we use another OptiX kernel that applies defocus to the result image (i.e., the obtained color values) considering the radii and the orientation or focal lines. There we distribute filter taps (number depends on the ellipse size) within the ellipse of confusion and weight them according to a multivariate Gaussian distribution. Additionally, we eliminate color leaking of sharp objects by weighting filter taps according to their depth and blur values. Note that in the non-astigmatic case the ellipse of confusion becomes a circle with $r = r_1 = r_2$.

Compared with the accurate variant of Section 6.2, this only provides an approximate solution. However, the performance is superior since only a single primary ray is required per pixel. While the defocus computation for a particular pixel is approximate, the location of defocus is still physically correct since it is based on wavefront tracing. We consider this particularly useful for an interactive eyeglass simulator that is designed to assess the quality of eyeglasses. There it is more important being able to locate defocus, rather than to compute its exact value.

7 Results

Our implementation uses NVIDIA OptiX 2.5.1 running on Windows 7. We tested our implementation on a NVIDA GTX 480 that is a two-year-old midrange GPU. Performance measurements are provided in frames per second and account for all runtime overhead including GUI display. In order to test our simulation system we use eyeglasses that are composed of spheric, toric and free-form surfaces. While spheric and toric surfaces have a trivial definition, we use an uniform B-spline surface that consists of 50×50 control points to describe a free-form surface. The optical properties of such a B-spline eyeglass, namely effective astigmatism and refractive power, are shown in Figure 4. Note that these parameters affect wavefront tracing, however, the wavefront parameters at the eye lens (see Section 6.1) are a distinct measure since they vary with the scene environment.

Wavefront tracing and visualization: For the wavefront tracing process we shoot a single primary ray per pixel. Each ray will then be refracted at both the back and front surface of the eyeglass. Thus, three rays need to be traced per pixel before hitting an object. A visualization of the wavefront parameters (effective astigmatism and power of refraction) at the eye lens is depicted in Figure 5. These parameters vary within a scene environment and provide information about the respective scene and lens setup. This is particularly

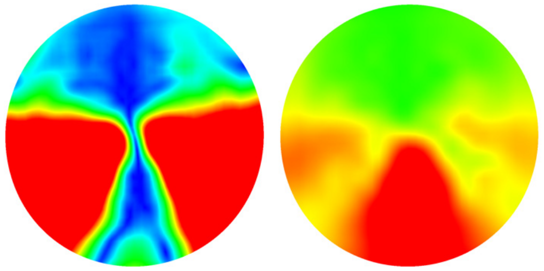

Figure 4: *Effective astigmatism (left) and refractive power (right) of a typical progressive addition lens. The section for near vision is located at the bottom center and is relatively small; the far vision area is larger and located at top of the lens.*

useful, when designing special purpose eyeglasses such as reading or sport glasses.

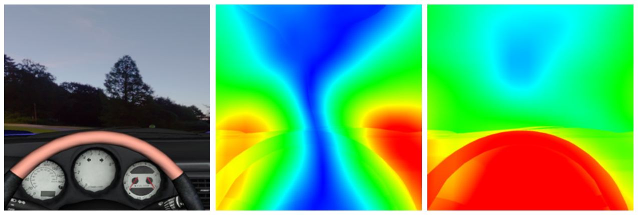

Figure 5: *From left to right: view through progressive addition lens using a pinhole camera; visualization of the effective astigmatism of the wavefront at the eye lens; visualization of the refractive power of the same wavefront.*

Defocus computation and simulation results: Defocus is obtained either exactly by progressive distributed ray tracing or approximately by screen space filtering (see Section 6). Figure 6 shows a comparison between those two approaches where an eye model with limited accommodation as well as a spheric eyeglass is used. Both variants achieve similar visual quality, however, the exact approach requires multiple frames in order to sample the eye lens.

