Beyond Blur: Real-time Ventral Metamers for Foveated Rendering

DAVID R. WALTON*, RAFAEL KUFFNER DOS ANJOS*, SEBASTIAN FRISTON, DAVID SWAPP, KAAN AKŞIT, ANTHONY STEED, and TOBIAS RITSCHEL, University College London, UK

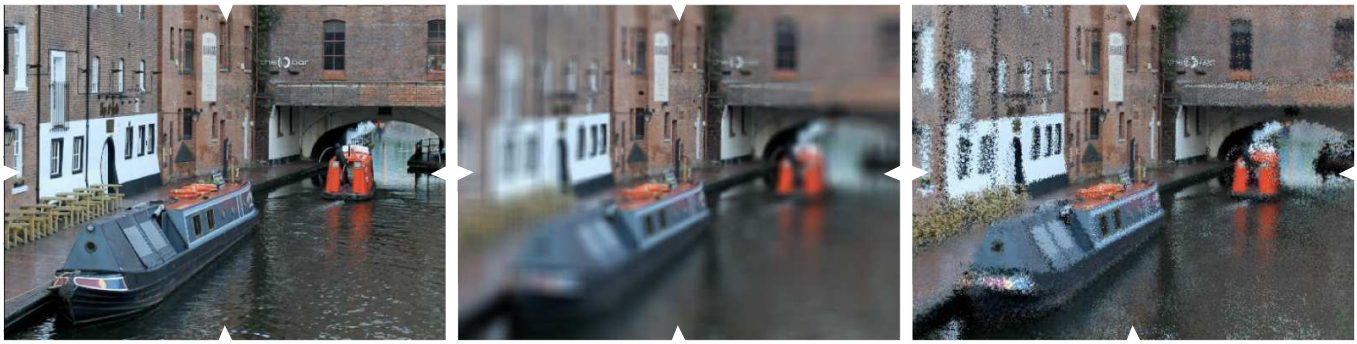

Fig. 1. Three images to be compared at a viewing distance of 30 cm in A4 print by fixating (foveating) on the location indicated by the arrows. The first image is a reference **(left)**. The second image is simulating peripheral vision using a Gaussian blur with bandwidth proportional to acuity **(middle)**. Third, our real-time ventral metamer where the periphery matches the reference in terms of statistics of multi-orientation and multi-scale feature activations **(right)**. Both can be computed in real-time frame rates, yet the metamer appears much closer to the reference. Timing for 512×512 on a Nvidia 2080 GPU.

To peripheral vision, a pair of physically different images can look the same. Such pairs are *metamers* relative to each other, just as physically-different spectra of light are perceived as the same color. We propose a real-time method to compute such ventral metamers for foveated rendering where, in particular for near-eye displays, the largest part of the framebuffer maps to the periphery. This improves in quality over state-of-the-art foveation methods which blur the periphery. Work in Vision Science has established how peripheral stimuli are ventral metamers if their statistics are similar. Existing methods, however, require a costly optimization process to find such metamers. To this end, we propose a novel type of statistics particularly well-suited for practical real-time rendering: smooth moments of steerable filter responses. These can be extracted from images in time constant in the number of pixels and in parallel over all pixels using a GPU. Further, we show that they can be compressed effectively and transmitted at low bandwidth. Finally, computing realizations of those statistics can again be performed in constant time and in parallel. This enables a new level of quality for foveated applications such as such as remote rendering, level-of-detail and Monte-Carlo denoising. In a user study, we finally show how human task performance increases and foveation artifacts are less suspicious, when using our method compared to common blurring.

CCS Concepts: • **Computing methodologies** → **Perception**; **Image compression**; *Ray tracing*.

Additional Key Words and Phrases: Foveated Rendering; Head-mounted displays; Texture synthesis; Human Visual perception

1 INTRODUCTION

In order to create a rich visual experience Virtual Reality (VR) often employs Near-Eye Displays (NEDs) or light projection systems such as VR caves to cover a large proportion of the user's visual field. Doing so at a high enough resolution to match the human vision in the fovea requires substantial compute and bandwidth resources.

A screen size of 4,000 pixels or higher would be required if a traditional screen and rendering pipeline were to be used. The Human Visual System (HVS) however, only resolves fine spatial details in its fovea but not in the periphery [Rosenholtz 2016; Strasburger et al. 2011]. The idea of *Foveated Rendering* [Albert et al. 2017; Friston et al. 2019; Guenter et al. 2012; Meng et al. 2018; Patney et al. 2016] is to focus compute effort to the fovea. Typically, foveated rendering shows a band-limited (i.e., blurry) version of the image in its periphery [Guenter et al. 2012; Patney et al. 2016], computed from fewer samples. Unfortunately, such blur can be perceived as unnatural and does not match well to what the HVS actually perceives: the periphery is not *just blurry* [Bouma 1970; Rosenholtz 2016].

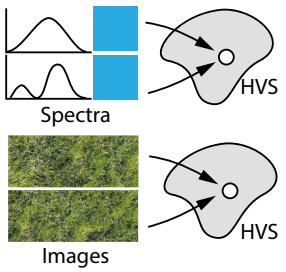

Fig. 2. Classic color **(top)** and ventral **(bottom)** metamers.

In this work, we seek to improve upon the fidelity of blurring, while retaining its efficiency. We propose a real-time method to compute images that appear identical to other images when observed in the periphery. Such images are called *Ventral Metamers* [Freeman and Simoncelli 2011] (Fig. 2). A stimulus is a metamer to another one if they are physically different but perceived identically. A well-known instance of metamerism is that different color spectra can map to the same color perception [Fairchild 2013]. But what is a good way to realize a peripheral metamer of another image?

*Joint First Author

Authors' address: David R. Walton; Rafael Kuffner dos Anjos; Sebastian Friston; David Swapp; Kaan Akşit; Anthony Steed; Tobias Ritschel, University College London, UK.

The Vision Science literature has shown how an image is a peripheral metamer to another one if certain image statistics in their periphery are identical. Statistics here refers to "soft counting", i.e., how *often* a feature, such as an edge, appears in a spatial *pooling* region of the visual field. Many different statistics, features and pooling regions have been proposed, leading to different models of peripheral metamerism in the literature [Greenwood et al. 2009; Rosenholtz et al. 2012; Schmid et al. 2009]. These models aim at understanding physiological processes and hence are often slow to compute, difficult to implement or hard to control. We aim for a simple-to-implement model with computational efficiency as required in real-time rendering. The key difficulty is the choice of statistics that match human perception.

We propose a form of statistics that is suitable for real-time analysis and synthesis of metamers: smooth steerable moments. These are applied in three steps: First, the method can analyze an input image in constant per-pixel time ($O(1)$). This is performed in parallel over all pixels. This step in inspired by variance shadow maps [Donnelly and Lauritzen 2006]. Second, the resulting statistics are processed, compressed and transmitted, depending on the application. Finally, random realizations of the metamer in accordance to the statistics can be generated, again in $O(1)$ time, by a process inspired by classic texture synthesis [Heeger and Bergen 1995].

After devising our theory, we will show a system that can metamerize any (stereo) image stream in real-time for VR. Our idea further enables a new level of peripheral quality which we demonstrate for three examples. First, foveated *metameric image compression* where the peripheral statistics are computed at a server and transmitted to a client that generates a metamer. While classic foveated compression is blurry in the periphery, our approach results in correct statistics. Second, foveated *metameric textures* where instead of storing means in MIP maps, we store the moments, allowing the generation of plausible peripheral texture details at low bandwidth. Third, foveated *Monte-Carlo denoising*, where a Convolutional Neural Network (CNN) maps noisy images to their reference statistics in the periphery instead of attempting the harder task of generating a correct noise-free image that cannot be perceived in the first place.

We evaluate how metamers generated with our method are perceived by human observers in a set of perceptual studies to confirm that (1) it improves task performance when recognizing patterns (2) it is preferred over other methods and (3) participants are more likely to classify an image metamerized using our method as consistent than images produced using blur.

2 PREVIOUS WORK

The aim of our work is to improve foveated rendering (Sec. 2.1) using findings from Vision Science on peripheral processing (Sec. 2.2) that relate to texture perception and synthesis (Sec. 2.3). We review the relevant work in the literature accordingly. Closely related but not in the scope of this research, foveated display hardware [Akşit et al. 2019; Kim et al. 2019] offer a non-uniform distribution of pixels to avoid rendering high resolution images at all eccentricities within a visual field of view, we refer our curious readers to the survey by Spjut et al. [2019] for more on foveated displays.

2.1 Foveated Rendering

Foveated rendering is motivated by the variation in acuity of the HVS. Uniform sampling in conventional displays means that the entire frame must be rendered and drawn at the highest resolution even though only a small region is visible at any time. Since the fovea has a high resolution, and as displays subsume more of the visual field - as with NED - the computational load increases quadratically or more. Foveated rendering aims to reduce this by targeting compute effort to where the HVS will resolve it.

This principle was first applied to video by Geisler and Perry [1998], who encoded frames as foveated multi-resolution pyramids to reduce bandwidth. Since then a number of gaze-contingent methods were proposed [Duchowski et al. 2004; Reingold et al. 2003].

For real-time rendering, a challenge has been to work around the assumed uniform sampling of the traditional pipeline. Methods have included drawing multiple passes at different resolutions [Guenter et al. 2012], using object Level-of-Detail (LoD) [Murphy and Duchowski 2001; Ohshima et al. 1996], and finding ways to support non-uniform rasterization [Friston et al. 2019; Meng et al. 2018] or multi-resolution rasterization [Vaidyanathan et al. 2014].

For ray-tracing [Fujita and Harada 2014; Levoy and Whitaker 1990; Weier et al. 2016; Zhang et al. 2011], foveation itself is trivial because a ray-tracer can sen rays in any layout. Though there are many practical difficulties in building immersive, real-time ray-tracing systems (e.g., eye-tracking and latency).

Other works have focused on reducing shading cost. He et al. [2014] adaptively evaluated different parts of the shading function over time and space, while Stengel et al. [2016] placed samples according to foveation. One of the latest developments is that of Kaplanyan et al. [2019], who use a neural network to reconstruct an image from samples that are dense in the fovea and sparse in the periphery. The method works with high temporal-resolution video, so every pixel is covered by a sample after a few frames. The method therefore solves a temporal in-painting problem. This is highly practical, though the loss is the same in periphery and fovea, and does not consider their different perceptual characteristics.

