Perceptual Rasterization for Head-mounted Display Image Synthesis

SEBASTIAN FRISTON, University College London
TOBIAS RITSCHEL, University College London
ANTHONY STEED, University College London

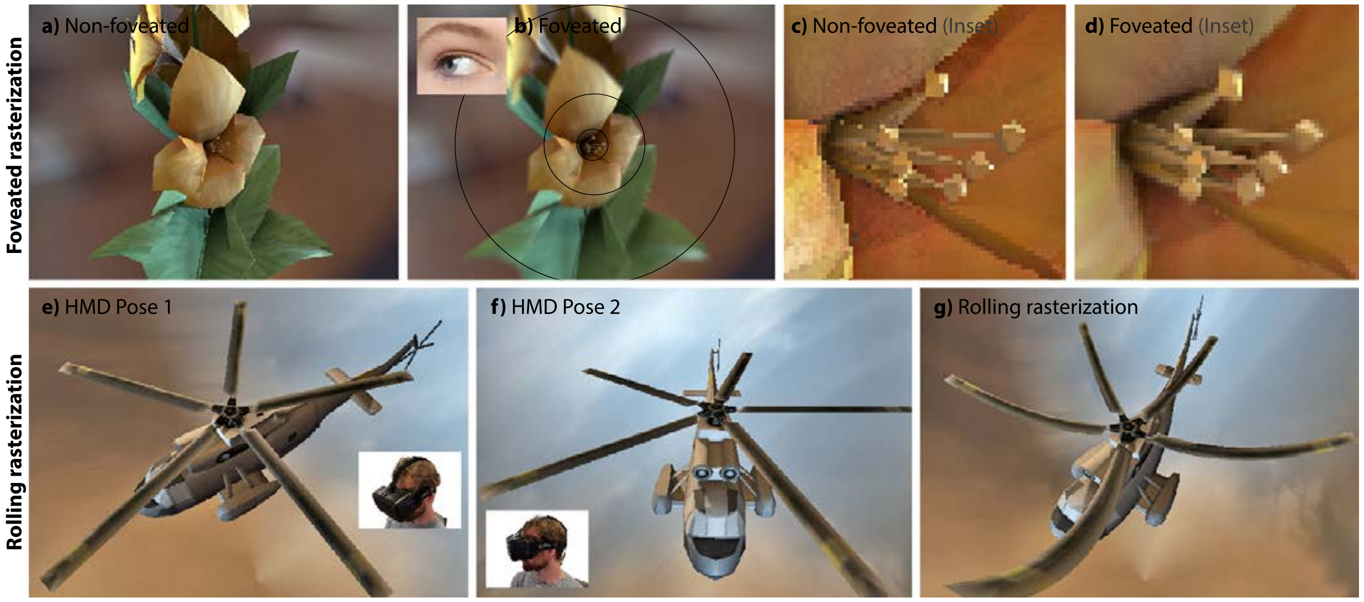

Fig. 1. Perceptual rasterization is a generalization of classic rasterization to the requirements of HMDs such as foveation *(top row)* and rolling image formation *(bottom row)*. On an HMD, most pixels appear in the periphery **(a)**. We rasterize images with continuously-varying pixel density **(b)**. A zoom of the the foveated area shows how a common same-shading-effort image has aliasing **(c)**, while our result benefits from higher pixel density, resulting in super-sampling **(d)**. In common rasterization, each pixel on the display is effectively sampled at the same simulation time ($t = 0$ for the first frame **(e)** and $t = 1$ for the next frame **(f)**). When displayed on a "rolling" HMD display, where pixels are illuminated at different points in time, latency is introduced: the rightmost pixel is outdated by ca. 16 ms. Our rolling rasterization **(g)** allows spatially-varying time: starting at $t = 0$ on the left of the image and increasing to 1 on the right.

We suggest a rasterization pipeline tailored towards the needs of HMDs, where latency and field-of-view requirements pose new challenges beyond those of traditional desktop displays. Instead of image warping for low latency, or using multiple passes for foveation, we show how both can be produced directly in a single perceptual rasterization pass. We do this with per-fragment ray-casting. This is enabled by derivations of tight space-time-fovea pixel bounds, introducing just enough flexibility for the requisite geometric tests, but retaining most of the simplicity and efficiency of the traditional rasterizaton pipeline. To produce foveated images, we rasterize to an image with spatially varying pixel density. To compensate for latency, we extend the image formation model to directly produce "rolling" images where the time at each pixel depends on its display location. Our approach overcomes limitations of warping with respect to disocclusions, object motion and view-dependent shading, as well as geometric aliasing artifacts in other foveated rendering techniques. A set of perceptual user studies demonstrates the efficacy of our approach.

CCS Concepts: •**Computing methodologies → Rasterization;** *Perception; Virtual reality;*

ACM Reference format:
Sebastian Friston, Tobias Ritschel, and Anthony Steed. 2019. Perceptual Rasterization for Head-mounted Display Image Synthesis. *ACM Trans. Graph.* 38, 4, Article 97 (January 2019), 14 pages.
DOI: 10.1145/3306346.3323033

1 INTRODUCTION

The use cases of HMDs have requirements beyond those of typical desktop display-based systems. Completely subsuming the user's vision, the HMD and system driving it must maintain low and predictable latency to facilitate a sense of agency and avoid serious negative consequences such as breaks-in-presence [Slater 2002], simulator sickness [Buker et al. 2012], and reduced performance [Ellis et al. 1999]. This challenge is exacerbated by other HMD characteristics, such as high Field-of-View (FOV) and resolution. Further, as human vision has varying spatial resolution with a rapid fall-off in the periphery, much of this computational effort is wasted.

Ray-tracing could cast more rays to the foveal area (foveation) and update the view parameters during image generation (low latency).

Regrettably, ray-tracing remains too slow in large and dynamic scenes. Traditional rasterization efficiently draws an image, but with uniform detail. It does not take advantage of how that image will be perceived. Here, we suggest *perceptual rasterization* that retains most of the efficiency of traditional rasterization, but has additional optimizations that are especially beneficial for HMDs: low-latency and foveation.

This is achieved by generalizing common OpenGL-style rasterization. Our foveated rasterization can work with HMDs that provide eye-tracking data, such as the FOVE [2018], allowing rasterization into a framebuffer with a non-constant pixel density that peaks at the fovea. Our rolling rasterization gives every column of pixels a different time and can be used on HMDs with rolling displays, such as the Oculus Rift DK2, that illuminate different spatial locations at different times. The techniques can be used together.

After discussing previous work (Sec. 2), we will, describe our novel perceptual rasterization pipeline (Sec. 3) before presenting the specific time, space and retinal bounds in Sec. 4. In Sec. 5 we present image results and analysis and in Sec. 6 we present four user studies that demonstrate the efficacy of perceptual rasterization.

2 PREVIOUS WORK

Foveated rendering. The wide FOVs (100 degrees and more) found in current HMDs [FOVE 2018; Patney et al. 2016; Toth et al. 2016; Weier et al. 2017] require higher resolutions and therefore increasing amounts of memory and bandwidth on the GPU. At the same time, only a small percentage of the screen falls onto the fovea, where the highest resolution is required. This makes foveated rendering particularly important for HMDs. In-HMD eye tracking [FOVE 2018; Stengel et al. 2015] is required to know the fovea's location.

Guenter et al. [2012] demonstrate a working end-to-end foveated system based on rasterization. To achieve foveation, they rasterize in multiple passes (three in their example) to individual images with different but uniform pixel densities. We also use rasterization, but into an image with continuously varying pixel density and in a single pass. The work of Patney et al. [2016] applies blur and contrast enhancement to the periphery to hide artifacts. In doing so, they can further reduce the size of the highest resolution foveal region without becoming noticeable. Reducing shading in the periphery is discussed by He et al. [2014]. However, this does not increase pixel density in the fovea, whereas our approach provides substantial super-sampling of both shading and geometry. When using light field display with focus cues, foveated rendering is increasingly important [Sun et al. 2017]. Kernelized foveated rendering [Meng et al. 2018] first rasterizes a G-buffer at foveal resolution, then re-samples it to the log-polar domain. It is shaded in this domain before being re-sampled again to the display. This achieves high shading and geometry rates, while also supporting the same accurate, physiologicaly-motivated and peaky foveation functions we use. The bottleneck however remains in generating the initial, full-resolution G-buffer, which we avoid. Hypothetically, Lens-Matched Shading (LMS) [Nvidia 2017] could be modified to realize foveated rendering, as it allows varying the pixel density across the image. Originally devised to compensate modest and linear changes in pixel density due to the optics of an HMD, we will show it is less suited to perform foveated rendering as the exponential foveation function is not well-approximated using a linear function as required by LMS.

Display latency. In Virtual Reality (VR) systems to date, an important delay that contributes to the end-to-end latency is the interval $[t_s, t_e]$ during which a pixel will be displayed. The longer the interval, the more "outdated" a stimulus will become: if each pixel holds a constant value for 1/60 of a second, at the end of the interval t_e the image may deviate significantly from the ideal representation of the state of the virtual world at the time it was rendered (at or before t_s). In combination with head or eye motion, this leads to hold-type blur [Didyk et al. 2010; Sluyterman 2006].

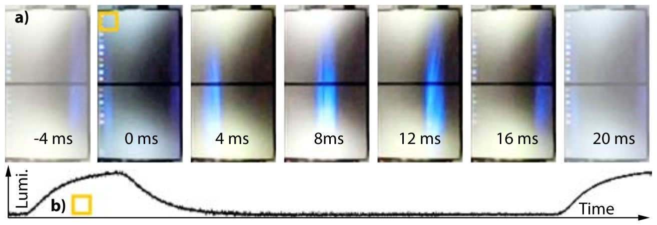

Fig. 2. **a)** Seven frames (24 ms) high-speed capture (Casio Exilim EX-ZR1000) of an HDK 2 HMD (twin) display. Specific locations are illuminated *(blue)* at specific points in time. **b)** Time-varying illumination of a 4 mm band of an Oculus DK2 display captured with a photodiode and a PicoScope 6402B. The yellow box locates the display pixel at which the illumination is captured. We see, that different spatial areas of the display are illuminated at different points of time during the display interval (0, 16), i. e., it is rolling.

To compensate for these negative effects, designers use displays with increasing refresh rates, and lower persistence. Increased refresh rates reduce apparent latency by limiting the maximum age of a given pixel. Low persistence displays illuminate the screen for a time far below the refresh period of the display. This reduces artifacts such as blur. Some of these low persistence displays use a "global scan", in which the entire display is illuminated at once. These have two complications: the display is much darker, and global changes in brightness can produce noticeable flicker. Low brightness is a relatively minor issue for HMDs because the user's vision can adapt. However flicker will be very noticeable, as the human ability to detect flicker is stronger if the target is large (the Granit-Harper [1930] law). An alternative low persistence display technology behaves similarly to traditional Cathode Ray Tubes (CRTs). That is, pixels are illuminated for a short period as they are updated. We consider such displays to have a "rolling scan" (Fig. 2). Drawbacks and benefits of such a display are discussed by Sluyterman [2006]. They exhibit less flicker (as the target is smaller [Granit and Harper 1930]) while remaining resistant to blur. Both global and rolling scan displays will show outdated stimuli, as there is still a delay between the time t a pixel is rendered, and t_s when it is displayed.