The results of our eyeglass simulation are shown in Figure 1. Therefore, we use a progressive addition lens as an example for a particularly challenging eyeglass. In addition, we limit the accommodation capability of the virtual eye lens. The two images depict near and far vision examples, respectively. This corresponds to real eye movement where particular regions of interest are being looked at. The usage of progressive addition lens demonstrates that focus and defocus are depending on both, astigmatism and the ability of the eye to accommodate appropriately. It is interesting to note that large areas of the result images are blurred, even though in reality the human brain manages to compensate for. We have also tested our simulation system for a variety of simpler eyeglasses (such as spheric, toric), however, we omitted the results due to reason of space.

Performance: The performance for our simulation system is shown in Table 1. In order to obtain the timings, we use the car scene environment of Figure 1 that consists of $300K$ triangles. For the B-spline lens surface, we employ the same lens design as shown in Figure 4. Performance measurements are provided in frames per second and resulting images have a resolution of 1024×1024 pixels. Since our approach is mainly based on ray tracing, the performance scales linearly with respect to the screen resolution. Each row of Table 1 corresponds to a particular eyeglass design that consists of two surfaces which are either spheric, toric, or free-form. Each column of the table depicts the performance of a specific rendering type: *ADOF* is computing the eye accommodation using wavefront tracing and approximating defocus by screen space filtering. Since the visual quality of ADOF is similar to that of distributed ray tracing (see Figure 6), we consider ADOF the default simulation variant. *VIS* refers to wavefront tracing and subsequent visualization of the wavefront parameters (see Figure 5). *PIN* is ray tracing the scene using a pinhole camera model, with neither wavefront tracing nor defocus computation. While this is an artificial lens design, it provides an upper bound for the performance. *DOF* stands for rendering a single frame using progressive distributed ray tracing. In order to obtain the correct defocus multiple frames need to be rendered. However, it is possible to move around freely in real-time. Defocus computation starts when user interaction stops. In the end all variants achieve real-time framerates, for all possible eyeglass compositions. Please note that these numbers correspond to mid-range hardware and our simulation would be considerably faster on high-end GPUs.

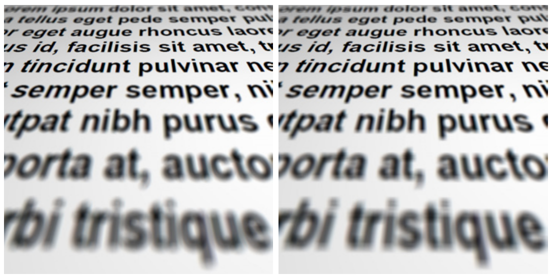

Figure 6: *View on a text using a (positive) spheric eyeglass. The eye that has limited accommodation; thus, only mid-range distances are in focus. The right-hand image is rendered using distributed ray tracing and the left-hand one is obtained using the approximate depth of field variant. While the quality of both images is similar, the accurate variant requires considerably more computational effort.*

Performance (fps)	ADOF	VIS	PIN	DOF
No Lens	**66.5**	70.5	84.2	83.5*
Sphere/Sphere	**42.5**	45.6	56.1	55.5*
Sphere/Torus	**35.5**	38.7	43.5	43.1*
B-spline/Sphere	**31.7**	32.5	38.2	37.5*
B-spline/Torus	**28.1**	28.8	31.6	30.8*

Table 1: *Performance measurements in frames per second for different eyeglass compositions. Eyeglasses consist of two surfaces which are either spheric, toric, or free-form. Timings are provided for computing eye accommodation and approximate defocus computation (ADOF); computation and visualization of incident wavefronts at the eye lens (VIS); ray tracing the scene with a pinhole camera model; and progressive distributed ray tracing. Note that ADOF is our default method and DOF requires multiple frames to obtain defocus.*

8 Conclusion and Future Work

We have presented a novel system that allows real-time simulation of human vision through all common types of eyeglasses. Since our approach works entirely on the GPU, we achieve real-time framerates which allows a user to interactively navigate in virtual environments. In contrast to previous methods we directly ray trace analytic lens geometry (e.g., bicubic B-splines) without intermediate triangulation. Thus, we obtain physically correct results and keep memory footprint low. To sum up, our system is ideally suited for assisting during a lens design process as well as providing an interactive feedback tool for eye shop customers.