The periphery is not simply a lower resolution version of the fovea however, so it is important to consider how artifacts introduced by foveation methods may be perceived. Hoffman et al. [2018] examined image degradation in near-eye displays. Patney et al. [2016] investigated aliasing in practical foveation techniques, and presented improvements based on contrast sensitivity.

2.2 Peripheral Perception

Perception of the central and peripheral visuals differs in many intriguing and complex ways as detailed in surveys by Strasburger et al. [2011] and Rosenholtz [2016]. What can be said with certainty, is that recognition of patterns in the periphery becomes increasingly difficult. In part, this is due to a loss in acuity. Often, peripheral vision is described as "blurry", with the blur matching the density of receptors.

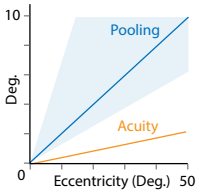

Fig. 3. Acuity and pooling (y) as function of eccentricity (x).

This effect is shown in the linear fit of acuity fall-off in Fig. 3 as orange: at an eccentricity of 50 degree, letters of a size around 2 degree are no longer discernible [Anstis 1974]. A similar relation is also found for contrast sensitivity: the contrast towards the periphery has to increase to make gratings discernible [Legge and Kersten 1987]. This acuity is commonly used to represent the foveal regions in foveated rendering [Guenter et al. 2012; Patney et al. 2016]. In this work, we will show it is beneficial to think beyond an acuity or blur-based model of peripheral vision and that real-time practical applications are already feasible.

Objects in the periphery do not simply appear blurry. For centuries it has been noted that objects appear "different" and "hard to see", but not blurry [Aubert and Förster 1857]. The reduction in acuity rather is attributed to an effect called *pooling* or crowding [Strasburger 2020]. Pooling means that the spatial location of features is irrelevant, and only their aggregate statistics matter. This is shown in Fig. 4, where the two large circles depict a *pooling region* with different features in each, depicted as dots. The statistics of those features are identical in the sense that the number of dots in each circle is nine. It becomes irrelevant where the dots are and only matters that they are present. Hence, the regions will be perceived equivalently; they are –simple– metamers to each other. The key to getting this effect right in images is to ask three main questions: First, *how large* the pooling regions are. Second, *which* statistics are to be represented. Third, *what* features are we to compute statistics on.

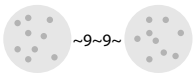

Fig. 4. Two equivalent pooling regions.

Size. The size of the pooling regions is described by Bouma [1970] as to depend linearly on eccentricity, shown as a blue line in Fig. 3. Remarkably, pooling region size increases much faster than acuity is decreasing. In other words, patterns turn into statistics much faster than acuity fails to resolve them. Depending on what features we consider, this function might look different, corresponding to the blue corridor of values in Fig. 3. Our model will follow the slope in the middle of the corridor which would mimic the effect of Visual Cortex area 1 (V1).

Statistics. In this work, we will assume the statistics in a pooling region are well described by their moments (mean and variance) because they can be efficiently analyzed and synthesized. More refined models further consider the correlation between features [Freeman et al. 1991; Tanaka 1996]. Correlation captures effects such as the probability that a vertical edge at a certain scale co-occurs with a horizontal edge at some other scale. The Gram matrix in style transfer serves the same purpose [Gatys et al. 2016]. Capturing correlation however comes at the expense of additional processing and storage. At the same time its visual importance can be called into question e.g., it is ignored in popular texture synthesis methods such as Heeger and Bergen [1995]. Storing correlation matrices needs memory quadratic in the number of features while moment vectors have storage requirements linear in the number features. Also the matrix can be made to tend to diagonal-dominant by using a decorrelated feature space like YCrCB for color. Hence, using moment vectors seems adequate for the real-time setting we target.

Features. We assume the features to build statistics from to be those used in the different areas of the visual cortex, in particular, the ventral stream [Ungerleider and Haxby 1994]. The early levels are scale- and orientation-sensitive linear filters [Carandini et al. 2005; Hubel 1982] while higher levels are concerned with their correlation. In this work, we assume all filters to be linear and do not look at higher-order correlations [Tanaka 1996].

The main inspiration of this work is the results of Freeman and Simoncelli [2011], who show that with a sophisticated model of human perception one can compute two images that are metamers. However, their optimization procedure computes a large number of statistics and complex features which are not amendable to real-time processing. Despite using gradient descent with custom gradients, this is orders of magnitude away (hours per image) from being applicable to VR in a real-time context (milliseconds per image). We seek to emulate their procedure, using minimal features and without the need to perform an optimization.

In this light, the step of Patney et al. [2016] which adjusts contrast to match a target after blurring [Kim et al. 2011] is the most basic version of the system by Freeman and Simoncelli [2011]: pixel color statistics are matched. We show that for a refined effect, features will need to be perceptual (scale and orientation-selective), and not just pixel colors. We also confirm this in our study showing improved performance of metamers in classification and discrimination tasks.

We note that according to Vision Science models, the Deep Fovea work Kaplanyan et al. [2019] solves important tasks, but might not be computing a metamer. Their method is trained to reproduce the value of sparse samples, which are increasingly sparse in the periphery. Instead of matching the specific peripheral statistics of a target, their adversarial term will seek to match the general statistics of natural images. Their network has access to several frames of a high frame-rate animation, while we work form a single image. We provide evidence that fooling the periphery might need less computational effort than a neural network and that a simpler real-time method is applicable to their and several other related problems.

In an attempt to speed up the work of Freeman and Simoncelli [2011], Deza et al. [2017] and Feather et al. [2019] use an AdaIN-based [Huang and Belongie 2017; Ulyanov et al. 2016] style transfer to generate metamers. It requires executing a neural network both for analysis and two networks (VGG inversion and pix2pix [Isola et al. 2017]) for synthesis. Foveation is addressed by blending style and content. However we were unable to produce plausible metamers using these methods and it was not computationally efficient enough for real-time applications i.e., performing within a few milliseconds on a typical frame.

In practical rendering, Tursun et al. [2019] combine a foveated rendering system with adaptive sampling where areas of low contrast are sampled less. The combination is attractive, as the thresholds for different channels in the periphery are different from the fovea. Their model accounts for acuity, but not for pooling. Swafford et al. [2016] have proposed a metric for foveated rendering. It is based on VDP [Mantiuk et al. 2011], but adapts the contrast thresholds

towards the periphery. Without a notion of pooling, it will overestimate the difference of image pairs that are metamers. Hoffman et al. [2018] have performed a user study to understand how typical rendering artifacts are perceived in the periphery.

2.3 Texture Synthesis

Our second main inspiration is texture synthesis, which is very related to metamer synthesis. A texture is an image with uniform (stationary) statistics [Portilla and Simoncelli 2000]. Texture synthesis means to generate new realizations with these statistics. Ideally a texture model is able to not only produce some, but also enumerate *all* instances of the texture (diversity). For textures, this generation has been done by summing up weighted noise [Perlin 1985] or by non-parametric sampling [Efros and Leung 1999]. Such procedures can be difficult to implement efficiently [Liang et al. 2001]. A survey is provided by Wei et al. [2009]. While similar to texture synthesis, the key difference in creating a metamer is that the statistics are more involved and most notably they are spatially varying. The size of the spatial regions over which statistics are constructed varies too (i.e., the pooling region size). In this sense, we are closer to by-example texture synthesis [Galerne et al. 2012; Lagae et al. 2010]. Our method is based on a localized (non-stationary) version of Heeger and Bergen [1995]. They match noise at different scales and orientations to the target statistics of an exemplar. They use histograms of filter responses which we compress to moments. We empirically found the visual benefit of a full histogram is not proportional, when targeting a real-time application .

Recently, learning-based methods have been proposed to generate textures, either by optimization [Gatys et al. 2016], which is too slow for our purpose, or by training a feed-forward NN to map noise into the style of a particular exemplar, which is not applicable to our task as we have field of statistics in across the image. Methods based on AdaIN [Huang and Belongie 2017; Ulyanov et al. 2016] generalize to arbitrary styles but would be too slow to apply to VR, and assume one homogeneous, stationary, texture. Wallis et al. [2016] have studied texture perception in the periphery, in particular comparing VGG base and simpler statistics, and found simple features, as long as they have multiple scales and orientations, to perform competitively.

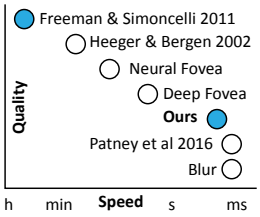

Fig. 5. Quality-speed trade-off.

Summary. The most relevant previous work is summarized on Fig. 5. We see a continuum of methods spanning the quality-speed space, starting from the original Freeman and Simoncelli [2011] ventral metamer work (requiring hours), over texture synthesis to newer methods based on neural network (requiring seconds to milliseconds) and finally blur, with speed close to `memcpy`. Our method is almost as fast as blur, but produces higher visual quality at modest computational overhead. It also is the only method to produce a metamer (marked in blue) for foveated rendering.

3 OUR APPROACH

3.1 Overview

We will explain all steps of our algorithm for an *exemplar task* of creating a metamer for a known image. We will generalize this exemplar task to practical applications e.g., computing analysis on a server, pre-compute it or emulate it using a CNN, in Sec. 5.

We look for a mapping $I_{\text{Out}} = \mathcal{M}(I_{\text{Ind}}, \xi)$ from an input image I_{Ind} and a random value ξ to an output image I_{Out}. The post-condition is that I_{Out} is a metamer of I_{Ind}. The mapping is to be ergodic, i.e., all metamers are generated as we put in all random numbers ξ. Without loss of generality, we assume foveation to the image center.

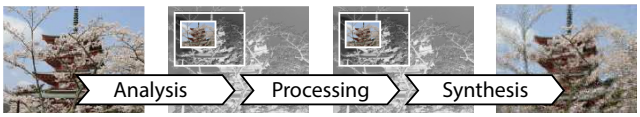

Fig. 6. Our approach has three steps: First, *analyzing* the input image to extract statistics. Second, *processing* those statistics depending on the application, and third, *synthesizing* a new image from the statistics.

Our approach proceeds in three steps: *Analysis* (Sec. 3.2) that extracts the statistics from the input, *Processing* (Sec. 3.3) that changes those statistics and *Synthesis* (Sec. 3.4) that samples from the distribution of metamers having the desired statistics.

3.2 Analysis

Our approach takes as input an *RGB* image I_{Ind} and outputs its statistics S_{In}. These are found in a specific color space, using feature pyramids and local moments for pooling, three aspects we will discuss in the next paragraphs.