Our solution is to produce a *rolling* image, where pixels at different spatial locations correspond to different points in time [Friston et al. 2016]. This is analogous to a rolling shutter sensor which captures light at different points in time for different sensor locations.

Ray-tracing. Both rolling and foveated images can be generated by ray-tracing: rays are free to use a different time value to intersect the virtual world and more rays could be sent to the fovea [Stengel et al. 2016; Weier et al. 2016]. Low-latency ray-casting has been demonstrated at interactive rates for simple scenes with specialized hardware [Friston et al. 2016]. Foveated ray-tracing is demonstrated by Stengel et al. [2016] in a system that adaptively sends more rays into perceptually important areas, including the fovea. Weier et al. [2016] also describe a solution that provides foveated ray-tracing for HMDs in real-time. Both systems require scenes that fit the assumptions of interactive ray-tracing.

Significant advances in ray-tracing have been made [Wald et al. 2014], but it is still typically considered too slow for modern interactive applications with complex dynamic scenes, such as computer games. A modern ray tracer, making use of bounding volume hierarchies (BVH), would handle a more continuous approximation of frame time by extending bounding volumes with primitive motion: the bounding volume of a single triangle moving across the scene will get much larger, i. e., ray-tracing it is much slower. Our tightest bound provides a better fit in space-time to avoid this.

Warping. One source of latency is the time expended between beginning a render and displaying it. One way to counteract this is to warp, i. e., deform, the final image, accounting for changes in viewpoint during the render. Early approaches changed which regions of an image were read out [Oculus VR 2017; Regan and Pose 1994], or drew points [Chen and Williams 1993] or grids [Mark et al. 1997]. Modern approaches such as Asynchronous Time Warp (ATW) [Antonov 2015] incorporate a number of these techniques to compensate for multiple sources of latency. In particular, ATW, compensates for rendering latency (the scene is rendered with a different transform than the one the image starts showing), but also for display latency (the transform is becoming outdated while the image is displayed) [Oculus 2018]. The main drawback of warping is that it suffers disocclusion artifacts. Gathering pixels instead of scattering can be faster [Yang et al. 2011], but will neither be able to resolve occlusion from a single image. Some techniques can help ameliorate these, such as perceptually improved hole filling [Didyk et al. 2010; Schollmeyer et al. 2017]. Alternatively the result can be improved by changing the images provided to the algorithm itself [Reinert et al. 2016]. No deformation however can reveal what is behind a surface. Our images have no disocclusion artifacts, and also support correct specular shading.

Shading latency. Due to latency, specular shading is also incorrect as highlights depend on the moving viewpoint that is frozen at the start of the frame in classic pipelines [Antonov 2015]. This could be resolved by ray-tracing, but would still produce problems if combined with warping. Perceptual rasterization correctly resolves specular shading.

Non-standard rasterzation. A simple solution to achieve both rolling and foveated images is to change the vertex shader [Brosz et al. 2007] from a linear to a non-linear projection, such as first done for shadow mapping [Brabec et al. 2002]. Doing this for latency compensation or foveation results in holes, in particular if primitives are large or close to the camera, as primitive edges remain straight [Brosz et al. 2007]. Our approach is a type of non-linear rasterization [Gascuel et al. 2008; Liu et al. 2011]. Toth et al. [2016] suggest single-pass rendering into spatially neighboring but multiple linear sub-projections [Popescu et al. 2009] to address the non-uniform pixel distribution in HMDs, but do not account for eye tracking. Rasterization has been made more flexible in stochastic rasterization [Akenine-Möller et al. 2007; Brunhaver et al. 2010; McGuire et al. 2010], but we are not aware of an approach to produce rolling or foveated images directly using rasterization in a single pass. In particular, we derive non-trivial bounds specific to our projection that drastically improve the sample test efficiency, i. e., how many fragments need to be tested against each primitive [Akenine-Möller et al. 2012; Laine et al. 2011; Pineda 1988].

3 PERCEPTUAL RASTERIZATION

We first describe the general perceptual rasterization pipeline before deriving specific bounds enabling its application to foveation, rolling and both. The key is to achieve just enough ray tracing-like flexibility while retaining the efficiency of rasterization.

Let us first recall rasterization and ray-tracing: ray-tracing iterates over pixels and finds the primitive mapping to them, while rasterization iterates over primitives and maps them to pixels. Our technique is a hybrid of these approaches. To decide what pixels a primitive maps to, the rasterization essentially performs ray-primitive intersections [Pineda 1988] followed by a z-test. A correct, but slow, solution would be to test all primitives against all pixels. Instead, the approach becomes fast by using tight *primitive-pixel bounds*: ideally, a compact, easy-to-compute subset of pixels is found for the projection of each primitive in a first step, and only the rays going through these pixels are tested against the primitive.

The idea of perceptual rasterization is to construct such pixel-primitive bounds for the requirements of HMDs. To this end, we will next propose different *ray-primitive models* we use (Sec. 3.1), before describing the pipeline in detail in Sec. 3.2. The actual bounds are then derived in Sec. 4.

3.1 Ray-primitive Models

The interaction between rays and primitives required on an HMD are not arbitrary, as, say, in path tracing, but have a very specific layout in time, space and the retina, which we will later exploit to construct appropriate bounds. We will now discuss the ray-primitive models required for common, as well as our foveated, rolling and jointly foveated-rolling rasterization.

3.1.1 *Foveated.*

To retain the simplicity of rasterization on a regular grid, we seek inspiration from cortical magnification theory [Daniel and Whitteridge 1961] also used in information visualization [Furnas 1986]: to give more importance to an area, it simply needs to be magnified. So instead of increasing the pixel density in the fovea, we just magnify it.

Domain. We suggest an image domain where the ray (or pixel) density depends on a function $p(d) \in (0, \sqrt{2}) \rightarrow \mathbb{R}^+$, where d is the distance to the foveation point $\mathbf{x}_\mathrm{f}$. In common rasterization, this function is a constant: 1 (Fig. 3 a, constant line). For foveated rendering, it is higher close to the fovea (d is small) and lower than 1 for the periphery (d is large) (Fig. 3, a, yellow line).

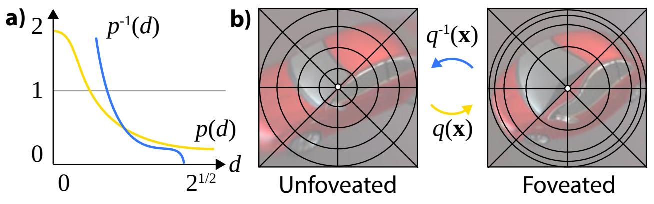

Fig. 3. Foveation and unfoveation function *(a)* and domains *(b)*.

p can be any foveation function, whether physiologically based [Daniel and Whitteridge 1961] or empirically based [Patney et al. 2016; Weier et al. 2017]. The size of the foveated region, and therefore p, must account for non-idealities such as imperfect tracking and suboptimal frame rates. These may also change over time. Therefore we refrain from using any analytic model and instead assume that the function is arbitrary, subject to the constraints below, and free to change every frame.

Given p, we define another function $f_\mathrm{f}(\mathbf{x}) \in (-1,1)^2 \to (-1,1)^2 : \mathbf{x}_\mathrm{f} + \mathtt{normalize}(\mathbf{x}-\mathbf{x}_\mathrm{f}) \cdot p(||\mathbf{x}-\mathbf{x}_\mathrm{f}||)$. This function essentially scales $\mathbf{x}$ by p, away from the gaze position $\mathbf{x}_\mathrm{f}$. Near the center, this results in stretching, as the pixel density is larger than 1. In the periphery, compression, as fewer pixels are required (Fig. 3, b). We also define f_f^{-1}, to be f_f but with p^{-1} in place of p. p^{-1} is the inverse of p. This necessitates that p is invertible. Any monotonic p can be inverted numerically in a pre-processing pass, if an analytic inversion is non-trivial. Note that d is not a scaling factor but an exact distance. Thus p maps an unfoveated distance to a foveated distance, and p^{-1} maps it back. f_f and f_f^{-1} use these functions to do the same for pixel locations. We refer to these pixel transformations as to "foveate" and "unfoveate".

Rendering. During shading, each fragment in the non-uniform buffer uses the unfoveate function to compute the ray origin. Consequently, more fragments cast rays to the region under the fovea, effectively magnifying it (Fig. 3, b, Foveated).

Display. After rendering all primitives, the foveated image I_f has to be converted back into an unfoveated I_u one for display. This imposes several challenges for filtering: f_f^{-1} is heavily minifying in the center and heavily magnifying in the periphery. A simple and fast solution is to create a MIP map for the foveated image I_f and then evaluate the display image as $I_\mathrm{u}(\mathbf{x}) = I_\mathrm{f}(f_\mathrm{f}(\mathbf{x}))$ using proper tri-linear MIP mapping and a 3-tap cubic filter (0.6 ms in 1024×1024 on an Nvidia GTX 980 GPU). A higher-quality version (1.6 ms in 1024×1024, same GPU) computes

$$I_\mathrm{u}(\mathbf{x}) = \sum_{y\in 5\times 5} I_\mathrm{f}(f_\mathrm{f}(\mathbf{x})+\mathbf{y}) \cdot r(\mathbf{x} - f_\mathrm{f}^{-1}(f_\mathrm{f}(\mathbf{x})+\mathbf{y}))),$$

where r an arbitrary, e. g., Gaussian, 2D reconstruction filter in the display image domain. Such an operation effectively computes the (irregular-shaped) mapping of the display's reconstruction filter into the cortical domain.

3.1.2 Rolling. Here, the ray direction and position at a certain pixel depends on the time that pixel is displayed. When testing a ray through a given pixel, the state of the primitive intersected also has to be its state at the time the pixel is displayed.

Display. We consider a rolling-scan display to have three properties: rolling illumination, a short hold-time, and we must be able to predict the absolute head pose at any point in the interval $[t_\mathrm{s}, t_\mathrm{e}]$.

First, a *rolling scan* implies that different parts of the display are visible at different times. The term "rolling" is chosen as an analogy to a camera's rolling shutter sensor. A classic CRT is an example of a rolling scan display. Most LCDs these days perform a globally synchronized illumination of all pixels at once. OLEDs, such as those used in the DK2 and other HMDs sometimes use rolling illumination.

