Perceptual Rasterization for Head-mounted Display Image Synthesis

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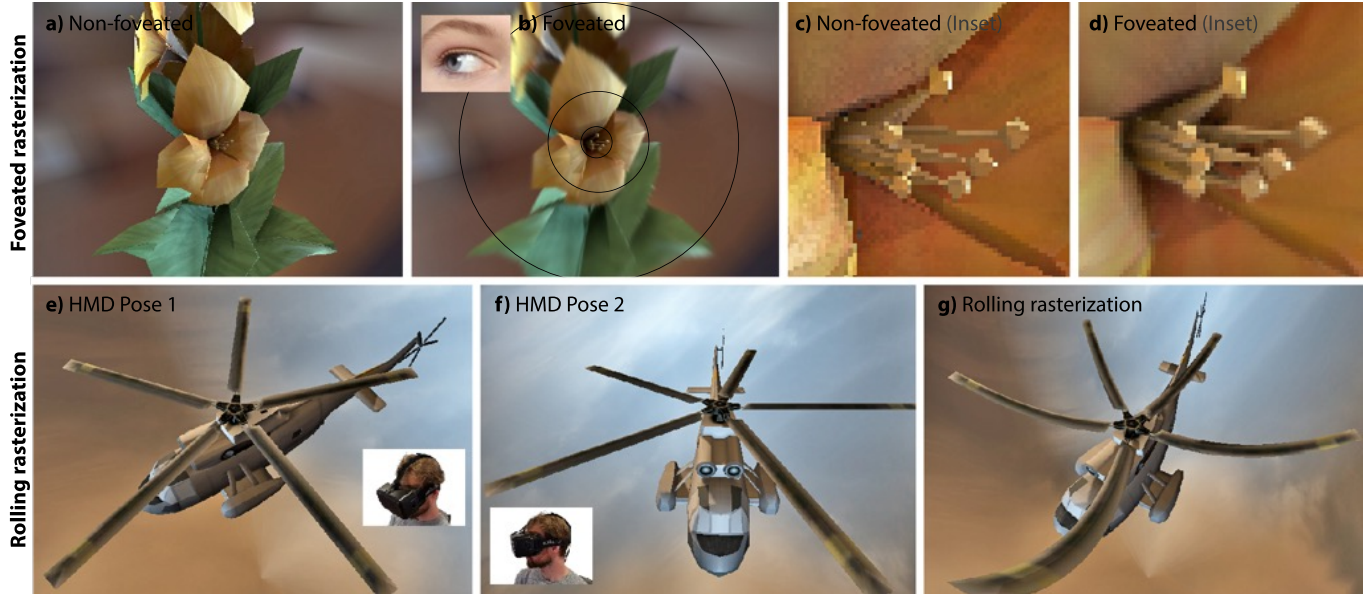ANTHONY STEED, University College London

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 a 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HMD display, where pixels are illuminated at different points in time, latency is introduced: the rightmost pixel is outdated by ca. 16 ms.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 need of HMDs, where latency and field-of-view requirements pose new challenges beyond those of traditional desktop displays. Instead of rendering and 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 requisite geometric tests, but retaining most of the the simplicity and efficiency of the traditional rasterizaton pipeline. To produce foveated images, we rasterize to an image with spatially varying pixel density. To reduce 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.

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 characteristics of HMDs, 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 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.

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).
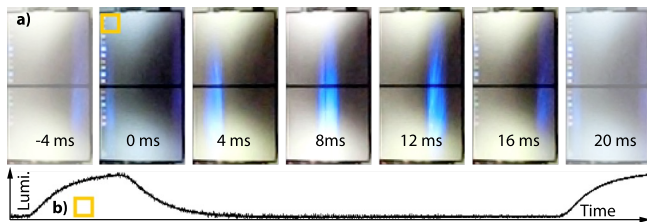

Fig. 2. **a)** Seven frames (24 ms) high-speed capture (Casio Exilim EX-ZR1000) of an HDK 2 HMD (twin) display. Specific locations are illuminated *(blue)* at specific points in time. **b)** Time-varying illumination of a 4 mm band of an Oculus DK2 display captured with a photodiode and a PicoScope 6402B.

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. 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-tracing 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. It is also not clear how a modern ray tracer making use of bounding volume hierarchies (BVH) would handle a more continuous approximation of frame time. Rebuilding the BVH for every pixel would certainly be less than ideal.

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 Warping* (ATW) (Antonov 2015) incorporate a number of these techniques to compensate for multiple sources of latency. The main drawback of warping is that it suffers disocclusion artefacts. 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 artefacts, 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). Toth et al. (2016) suggest single-pass rendering into spatially neighboring but linear sub-projections 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 (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 rasterisation 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 information visualization (Furnas 1986) and directly from cortical magnification theory (Daniel and Whitteridge 1961): 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.
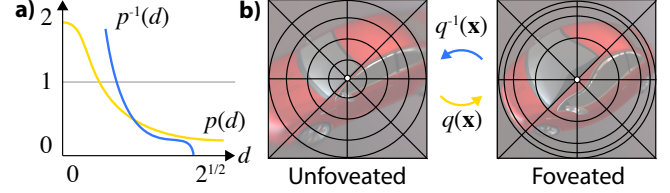

Fig. 3. Foveation and unfoveation function *(a)* and domains *(b)*.

Domain. We suggest an image domain where the ray (or pixel) density depends on a function $p(d) \in (0, \sqrt{2}) \to \mathbb{R}^+$, where d is the distance to the foveation point $\mathbf{x}_f$. 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).

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 $q(\mathbf{x}) \in (-1, 1)^2 \to (-1, 1)^2$: $\mathbf{x}_f + norm(\mathbf{x} - \mathbf{x}_f) \cdot p(||\mathbf{x} - \mathbf{x}_f||)$. This function essentially scales $\mathbf{x}$ by p, away from the gaze position. 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 q^{-1}, to be q but with p^{-1} in place of p. p^{-1} is the inverse of p. 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. q and q^{-1} use these functions to do the same for pixel locations. We refer to these pixel transformations as to "foveate" and "unfoveate". 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.