Color space. In a first step, the input image is converted to YCbCr, a decorrelated color space [Poynton 2012]. This is important, as we will later not capture co-statistics between feature channels, a strategy most beneficial when channels are maximally decorrelated. In the HVS, this step is related to low-level processing found in receptive fields providing color opponency [Ruderman et al. 1998].

Pyramid. The ventral stream, the next step in the HVS, is sensitive [Güçlü and van Gerven 2015] to features at all scales (Property 1) and features are related to changes over space (Property 2). Image pyramids are ideal to capture both properties.

Hence the input is converted into a steerable pyramid [Freeman et al. 1991]. A steerable pyramid applies a pair of direction sensitive filters (horizontal and vertical) to every level, followed by a subsampling step. Steerability assures that the response at in-between orientations is a linear combination of the response at the two main directions. The original steerable pyramid designs filter in the Fourier domain to be optimal. We however have to limit ourselves to extremely compact filters (small kernels) for a real-time application. We found that the optimal 5×5 filters are sufficient for metamers in practice for synthesis. If analysis aims to be real-time, we also use 5×5 filters, but also have the option to use ground truth (Fourier-based) steerable filters if the statistics are to be produced in a preprocess, such as in the texturing application (Sec. 5.2). We find those optimal filters by optimizing for the filter deck that has the response

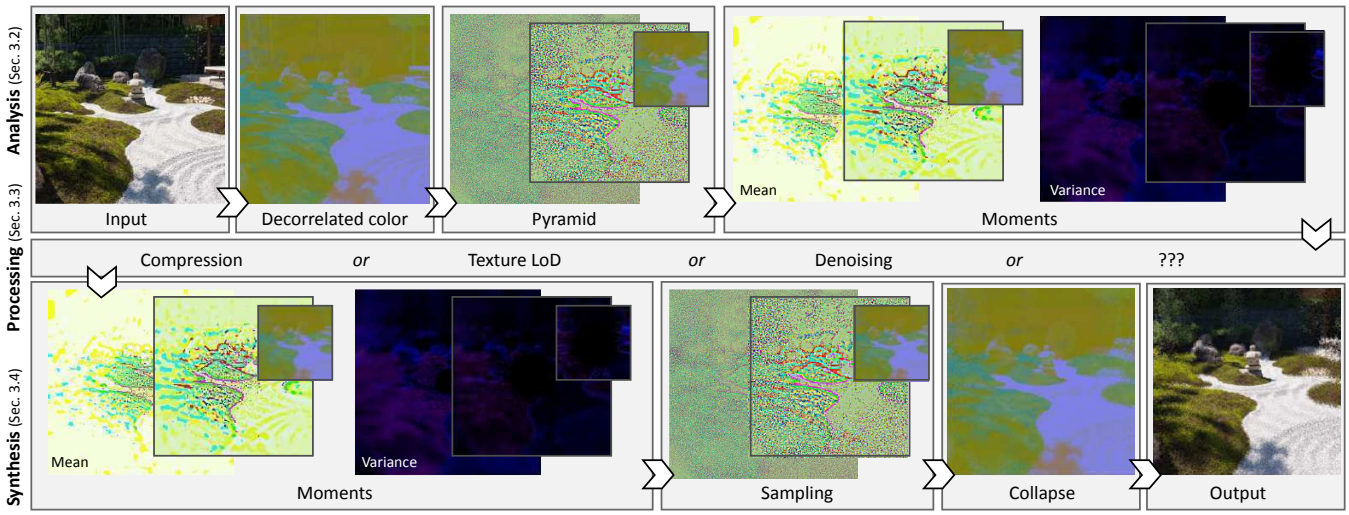

Fig. 7. Visualization of our approach: We start from an RGB image, which is converted to a decorrelated color space. Next, we compute a feature pyramid comprising of steerable filter responses on multiple scales, and a low-res residual. Moments (mean and variance) of statistics of arbitrary pooling regions can be computed in constant time, we here visualize mean and variance with a radial fall-off. The next step is an application making use of that representation. The synthesis step uses the new statistics to compute a matching activation pyramid of noise. Collapsing this pyramid instantiates a metamer.

most similar to the ground truth filter from the original approach. The optimization is a gradient descent on a $2 \times 5 \times 5$ space of filters using the ADAM solver [Kingma and Ba 2014] with the L2 image difference as a loss. This filter kernel optimization is only performed once and can later be used on any image. The conversion require constant time and constant memory for arbitrary-sized images. This requires below a milliseconds for 512×512 images on recent GPUs in practice.

Pooling. All prior operations are linear operations. In the periphery, non-linear functions *pool* the information over so called *pooling regions*. The HVS does not perceive individual features but their statistics aggregated spatially over the pooling region. For simplicity, we here only look at the *moments* of those statistics, in particular the *mean* and the *variance*. We further assume the pooling region shape to be parameterized by Gaussians, so a per-pixel map $R(\mathbf{x})$ to hold the standard deviation of a Gaussian approximating the pooling region around location $\mathbf{x}$. This map has high values in the periphery where pooling is over large regions, and small values in the center, where less pooling is found.

Under these conditions, the statistics can be computed efficiently, too. We here take inspiration from shadow map filtering [Donnelly and Lauritzen 2006]. The same operation is applied to all levels L of the input pyramid P_{In}, producing the a per-level *moment map*. We will describe the two steps for each level.

First, a cubic MIP map $\texttt{mip}(L)$ of each level L is constructed. The first moment map M_0, the mean map, can be read directly from this MIP map using $\texttt{select}(\texttt{mip}(L), \sigma)$, where $\texttt{select}(\texttt{L}, \sigma)$ simply copies pixel values from the per-pixel MIP level corresponding to bandwidth σ using tri-cubic interpolation.

Second, in a similar spirit, the variance map M_1, or other higher-order moments, can be computed efficiently. The underlying trick was used in variance shadow maps [Donnelly and Lauritzen 2006] for real-time shadow rendering. Instead of computing a MIP map $\texttt{mip}(L)$, we compute the MIP map $\texttt{mip}(L^2)$ of the squares of each level. Recalling that $\mathbb{V}[\mathbf{x}] = \mathbb{E}[\mathbf{x}^2] - \mathbb{E}[\mathbf{x}]^2$, we can now blur the square map with the spatially-varying pooling blur as well to produce $\texttt{select}(\texttt{mip}(L^2), r)$, and subtracting the square-of-mean from the mean-of-squares to arrive at variance.

The result is a pyramid that holds at every pixel and all levels the feature statistics across each pooling region described by first and second moments.

3.3 Processing

Processing maps the statistics to new statistics. This step is particularly important in different applications. Specifically, the input statistics S_{In} can be altered into output statistics S_{Out}, without affecting the peripheral perception. This opens opportunity for compression, for baking information into textures or for inferring the values from using a CNN as explained in Sec. 5. For the didactic exposition, we just leave the statistics unaltered here.

3.4 Synthesis

The synthesis step converts the output statistics S_{Out} back into an image. As there are infinitely many images that have the same statistics (at least for finite-order moments), this is a generative process conditioned on random value ξ, in our case a pyramid with random values as used in texture synthesis [Heeger and Bergen 1995]. This random pattern is held constant over time, providing temporal coherence.

First, ξ is reshaped to have the mean and variance of each level respectively. This is easy for Gaussians, by shifting and scaling as in $\xi \cdot M_1 + M_0$. Note, how this differs from classic texture synthesis, where the statistics are, by definition of a stationary texture, constant across space. In our case, the mean and variance change across image

space: in a typical photo, the sky might be uniform, so the variance of color will be lower than on the forest part of the picture below it.

Finally, all levels of the pyramid are collapsed back into an image. We start at the coarsest desired level, sample it up, add level i and apply the analysis filters until arriving at the final image. Note that the analysis filters are self-inverting in steerable filters and by design our optimization inherits this property.

4 IMPLEMENTATION

We have implemented two versions of our approach, one in Python and one in Unity/C#/OpenGL. These both demonstrate the full pipeline as described in Sec. 3. The Python implementation takes as input any image (photo, video, interactive 3D content) generated by a renderer and a pre-computed noise map. The first pass computes the statistics from that input image. The applications then manipulate the statistics. The second pass collapses the statistics into an image again. Stimuli for the studies in Sec. 6 and compression results in Sec. 5.1 were produced using this implementation.

The Unity implementation (Unity 2018.4.26f1) is real-time and follows the same approach. The scene is rendered as normal by scene camera(s). The metamer is calculated as a post-process: the first pass that computes the statistics is run as a sequence of shaders on the full-screen image, no manipulation of statistics is done, and then the second pass runs as a second rendering pass on a full screen image. Because it runs as a post-process the metamer can be computed on all scenes supported in Unity. The texture results in Sec. 5.2 develop upon this framework.

The Unity implementation was extended to support the Varjo XR1 NED. For this stereo display each eye sees both a wide-field of view context display and a static high resolution focus display. The display also contains a stereo eye-tracker. Thus four images are rendered per frame. For the demonstration we apply metamers to the context displays' images only. See the accompanying video for examples of output, achieved at 60 Hz for the XR1, using a Intel i9-9900 CPU at 3.10Ghz, 64GB RAM and an NVIDIA GeForce 2080Ti with 11 GB of GDDR 6 memory (this GPU being the minimum specification for the XR1). The scene shown uses a panoramic image at 8K×4K, rendering to graphics contexts of 2K×2K for the context displays.

While metamerizing an image that is already computed at full resolution appears to provide no practical benefits (these will come from the applications in Sec. 5), our system is the first to allow study of the impact of metamerized images in an interactive setting. As it runs on modern NEDs, it opens an avenue for several experiments in VR, Vision Science and Psychology that were not possible before.

Both implementations will be made available upon publication.

Evaluation. To better understand the impact of screen size and thus the size of the image to be metamerized, Fig. 8 measures the time required to metamerize an image, with a breakdown between analysis and synthesis. These were computed on a machine with Intel Core i7-10750H CPU at 2.60Ghz, 12 GB RAM and an Nvidia GTX 1650 Ti GPU. We see that time is roughly linear in the number of pixels, as expected and required for a real-time system. Our approach requires roughly five times more memory than blur, due to the need to maintain multiple filtered versions of each level and a separate MIP map for each level. Recall, [Williams 1983] that MIP maps only ever add a constant factor of ×1.3. This is true at ×1.6 also for the pyramid of pyramids we use.

It also requires five times more compute time, across all resolutions. The fact that memory and time scale similarly, indicates that the bottleneck to address is the chip bandwidth to access the statistics for multiple bands and multiple orientations. Improving upon both compute time and memory usage is important future work to make our approach more practical. A detailed analysis of how the number of bands, orientations and kernel filters affect result quality in which condition would complement such effort.