We will formalize this as a *rolling*-function $r(\mathbf{x}) \in (0,1)^2 \to (0,1) : \mathbf{x}\cdot\mathbf{d}$ that maps a (unit) spatial location $\mathbf{x}$ to a (unit) point in time at which the display will actually show it by means of a skew direction $\mathbf{d}$. $\mathbf{d}$ depends on the properties of an individual display. For example $\mathbf{d} = (0, .9)$ describes a display with a horizontal scan-out in the direction of the x-axis and a (blank) sync period of 10 % of the frame period. For the DK2, $\mathbf{d} = (1,0)$ based on behavior profiled with an oscilloscope (Fig. 2).

Second, the display has to be low persistence (*non-hold-type*), i. e., a pixel is visible for only a short time relative to the total refresh period. A CRT is typically of this type. CRT phosphor has a decay that typically reduces brightness by a factor of 100 within one millisecond (Fig. 1 in [Sluyterman 2006]).

Third, we assume that the model-view transformation can be linearly interpolated across the animation interval and that consequently vertices move along linear paths $\mathbf{x}(t)$ during that time.

Forward map. It is non-obvious, to which 2D image position a moving 3D point will map on a rolling display. Fortunately, we can show that this mapping is unique and simple to compute in closed form. The problem of finding where the rolling scan will "catch up" with the projection of a moving 3D point has similarity with Zenon's paradoxon where Achilles tries to catch up with the tortoise [Wicksteed and Cornford 1929] (Fig. 5, a).

If Achilles starts at x_s and moves at constant speed $\dot{x}_\mathrm{s}$, it will reach (other than what the paradoxon claims) a tortoise at position x_p with 1D speed $\dot{x}_\mathrm{p}$ at the time t where

$$x_\mathrm{s} + t\dot{x}_\mathrm{s} = x_\mathrm{p} + t\dot{x}_\mathrm{p}, \quad \text{which occurs at} \quad t = \frac{x_\mathrm{s} - x_\mathrm{p}}{\dot{x}_\mathrm{s} - \dot{x}_\mathrm{p}}.$$

The same holds for a rolling scan (Achilles) catching up with a vertex (tortoise). Regrettably, in our case, the rolling scan moves in 2D image space, while the point moves in 3D and gets projected to 2D i. e., it moves in a 2D projective space (horizontal x component and homogeneous coordinate w) from spatial position x with speed $\dot{x}$ and homogeneous position w with homogeneous speed $\dot{w}$ (Fig. 5, b). This can be stated as

$$x_\mathrm{s} + t\dot{x}_\mathrm{s} = \frac{x_\mathrm{p} + t\dot{x}_\mathrm{p}}{w_\mathrm{p} + t\dot{w}_\mathrm{p}},$$

which is a rational polynomial with a unique positive solution

$$t = -\frac{(\sqrt{4x_\mathrm{s}\dot{w}_\mathrm{p} + \dot{x}_\mathrm{s}^2 - 2\dot{x}_\mathrm{s}w_\mathrm{p} + w_\mathrm{p}^2} - \dot{x}_\mathrm{s} + w_\mathrm{p})}{2\dot{w}_\mathrm{p}}. \qquad (1)$$

This equation is non-linear, as linear 3D motion becomes non-linear in 2D under perspective projection.

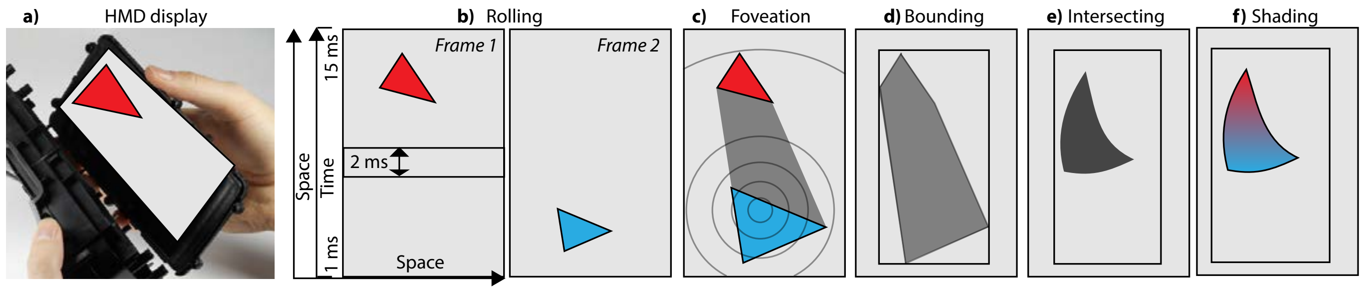

Fig. 4. Overview of perceptual rasterization. Common rasterization **(a)** produces images at a fixed time and uniform pixel density. We suggest to account for primitive motion, here shown as two frames **(b)** and non-uniform pixel density, here visualized as iso-lines (**c**). Primitive-ray interaction is bound, here using a rectangle **(d)** and intersected **(e)** to produce a rolling and foveated image to be shaded (**f**). (Depiction uses a monoscopic HMD display for simplicity.)

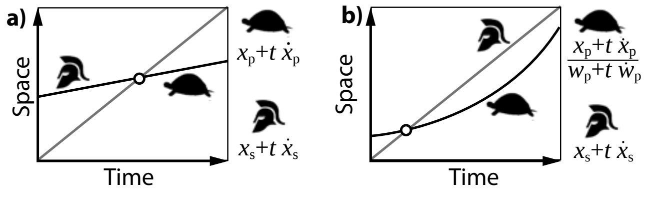

Fig. 5. Linear (**a**) and perspective (**b**) Zenon's paradoxon (see text below).

Henceforth, to compute the image position $f_r(\mathbf{x})$, we first compute the collision time t, from this the position $\mathbf{x}(t)$ and finally project it to 2D.

3.1.3 Lens distortion. Our approach supports a third aspect of image generation in HMDs: lens distortion [Oculus VR 2017]. Same as for foveation, we use a polynomial f_l distortion model. Same as for rolling, and different from foveation, the inverse f_l^{-1} does not need to be applied to the image as this already happens optically i. e., the display shows a distorted image that will ultimately be deformed by the lens. Only the – much smaller – chromatic blur still needs to be applied, as the effort of rasterizing three channels independently does not appear justified.

3.1.4 Compositions. Our pipeline supports the foveation-rolling-lens composition $f = f_l \circ f_r \circ f_f(\mathbf{x})$ of the three functions above. Without loss of generality, we will from now on use f, an arbitrary composition of the three perceptualizations and its inverse f^{-1}. Order is important: the rolling time coordinate has to depend on where the pixel will effectively be displayed on the display (not the cortical domain). Lens distortion should happen last.

3.2 Pipeline

An overview of perceptual rasterization is seen in Fig. 4, d–f. We extend a classic OpenGL-style rasterization pipeline using vertex, geometry and fragment programs (VP, GP and FP) to produce a typical deferred shading buffer from primitives in two steps: *bounding* and *intersecting*. We will explain how to bound tightly and efficiently for the different models later in Sec. 4.

Bounding. Input to the VP are the world-space vertex positions v_s at the beginning and v_e at the end of the frame interval. Additionally, the VP is provided two model-view-projection matrices M_s and M_e that hold the model and view matrices at the beginning and the end of the frame interval. The VP transforms both the start and the end vertex, each with the start and the end matrix ($\mathsf{M}_s v_s$ and $\mathsf{M}_e v_e$), and passes this information on to the GP. Please note, how this procedure captures both object and viewer motion. Please also note, that no projection is required at this step.

Input to the GP is the tuple of animated camera-space vertices $S = (v_{s,0}, v_{e,0}, v_{s,1}, v_{e,1}, v_{s,2}, v_{e,2})$, i. e., an animated camera space triangle. The GP *bounds* the projection of this space-time triangle with a six-sided convex polygon B, such that all pixels that would be affected by the triangle are covered. We describe the bounding computation in Sec. 4. The geometry program passes the space-time triangle on to the fragment program as (`flat`) attributes. Note, that the bounding primitive B is not passed on from the GP to the FP: It is only required as a proxy to determine the pixels to test directly against S (and not B) i. e., what pixels to rasterize. The fragment program then performs the intersection test described next.

Intersection. The fragment program is now executed for every pixel i affected by the primitive. This happens in three steps.

First, we note that for a fixed pixel i at 2D image position $\mathbf{x}_i$, the time is fixed to $t_i = r(\mathbf{x}_i)$ as well, and so the space-time primitive $\mathcal{S}$ becomes a common space-only triangle $\mathcal{T}_i = (v_{s,0} + t_i(v_{e,0} - v_{s,0}), \ldots)$. Second, we compute the ray R_i. To this end we construct a ray through the 2D image location $f^{-1}(\mathbf{x}_i)$. Third and finally, we intersect R_i and $\mathcal{T}_i$ using a common ray-triangle intersection test. If the test fails, nothing happens. If the test passes, the fragment is written with the actual z value of the intersection and with common z buffering enabled. Recall, that the entire space-time triangle, its normals, texture coordinates and material information, were passed on as `flat` attributes from the GP and can now be computed using barycentric interpolation.

Please note, how the depth test will resolve the correct (i. e., nearest to the viewer) fragment. For every pixel there is a unique time and fovea location, and hence the distances of primitives mapping to that pixel are z-comparable. This is key to make perceptual rasterization possible when primitives are submitted in a streaming fashion in an arbitrary order, combining the flexibility of ray-tracing with the efficiency of z-buffered rasterization.

Shading. Shading has to respect the ray-primitive model as well: the time at every pixel is different for the rolling and joint model, having the implication that parameters used for shading, such as light and eye position, should also be rolling and differ per-pixel. This again can be done with linear interpolation. Note that shading is not affected by foveation.

4 BOUNDS

A key technical contribution of this paper is the derivation of tight and efficiently computable bounds for the ray-primitive model required for modern HMDs.

Similar bounds have been derived for other non-linear projections before [Gascuel et al. 2008; McGuire et al. 2010]. The non-linearities we consider here, are different. While previous work was interested in fish-eye-like or spherical mappings [Gascuel et al. 2008] as well as bounds for depth-of-field and motion blur [McGuire et al. 2010], the problem of rolling and foveation has not been addressed from this angle.

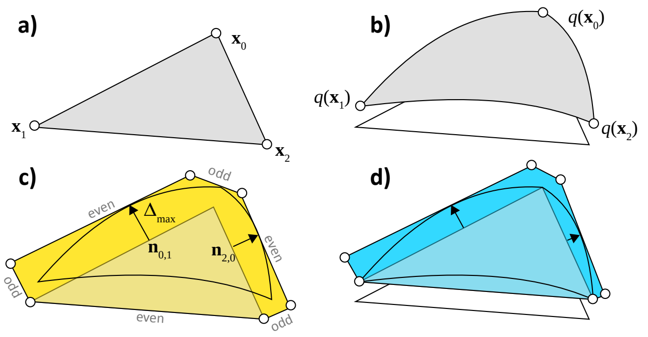

Fig. 6. Search-and-displace. **a)** the original straight-edge primitive. **b)** curved-edge primitive. **c)** simple bounds displaces the original straight edges. **d)** advanced bound first maps the end-points and then bounds the displacement. Note the blue area to be smaller than the yellow one.