Display. After rasterizing 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: q^{-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 and then evaluate $I_u(\mathbf{x}) = I_f(q^{-1}(\mathbf{x}))$ 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

$$L_d(\mathbf{x}) = \sum_{y \in 5\times5} L_c(q(\mathbf{x}) + \mathbf{y}) \cdot r(||\mathbf{x} - q^{-1}(q(\mathbf{x}) + \mathbf{y}))||),$$

where L_d is the display image, L_c the foveated imaged, and r an arbitrary, e. g., Gaussian, reconstruction filter parametrized by distances in the display image domain. Such an operation effectively
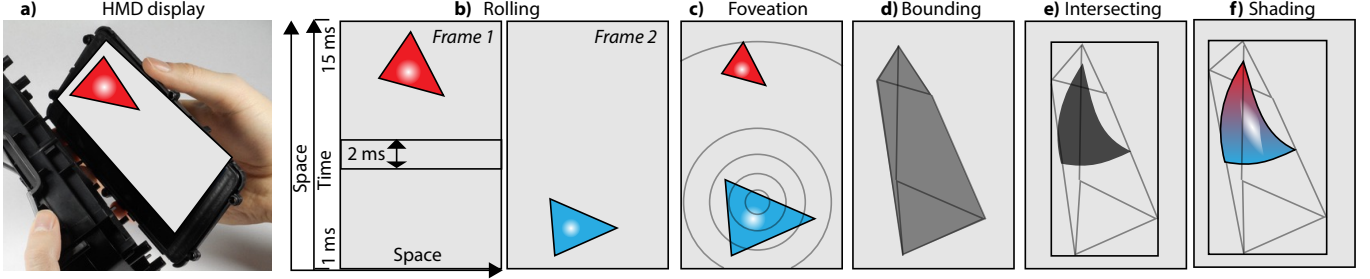
Fig. 4. Overview of perceptual rasterization. Common rasterization **(a)** produces images at a fixed time and uniform pixel density. We suggest to account for primitive motion, here shown as two frames **(b)** and non-uniform pixel density, here visualized as iso-lines **(c)**. Primitive-ray interaction 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.)

computes the (irregular-shaped) projection 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 $[t_s, t_e]$.

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 Cathode Ray Tube (CRT) is an example of a rolling scan display. Most LCDs these days perform a global 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 $r(\mathbf{x}) \in (0,1)^2 \rightarrow (0,1) : \mathbf{x} \cdot \mathbf{d}$ 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 scanout 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 vertices move along linear paths during that time. More general motion is possible, but not for the tightest bound (Zenon's bound), which uses an analytic derivation requiring linearity.

3.1.3 Joint foveated-rolling. The composition $r \circ q(\mathbf{x})$ of the above.

3.2 Pipeline

An overview of perceptual rasterization is see 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 ($M_s v_s$ and $M_e v_e$), and passes this information on to the GP. 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 2D primitive, such that all pixels that would at any point in time be affected by the triangle are covered by the new bounding primitive B. 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 that could be affected by the primitive's bound. Note that this test is the same regardless of what bounding is used. To decide if the pixel $\mathbf{x}_i$ actually is affected by the space-time triangle, we intersect the ray R_i at this pixel with the triangle at time $r(\mathbf{x}_i)$. The entire triangle, its normals, texture coordinates and material information, were emitted as `flat` attributes from the GP. Note, that R depends on the time as well: every pixel i has to ray-trace the scene at a different time following r. For foveation, R_i is not formed by a pin-hole model but follows q. The joint model distributes rays according to $r \circ q$. The position of the entire triangle at time $r(\mathbf{x}_i)$ is easily found by linear interpolation of the vertex motion. This results in a camera-space triangle T_i, that can be intersected with R_i using a 3D 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. This will resolve the correct (i. e., nearest to the viewer) fragment information. For every pixel there is a unique time and fovea location, and hence distances of multiple 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.

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 by simple 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.
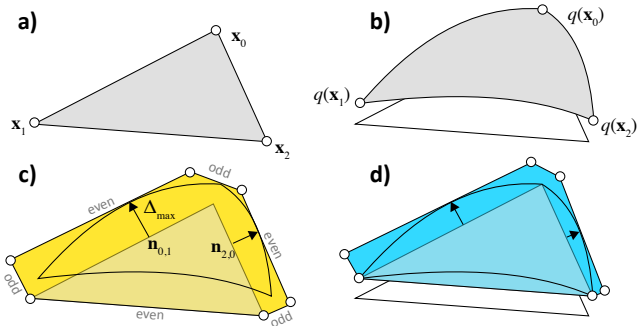
4.1 Foveation bounds

Fig. 5. Foveated bounding. **a)** the original primitive. **b)** the foveated primitive. **c)** the simple bounds displaces the original edges. **d)** the advanced bound first foveates the edges and then bounds the displacement.

As there is no closed-form foveation function available, we cannot derive a closed-form solution as will be done for rolling rasterization. This might become possible in future work when using an analytic foveation function. For now, the monotonicity property still allows for tight bounds that are quick to compute (Fig. 5).

The key is to use q and q^{-1}. The bounding geometry we generate 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. The remaining task is then to bound a single triangle edge from $\mathbf{x}_0$ to $\mathbf{x}_1$. We have derived two bounds, a simple and a tighter recursive bound.

4.1.1 Simple. Here, the bounding edge is assumed to be parallel to the original edge (Fig. 5,c) All we need to find is the maximal positive distance along the normal from the edge joining $\mathbf{x}_0$ and $\mathbf{x}_1$

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

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

where n creates a direction orthogonal to the line between its two arguments. As the distance is a convex function, it can be minimized using a ternary search that 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.