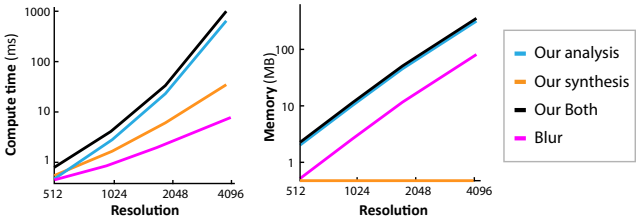

Fig. 8. Compute time and memory (in log scale) for our approach and its passes when metamerizing images at different resolutions.

5 APPLICATIONS

We apply our approach to three tasks: metameric image compression (Sec. 5.1), metameric textures (Sec. 5.2) and peripheral denoising (Sec. 5.3).

We explain applications in the logic of a client-server setup. The *client* is less powerful and consumes the representation produced by the server, mapping it to a metamer. Client and server do not need to be physically separated by a network, but can also be logical parts of one software system on one machine.

The *server* has more significant computational power and access to the non-metamerized, Ground Truth (GT) image. It transforms the GT image into some representation related to the statistics. It further knows or is able to predict the fixation point sufficiently, which, without loss of generality, we will keep assuming to be located at the image center.

We note that, unfortunately, there is no metric yet to quantify the visual success of our methods, other than looking at the images in the right viewing conditions. We can measure speed and compression rate and bandwidth, but there is no way to measure successful metamerization that has been proposed in the literature to our knowledge. Previous metrics for foveation are based on acuity only, i.e., they blur the periphery by construction, a limitation we want to overcome in the first place. Paradoxically, our method, more precisely its activations, might provide the metric itself, but this is left to future investigations. The user studies in Sec. 6 will provide further insight into how users perceive the imagery we generate.

5.1 Application: Compression

This application targets transferring images such as plain photos and video frames, including remote-rendered content from a server to a client. The metamer analysis and compression happens at a server. Instead of sending updated images, the server sends updated

statistics. The client is then free to realize any metamer to fit the statistics. As the statistics are much smaller than the image, bandwidth is reduced while still producing plausible details in the periphery.

To reduce the size of the statistics, three steps are employed for encoding: warping (Sec. 5.1.1), sampling (Sec. 5.1.2) and quantization (Sec. 5.1.3). The inverse of those steps is used in reverse order at decoding time. Next, we detail these three steps.

5.1.1 Warping. We recall the pooling region size to vary over the image. If a pooling region, for example, in the periphery is maybe 10×10 pixels in size, we seek not to store all 100 pixels, but only its statistics, a much smaller set of values. To achieve this, we *warp* images, such that the local pixel density, which is constant in a common image, becomes proportional to the pooling [Anstis 1974]. In practice, areas that are in the periphery have a density below one and hence shrink.

Warping is a common approach for compression of foveated images [Traver and Bernardino 2010] when applied to the image alone. Typically, the acuity function is used for warping where multiple input pixels in the periphery are being mapped to a single output pixel, hence averaged, and ultimately blurred; in our case we would like to also preserve the statistics of these averaged areas instead. We hence look for a way to compress both the image in the fovea, and the statistics in the periphery.

To this end, we not only compress the image, but an entire pyramid to preserve the statistics. This has two goals: sufficiency and compactness. First, we want it to be *sufficient*, i.e., the statistics we need have to be preserved to the level a metamer needs. Second, it also has to be *compact*, i.e., we want those statistics only to have the resolution that is required, not more. If it was not compact, we would lose the compression advantage of foveation. If it was insufficient, we would produce blur. Hence, the question becomes how to warp the entire pyramid to meet both of these requirements?

For formalization, we will work in the *polar domain* where the horizontal axis is radius r and the vertical axis is angle θ as seen in Fig. 10, a. In such a setting, pixel density is constant along angle θ, and only varies with radius r.

We hence look into functions mapping radius r to pixel density $d(r)$. These functions are different for the image and its pyramid levels. For the image itself, it is the classic acuity / pooling function that drops of from the center, for example $d_0(r) = r^{-2}$. This function is seen in Fig. 10a as a blue curve. Note the log scale on the vertical density axis.

What would the density function d_l of the stat map of level l need to look like, given this curve d_0 for the image, to be sufficient and compact? Three things play together here. First, if the statistics map level has a pixel density lower than the image, the density can be 0. This is because we do not need statistics at radii where the original image signal is present in the image. So all density functions for the pyramid can be 0 in the fovea as this is transmitted unchanged. This already eliminates storage for the vast part of the pyramid and allows us to steer bandwidth to the periphery. Second, if the original image loses details of scale 2^l at some radius r (say 8 pixels compress to 1), the statistics map has to represent them, so the pixel density d_l at r has to be larger than zero (we want the statistics of those 8 pixels). Finally, the resolution at which statistics are required also falls off, as statistics are pooled over increasingly large regions, just as the image is. So while having to increase and peak at the point where statistics are most important, they can also fall down rapidly as pooling regions grow. This leads to shapes seen as red plots in Fig. 10b for different levels.

5.1.2 Sampling. To apply the warp to the original images as well as to every level of the pyramid, we proceed as follows: First, we compute the Cumulative Density Function (CDF) $D(r)$ of $d(r)$. This is seen in Fig. 10, b. This functions D holds the accumulate density up to radius r. Let $D^{-1}(y) = r$ be its inverse, which exists as a cumulative function is monotonically increasing. We sample the image or the pyramid levels at regular levels $D^{-1}(y)$ for $y \in (0, 1)$.

$D^{-1}(y)$ is many-to-one, i.e., many input pixels from the image or pyramid level map to one output pixel. Simply picking the single pixel nearest to the inversely-mapped position will both alias and will not produce the statistics we need. Instead, we handle this in two steps: First, if the input is $N \times M$ pixels, we sample to an output resolution of size $(R \cdot N) \times M$ where R is some bound for the compressiveness, we use $R = 32$. Note that we work in the polar domain, where radius and angle can be handled differently. In this approach, aliasing is prevented as for every output pixel there is no more than one input pixel. Second, this temporary image is resampled to the desired output by averaging groups of R pixels into one. Instead of just averaging pixels, we also average their squares. This produces a map of statistics with a controlled pixel density.

5.1.3 Quantization. The resulting warped image and statistics maps can now be further compressed. Currently, each channel of each statistics map is remapped linearly to $[0, 255]$. Each map is then quantized to 8 bits per channel and compressed using JPEG. Further signal-dependent equalization or specific custom quantization tables would likely further improve our results.

5.1.4 Results. Results are seen in Fig. 11. Input images were 1024×1024 pixels, meaning an uncompressed filesize of roughly 3 MB. The quality setting of the JPEG examples was adjusted to give a filesize as close as possible to that of our approach. We see that at the rates of 41 kB per image (i.e., roughly a 1:75 ratio), naive JPEG suffers and a naive foveated JPEG fairs better, but blurs the periphery. At same effort, our approach achieves the same by making a slight concession to the periphery, but plausible periphery. Unfortunately, as explained in Sec. 2 no metric to compare peripheral stimuli is available. We kindly refer the reader to the experimental evaluation of our approach in Sec. 6.

5.2 Application: Texturing

In this section, we synthesize metamers in screen space, but from statistics stored in textures. The key observation is, that the pooled statistics of an image are more compact than the image itself as pooling is blurring which removes details and can be stored in a lower resolution. Hence, accessing the statistics can save bandwidth. To get plausible details back, we metamerize the texture reads on-the-fly. This can be realized using a *pre-computation* and a *run-time* step. The idea is illustrated and explained in Fig. 12.

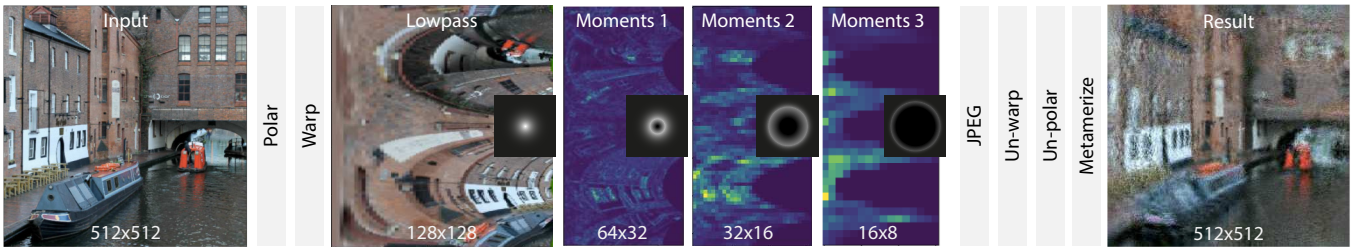

Fig. 9. Metamerized foveal image compression: Starting from an input image in some resolution e.g., 512×512, we change to the polar domain and apply different carefully-crafted warping functions (as seen in Fig. 10) that match output pixel density to what needs to be captured to produce a metamer. The lowpass captures the fovea (the gray-scale inset shows this density), but shrinks the periphery, hence it can be sub-sampled by factor 4. The moments for the periphery are smooth due to pooling, and hence can be sub-sampled progressively and aggressively, providing the compression. The resulting images can be compressed with any lossy or lossless compression. Applying inverse warp the polar transform and sampling an instance of the metamer produces a result in the original resolution. More compression result details are seen in Fig. 11.

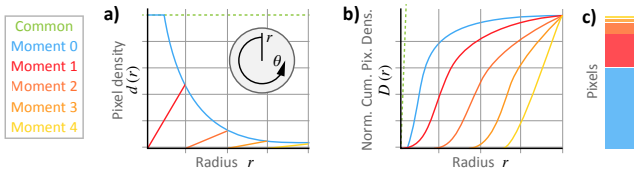

Fig. 10. Pixel density **(a)** and normalized cumulative density **(b)** on the vertical axis for different eccentricity on the horizontal axis. Colors indicate pyramid levels (red), respectively, the image itself (blue) and a non-foveated baseline (dotted green). **(c)**: Pixel proportion per level.

Pre-computation. First, we build a pyramid of the texture as done in our analysis step (Sec. 3.2). This texture can have any arbitrary size, in theory it can be infinite-resolution, as long as we can compute the statistics, i.e., estimate the variance of filter activations. As this is a pre-process, we can also use any filter in the Fourier basis to create the pyramid, without resorting to our optimized filters for fast analysis. A typical example would be 8192 pixels for the planet texture we demonstrate. We compute the moment map for this texture and store it as defined in the analysis in Sec. 3.2. This requires additional memory (30 %, as in common MIP maps), but with the right paging and caching (which we did not implement in our prototype), only the foveated part of that texture ever needs to be accessed and held in memory.