Bounding is done with respect to the perceptual mapping f that composes rolling, foveation and lens distortion, mapping straight primitive edges to curves (Fig. 6 a and b). We would like to find a tight straight-edged polygon to cover this curved-edge primitive.

To this end, we employ a search-and-displace strategy (Fig. 6 c) where every straight edge is displaced perpendicular along its 2D normal by the minimal amount required so that the entire curved edge is below the new straight edge when looking outwards. This minimal displacement is found in a search along the straight primitive edge. An advanced *adaptive* bound is even tighter: It first maps the two end-points and then search-and-displaces relative to this new straight edge (Fig. 6 d). We will next detail the search (Sec. 4.1) and displace (Sec. 4.2) steps.

4.1 Search

We use either the original ("Simple") or the already perceptualized ("Adaptive") start and end points of each edge. We explain both options next.

Simple. Here, we suggest to bound by finding the maximum positive signed distance along the normal from a primitive edge joining $\mathbf{x}_0$ and $\mathbf{x}_1$

$$\Delta_{\max} = \max_{s\in(0,1)} \{\Delta(s) = (\eta_s(s) - \eta_c(s)) \cdot n(\eta_s(0), \eta_s(1))\}$$

$$\eta_s(s) = \mathbf{x}_0 + s(\mathbf{x}_1 - \mathbf{x}_0) \quad \text{and} \quad \eta_c(s) = f(\mathbf{x}_0 + s(\mathbf{x}_1 - \mathbf{x}_0)),$$

where n creates a direction orthogonal to the line between its two arguments.

This is shown for all three edges in Fig. 6,c. We consider signed instead of unsigned distance. Unsigned distances could not tell apart a curve that is bending in from a curve that is bending out in respect to the straight edge. While an out-bending edge now will extend the bounds, an in-bending should not (the maximal positive distance is zero, at both end points). An example of such an in-bending edge is the lowest edge in Fig. 6, c.

As the distance is a convex function, the offset $\Delta(s)$ can be minimized over s using a ternary search. This procedure starts by evaluating Δ on both ends $s = 0$ and $s = 1$ as well as in the middle $s = .5$, followed by a recursion on the left or right segment. The approach converges to a pixel-precise result in $\log(n)$ steps, if n is the number of possible values, here, the number of pixels on the edge. Consequently, for a 4 k image example, bounding requires $3 \times 2 \times \log(4096) = 96$ multiply-adds and dot products per triangle at most, but typically much less as triangle edges are shorter.

Adaptive. In the adaptive case, the straight edge does not join the original, but the perceptualized end-points

$$\eta_s(s) = f(\mathbf{x}_0) + s(f(\mathbf{x}_1) - f(\mathbf{x}_0)).$$

This is seen in Fig. 6, d, where displacement is relative to edges that join vertices that were already mapped by f. The bound is more tight as seen when comparing the yellow to the blue area. In particular, the lower edge moved up, and, bending in, produces a much tighter bound.

Note, how this adaptive bound is only possible thanks to our perspective Zenon mapping (Eq. 1) that can predict where moving 3D vertices fall on a rolling 2D display.

Please also note, that the normal for a different straight edge is also different, as f is a nonlinear function: an edge joining a point close to the origin and a point farther from the origin will change its slope as both are scaled differently.

4.2 Displace

Displace moves the 2D edge by the minimal required distance along its normal. Our bounding geometry will always consist of a convex polygon with six vertices, and does not require a convex hull computation. Every even pair of vertices is produced by bounding a single edge of the original triangle. Every odd pair joins the start and end of a bounding edge produced from a primitive edge.

We also compare to a simpler strategy, that bounds the perceptualized primitive using the 2D bounding box of all pixels on all curved edges.

For primitives intersecting the near plane we proceed similar to McGuire et al. [2010]: all primitives completely outside the frustum are culled; primitives completely in front of the camera (but maybe not in the frustum) are kept, and those that intersect the near plane are split by this plane and a box is used for bounding.

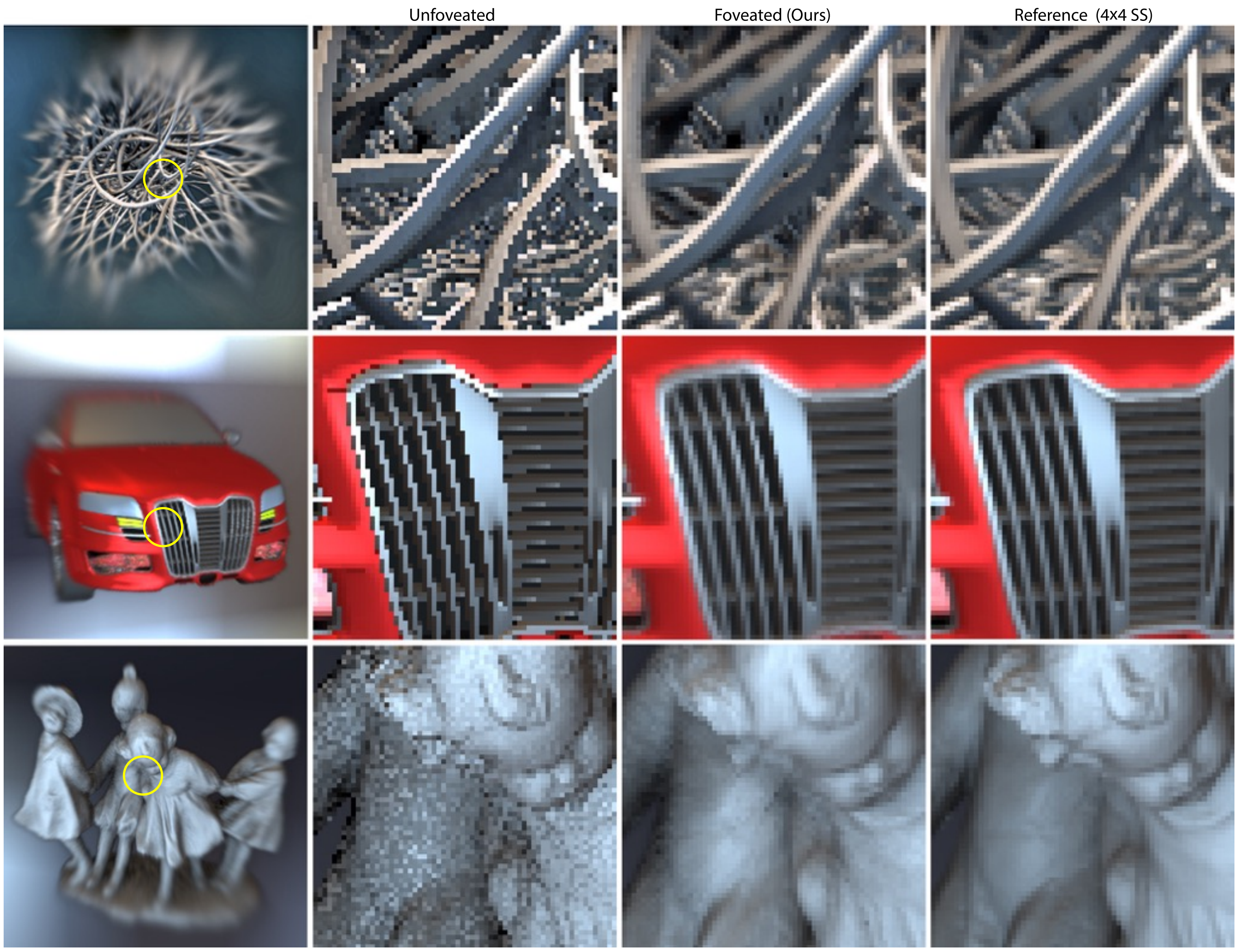

Fig. 7. Foveation results. The first column shows the result we produce, fovea marked in yellow. The second to fourth columns shows the foveated region using non-foveated rendering, our approach, and a 4×4 super-sampling reference. Quantitative evaluation is found in Fig. 12.

5 RESULTS

We discuss qualitative (Sec. 5.1) and quantitative (Sec. 5.2) results.

5.1 Qualitative

Foveation. Results of our foveated rasterization approach are seen in Fig. 7. Our image was produced by foveating the center using a simple power-falloff $p(x) = x^2$ foveation function. The inset shows a 32×32 patch. The reference was produced by 4×4 super-sampling.

We see that the amount of detail varies across the image in the first column. While the center is sharp, yet super-sampled, the periphery has less detail, yet blurred with a high-quality cubic filter. In the unfoveated condition (second column) the fine hairs of the hairball lead to almost random results without super-sampling, while our result remains smooth and similar to the reference. The same is true for the fine geometric details in the car's grill. In the CHILDREN scene, the super-sampling of shading is salient.

The common images were produced using the same memory, the same shading effort and not less than half the compute time than ours (third column), yet the differences are visible. At the same time, the reference (fourth column), uses 16 times more memory and shading effort and is more than twice the compute time than ours, yet the differences are subtle.