4.1.2 Recursive. Consider the original vertices $\mathbf{x}_0$ and $\mathbf{x}_1$ (Fig. 5, a) and the foveation $q(\mathbf{x}_0)$ and $q(\mathbf{x}_1)$ of these vertices (Fig. 5, b). While the simple bound displaces relative to the straight original edge from $\mathbf{x}_0$ to $\mathbf{x}_1$ (Fig. 5, c) the new recursive bound will displace relative to the straight edge $q(\mathbf{x}_0)$ to $q(\mathbf{x}_1)$ (Fig. 5, d):

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

This is possible, as the edge has to be straight, but not necessarily the "original" one. The resulting bound is tighter, i. e., the blue area is smaller than the yellow one in Fig. 5. Note, that the normal for a different straight edge is also different, as q 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.
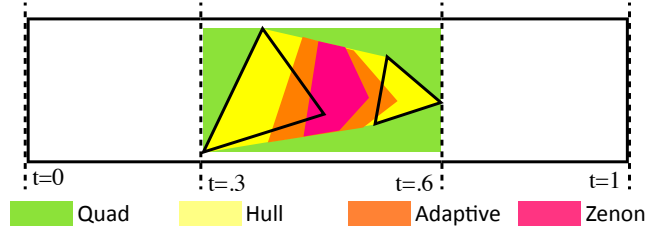
4.2 Rolling bounds

Fig. 6. Rasterization bounds for a space-time triangle moving across the screen. The triangle starts at a position where the frame time already is .3 and ends where frame time is .6. Consequently, it can not cover the full convex hull, but only the convex hull of a spatio-temporal subset. We identify this region, resulting in an increased sample test efficiency (cf. the ratio of areas of "Hull" and "Adaptive"). Finally, there is an analytic solution to when exactly the rolling beam will catch up with a moving primitive allowing for even tighter bounds ("Zenon").

4.2.1 Boxes. A reasonably tight bound for the time-space triangle S as defined in Sec. 3.2, is the 2D bounding box

$$B = \mathrm{bbox}\{\mathcal{P}(S_{i,j}, t) | i \in \{\mathrm{s, e}\}, j \in \{0, 1, 2\}, t \in \{0, 1\}\}$$

of all vertices in the start and end of the frame, where bbox builds the 2D bounding box of a set of points and $\mathcal{P}$ is the projection of a point at time t, i. e., multiplication with a time-varying matrix followed by a homogeneous division ("Quad" in Fig. 6).

4.2.2 Convex hull. A bounding box would create substantial overdraw for thin and diagonal primitives. Investing time to produce tighter bounding primitives can be worthwhile as it reduces the amount of work done for each pixel ("Hull" in Fig. 6). Fortunately, all points of a triangle under linear motion fall into the convex hull of its vertices (Akenine-Möller et al. 2007). We can therefore replace the operator bbox by the convex hull of a set hull that could be implemented efficiently (McGuire et al. 2010) (our current implementation uses a GLSL quick hull implementation). For primitives intersecting the near plane we proceed as 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 their convex hull is used. We found using a convex hull of up to 15 points (there are 15 edges between 6 space-time vertices (McGuire et al. 2010)) resulted in higher overall performance than when using the simpler bounding box.

4.2.3 Adaptive. While convex hulls are tight spatially, the rolling case allows for a surprisingly tighter bound under some simple and reasonable assumptions on w, the mapping from pixel locations to frame times ("Adaptive" in Fig. 6). The key observation is that a rolling space-time triangle only has to cover

$$B = \text{hull}\{\mathcal{P}(S_{i,j}, t) | i \in \{s, e\}, j \in \{0, 1, 2\}, t \in \{t_{\min}, t_{\max}\}\},$$

where the triangle-specific time interval $(t_{\min}, t_{\max})$ is found by mapping back 2D position to time

$$t_{\min} = \min\{w^{-1} \mathcal{P}(S_{i,j}, t) | i \in \{s, e\}, j \in \{0, 1, 2\}, t \in \{0, 1\}\}.$$

The maximal time $t_{\max}$ is defined by replacing the minimum with a maximum operation. In other words, to bound, we first project all six vertices with time 0 and 1, to get bounds in 2D but then find the maximal and minimal time at which these pixels would be relevant. As this time span is usually shorter than the frame i. e., $t_{\min} \gg t_s$ and $t_{\max} \ll t_e$, the spatial bounds also get tighter.

4.2.4 Zenon's hull. The problem of bounding where the rolling scan will "catch up" with the projection of a moving triangle has similarity with Zenon's paradoxon where Achilles tries to catch up with the tortoise (Wicksteed and Cornford 1929) (Fig. 7, a).
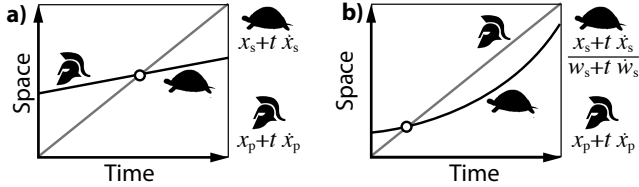

Fig. 7. Linear (**a**) and perspective (**b**) Zenon's paradoxon (see text below).

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

$$x_s + t\dot{x}_s = x_p + t\dot{x}_p, \quad \text{which occurs at} \quad t = \frac{x_s - x_p}{\dot{x}_s - \dot{x}_p}.$$

The same holds for a rolling scan (Achilles) catching up with a vertex (tortoise). Regrettably, in our case, the rolling scan moves in image space, while the primitive moves in a 2D projective space (horizontal x component and projective coordinate w) from spatial position x with speed $\dot{x}$ and projective position w with speed $\dot{w}$ (Fig. 7, b). This can be stated as

$$x_s + t\dot{x}_s = \frac{x_p + t\dot{x}_p}{w_p + t\dot{w}_p},$$

which is a rational polynomial with a unique positive solution

$$t = -\frac{(\sqrt{4x_s \dot{w}_p + \dot{x}_s^2 - 2\dot{x}_s w_p + w_p^2} - \dot{x}_s + w_p)}{2\dot{w}_p}. \tag{1}$$

To produce the final bounds, the time t_i, and the 2D position x_i at this time, is computed for each of the six vertices of the space-time triangle. The convex hull of the x_i is the final bounding geometry.