Runtime. The key idea is to fetch only the moments we need in the framebuffer from the texture without the need to ever compute them in the framebuffer. This has two aspects: picking the right pyramid level and picking the right pooling size.

Let us first approach picking the right *pyramid level*. Consider an image with texture resolution N (e.g., 8192), a rendering resolution M (e.g., 1024). Consider a pixel in the rendered framebuffer. Assume this pixel has a *pixel-to-texel* ratio logarithm ρ. This value depends on view, texture coordinate and geometry in a complex way but can be computed from m, N and the texture coordinate derivatives following the OpenGL specification for MIP level selection. In our example, for an orthographic fronto-parallel view on a texture quad textured geometry fitting the screen, we would have $\rho = 3$, as every pixel maps to $(2^3)^2$ texels. To fill the framebuffer at resolution M resp. level 0, we hence have to fetch the pyramid level ρ. So far this is classic MIP mapping [Williams 1983]. Remember the highest-resolution MIP level is typically (e.g., in OpenGL) indexed at 0.

To pick the right *pooling* for every pixel in the framebuffer pyramid, we have to consider the spatial position. Pixels close to the fovea pool over small regions, pixels at the periphery pool over large regions. Note, that this is not a property inside the texture, but inside the framebuffer. Consider a pixel that has the quadratic pooling region of log-edge length η in screen space. For example, a pixel in the periphery grouping 16×16 values would have $\eta = 4$. Classic foveated rendering could now fetch the texture MIP level $\rho+\eta$, a combination of blur for filtering and a combination of further blur for foveation. This would reduce bandwidth, as operating at higher MIP levels is beneficial i.e., saves memory.

The downside is, that the stats of the texture between ρ and η are lost. We can access them without accessing ρ, by looking up the MIP level $\eta - \rho$ in pyramid level $\rho + \eta$. This texture holds exactly the pooled (hence low-resolution) stats we are missing. So we use values from $\rho+\eta$ as the low-pass and add the details fetched between ρ and $\rho + \eta$. Instantiating a metamer with those stats produces a texture signal of ρ.

We have made a further optimization, in which we do not even fill a pyramid of moments to give to the synthesis step, but a shader that generates the texture value by simply adding up $\eta - \rho$ noise values, scaled by the mean and variance and the low-pass value ρ. We use this variant in all our texturing results.

Results. Please see Fig. 13 for results and their discussion. The bandwidth saved for a 512×512 framebuffer using the Mars texture is 52 %, for a 4k framebuffer, it is 79 %.

5.3 Application: Denoising

The final application is to de-noise Monte Carlo (MC) path-traced images, such that they are noise-free, but metameric to the reference image instead of minimizing any classic image error. Typical denoising approaches map from noisy path traced images to clean images. We instead map from noisy path traced images to a moments map, which can then be turned into a metamer. The intuition is, that finding the statistics of an image is an easier task for a neural net than finding the image itself.

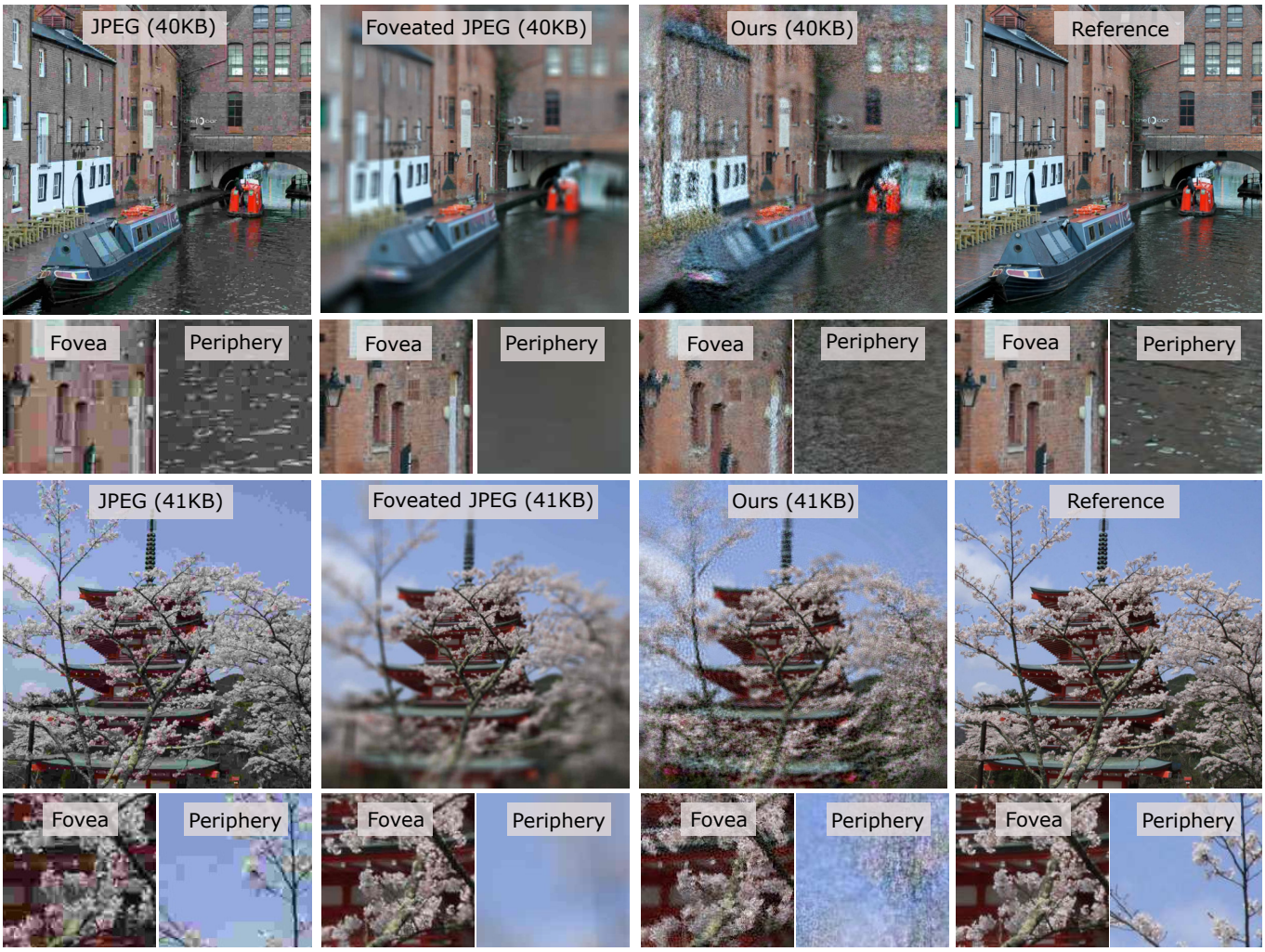

Fig. 11. Compression results for different methods **(rows)** on different inputs **(columns)**. All files have a size around 40kB i.e., a 1:75 ratio. In the insets, we note that, first, the fovea is good in all methods, except JPEG and, second, the periphery is plausible only in ours.

Training Data. We first sample pairs of noisy path-traced images with a finite number of samples and a noise-free reference. All training images are renders of the same 3D scene show in Fig. 14. The number of samples increases from 1 in the periphery to 32 in the center. We apply our analysis to the reference images to generate corresponding moments maps. These moments maps are the desired output of our network. Here our baseline is an identical network, attempting to map noisy images to clean RGB images.

Network. We train a conventional U-net [Ronneberger et al. 2015] under L2 loss with 8 layers, 64 internal activations and residual links to map from the reference images to the moment maps.

We do not make use of a guidance signal (G-buffer), nor any of the many other exciting inventions made in deep MC denoising [Bako et al. 2017; Chaitanya et al. 2017; Kalantari et al. 2015].

Results. As there is no published way to quantify peripheral results, we show and discuss qualitative results in Fig. 14.

Please note that our intended purpose here is to compare estimation of moments with direct estimation of RGB values. Consequently the absolute performance of the networks is less important than the comparison between them. Both of our networks use the same architecture, therefore, any improvement in the architecture would be expected to improve the results of both networks.

Our result with the network that maps noisy images to moment maps indicates that metamerism is a process that can be effectively learned by CNNs. Investigating denoising of general scenes with a state-of-the-art architecture remains future work and we explicitly do not claim state-of-the-art denoising here. Our contribution is a practical method that denoises while targeting the characteristics of peripheral vision.

6 PERCEPTUAL EVALUATION

We conducted a number of user studies to evaluate the effectiveness of our method, as well as others, on perception in the periphery.

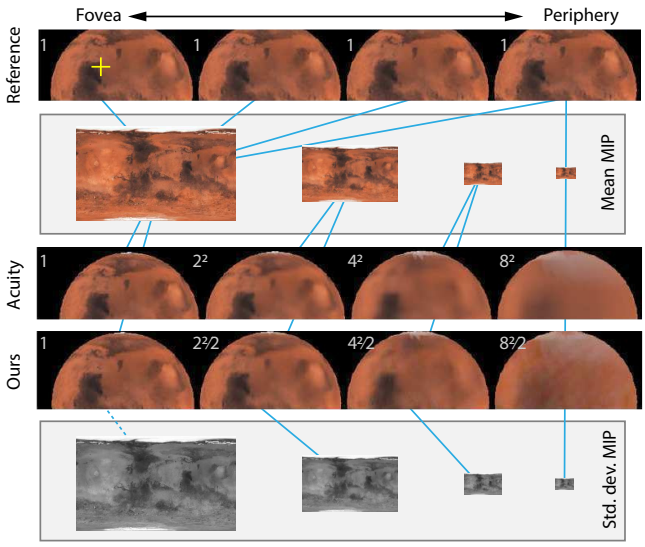

Fig. 12. Metamerized textures for foveated rendering. The top row shows a rendering of four planets sharing a texture and a subject fixating the yellow cross, i.e., foveation is left, periphery is right. In a ground-truth rendering (top row), all planets have the same distance and same texture, hence access (blue arrows) the same MIP level 0. The number 1 denotes bandwidth saving, which remains the same for reference rendering. In an acuity-based foveated rendering system (middle row), the foveated planet would access MIP level 0, but towards the periphery, the MIP level can increase up to level 3. Bandwidth is only $1/2^3 \times 2^3 = 1/64$ but details get blurred. In our approach (bottom row), the same MIP levels as in the acuity-based approach are accessed, but additionally, we access the smooth moments of the texture. This requires more bandwidth, but still saving factor $1/2 \times 2^3 \times 2^3 = 1/32$ but produces details in the periphery.