Rolling. Images produced by our rolling rasterization approach can be seen in Fig. 8. A non-rolling image is seen in the first column. The second and third columns contain rolling images where the camera has both translated and rotated during a rolling scan-out from left to right. The second column shows image warping using a pixel-sized grid, where triangles that have a stretch that differs by more than a threshold are culled entirely [Mark et al. 1997]. Disoccluded areas are marked with a checkerboard pattern. The third column shows the results produced by our approach. The fourth and fifth columns show insets from the second and third

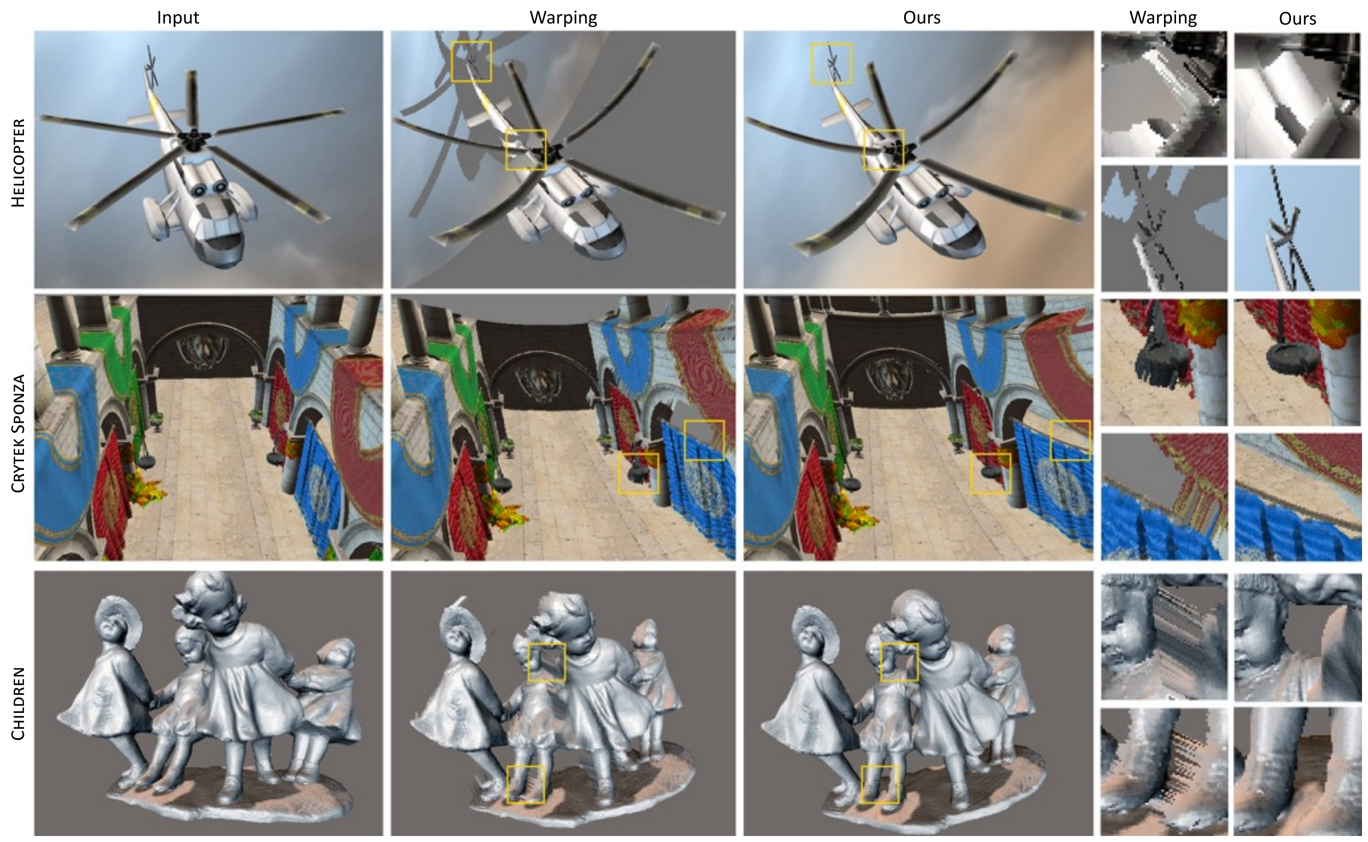

Fig. 8. Results of our rolling rastrization approach. Different rows show different scenes. The first column shows the input image. The result of warping is shown in the second, where disocclusions were filled with gray. The third column shows our approach. The fourth and fifth columns shown the inset areas from columns two and three. Quantitative evaluation is found in Fig. 12. Please, see the supplemental video for animated versions of these results.

row. Our images are identical to a ray-traced reference, which is not shown.

We see that rolling images contain the expected non-linear projection effects: long edges that are straight in 3D appear as curves in the image. As this mapping is consistent, other effects such as shadows and specularities appear consistent for all approaches. Warping however has difficulties with disocclusions, edges and fine details. We see that large parts of the background are missing. The biggest challenge are areas occluded in the input image. Large parts are missing in warping, e. g., the sky background in HELICOPTER condition, that are easily resolved by our approach. Current Warping techniques always have difficulties with edges, where a pixel can only be either warped or not, resulting in jagging artifacts such as on the edges of CHILDREN. When motion, occlusion and fine edge structures come together, such as in the area around the HELICOPTER's rotor, the warped images bear little resemblance to the reference.

Rolling+foveation. Results for joint rolling+foveated images are show in Fig. 9. We see both the expected improvement in the foveal inset and the global rolling: the car and fence have straight 3D edges that turn into curves under viewer motion. Those scenes have around 100 k faces and render in less than 50 ms.

Lens Distortion. Results for lens distortion are seen in Fig. 10.

Rolling shading. Here we compare rolling shading, included in all the above results, to rolling rasterization without rolling shading in Fig. 11. Specular inconsistencies will result in popping artifacts over time [Antonov 2015], where a highlight does not slide across the side of the car but judders between frames.

5.2 Quantitative

Here we compare alternatives to our approach for both foveation and latency compensation, in terms of image similarity and speed.

Competitors. For foveation, we compare our technique to LMS [Nvidia 2017] and log-polar warping [Meng et al. 2018] called "Kernel". We report both the speed assuming LMS is as fast as common rasterization, and the very upper-bound speed LMS could have if it was able to rasterize four images at the speed of one when using `GL_NV_clip_space_w_scaling` and `NV_viewport_array2`. We also evaluate the performance of a three-layered method [Guenter et al. 2012], but assume image similarity is ideal. For latency compensation, we compare to traditional image warping. We compare three variations of our technique with different bounds. We also compare to a traditional rasterization as a benchmark (Common) rendered with the same resolution as ours.

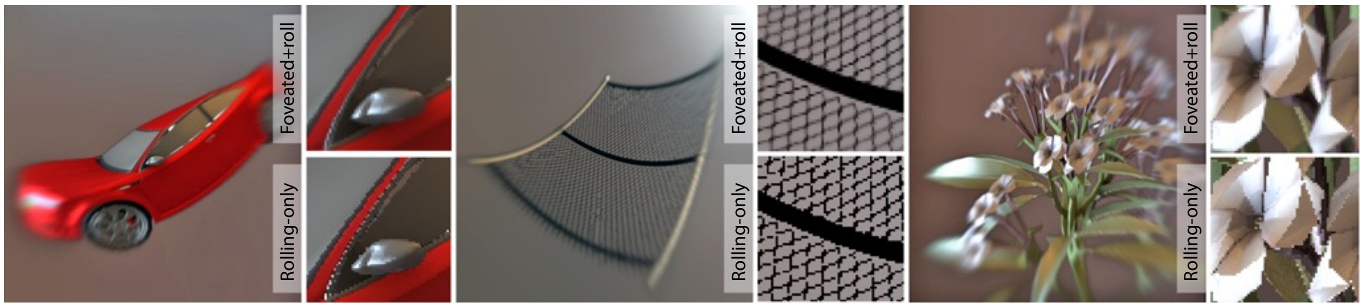

Fig. 9. Rolling+foveated, perceptual rasterization for three scenes. The insets compare rolling+foveation and rolling-only results.

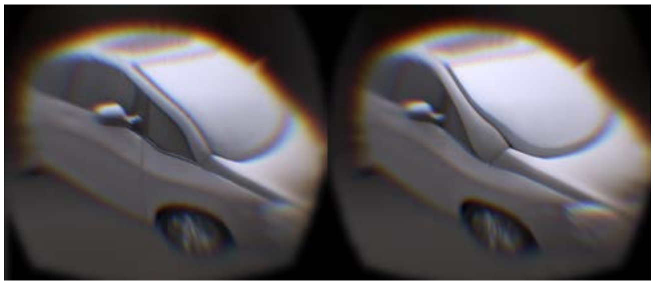

Fig. 10. This stereo image is both rolling and foveated, in addition it will appear undistorted in space and chroma when observed through the lenses of an HMD.

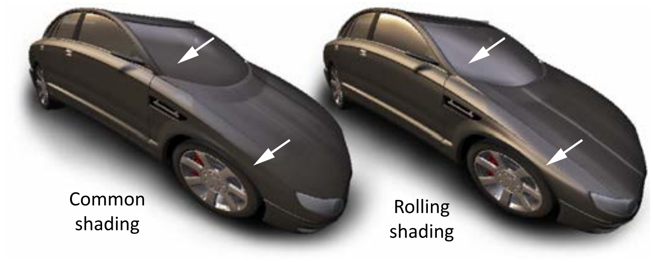

Fig. 11. Rolling rasterization without rolling shading *(left)* lacks some specular effects. Rolling shading *(right)* produces highlights that change across the image due to the change in view over time.

Methods. Image similarity is measured with an adapted SSIM [Wang et al. 2004]. All techniques were compared to an ideal reference, which was a ray-traced image for rolling and an 8×8 super-sampled rasterization for foveation. The SSIM ignores all disoccluded pixels, providing an upper bound on quality to what any hole filling, however sophisticated, could do [Didyk et al. 2010; Schollmeyer et al. 2017]. For foveated comparisons, the SSIM is computed for the 64×64 foveal pixels. While SSIM is not designed to compare non-aligned images, we use it as unfortunately, no better technique is available either. Performance was measured as the rasterization and shading execution times on an NVidia Quadro K6000. We state the ray-tracing time of a reasonably implemented GPU traversal of an SAH-optimized BVH.

Similarity and Speed. Performance and similarity measures for foveation and rolling are shown are shown in (Fig. 12), along with the Sample Test Efficiency (STE) measures of our three boundings. The column "Total" is the compute time sum in millisecond (less is better) of the second column "Rasterization" and the third column "Shade". The fourth column is STE (more is better) and the fifth similarity (more is better). A white diagonal in a bar indicates the time exceeds the range of the plot. The first three rows are foveation, the next are rolling, followed by combined techniques. Each row is a different scene.

For foveation, our approach is more similar in appearance to the reference than common rasterization (Fig. 12, a). Quality is lower both for "Kernel" and "LMS", even when operating both at their optimal foveation settings according to the experimentation in Fig. 14. Kernel foveated rendering performs better as it is a better fit to the foveation shape than a a linear function of LMS. Furthermore, our approach achieves speed that is roughly half as fast rasterizing multiple layers and very similar to rendering at full resolution (Fig. 12, b). LMS is shown in two variants: the optimum one is shown solid, the one we were able to measure is shown in transparent. The measured LMS is always four times slower than common and between 10 % and 50 % faster than ours. Kernel speed is limited by having to rasterizing a large image before resampling it. Shading effort (SSAO and IBL) is the same for ours and common, while it is three times larger for layered and 16 times larger for the reference (Fig. 12, c). The following rows of "Shading" are greyed as they repeat the first row. Finally, we see that refined bounds increase sample test efficiency as well as actual compute time (Fig. 12, d).

We see that rolling and common non-rolling images are substantially different according to the SSIM metric. When warping the image, the similarity increases but remains behind ours (Fig. 12, e). Note, that our SSIM is always 1 as our rasterization has been verified to be identical to ray-tracing a rolling shutter image. Common rasterization is fastest, but warping requires two primitives per pixel [Mark et al. 1997] and turns out slower (Fig. 12, f). Differences in STE between our method and other are even more pronounced for rolling (Fig. 12, g). We also note that scenes with many polygons, such as CHILDREN (1.4 M) are feasible, but noticeably slower, likely due to the straightforward convex hull implementation used in the GP. LMS is not applied to rolling in our tests.