4.3 Joint Foveated-rolling bounds

A joint approach for rolling and foveation operates similarly to the foveation-only approach. To add rolling to foveation, we add the rolling transformation to q (Fig. 8). Order is important: the rolling time coordinate has to depend on where the pixel will effectively be displayed in the non-foveated domain. Let $\mathbf{x}_0$ and $\mathbf{x}_1$ be the original world coordinates of that edge The new edge functions are therefore

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

where Q is the joint action of rolling and foveation $Q(\mathbf{x}) \in \mathbb{R}^3 \to \mathbb{R}^2 : Q(\mathbf{x}) = q(\mathcal{P}(\mathbf{x}, t))$. The time t can be found using Eq. 1.
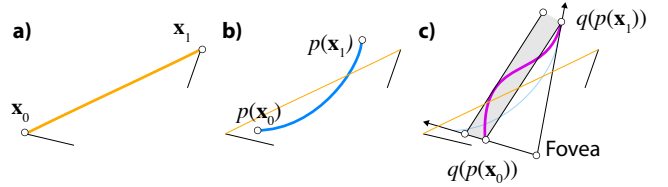

Fig. 8. Joint rolling-foveated rasterization. *a)* One original edge of a primitive in orange. *b)* rolling of the same edge results in the blue curve. *c)* Foveation of that curve leads to another pink curve, that is bound from the line joining its ends, adding the gray area.

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. 9. 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 common 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. 10. 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
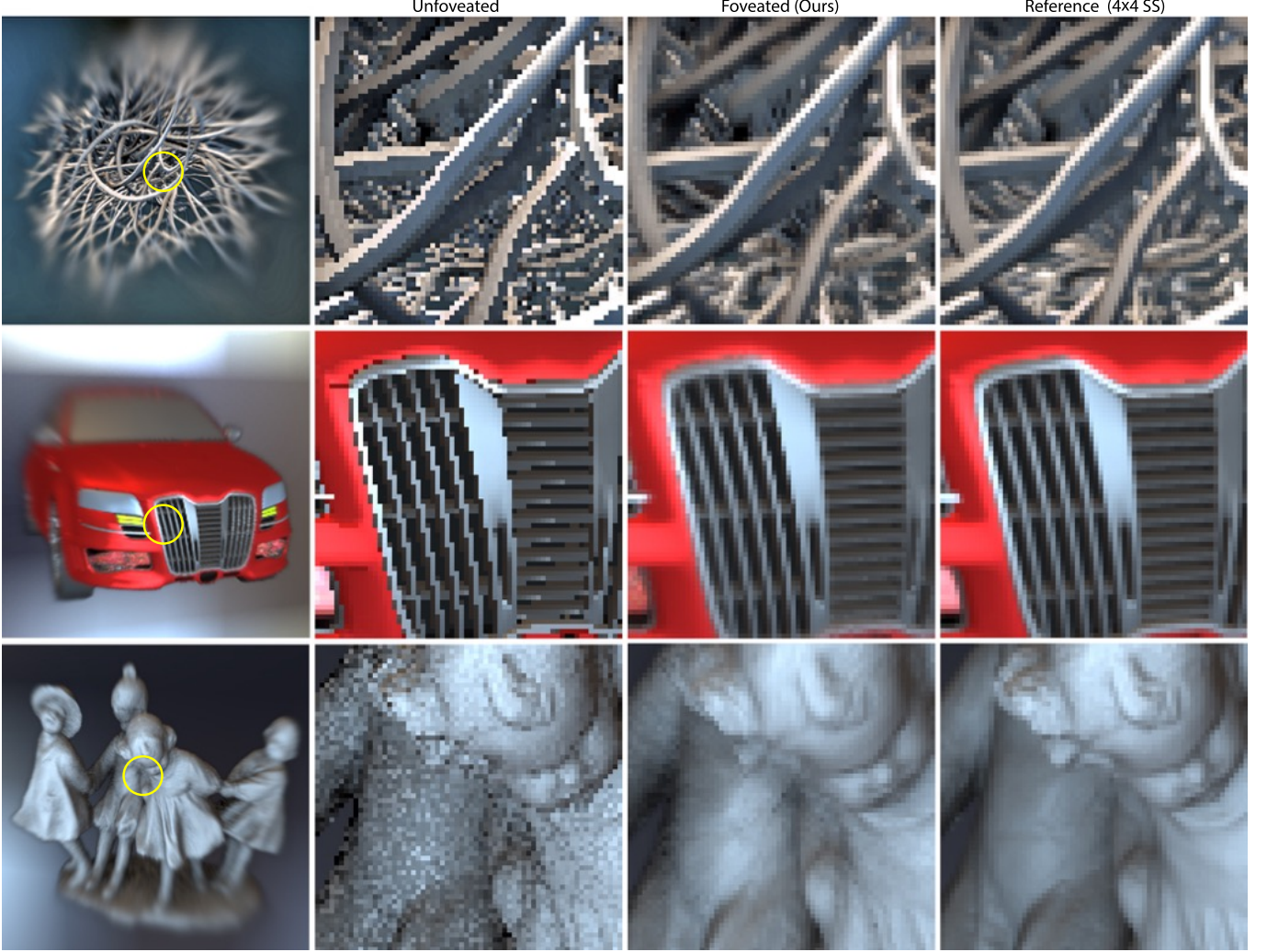
Fig. 9. Foveation results. The first column shows the result we produce, fovea marked in yellow. The second to fourth columns shows the foveated region using non-foveated rendering, our approach, and a 4 × 4 super-sampling reference. Quantitative evaluation is found in Tbl. 1.