In the first (Sec. 6.1) we measured the effects on performance in an acuity dependent classification task. In the second (Sec. 6.2) we measured image preferences. In the third (Sec. 6.3) we measure acuity using a within-frame detection task.

All experiments were run in VR on Oculus Quest V1. Experiments were distributed, with volunteers running the trials on their own devices. Demographics were otherwise uncontrolled as we did not expect any effect on low-level vision [Shaqiri et al. 2018]. In all studies participants saw a small dot (1 deg or so in size) at the center of their vision, and were told to focus on this dot until the experiment completed. An example view is shown in Fig. 15. All experiment responses were binary, given with hand controllers. Each experiment lasted 15-20 minutes. Experiments were approved by UCL Ethics Board (4547-013).

. Across three experiments, we compared four methods. i) classic Blur, (implemented as a filter pyramid) potentially at different bandwidths, ii) Our method, iii) the original offline metamer method F&S as described by Freeman and Simoncelli [2011], and iv) a Reference. All methods operate on non-linear color (subject to display gamma) as they are concerned with perceived contrasts, not with preserving energy.

Table 1. Comparison of all methods to Reference when fitted to a Logistic Model seen in Fig. 17. The first three values are the resulting fit parameters. The column p is the significance of the fit, i.e., where a low value indicates probability of no difference.

Method	Estimate	Std. Err.	T-Stat.	p
Blur 0.35	0.105	0.138	0.759	0.447
Blur 0.70	-0.391	0.131	-2.987	0.003
Blur 1.05	-0.611	0.129	-4.473	<0.001
Blur 1.40	-0.846	0.127	-6.631	<0.001
Ours	0.258	0.141	1.827	0.067

6.1 Task Performance Experiment

This study investigated how metamerism affects task performance. Metamers should contain the same information as the reference, and so support equivalent performance (Hyp. i). We also hypothesize that the blur typically applied in foveated rendering could obscure information [Rosenholtz 2016] and so reduce performance (Hyp. ii). We test these with a symbol discrimination task. Participants (N = 14) were asked to classify symbols in their periphery with different foveation effects applied, while we measured their success.

Protocol. A major application of foveated rendering will be for VR. However, most current headsets do not have sufficiently high quality eye trackers. We therefore designed our study to be robust to uncontrolled eye gaze, and so enabled its running on Oculus Quests.

Landolt circles (circles with or without a gap) were displayed at random polar-angles at given eccentricities for 150 ms, after which participants were asked to classify them as open or closed. The random polar coordinate means that even if a participant attempted to cheat, they would not gain a statistically significant advantage.

We tested four blur rates covering the range of Patney et al. [2016]. Each participant saw an equal number of open and closed symbols (5 each), for each method (6) and eccentricity (5), for a total of 300 observations. Symbols were displayed with a height of 1.25 deg between 5 and 25 deg eccentricity. At 30 deg, this height is the threshold of detectibility [Anstis 1974], and even control answers should reduce to chance level. Example symbols are shown in Fig. 16.

Analysis. Fig. 17 shows how the probability of success varies with eccentricity and foveation method. At the most extreme eccentricities, performance tends towards chance for all conditions, as the stimuli reaches the limits of the HVS. To test for significance, we fit a Generalized Linear Model [Dobson and Barnett 2018] with a binomial distribution, i.e.,

$$\texttt{logit(correct)} \sim 1 + \texttt{eccentricity} + \texttt{method},$$

in Wilkinson notation [Wilkinson and Rogers 1973], treating eccentricity as a co-variate, so we can compare the significance of the foveation methods (Tbl. 1). We see that compared to the reference, there is no significant difference in performance with our stimuli (Hyp. i), or with the baseline blur rate. However, performance degrades significantly as blur increases (Hyp. ii). A Spearman's rank test on the correlation of performance with time did not show any learning effects ($p > 0.61$).

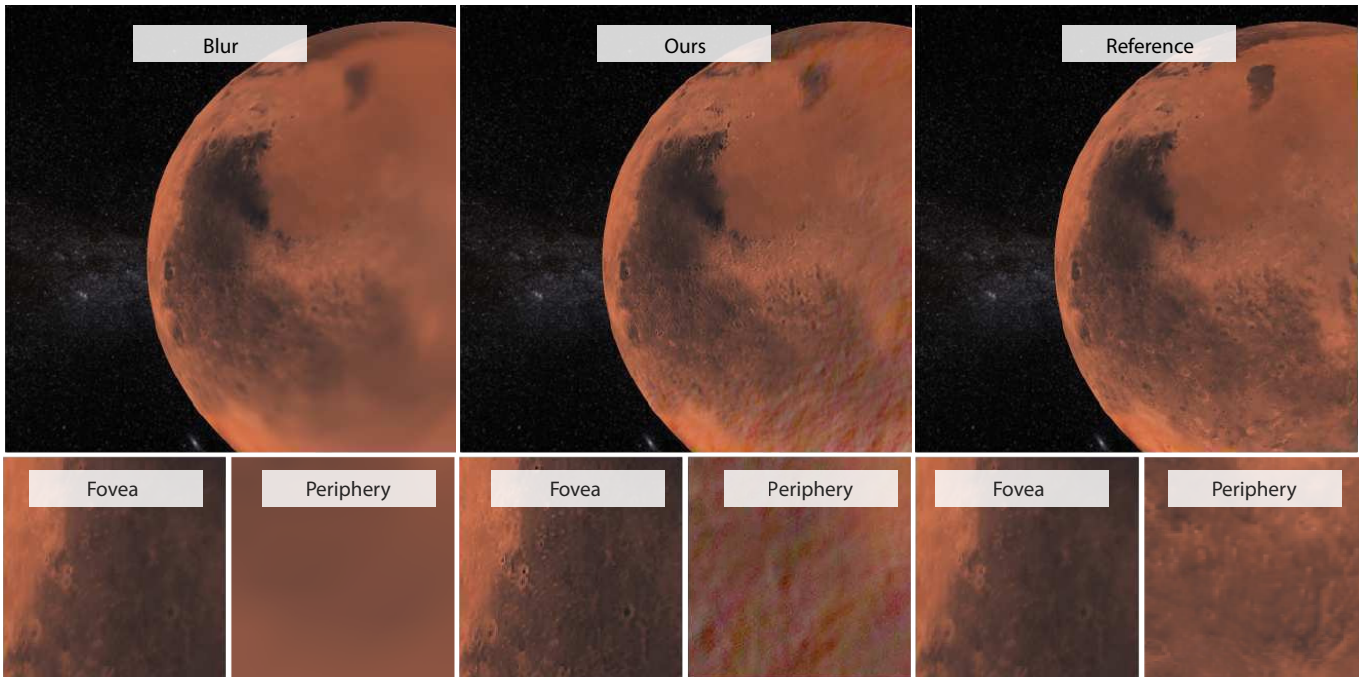

Fig. 13. Our ventral metamer textures applied to a model of planet Mars. All methods **(columns)** reproduce a sharp fovea. Foveated rendering reduces bandwidth and memory by accessing only high MIP levels in the periphery. This leads to a blurry periphery. We also ever only access high MIP levels in the fovea, but these store statistics, allowing us to produce details no the fly, which are perceived similar to a reference by peripheral vision.

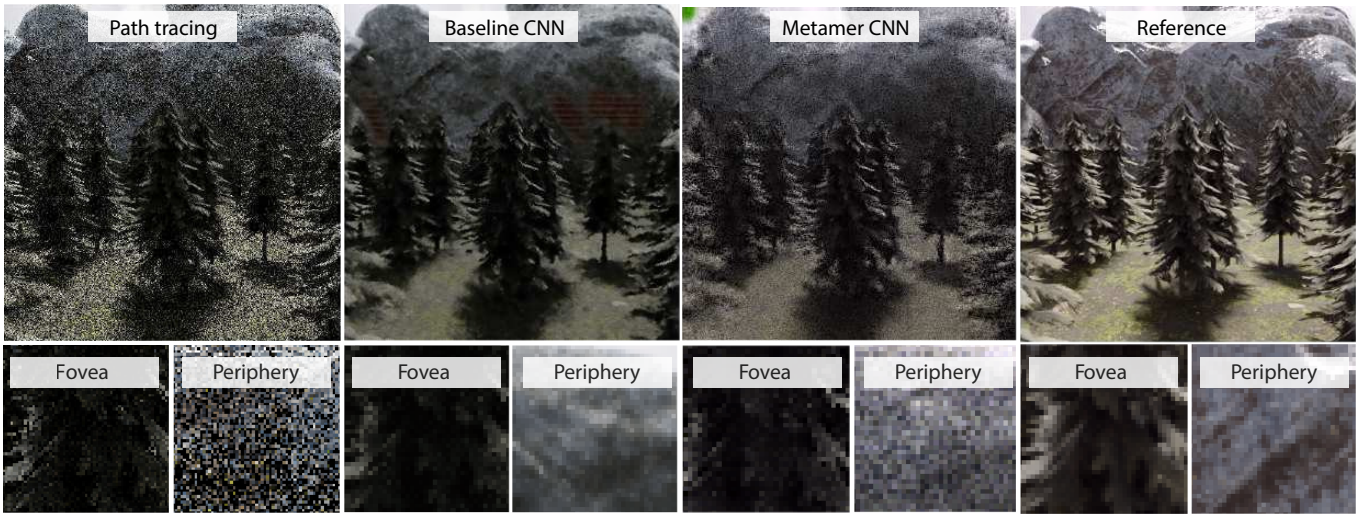

Fig. 14. Path tracing denoising application. Here, we compare two CNNs that denoise MC path-traced images **(1st column)**. The baseline CNN **(2nd column)** maps noisy images to to complete RGB images. In the periphery, all noise is cleaned, but in the periphery, edges are hallucinated and smooth values are put in-between. Our CNN **(3rd column)** maps a noisy MC image to a moment map, that is then metamerized. The resulting image has no noise in the fovea. In the periphery, however, the result contains noise matching the statistics of the reference **(4th column)**.

6.2 Preference Experiment

This study measured how users judged images with foveation methods applied. Previous works, e.g., Patney et al. [2016], reported anecdotally that blur resulted in a sense of "tunnel vision". We set out to determine if we saw similar effects for the methods we study by asking user for preferences.