For both foveation and rolling, ray-tracing – while very convenient and clean to implement – is slower than all versions of rasterization. Note, that the ray-tracing numbers do not include the SAH building time required, which is likely substantially larger (Fig. 12, h).

In combined foveation and rolling, our method again provides much better quality at slightly lower speed compare to the two alternatives LMS and Kernel (Fig. 12, i).

Finally, for foveated shadow mapping, all methods come out at similar speed (Fig. 12, j), but again ours provides the highest quality.

Overall, perceptual rasterization achieves quality similar to a super-sampled reference, while being slower than highly-optimized, fixed-pipeline rasterization by a moderate factor, but much faster than ray-tracing and super-sampling.

Foveation. Fig. 14 compares our technique, LMS and log-polar warping at different foveation strengths. The application of LMS [Nvidia 2017] to foveation is hypothetical and not described in any publication we are aware of.

We tested different linear functions, as used in LMS in Fig. 14 and observed that none surpass the similarity to a reference that we achieve (blue vs. orange line, top plot). We note that both high and low foveation performs worse, as expected from fitting lines of different slope to a non-linear function: One is the best, but still not a good, fit (around 2.0, orange maximum) all others deviate more. At the same time, using four linear projections requires four passes and four times the compute. Still due to its simplicity, performance is competitive to ours for a high level of foveation (the orange and blue curves almost cross in the top plot of Fig. 14). When LMS is available and if it operates with zero overhead, the compute time should be reduced by a factor of 4 (dotted orange line in Fig. 14).

The second alternative is log-polar warping [Meng et al. 2018]. Here an entire image is first rasterized at foveal resolution, then warped and sub-sampled into an alternative polar cortical domain. Shading is done in this reduced domain and the image is transformed back. This does not reduce rasterization time, but does reduce shading time, similar to our approach. Rasterizing the entire image at the foveal resolution is simple but requires excessive amounts of memory: If the fovea is oversampled at 16×16, the approach needs 64 times more memory and fill-rate. For an upper bound on quality, our implementation of this method shades the full high-resolution image and samples this using a log-polar mapping. For timing, we only measure rasterization of the full-fovea resolution image, again, an upper bound. This is as implementing screen-space shading in the log-polar domain appears substantially non-trivial: the most foveal pixel fills the entire left column, etc. It also appears likely that the idea of first rasterizing with uniform resolution and then shading in a warped domain could be explored in future work using our cortical mapping, which does not require polar coordinates and is a better fit to screen space shading as it preserves topology. Besides higher memory requirements, the approach is fast, saving substantial shading cost (green curve lower than blue in second plot of Fig. 14) at almost the same quality as ours (green line below blue line in first plot in Fig. 14). Still, we found the results to be slightly more blurry, due to the difficulty of filtering the highly anisotropic log-polar map. For no foveation, this results in a slight blur, that in this example happens to produce a result closer to the reference (green curve higher than any other at 0 in top plot at Fig. 14).

We conclude that a linear foveation function does not provide the speed and quality of a non-linear function, and that log-polar re-sampling uses prohibitive amounts of memory, while still not providing the optimal approximation of the true foveation function. It is finally to be noted, that neither method has demonstrated rolling or combination with rolling or lens distortion, which our method offers without overhead.

Sample Test Efficiency. We also compute the STE [Akenine-Möller et al. 2007; Fatahalian et al. 2009; Laine et al. 2011; McGuire et al. 2010], defined as the ratio of pixels belonging to a primitive to the number of pixels tested. An STE of 100 % would mean that only necessary test were made, i. e., the bounds were very tight. A low STE indicates that unnecessary tests occurred. Comparing the bounding approaches in Fig. 12 and Fig. 12 (STE fourth column), it can be seen that investing computational effort into tight bounds, pays off with a higher STE and is ultimately faster overall. Visualizations of the STE for rolling rasterization are seen in Fig. 15.

Scalability. Dependency of speed and image similarity on external variables is plotted for different approaches in Fig. 13.

The first plot shows how image resolution affects computation time (Fig. 13, a). We see that our approach is, as expected, slower than common rasterization, which is highly-optimized in GPUs. At the same time warping does not scale well with resolution due to the many pixel-sized triangles to draw. At high resolutions, the warping method is worse both in terms of speed, as well as image quality.

Next, we analyze computation time as a factor of the transformation occurring during the scan-out (Fig. 13, b). We quantify this as view rotation angle around the vertical axis. We see that classic rasterization is not affected by transformation at all. Warping adds an almost-constant time overhead that only increases as larger polygons are to be drawn. Our approach is linearly dependent. The amount of pixel motion is expected to be linear in small angles. Our tighter bounds can at best reduce the magnitude of the linear relationship. For large motions our approach is approximately half as fast as fixed-function rasterization plus warping, or six times slower than fixed-function rasterization alone.

Next, we analyze similarity (higher is better) depending on the transformation, again parametrized as an angle (Fig. 13, c). We find that our approach, as expected, has no error relative to the ray-tracing reference. This is as we use the same ray-primitive intersection, the only difference is that one is classic BVH-accelerated ray-tracing starting at pixels, while our approach conservatively bounds the pixels on-screen and tests against the same primitives using the same code. With no user motion, common rasterization has no error either, while warping still introduces sampling artifacts. As motion becomes more extreme warping reduces error with respect to common rasterization, but similarity still decreases, as disocclusions cannot be resolved from a single image.

Finally, we see the dependency of similarity and compute time on foveation strength α (Fig. 13, d), in the power foveation function $p(d) = d^{\alpha}$. We find that similarity is a convex function, peaking around the value $\alpha = 2$ that we use. Too low-a foveation does not magnify enough to benefit from the super-sampling. Too high-a values magnify so much, that only the central part of the fovea benefits, reducing SSIM again. Compute time is a linear function of foveation strength, as polygonal bounds to increasingly curved triangles are decreasingly tight.

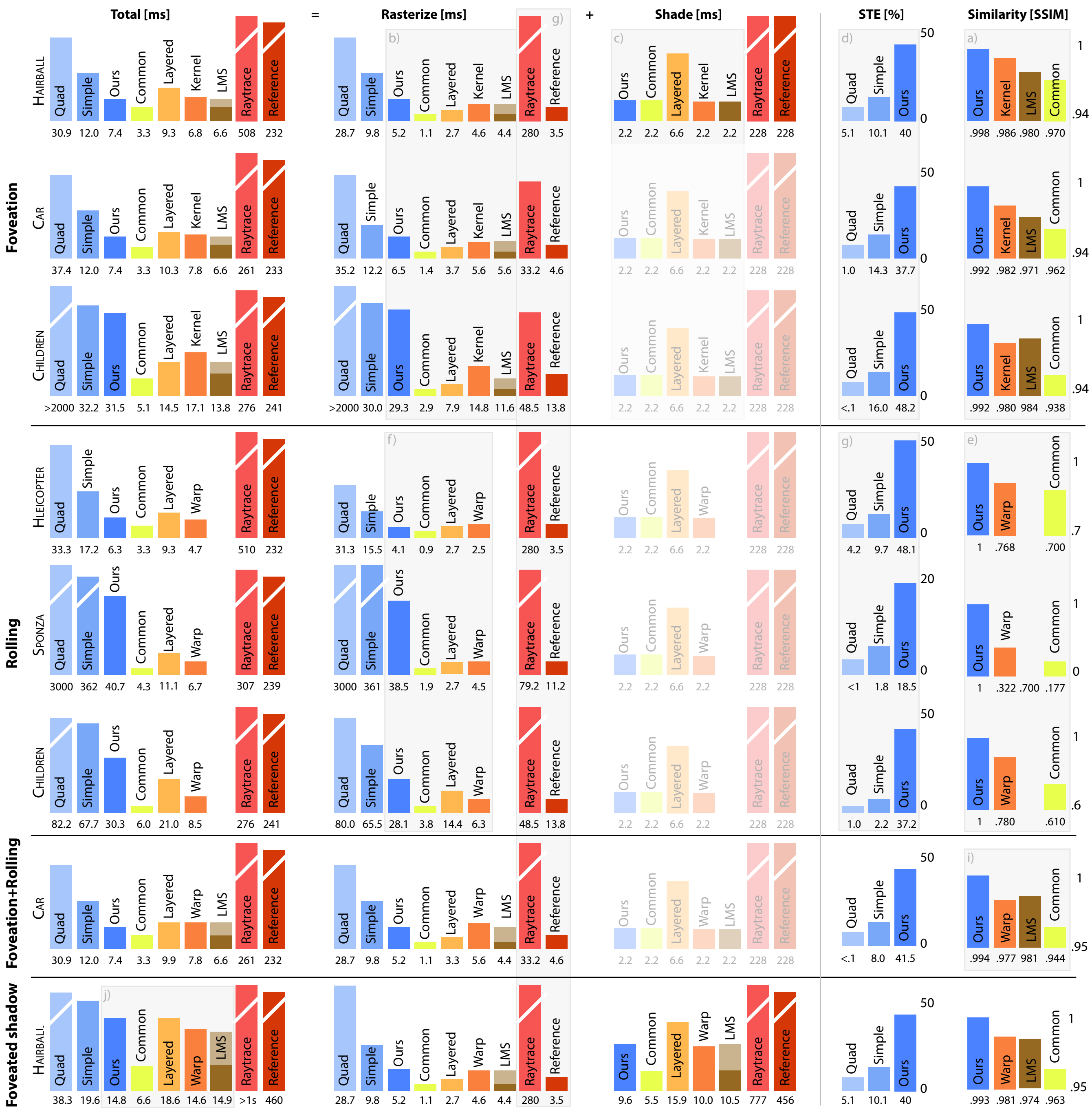

Fig. 12. Measurements of time (less is better; total, and split in rasterizaiton and shade), STE (more is better) and similarity (more is better) (**horizontal blocks**) resulting from different methods (**columns)**) for different perceptualizations **(vertical blocks)** in different scenes **(rows)**. See the text for discussion.