In the future we would like to incorporate contact lenses into our simulation system. This is particularly challenging since contact lenses are very thin which makes the ray tracing process difficult due to numerical issues. In addition, our system could be integrated into virtual reality glasses. In combination with head and eye tracking this could enhance the quality of visual impression and improve upon realism.

Acknowledgments

This project is part of a collaboration with *Rupp + Hubrach* (premium spectacle lens manufacturer). We would like to thank for continuous support.

References

ABERT, O., GEIMER, M., AND MULLER, S. 2006. Direct and fast ray tracing of nurbs surfaces. In *Interactive Ray Tracing 2006, IEEE Symposium on*, Ieee, 161–168.

AILA, T., AND LAINE, S. 2009. Understanding the efficiency of ray traversal on gpus. In *Proc. High-Performance Graphics 2009*, 145–149.

BAIRSTOW, L. 1920. *Applied aerodynamics*. Longmans, Green and co.

BARSKY, B. 2004. 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*, ACM, 73–81.

BOEHM, W. 1980. Inserting new knots into b-spline curves. *Computer-Aided Design 12*, 4, 199–201.

COOK, R., PORTER, T., AND CARPENTER, L. 1984. Distributed ray tracing. In *ACM SIGGRAPH Computer Graphics*, vol. 18, ACM, 137–145.

DEMERS, J. 2004. Depth of field: A survey of techniques. *GPU Gems 1*, 375–390.

DO CARMO, M. 1976. *Differential geometry of curves and surfaces*, vol. 1. Prentice-Hall.

GEIMER, M., AND ABERT, O. 2005. Interactive ray tracing of trimmed bicubic bézier surfaces without triangulation. *WSCG (Full Papers)*, 71–78.

HECHT, E., AND ZAJAK, A., 2002. Optics.

KAKIMOTO, M., TATSUKAWA, T., MUKAI, Y., AND NISHITA, T. 2007. Interactive simulation of the human eye depth of field and its correction by spectacle lenses. In *Computer Graphics Forum*, vol. 26, Wiley Online Library, 627–636.

KAKIMOTO, M., TATSUKAWA, T., AND NISHITA, T. 2010. An eyeglass simulator using conoid tracing. In *Computer Graphics Forum*, vol. 29, Wiley Online Library, 2427–2437.

KNEISLY, J., ET AL. 1964. Local curvature of wavefronts in an optical system. *Journal of the Optical Society of America (1917-1983) 54*, 229.

LOOS, J., SLUSALLEK, P., AND SEIDEL, H. 1998. Using wavefront tracing for the visualization and optimization of progressive lenses. In *Computer Graphics Forum*, vol. 17, Citeseer, 255–266.

MARTIN, W., COHEN, E., FISH, R., AND SHIRLEY, P. 2000. Practical ray tracing of trimmed nurbs surfaces. *Journal of Graphics Tools 5*, 1, 27–52.

MITCHELL, D., AND HANRAHAN, P. 1992. Illumination from curved reflectors. *ACM SIGGRAPH Computer Graphics 26*, 2, 283–291.

MOSTAFAWY, S., KERMANI, O., AND LUBATSCHOWSKI, H. 1997. Virtual eye: retinal image visualization of the human eye. *Computer Graphics and Applications, IEEE 17*, 1, 8–12.

PARKER, S., BIGLER, J., DIETRICH, A., FRIEDRICH, H., HOBEROCK, J., LUEBKE, D., MCALLISTER, D., MCGUIRE, M., MORLEY, K., ROBISON, A., ET AL. 2010. Optix: A general purpose ray tracing engine. *ACM Transactions on Graphics (TOG) 29*, 4, 66.

RIGUER, G., TATARCHUK, N., AND ISIDORO, J. 2003. Real-time depth of field simulation. *ShaderX2: Shader Programming Tips and Tricks with DirectX 9*, 529–556.

STAVROUDIS, O. 1972. The optics of rays, wavefronts, and caustics. Tech. rep., DTIC Document.

WHITTED, T. 1980. An improved illumination model for shaded display. *Communications of the ACM 23*, 6, 343–349.