Protocol. Participants ($N = 10$) were shown pairs of images, time-divided, for 0.5 seconds each with a randomized display order. Balanced numbers ($N = 6$) of six method-combinations (Fig. 18) were shown, for each scene ($N = 7$), for a total of 252 decisions per participant. For blur-rate, we chose the baseline of 0.35 (Patney et al.

Fig. 15. Participant view of experiment. Introduction screen for task performance study. To avoid Troxler fading [Clarke 1960], an undistorted skybox was presented as the background.

Fig. 16. Task performance experiment stimuli at 25 deg eccentricity. Left to right: Reference, Blur at 0.35 and Ours.

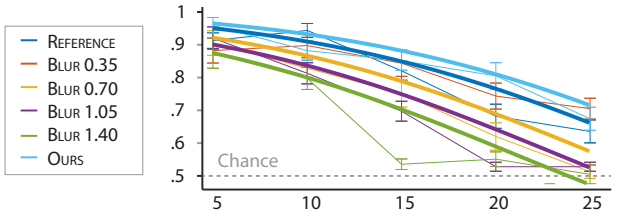

Fig. 17. Probability of correctly classifying the symbol as a function of eccentricity, for Reference, four Blur, and Our method. Thin lines are the data, thick lines the Logistic Fit. Error bars indicate standard error.

[2016]). We also modified the classic metamer F&S method to work with epirectangular projections.

It is not feasible to make this study robust without an eye tracker. Instead, we proceeded under the expectation that there would be bias in the results, and our analysis would focus on relative responses.

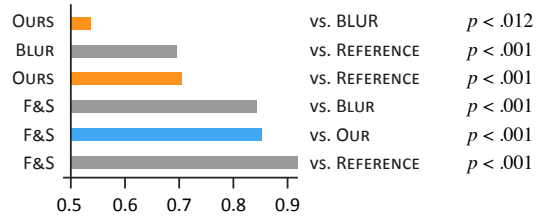

Fig. 18. Preferences as proportions for different treatments.

Analysis. Fig. 18 shows preferences as probabilities for each combination. For each pair we perform a binomial test to check significance compared to chance. A number of significant results validate the protocol and suggest that lack of significance in others could be due to perceptual equivalence. A Spearman's rank correlation with time showed no evidence of learning for Blur, F&S or Ours ($p > 0.85, 0.35, 0.76$). As expected there is a bias towards the Reference, as we did not control for gaze. There is no significant difference between Ours and Blur at the 0.01 level. Our method is strongly preferred over classic metamers F&S however.

This outcome is at odds with the common observation that blur is an artifact in foveated rendering. However, it could either be because there is no perceived difference and hence preference does not matter or because participants perceived the difference but simply did not prefer it e.g., because they like blurry peripheries, reminding them of depth-of-field. We conducted an additional detection experiment to separate these possible explanations.

6.3 Detection Experiment

Our preference study revealed participants made distinctions between images with significance, however it is difficult to know the criteria with which naïve participants make such judgments, especially as we could not control for gaze. Apparently, subjects seem to prefer blur, probably associated with depth-of-field and the absence of artifacts. To explore peripheral perception further, we perform a study focused on effect detection which is more relevant to our use case of foveated rendering.

Protocol. Participants ($N = 10$) were shown images bisected at a random angle $[0, 360]$ for 0.5 seconds. Each image was either a control image (Reference), or one side was foveated, while the other had either the same foveation (consistent), or showed the reference (inconsistent). Control images are inherently consistent. Three foveation methods were used: Ours, Blur and F&S. The area around the bisector was alpha-blended to avoid introducing additional frequencies. Participants were asked to indicate whether or not the overall image was *consistent*. Each participant saw 6 consistent and 6 inconsistent images for each method ($N = 3$), in addition to 6 reference images, for each scene ($N = 7$), for a total of 294 decisions. The random angle made the study robust to lack of eye tracking. Fig. 19 shows how to construct an inconsistent bisected image with one of the three foveation methods. Consistent stimuli are just the Reference image, or the full image processed with one of the three foveation methods.

Analysis. The measures of interest were (a) whether users could *correctly* detect a method is consistent and (b) the amount of *bias* by which a method is considered consistent, regardless if it was or not. The first one measures if answers are correct. The second measures if answers are wrong, how they are biased.

The two measures are closely related. Conditions were balanced, so if participants answered correctly the majority of the time, there would be an equal distribution of choices between Consistent and Inconsistent. If participants answered incorrectly however, they may tend to over- or underestimate the consistency of particular methods. This would be reflected in the total proportion of Consistent to

Fig. 19. Preparation of inconsistent stimuli for detection experiment. A random method X is selected out of three foveation methods. This is blended with the Reference image according to an alpha gradient along a randomly chosen direction.

Inconsistent responses. If a response is biased towards a method, that may indicate a subliminal preference.

Fig. 20 shows the probability of answering correctly and the bias. The control treatment (Reference-Reference) shows that the protocol is working, as participants judged correctly the vast majority of the time, above 80 %. Participants were unable to make accurate conscious judgments as correctness was not significant for any foveation method. When we look at the bias however, we see participants rated Ours as consistent with a significantly higher chance than Blur, or F&S (Fig. 20, right). So even if subjects did not express preference and the task of detection is hard, they significantly and with a strong effect tend to perceive Ours to be more consistent, which is the aim of this work and foveated rendering in general. The significance of some treatments is evidence that the lack of significance in others could be attributed to perceptual equivalence, rather than a protocol failure.

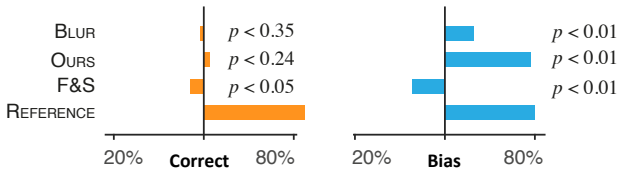

Fig. 20. Proportion of images judged correctly as *consistent* (orange), and the *bias* towards or away from judging a method as consistent (blue). Bars are annotated with the p-values of a binomial test comparing to chance.

6.4 Discussion

Our evaluation shows that our metamers retain information necessary match the ground truth in a task performance study, whereas performance quickly degrades with blur rate (Sec. 6.1). When participants were conscious of the difference between our metamers and blur, our metamers were no less desirable than blur (Sec. 6.2). When asked to detect the effect however, participants were unable to make accurate conscious judgments, but did show a bias in favor of our metamers, with the overall ratings closest to the reference of any foveation method (Sec. 6.3). This demonstrates the potential of our method, however we still need to evaluate it within a full VR pipeline with a working eye tracker to confirm the results with varying gaze. Additionally these experiments focused on static scenes; quantifying the perception of metamers in motion remains future work. An important building block, would be to rely on temporally coherent noise [Kass and Pesare 2011].

7 Conclusion

We present a new rendering method for generating high-quality ventral metamers for foveated rendering at real-time rates. Our method relies on the key idea of accounting for the pooling characteristics of the HVS. Furthermore, with the help of user studies, we show that we match the pooled statistics in a realistic and accurate manner. Our approach demonstrates that ventral metamers can address an inherent problem in the foveated rendering literature, and open the gate towards a series of important applications in graphics ranging from compression to texturing or de-noising. We believe our approach can play a key role in bridging the gap between foveated rendering pipelines and their counterparts in next generation foveated near-eye displays.

Acknowledgments

This work was funded by the EPSRC/UKRI project EP/T01346X/1.

References

Kaan Akşit, Praneeth Chakravarthula, Kishore Rathinavel, Youngmo Jeong, Rachel Albert, Henry Fuchs, and David Luebke. 2019. Manufacturing application-driven foveated near-eye displays. *IEEE Trans Vis and Comp Graph* 25, 5 (2019), 1928–1939.

Rachel Albert, Anjul Patney, David Luebke, and Joohwan Kim. 2017. Latency requirements for foveated rendering in virtual reality. *ACM Trans App Perc* 14, 4 (2017).

Stuart M Anstis. 1974. A chart demonstrating variations in acuity with retinal position. *Vis Res* 14, 7 (1974), 589–592.

H Aubert and R Förster. 1857. Beiträge zur Kenntniss des indirecten Sehens.(I). Untersuchungen über den Raumsinn der Retina. *Archiv für Ophthalmologie* 3, 2 (1857), 1–37.

Steve Bako, Thijs Vogels, Brian McWilliams, Mark Meyer, Jan Novák, Alex Harvill, Pradeep Sen, Tony Derose, and Fabrice Rousselle. 2017. Kernel-predicting convolutional networks for denoising Monte Carlo renderings. *ACM Trans Graph* 36, 4 (2017), 97–1.

Herman Bouma. 1970. Interaction effects in parafoveal letter recognition. *Nature* 226, 5241 (1970), 177–178.

Matteo Carandini, Jonathan B. Demb, Valerio Mante, David J. Tolhurst, Yang Dan, Bruno A. Olshausen, Jack L. Gallant, and Nicole C. Rust. 2005. Do we know what the early visual system does? *J Neuroscience* 25, 46 (2005), 10577–10597.

Chakravarty R Alla Chaitanya, Anton S Kaplanyan, Christoph Schied, Marco Salvi, Aaron Lefohn, Derek Nowrouzezahrai, and Timo Aila. 2017. Interactive reconstruction of Monte Carlo image sequences using a recurrent denoising autoencoder. *ACM Trans Graph* 36, 4 (2017), 1–12.

FJJ Clarke. 1960. A study of Troxler's effect. *Optica Acta: Int J Optics* 7, 3 (1960), 219–236.

Arturo Deza, Aditya Jonnalagadda, and Miguel Eckstein. 2017. Towards metamerism via foveated style transfer. *arXiv:1705.10041* (2017).

Annette J. Dobson and Adrian G. Barnett. 2018. *An Introduction to Generalized Linear Models.* Chapman and Hall/CRC.

William Donnelly and Andrew Lauritzen. 2006. Variance shadow maps. In *Proc. i3D.* 161–165.

Andrew T Duchowski, Nathan Cournia, and Hunter Murphy. 2004. Gaze-Contingent Displays: A Review. *CyberPsychology & Behavior* 7, 6 (2004), 621–634.

Alexei A Efros and Thomas K Leung. 1999. Texture synthesis by non-parametric sampling. In *ICCV*, Vol. 2. 1033–1038.

Mark D Fairchild. 2013. *Color appearance models.* John Wiley & Sons.

Jenelle Feather, Alex Durango, Ray Gonzalez, and Josh McDermott. 2019. Metamers of neural networks reveal divergence from human perceptual systems. *NeurIPS* 32 (2019), 1–25.