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 row. The scenes were intentionally chosen to contain large polygons, which are difficult for non-linear projections (Brosz et al. 2007; Gascuel et al. 2008).

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, and the ground plane in HOUSES, 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.

Joint rasterization. Results for joint rolling-foveated images are show in Fig. 11. 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 100,00 k faces and render in less than 50 ms.

Lens Distortion. Including a barrel lens distortion (Oculus VR 2017) in the joint approach is simple (Fig. 12): we just use a foveation function p that is a composition $p(d) = p_c \circ p_l(d)$ of the cortical foveation p_c function and a lens distortion function p_l. When sampling back from the foveated domain, only p_c is applied, as p_l will happen optically. Only the – much smaller – chromatic blur still

7

| Input | Warping | Ours | Warping | Ours |

Fig. 10. 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 Tbl. 2. Please, see the supplemental video for animated versions of these results.
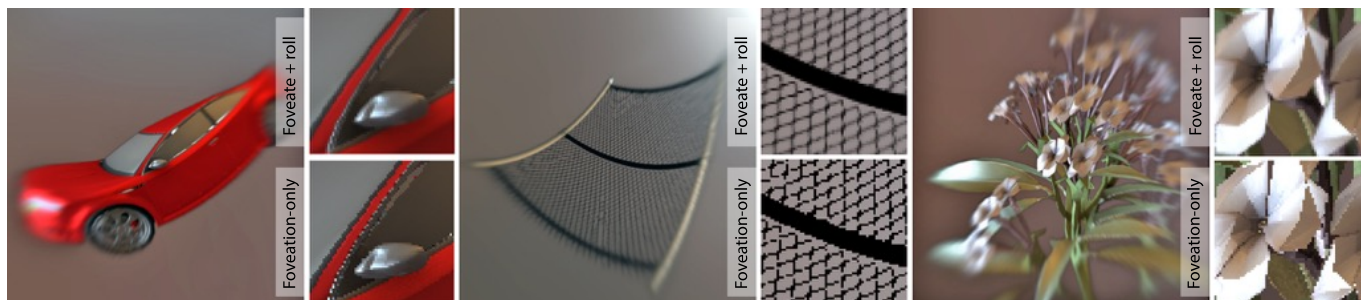

Fig. 11. Joint i. e., rolling and foveated, perceptual rasterization for three scenes. The insets compare joint and rolling-only results.

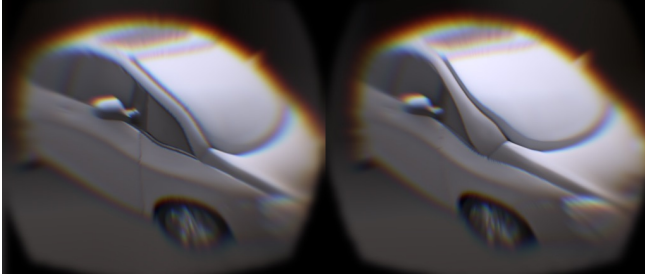needs to be applied, as the effort of rasterizing three channels independently does not appear justified.

Fig. 12. This stereo image is both rolling and foveated, as well as it will appear lens-undistorted in space and chroma when observed in a HMD.

Rolling Shading. Here we compare rolling shading, included in all the above results, to rolling rasterization without rolling shading in Fig. 13. 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.
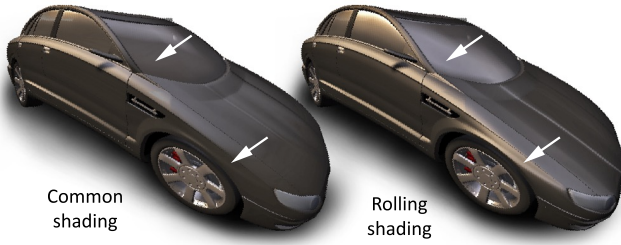

Fig. 13. Rolling rasterization without rolling shading *(left)* lack some specular effects. Rolling shading *(right)* produces highlights that change across the image due to the change in view over time.

5.2 Quantitative

Here alternatives and variants of our approach are compared in terms of speed and image similarity.

Methods. We tested our approach on a Nvidia Quadro K6000. Image similarity is measured in terms of an adapted SSIM (Wang et al. 2004) metric. It ignores all disoccluded pixels, i. e., it provides 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, SSIM is computed for the 64×64 foveal pixels. For foveation, we compare speed and quality to a common method, that directly operates at the same resolution as ours, and speed to a three-layered method (Guenter et al. 2012) assuming it will provide similar quality. The foveated reference is a 8 × 8 super-sampled rasterized image. 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. As rolling methods we compare "No rolling" corresponding to the first column in Fig. 10, "Warping" from the second column in Fig. 10 and our "Rolling" approach from the third column in Fig. 10. The rolling reference is ray-traced, that is, identical to all images perceptual rasterization produces. We state

the ray-tracing time of a reasonably implemented GPU traversal of an SAH-optimized BVH.

Comparison. Foveation results are shown in (Tbl. 1). Our approach is more similar to the reference than common rasterization. Furthermore, it achieves speed that is roughly half as fast rasterizing multiple layers and very similar to rendering in a full resolution. Finally, we see that refined bounds increase sample test efficiency as well as actual compute time.

Rolling results are stated in Tbl. 2. First, we see that rolling and non-rolling images are substantially different according to the SSIM metric. At the same time, classic GPU rasterization is highly optimized and produces images very quickly. When warping the image, the similarity increases, but time is increased by two milliseconds: high-quality warping requires two primitives per pixel (Mark et al. 1997). Next, we compare our method using different bounds. Note, that the SSIM is always 1 as our rasterization has been verified to be identical to ray-tracing a rolling shutter image. 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.