Head Pose Estimation. Finally, we investigate the effect of head pose prediction error on our approach. Before (e. g., Fig. 13, c), we have seen that the image error is proportional to the error in transformation. Therefore, we sampled head motion using the DK2 at approximately 1000 Hz. At each time step we used the SDK's predictor - the same that drives the rolling rasterization - to predict the pose one frame ahead. We use these captures to determine how the linearly interpolated pose and a time-constant pose differ from the actual pose. For 3,459 frames of typical DK2 motion, we we found the LINEAR prediction to have an error of .001 meter in translation and .25 degree in rotation while the error of a CONSTANT

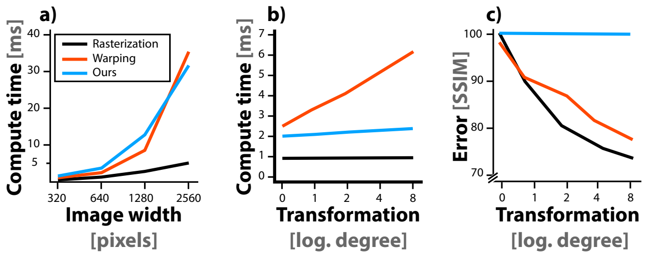

Fig. 13. Comparison of different rolling approaches in Helicopter: Classic rasterization, warping and rolling. **a)** Image resolution and compute time (less is better). **b)** Transformation (camera rotation) and compute time (less is better). **c)** Transformation and image similarity in SSIM (more is better).

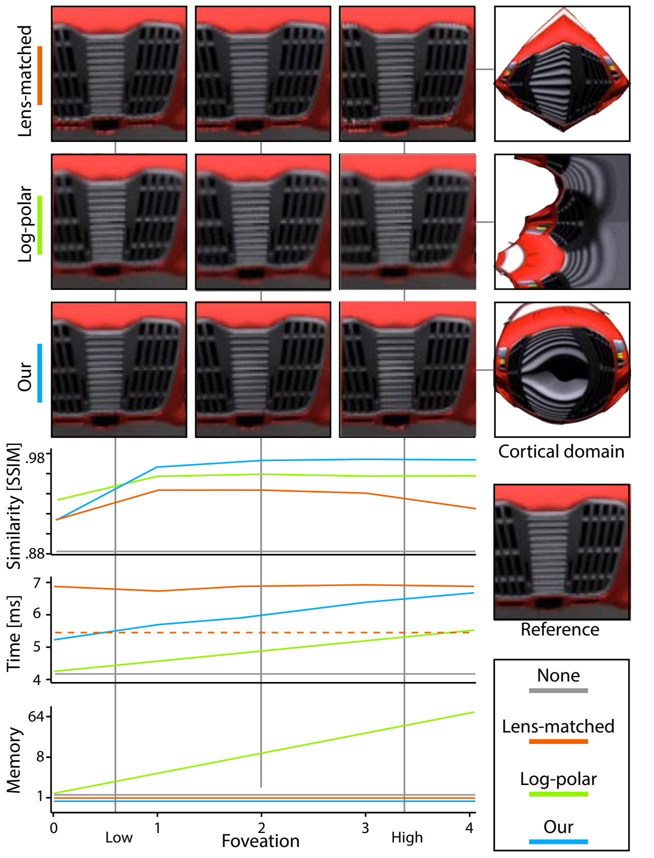

Fig. 14. Comparison to alternative approaches: The first to third row, show insets using three different methods at three increasing levels of foveation in columns one to three. The fourth row shows the entire image in the cortical domain. The plots show the resulting similarity (more is better), compute time (less is better) and memory requirement (less is better).

prediction is much larger, at .05 meter and 1.3 degrees, indicating a linear model already removes most of the overall error.

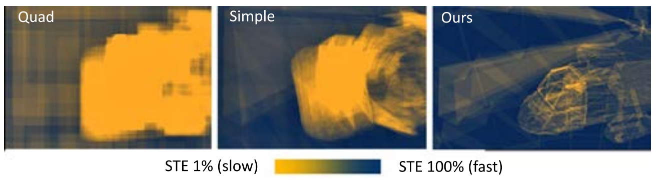

Fig. 15. Sample test efficiency of different rolling bounds in Helicopter. We see that quads are not a tight bound and while the simple bound improves it overestimates motion bounds in areas of vertical motion. Our bounds is most tight as it better localizes in time and rolling display space.

6 PERCEPTUAL EVALUATION

To quantify the perceptual effect of our technique, we conducted three user studies. Detailed apparatus, stimuli and analysis, including significance statements are found in the supplemental materials.

Foveation strength. The first is a threshold estimation experiment to establish the optimal foveation in an eye-tracked desktop setting. Subjects were asked to adjust foveation α to their preference in three scenes. We could establish, that users prefer an average foveation of $\alpha = .23$. This is lower than the physiologically-expected 2.0, but in agreement with previous studies of foveation using a desktop [Patney et al. 2016]. Please see the blue lines in Fig. 14 for an example of what such a foveation looks like, its compute time and memory budget as well as the SSIM in respect to a reference.

Foveation preference. The second is an image judgment experiment comparing images with no foveation and our foveation to referenced images, again in an eye-tracked desktop setting. Here, subjects clearly (in 90.0 % of the cases) prefer our treatment over no foveation.

Object tracking. The third is an object tracking experiment with and without rolling rasterization, performed on a HMD. Subjects were asked to track moving objects, rendered using our technique, warping and no latency compensation. In natural viewing conditions, humans would lead the target. This effect was strongest when using our method.

7 DISCUSSION

Rendering latency vs. display latency. There are good reasons to decouple display and rendering frequencies when the rendering frequency is below the display frequency. In this "rendering latency" cases, warping is adequate. Rolling rasterization however, does not address rendering latency, but assumes the rendering is faster than the display. It compensates for "display latency": the amount of time a pixel is outdated when shown on screen.

The differences between an ideal ground truth zero-latency rasterizer, rendering latency compensated by warping and our approach to compensate display latency are illustrated in Fig. 16. The ground truth approach without rendering or display latency (orange), would instantaneously produce an image that at every position in space-time will match the view transform. Normal rasterization preceding warping (light blue) will render frame $n + 1$ with the transform known at time t_1. By t_3, the end of frame $n + 1$, the display image will be severely outdated (difference E1). This will not change by making a renderer faster, but is a property of the display rate.

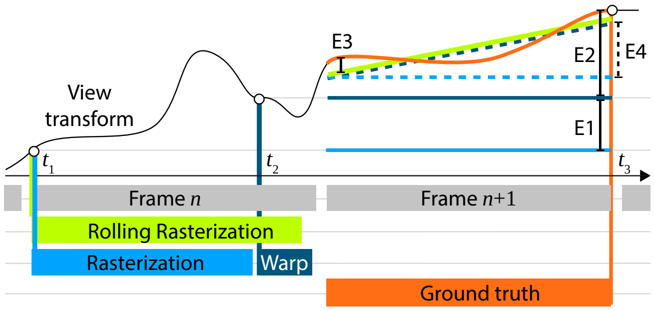

Fig. 16. Conceptual differences of our approach and warping to ground truth when producing frame $n + 1$. Time is the horizontal axis and the vertical axis is view transformation. Differences in images, and by this the perceived error, are likely proportional to differences in view transform (dotted lines). The ground truth view transform is shown as a single function curve for frame n and $n + 1$. Different methods are encoded as colors. Colored horizontal blocks are working time, colored lines are approximations of the view transform for images on display.

Warping (dark blue), will move pixels to compensate for the transformation at t_2, but can still not mitigate the image to become outdated during $n + 1$, (difference E2) and it has no way to remedy disocclusions occurring between t_1 and t_2. Our approach (green) also starts work at t_1, but using the transformation predicted for continuous points in time on frame $n + 1$, removing all occlusion and shading error and leaving only the transformation prediction error $E3$ but no error due to display latency. Even when assuming a hypothetical and unpublished competitor that rasterizes using a predicted view transform (dotted light blue line) and a rolling form of warping (dark blue dotted line), there remains an appearance error E4 at t_4 that can not ever be resolved by rasterizing outdated (i. e., non-rolling) occlusion and shading.

Fast Rendering. It is tempting to just hope faster rendering will make rolling rasterization obsolete. But any common non-rolling method will never reduce latency below the scan-out duration, typically 8.3-16 ms. Even if a fast non-rolling rasterization takes only 1 ms (a short light-blue bar in Fig. 16), the scan-out still takes 16 ms, and the latency will remain to be 17 ms by the end of the display period.

Our method slightly increases image synthesis time, but this does not matter, as long as it terminates before the buffer swap. Assuming image synthesis to take 10 ms, our approach might increase it to 14 ms, but both are below cut-off at 16 ms, so there is no drawback here. However, our image will not have any latency for the next 16 ms during display, while a common image, computed 4 ms faster for no reason, will accumulate up to 16 ms by the end of the display interval.

Prediction. Like any method that has to finish before the scan-out starts, we require a prediction of scene and viewer motion during scan-out. Grossmann et al. [1988] have measured the velocity and acceleration of head motions. Their results show that rotational and translational head velocity can be substantial, indicating that the rendering with a view transform that changes during the display interval is useful. They also find, that the acceleration i. e., derivation form a linear model, is small as it requires force. This indicates that our first order-model, with substantial velocity but limited acceleration, is physiologically plausible.

Eye/head/render/display constraints. Please also note, that while there are pros and cons of decoupling the display and rendering rate to compensate display and rendering latency, foveated rendering puts certain additional limits on how decoupled they can become: if foveated rendering should be tightly coupled with (predicted) eye position, there is no reason to not also tightly couple rolling rasterization and (predicted) head pose. Our work jointly does both, and while it is possible to decouple both, it is difficult to imagine how to decouple only one of them. Hence, as foveated rendering has the same coupling requirements rolling rasterization has, it is appropriate and efficient to do both jointly, as we suggest here.

Streaming. Friston et al. [2016] update the view matrix for each scan-line and ray-trace a simplistic scene in a period far below that of the display's scan out. It would not be clear how to ray-trace a complex scene in this time. Geometry in animated scenes changes for every scan line, which would require very high frequency BVH rebuilds. In our case of streaming OpenGL rasterization, which maps primitives to pixels, we have no guarantees on the space or time layout of the primitive stream. Consequently, we need to predict the head pose across the scan-out. Prediction is essential and cannot be omitted. Even if a sensor could give the absolute viewpoint continuously, there is still the delay due to rendering the image from this viewpoint, and therefore an interval between the rasterization and the actual scan-out. We further assume the change in transformation is small enough that the transform matrices can be linearly interpolated; an optimization that could be replaced with a more advanced interpolation.

Speed. We demonstrate a prototypical implementation using a GPU, which has speed comparable non-rolling or non-foveated implementations. Our current implementation runs at interactive rates, suggesting a full hardware implementation (with optimizations such as tiling, etc. [Akenine-Möller et al. 2007]) could achieve speeds similar to a traditional rasterizer.

Joint analysis. We have derived bounds for joint foveated-rolling rasterization and show example results in Fig. 9, but did not conduct a perceptual (stereo) experiment for this combination.

Periphery. Similar to other methods [Guenter et al. 2012; Patney et al. 2016; Stengel et al. 2016] our foveated rasterization can create temporal aliasing in the periphery, where humans are unfortunately particularly sensitive. Future work will investigate specialized spatio-temporal filters to circumvent this issue.