Jeremy Freeman and Eero P Simoncelli. 2011. Metamers of the ventral stream. *Nature Neuroscience* 14, 9 (2011), 1195–1201.

William T Freeman, Edward H Adelson, et al. 1991. The design and use of steerable filters. *IEEE PAMI* 13, 9 (1991), 891–906.

Sebastian Friston, Tobias Ritschel, and Anthony Steed. 2019. Perceptual rasterization for head-mounted display image synthesis. *ACM Trans Graph* 38, 4 (2019), 1–14.

Masahiro Fujita and Takahiro Harada. 2014. Foveated real-time ray tracing for virtual reality headset. *SIGGRAPH Asia Posters* 14 (2014).

Bruno Galerne, Ares Lagae, Sylvain Lefebvre, and George Drettakis. 2012. Gabor noise by example. *ACM Trans Graph* 31, 4 (2012), 1–9.

Leon A Gatys, Alexander S Ecker, and Matthias Bethge. 2016. Image style transfer using convolutional neural networks. In *CVPR*. 2414–2423.

Wilson S Geisler and Jeffrey S Perry. 1998. Real-time foveated multiresolution system for low-bandwidth video communication. In *HVIE III*, Vol. 3299. 294–305.

John A. Greenwood, Peter J. Bex, and Steven C. Dakin. 2009. Positional averaging explains crowding with letter-like stimuli. *Proc NAS US* 106, 31 (2009), 13130–13135.

Umut Güçlü and Marcel A.J. van Gerven. 2015. Deep neural networks reveal a gradient in the complexity of neural representations across the ventral stream. *J Neuroscience* 35, 27 (2015), 10005–10014.

Brian Guenter, Mark Finch, Steven Drucker, Desney Tan, and John Snyder. 2012. Foveated 3D graphics. *ACM Trans Graph* 31, 6 (2012), 1–10.

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

David J Heeger and James R Bergen. 1995. Pyramid-based texture analysis/synthesis. In *Proc. SIGGRAPH*. 229–238.

David Hoffman, Zoe Meraz, and Eric Turner. 2018. Limits of peripheral acuity and implications for VR system design. *J SID* 26, 8 (2018), 483–495.

Xun Huang and Serge Belongie. 2017. Arbitrary style transfer in real-time with adaptive instance normalization. In *ICCV*. 1501–1510.

David H. Hubel. 1982. Exploration of the primary visual cortex, 1955–78. *Nature* 299, 5883 (oct 1982), 515–524.

Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. 2017. Image-to-image translation with conditional adversarial networks. In *CVPR*. 1125–1134.

Nima Khademi Kalantari, Steve Bako, and Pradeep Sen. 2015. A machine learning approach for filtering Monte Carlo noise. *ACM Trans Graph* 34, 4 (2015), 122–1.

Anton S Kaplanyan, Anton Sochenov, Thomas Leimkühler, Mikhail Okunev, Todd Goodall, and Gizem Rufo. 2019. DeepFovea: Neural reconstruction for foveated rendering and video compression using learned statistics of natural videos. *ACM Trans Graph* 38, 6 (2019), 1–13.

Michael Kass and Davide Pesare. 2011. Coherent noise for non-photorealistic rendering. *ACM Trans.Graph. (TOG)* 30, 4 (2011), 1–6.

Jonghyun Kim, Youngmo Jeong, Michael Stengel, Kaan Akşit, Rachel Albert, Ben Boudaoud, Trey Greer, Joohwan Kim, Ward Lopes, Zander Majercik, et al. 2019. Foveated AR: dynamically-foveated augmented reality display. *ACM Trans Graph* 38, 4 (2019), 1–15.

Min H Kim, Tobias Ritschel, and Jan Kautz. 2011. Edge-aware color appearance. *ACM Trans Graph* 30, 2 (2011), 1–9.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv:1412.6980* (2014).

Ares Lagae, Peter Vangorp, Toon Lenaerts, and Philip Dutré. 2010. Procedural isotropic stochastic textures by example. *Computers & Graphics* 34, 4 (2010), 312–321.

Gordon E Legge and Daniel Kersten. 1987. Contrast discrimination in peripheral vision. *J OSA A* 4, 8 (1987), 1594–1598.

Marc Levoy and Ross Whitaker. 1990. Gaze-directed volume rendering. In *Proc. i3D*. 217–223.

Lin Liang, Ce Liu, Ying-Qing Xu, Baining Guo, and Heung-Yeung Shum. 2001. Real-time texture synthesis by patch-based sampling. *ACM Trans Graph* 20, 3 (2001), 127–150.

Rafał Mantiuk, Kil Joong Kim, Allan G. Rempel, and Wolfgang Heidrich. 2011. HDR-VDP-2. *ACM Trans Graph* 30, 4 (2011), 1–14.

Xiaoxu Meng, Ruofei Du, Matthias Zwicker, and Amitabh Varshney. 2018. Kernel foveated rendering. *Proc. i3D* 1, 1 (2018), 1–20.

Hunter Murphy and Andrew T Duchowski. 2001. Gaze-contingent level of detail rendering. *Proc. Eurographics* (2001).

Toshikazu Ohshima, Hiroyuki Yamamoto, and Hideyuki Tamura. 1996. Gaze-directed adaptive rendering for interacting with virtual space. *Proceedings - Virtual Reality Annual International Symposium* (1996), 103–110.

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* 35, 6 (2016), 179.

Ken Perlin. 1985. An image synthesizer. *ACM Siggraph Computer Graphics* 19, 3 (1985), 287–296.

Javier Portilla and Eero P Simoncelli. 2000. A parametric texture model based on joint statistics of complex wavelet coefficients. *Int J Comp Vis* 40, 1 (2000), 49–70.

Charles Poynton. 2012. *Digital Video and HD: Algorithms and Interfaces*. Morgan Kaufmann. 752 pages.

Eyal M. Reingold, Lester C. Loschky, George W. McConkie, and David M. Stampe. 2003. Gaze-contingent multiresolutional displays: An integrative review. *Human Factors* 45, 2 (2003), 307–328.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *MICCAI*. Cham, 234–241.

R Rosenholtz. 2016. Capabilities and Limitations of Peripheral Vision. *Annual review of vision science* 2 (2016), 437.

Ruth Rosenholtz, Jie Huang, Alvin Raj, Benjamin J. Balas, and Livia Ilie. 2012. A summary statistic representation in peripheral vision explains visual search. *J Vision* 12, 4 (2012), 14–14.

Daniel L. Ruderman, Thomas W. Cronin, and Chuan-Chin Chiao. 1998. Statistics of cone responses to natural images: implications for visual coding. *J OSA A* 15, 8 (1998), 2036–2045.

Anita M. Schmid, Keith P Purpura, Ifije E Ohiorhenuan, Ferenc Mechler, and Jonathan D Victor. 2009. Subpopulations of neurons in visual area V2 perform differentiation and integration operations in space and time. 3 (2009), 1–16.

Albulena Shaqiri, Maya Roinishvili, Lukasz Grzeczkowski, Eka Chkonia, Karin Pilz, Christine Mohr, Andreas Brand, Marina Kunchulia, and Michael H. Herzog. 2018. Sex-related differences in vision are heterogeneous. *Scientific Rep* 8, 1 (2018), 7521.

Josef Spjut, Ben Boudaoud, Jonghyun Kim, Trey Greer, Rachel Albert, Michael Stengel, Kaan Aksit, and David Luebke. 2019. Toward standardized classification of foveated displays. *arXiv:1905.06229* (2019).

Michael Stengel, Steve Grogorick, Martin Eisemann, and Marcus Magnor. 2016. Adaptive image-space sampling for gaze-contingent real-time rendering. 35, 4 (2016), 129–39.

Hans Strasburger. 2020. Seven Myths on Crowding and Peripheral Vision. *i-Perception* 11, 3 (2020).

Hans Strasburger, Ingo Rentschler, and Martin Jüttner. 2011. Peripheral vision and pattern recognition: A review. *J Vision* 11, 5 (2011), 13–13.

Nicholas T Swafford, José A Iglesias-Guitian, Charalampos Koniaris, Bochang Moon, Darren Cosker, and Kenny Mitchell. 2016. User, metric, and computational evaluation of foveated rendering methods. In *Proc. SAP*. 7–14.

Keiji Tanaka. 1996. Inferotemporal Cortex and Object Vision. *Ann Rev Neuro* 19, 1 (1996), 109–39.

V Javier Traver and Alexandre Bernardino. 2010. A review of log-polar imaging for visual perception in robotics. *Robotics and Autonomous Systems* 58, 4 (2010), 378–398.

Okan Tarhan Tursun, Elena Arabadzhiyska-Koleva, Marek Wernikowski, Radosław Mantiuk, Hans-Peter Seidel, Karol Myszkowski, and Piotr Didyk. 2019. Luminance-contrast-aware foveated rendering. *ACM Trans Graph* 38, 4 (2019), 1–14.

Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. 2016. Instance normalization: The missing ingredient for fast stylization. *arXiv:1607.08022* (2016).

Leslie G Ungerleider and James V Haxby. 1994. 'What' and 'where'in the human brain. *Current Opinion in Neurobiology* 4, 2 (1994), 157–65.

Karthik Vaidyanathan, Marco Salvi, Robert Toth, Tim Foley, Jim Akenine-Möller, Tomas Nilsson, Jacob Munkberg, Jon Hasselgren, Masamichi Sugihara, Petrik Clarberg, Tomasz Janczak, and Aaron Lefohn. 2014. Coarse Pixel Shading. In *Proc. HPG*.

Thomas SA Wallis, Matthias Bethge, and Felix A Wichmann. 2016. Testing models of peripheral encoding using metamerism in an oddity paradigm. *J Vision* 16, 2 (2016), 4–4.

Li-Yi Wei, Sylvain Lefebvre, Vivek Kwatra, and Greg Turk. 2009. State of the Art in Example-based Texture Synthesis. In *Eurographics STAR*. 93–117.

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. 35, 7 (2016), 289–298.

G. N. Wilkinson and C. E. Rogers. 1973. Symbolic Description of Factorial Models for Analysis of Variance. *J Royal Stat Soc C* 22, 3 (1973), 392–399.

Lance Williams. 1983. Pyramidal parametrics. In *Proc. SIGGRAPH*. 1–11.

Xin Zhang, Wei Chen, Zhonglei Yang, Chuan Zhu, and Qunsheng Peng. 2011. A new foveation ray casting approach for real-time rendering of 3D scenes. *Proc CAD/Graphics* (2011), 99–102.