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.

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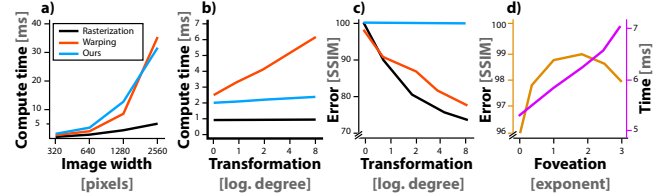

Fig. 14. Comparison of different rolling approaches in HELICOPTER: Classic rasterization, warping and rolling. **a)** Image resolution and compute time (less is better). **b)** Transformation (a camera rotation) and compute time (less is better). **c)** Transformation and image similarity in SSIM (more is better). Scalability of foveation in CAR: Similarity *(yellow line)* and compute time *(pink line)* as a function of foveation.

Sample Test Efficiency. We also compute the sample test efficiency (STE) (Akenine-Möller et al. 2007; 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 Tbl. 1 and Tbl. 2, 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. 14. The first plot shows how image resolution affects computation time (Fig. 14, a).

Table 1. Qualitative evaluation of foveated rasterization from Fig. 9. Layered SSIM is assumed to be 0.

	Tris	Ours							Common			Layered		Raytrace	Reference	
			Trivial	Quad		Recursive										
		Sim.	Shade	Raster	STE	Raster	STE	Raster	STE	Sim.	Shade	Raster	Shade	Raster	Raster	Shade	Raster

	Tris	Sim.	Shade	Raster	STE	Raster	STE	Raster	STE	Sim.	Shade	Raster	Shade	Raster	Raster	Shade	Raster
Hairball	115 k	.988	2.2 ms	28.7 ms	5.0 %	9.8 ms	10.1 %	5.2 ms	40.0 %	.970	2.2 ms	1.1 ms	6.6 ms	2.7 ms	280.0 ms	228.4 ms	3.5 ms
Car	178 k	.992	2.2 ms	35.2 ms	1.0 %	12.2 ms	14.3 %	6.5 ms	37.3 %	.962	2.2 ms	1.4 ms	6.6 ms	3.3 ms	33.2 ms	228.4 ms	4.6 ms
Children	1,400 k	.992	2.2 ms	3 s	0.0%	30.0 ms	16.0 %	29.3 ms	48.2 %	.938	2.2 ms	2.9 ms	6.6 ms	7.9 ms	48.5 ms	228.4 ms	13.8 ms

Table 2. Qualitative evaluation of rolling rasterization from Fig. 10. Our SSIM is 1 in all conditions.

| Scene	Tris	No Rolling		Warping		Rolling (ours)									Raytrace
							Quad		Hull		Adaptive		Zenon		
		Sim.	Time	Sim.	Time	Sim.	Time	STE	Time	STE	Time	STE	Time	STE	Time
Helicopter	15 k	.768	0.9 ms	.700	2.5 ms	1.00	31.2 ms	4.2%	15.5 ms	9.7%	5.8 ms	36.9%	4.1 ms	48.1%	30.4 ms
Sponza	223 k	.177	1.9 ms	.322	4.5 ms	1.00	3 s	0.0%	361.9 ms	1.8%	134.0 ms	5.5%	38.5 ms	18.5%	113.5 ms
Houses	13 k	.674	0.7 ms	.727	2.5 ms	1.00	25.0 ms	5.6%	13.9 ms	12.2%	6.7 ms	28.2%	5.2 ms	39.1%	16.0 ms
Children	1,400 k	.610	3.8 ms	.780	6.3 ms	1.00	80.0 ms	1.0%	65.5 ms	2.2%	36.7 ms	26.7%	28.1 ms	37.2%	45.3 ms
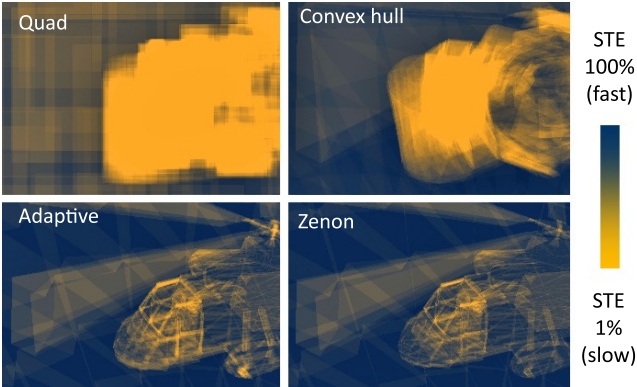
Fig. 15. Sample test efficiency of different rolling bounds in Helicopter. We see, that convex hulls are tighter than quads, but only bounds that adapt to the space-time structure have a workable STE, where Zenon's is more tight to the right of the image where motion is largest.

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. 14, 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 (more is better) depending on the transformation, again parametrized as an angle (Fig. 14, c). We find that our approach, as expected, has no error relative to the ray-tracing reference. With no user motion, common rasterization has no error either, while warping still introduces pixel-sampling problems. 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. 14, d), in the power foveation function $p(d) = d^\alpha$. We find that similarity is a convex function, peaking around the value $\alpha = 2$ we use. Too low-a foveation does not magnify enough to benefit from the super-sampling. Too high values magnify so much, that only the central part of the fovea benefits, reducing SSIM again. Time is a linear function of foveation strength, as polygonal bounds to increasingly curved triangles are decreasingly tight.

Head Pose Estimation. Finally, we investigate the effect of head pose prediction error on our approach. Before (e. g., Fig. 14, 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 prediction is much larger, at .05 meter and 1.3 degrees, indicating a linear model already removes most of the overall error.