Note, that our filtering from the cortical to the display domain, used both for kernelized rendering and ours, employs non-linear filters, that properly handle the complex anisotropic shape (curved ellipsoid) to which a display maps in the periphery of the cortical domain.

Screen-space effects. Screen space shading needs to be adapted to support perceptual rasterization and kernelized foveated rendering [Meng et al. 2018]. We have done so for SSAO by multiplying all image distances by the pixel density $p(\mathbf{x})$.

8 CONCLUSION

In this paper we introduced a new efficient rasterization technique that exploits the spatio-temporal-retinal relationship of rays and primitives found in HMDs. It prevents the artifacts and overhead of warping and works in a single pass while supporting moving objects, viewer translation and rotation as well as specular shading and lens distortion - all of which are challenging for warping. The main technical contribution is the derivation of tight and efficiently computable primitive bounds.

Future investigations could extend the rolling concept to physics and other simulations, and would also need to seek better understanding of the relationship between latency and motion blur, focus and the role of eye and head motion, and the behavior with other types of stereo display.

ACKNOWLEDGMENTS

UK EPSRC (grant reference EP/P004016/) project *Context Aware network architectures for Sending Multiple Senses* (CASMS).

REFERENCES

Tomas Akenine-Möller, Jacob Munkberg, and Jon Hasselgren. 2007. Stochastic rasterization using time-continuous triangles. In *Proc. Graphics Hardware.* 9.

Tomas Akenine-Möller, Robert Toth, Jacob Munkberg, and Jon Hasselgren. 2012. Efficient Depth of Field Rasterization Using a Tile Test Based on Half-Space Culling. *Comp. Graph. Forum* 31, 1 (2012).

Michael Antonov. 2015. https://developer3.oculus.com/blog/asynchronous-timewarp-examined/. (2015).

Stefan Brabec, Thomas Annen, and Hans-Peter Seidel. 2002. Shadow mapping for hemispherical and omnidirectional light sources. *Advances in Modelling, Animation and Rendering* (2002), 397–408.

John Brosz, Faramarz F Samavati, M Sheelagh T Carpendale, and Mario Costa Sousa. 2007. Single camera flexible projection. In *Proc. NPAR.* 33–42.

J. S. Brunhaver, K. Fatahalian, and P. Hanrahan. 2010. Hardware Implementation of Micropolygon Rasterization with Motion and Defocus Blur. In *Proc. HPG.* 1–9.

Timothy J. Buker, Dennis A. Vincenzi, and John E. Deaton. 2012. The Effect of Apparent Latency on Simulator Sickness While Using a See-Through Helmet-Mounted Display: Reducing Apparent Latency With Predictive Compensation. *Human Factors* 54, 2 (2012), 235–249.

Shenchang Eric Chen and Lance Williams. 1993. View interpolation for image synthesis. In *Proc. SIGGRAPH.* 279–88.

PM Daniel and D Whitteridge. 1961. The representation of the visual field on the cerebral cortex in monkeys. *J Physiology* 159, 2 (1961), 203–21.

Piotr Didyk, Elmar Eisemann, Tobias Ritschel, Karol Myszkowski, and Hans-Peter Seidel. 2010. Perceptually-motivated Real-time Temporal Upsampling of 3D Content for High-refresh-rate Displays. *Comp. Graph. Forum* 29, 2 (2010), 713–22.

S.R. Ellis, B.D. Adelstein, S. Baumeler, G.J. Jense, and R.H. Jacoby. 1999. Sensor spatial distortion, visual latency, and update rate effects on 3D tracking in virtual environments. In *Proc. VR.* 218–21.

Kayvon Fatahalian, Edward Luong, Solomon Boulos, Kurt Akeley, William R Mark, and Pat Hanrahan. 2009. Data-parallel rasterization of micropolygons with defocus and motion blur. In *Proc. HPG.* 59–68.

FOVE. 2018. https://www.getfove.com/. (2018).

Sebastian Friston, Anthony Steed, Simon Tilbury, and Georgi Gaydadjiev. 2016. Construction and Evaluation of an Ultra Low Latency Frameless Renderer for VR. *IEEE TVCG* 22, 4 (2016), 1377–86.

George W Furnas. 1986. Generalized fisheye views. In *Proc. CHI.*

Jean-Dominique Gascuel, Nicolas Holzschuch, Gabriel Fournier, and Bernard Peroche. 2008. Fast non-linear projections using graphics hardware. In *Proc. i3D.* 107–14.

Ragnar Granit and Phyllis Harper. 1930. Comparative studies on the peripheral and central retina. *J Physiology* 95, 1 (1930), 211–28.

Gerard E Grossman, R John Leigh, LA Abel, Douglas J Lanska, and SE Thurston. 1988. Frequency and velocity of rotational head perturbations during locomotion. *Exp. Brain Res.* 70, 3 (1988), 470–6.

Brian Guenter, Mark Finch, Steven Drucker, Desney Tan, and John Snyder. 2012. Foveated 3D graphics. *ACM Trans. Graph. (Proc. SIGGRAPH)* 31, 6 (2012), 164.

Yong He, Yan Gu, and Kayvon Fatahalian. 2014. Extending the graphics pipeline with adaptive, multi-rate shading. *ACM Trans. Graph (Proc. SIGGRAPH)* 33, 4 (2014), 142.

Samuli Laine, Timo Aila, Tero Karras, and Jaakko Lehtinen. 2011. Clipless dual-space bounds for faster stochastic rasterization. *ACM Trans. Graph (Proc. SIGGRAPH)* 30, 4 (2011), 106.

Baoquan Liu, Li-Yi Wei, Xu Yang, Chongyang Ma, Ying-Qing Xu, Baining Guo, and Enhua Wu. 2011. Non-Linear Beam Tracing on a GPU. *Comp. Graph. Forum* 30, 8 (2011), 2156–69.

William R Mark, Leonard McMillan, and Gary Bishop. 1997. Post-rendering 3D warping. In *Proc. i3D.* 7–ff.

Morgan McGuire, Eric Enderton, Peter Shirley, and David Luebke. 2010. Real-time stochastic rasterization on conventional GPU architectures. In *Proc. HPG.* 173–82.

Xiaoxu Meng, Ruofei Du, Matthias Zwicker, and Amitabh Varshney. 2018. Kernel Foveated Rendering. *Proc. ACM Comput. Graph. Interact. Tech.* 1, 1 (2018), 5:1–5:20.

Nvidia. 2017. NV_clip_space_w_scaling OpenGL extension. (2017).

Oculus. 2018. https://developer.oculus.com/documentation/mobilesdk/0.4/concepts/mobile-timewarp-overview/. (2018).

Oculus VR. 2017. Asynchronous TimeWarp. (2017).

Anjul Patney, Marco Salvi, Joohwan Kim, Anton Kaplanyan, Chris Wyman, Nir Benty, David Luebke, and Aaron Lefohn. 2016. Towards foveated rendering for gaze-tracked virtual reality. *ACM Trans. Graph. (Proc. SIGGRAPH)* 35, 6 (2016), 179.

Juan Pineda. 1988. A parallel algorithm for polygon rasterization. *ACM SIGGRAPH Computer Graphics* 22, 4 (1988), 17–20.

Voicu Popescu, Paul Rosen, and Nicoletta Adamo-Villani. 2009. The graph camera. *ACM Trans. Graph.* 28, 5 (2009), 158.

Matthew Regan and Ronald Pose. 1994. Priority rendering with a virtual reality address recalculation pipeline. In *Proc. SIGGRAPH.* 155–162.

Bernhard Reinert, Johannes Kopf, Tobias Ritschel, Eduardo Cuervo, David Chu, and Hans-Peter Seidel. 2016. Proxy-guided Image-based Rendering for Mobile Devices. *Comp. Graph. Forum (Proc. Pacific Graphics)* 35, 7 (2016).

Andre Schollmeyer, Simon Schneegans, Stephan Beck, Anthony Steed, and Bernd Froehlich. 2017. Efficient Hybrid Image Warping for High Frame-Rate Stereoscopic Rendering. *IEEE Trans. Vis. and Comp. Graph.* 23, 4 (2017), 1332–41.

Mel Slater. 2002. Presence and The Sixth Sense. *Presence* 11, 4 (2002), 435–439.

AAS Sluyterman. 2006. What is needed in LCD panels to achieve CRT-like motion portrayal? *J SID* 14, 8 (2006), 681–686.

Michael Stengel, Steve Grogorick, Martin Eisemann, Elmar Eisemann, and Marcus A Magnor. 2015. An affordable solution for binocular eye tracking and calibration in head-mounted displays. In *Proc. ACM Multimedia.* 15–24.

Michael Stengel, Steve Grogorick, Martin Eisemann, and Marcus Magnor. 2016. Adaptive Image-Space Sampling for Gaze-Contingent Real-time Rendering. *Comp. Graph. Forum* 35, 4 (2016), 129–39.

Qi Sun, Fu-Chung Huang, Joohwan Kim, Li-Yi Wei, David Luebke, and Arie Kaufman. 2017. Perceptually-guided foveation for light field displays. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 36, 6 (2017), 192.

Robert Toth, Jim Nilsson, and Tomas Akenine-Möller. 2016. Comparison of projection methods for rendering virtual reality. In *Proc. HPG.* 163–71.

Ingo Wald, Sven Woop, Carsten Benthin, Gregory S Johnson, and Manfred Ernst. 2014. Embree: a kernel framework for efficient CPU ray tracing. *ACM Trans. Graph (Proc. SIGGRAPH)* 33, 4 (2014), 143.

Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. 2004. Image quality assessment: from error visibility to structural similarity. *IEEE Trans Image Proc.* 13, 4 (2004), 600–12.

Martin Weier, Thorsten Roth, Ernst Kruijff, André Hinkenjann, Arsène Pérard-Gayot, Philipp Slusallek, and Yongmin Li. 2016. Foveated Real-Time Ray Tracing for Head-Mounted Displays. *Comp. Graph. Forum* 35, 7 (2016), 89–298.

Martin Weier, Michael Stengel, Thorsten Roth, Piotr Didyk, Elmar Eisemann, Martin Eisemann, Steve Grogorick, André Hinkenjann, Ernst Kruijff, Marcus Magnor, et al. 2017. Perception-driven Accelerated Rendering. *Comp. Graph. Forum* 36, 2 (2017), 611–43.

P.H. Wicksteed and F.M. Cornford. 1929. *Aristotle. Physics.* W. Heinemann.

Lei Yang, Yu-Chiu Tse, Pedro V Sander, Jason Lawrence, Diego Nehab, Hugues Hoppe, and Clara L Wilkins. 2011. Image-based bidirectional scene reprojection. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 30, 6 (2011), 150.