6 PERCEPTUAL EVALUATION

To quantify the perceptual effect of our technique, we conducted four user studies: a threshold estimation experiment to establish the optimal foveation for a particular apparatus (Sec. 6.1); an image judgment experiment comparing super-sampling in our foveation

approach to a reference super-sampled image (Sec. 6.2); an object tracking experiment with and without rolling rasterization (Sec. 6.3) and an image preference experiment comparing our rolling approach to other approaches such as warping in an HMD (Sec. 6.4).

6.1 Foveation Strength

This study demonstrated that there was no perceptual difference between a non-trivially foveated image and a traditionally rendered image. We based our protocol on that of Patney et al. (2016), performing a 2AFC staircase task to identify the Just Noticeable Difference (JND) threshold - the foveation strength at which participants begin to reliably detect foveation artefacts.
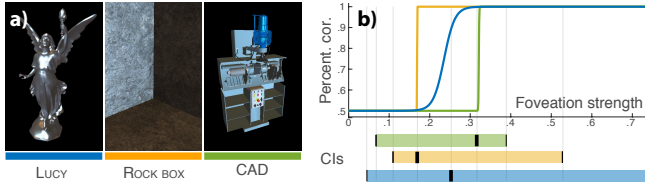

Fig. 16. Foveation study stimuli **(a)** and analysis (Please see text) **(b)**.

Procedure. After being fitted with an eye-tracker, participants engaged in a 2AFC task. In each trial, participants were exposed to two 1.5 second sequences of a rotating model - one with foveation and one traditionally rendered - with a .75-second gap in-between. The rotation was around the vertical axis at one revolution every 14 seconds. After viewing both sequences, participants were asked to indicate via the keyboard which of the two was "higher quality". The order of rendering technique was randomized. Participants each completed 180 trials on one of three models. Foveation strength was determined by a 1-up/3-down staircase following the guidelines of Garcia-Perez & Alcala-Quintana (2007).

Apparatus. We used a typical desktop PC with a GTX 980 GPU and an Asus VG248 144 Hz monitor to render the scenes with image-based lighting and specular materials (Fig. 16,a) under natural HDR illumination. The eye-tracker was an SR-Research EyeLink II connected via Ethernet directly to our application.

Participants. 25 naïve participants successfully completed our study across three conditions: Lucy (7), Rockbox (9), CAD (9).

Analysis. We opted for a fixed-size staircase with empirically set step-sizes, as our technique is novel and we do not have any reasonable priors for parametric sampling schemes. For our analysis though we fit a logistic psychometric function (García-Pérez and Alcalá-Quintana 2007) for simplicity and comparability to estimate thresholds and confidence intervals at 95 %.

Results. Fig. 16 shows an approximate psychometric function computed by averaging the function parameters for each participant, for each condition. A psychometric function describes the probability of detecting a distortion (vertical axis) depending on the foveation strength (horizontal). We see that the 75 % detection probability JND threshold occurs at non-zero levels of foveation. This indicates subjects cannot detect our foveation even when present at such strengths. The confidence intervals (colored bars) show the significance of this observation. Participant's individual functions, staircase results and further analysis are included in our supplementary materials.

6.2 Super-sampled Image Comparison

We performed a second study to determine how users compare our foveally super-sampled images to traditionally rendered images, when presented with a super-sampled image as a reference.

Protocol. After being fitted with an eye-tracker, participants engaged in a series of Two-Alternative Forced Choice (2AFC) trials. In each trial, participants viewed three instances of a slowly rotating model side-by-side (Fig. 16). The center model was a 4×4 super-sampled reference, with common and foveated to the sides in a randomized order. Participants were asked to indicate via the keyboard which side appeared most similar to the reference. Subjects each completed 45 trials spread evenly across three conditions, randomly interleaved.

Participants. 7 naïve participants completed this study.

Apparatus. We used the same apparatus as the previous study.

Analysis. We compute the preference for foveation as the proportion of aggregated trials in which the foveated image was chosen, for each condition. Two-tailed binomial tests ($n = 105$) indicated that the preferences are significantly different to chance ($p = .5$) for all conditions ($p < .001$);

Results. The results show a strong and consistent preference for foveation across all models ($\overline{\text{Lucy}}$: 90%, $\overline{\text{Flower}}$: 94%, $\overline{\text{CAD}}$: 87%). The slight reduction for CAD is likely because when viewing the back of the model there were few details to distinguish the techniques.

6.3 Rolling Rasterization Task Performance

We conducted a user study to examine how rolling rasterization affects perception in VR. We used an object tracking task to measure how behavior is influenced by both rolling rasterization and the asynchronous time-warping.

Protocol. Participants were shown a simple virtual environment in which a 50 cm box moved along a 180° curve, 8 m in front of them, just above eye level. The box reversed direction at the extents and moved at $85.9 \pm 68.7\,°\,\text{s}^{-1}$, the rate changing randomly every second. A head-fixed reticle was visible 8 m ahead.

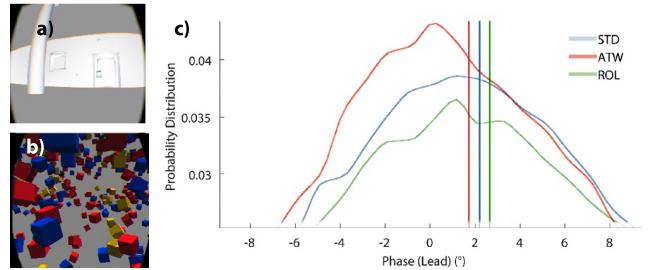Fig. 17. Stimuli of the first **(a)** and second **(b)** experiment. **c)** Probability Functions of Phase for each condition.

Participants were told to use their head to keep the reticle in the middle of the box. Participants followed the box in two trials, each lasting 4 minutes. Between the trials participants had a short break outside the HMD. There were three conditions pertaining to the rasterization method used: traditional (STD), Oculus' Asynchronous Time-warping (ATW) and our Rolling-Rasterization (ROL). The conditions were presented in 30-second-blocks, randomly interleaved.

Participants. 20 naïve participants completed the study.

Apparatus. Our experiment was performed with an Oculus Rift DK2. This HMD has a single low-persistence rolling-scanout display that scans right-to-left at 75 Hz with a persistence of 4 ms (Fig. 2). The DK2 has an inertial measurement unit (IMU) that samples at 1 kHz. The head and box positions were sampled at 75 Hz.

Analysis and Results. We began by analyzing the phase of the head motion. This is the instantaneous angular difference between the head and the box, positive when the head is leading the box, and negative when it is following. If participants were tracking the box exactly, we would expect a symmetrical distribution with a slight negative bias due to latency. Instead, a set of Kolmogorov-Smirnov tests show a non-normal distribution for the conditions separately and cumulatively ($P < 0.05$). All conditions also show a positive bias (Fig. 17, c). This indicates a tendency to lead the target.

Table 3. ANOVA test results for mixed-model terms

Variable	Effect	F-Stat	DoF	DoF	p
Per-Parti. Inter.	Random	1394.14	1	501,799	<.0001
Condition	Fixed	114.53	2	501,799	<.0001
Speed	Fixed	7984.79	1	501,799	<.0001
Condition: Speed	-	134.06	2	501,799	<.0001

In this case, we would expect the lead to increase as apparent latency decreases, and this is what is shown. ROL enables the largest anticipatory behavior (a lead of $\overline{2.7°}$). STD presents the second largest ($\overline{2.2°}$), having a latent but head-fixed image. ATW has the smallest lead ($\overline{1.7°}$), because while it compensates for latency, it does so by moving the entire image - including the target - counter to head rotation introducing apparent lag into the target.

To test the significance of this, we performed an ANOVA on the terms of a linear-mixed model, to control for per-participant biases and speed. The results of this test are shown in Tbl. 3, indicating a highly significant effect of rendering condition ($p < .0001$), as well as an interaction between rendering condition and speed ($p < .0001$).

6.4 Qualitative Rolling Study

This study compares fidelity of rolling rasterization to a traditional warping technique using a set of two-alternative forced choice trials. In this study the stimuli were head-fixed and in mono, to avoid any effects of latency or head motion.

Protocol. Participants were provided with the same HMD as before (Sec. 6.3) and exposed to a set of video pairs. Each video pair was seen consecutively and preference indicated using a keyboard.

Stimuli. The videos all showed moving objects (Fig. 17). The videos were one of three types. STD rendered each frame at a fixed time. WARP took each frame from STD and warped it from t_s to t_e with a traditional warping algorithm (Mark et al.'s (1997) pixel-sized grid warping, where triangles stretching by more than a threshold are culled). ROL rendered an image for t_s to t_e with our technique. Each video was presented for 2.5 seconds. Both the pairings of the types, and the order in which they were presented within the pair, were balanced an random.

Participants. 9 naïve participants completed the study seeing all combinations five times.

Analysis and Results. We summed the number of times each condition was preferred for each participant, and performed a one-way ANOVA on these three groups. We show a significant effect of rendering technique [$F(2, 24) = 13.45, p = .0001$], with users preferring STD or ROL over WARP. There is no significant difference between STD or ROL ($p = .993$). These results indicate that rolling rasterization can be indistinguishable from a traditional render in this protocol.

7 DISCUSSION

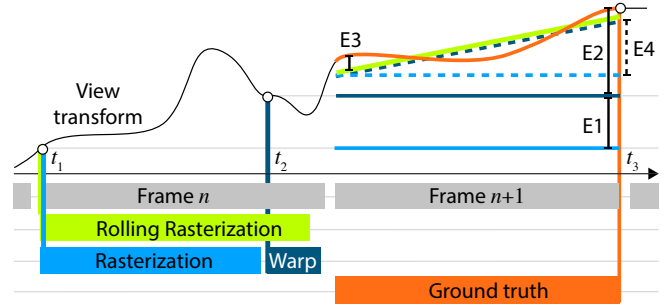*Comparison to warping.* The differences between an ideal ground

Fig. 18. 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.

truth zero-latency rasterizer, warping and our approach is seen in Fig. 18. The ground truth approach (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). 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 prediction error E3. 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-our duration, typically around 16 ms. Even if a fast non-rolling rasterization takes only 1 ms (a short light-blue bar in Fig. 18), the scan-out still takes 16 ms, and the latency will remain to be 15 ms. Using rolling

rasterization, that might be slower, say 4 ms, (green bar longer than the light-blue bar in Fig. 18) would be better deal, as the latency can get arbitrarily small if a sufficiently correct prediction is made.

Prediction. Like any method that has to finish before the scan-out starts, we require a prediction of scene and viewer motion during the 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.

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 a very high frequency of BVH rebuilds when using ray-tracing. 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 implementation. Our current implementation runs at real-time 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. 11, 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.

Screen-space effects. Screen space shading needs to be adapted to support perceptual rasterization. We have done so for SSAO by multiplying all image distances by the pixel density $p(x)$.

8 CONCLUSION

In this paper we introduced a new efficient rasterization technique that exploits the spatio-temporal-retinal relation 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 pixel-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. We only touched upon the relation to stereo or even light field displays.

REFERENCES

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

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. In *Comp. Graph. Forum*, Vol. 29. 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. *Proc. VR* (1999), 218–21.

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.* Vol. 17.

Miguel García-Pérez and Rocío Alcalá-Quintana. 2007. A comparison of fixed-step-size and Bayesian staircases for sensory threshold estimation. *Spatial Vis.* 20, 3 (2007), 197–218.

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.

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.

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. In *ACM SIGGRAPH Computer Graphics*, Vol. 22. 17–20.

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)* (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. In *Comp. Graph. Forum*, Vol. 35. 129–39.

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. In *Comp. Graph. Forum*, Vol. 35. 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.