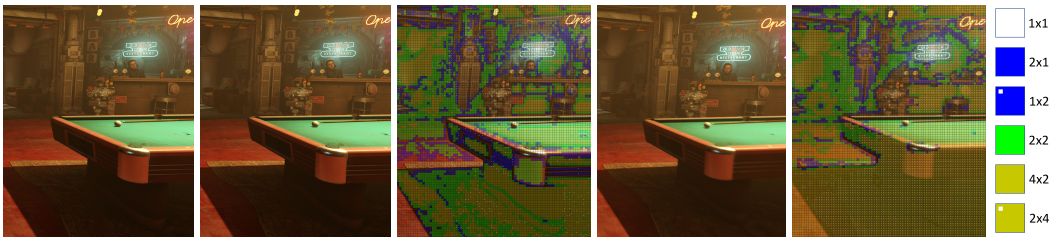
Visually Lossless Content and Motion Adaptive Shading in Games

LEI YANG, DMITRY ZHDAN, EMMETT KILGARIFF, ERIC B. LUM, YUBO ZHANG, MATTHEW JOHNSON, and HENRIK RYDGÅRD, NVIDIA Corporation



(a) Full rate (stationary) (b) Adaptive (stationary) (c) Shading Rate (stationary) (d) Adaptive (in motion) (e) Shading Rate (in motion)

Fig. 1. Content and motion adaptive shading in *Wolfenstein II: The New Colossus*. Adaptive shading (*b*) using per-tile shading rate (*c*) produces an image visually equivalent to one shaded at full rate (*a*). With motion present, more aggressively lowered shading rates may be used (*e*); imperfections are hidden by display or engine-generated motion blur (*d*). Shading rate per-tile is color-coded based on shading region size.

We present a technique that adaptively adjusts the shading rate during rendering based on the scene content and motion. Our goal is to improve performance with no loss in perceived quality. We determine per-screen-tile shading rate by testing an error estimate against a perceptually-corrected just-noticeable difference threshold. Our design features an effective and efficient error estimate using spatial and frequency analysis of half and quarter rate shading. We also study the effect of motion in reducing perceived error, a consequence of display-persistence and/or motion blur effects. Our implementation uses the computed per-tile shading rate with variable rate shading (a recent GPU feature) to lower shading cost. We demonstrate the quality and performance of our technique on two high-end game engines and shipped games.

CCS Concepts: • **Computing methodologies** → **Rendering**;

Additional Key Words and Phrases: variable rate shading, adaptive shading, real-time rendering, rasterization, motion blur, frequency domain, perception, image quality, rendering optimization

ACM Reference Format:
Lei Yang, Dmitry Zhdan, Emmett Kilgariff, Eric B. Lum, Yubo Zhang, Matthew Johnson, and Henrik Rydgård. 2019. Visually Lossless Content and Motion Adaptive Shading in Games. *Proc. ACM Comput. Graph. Interact. Tech.* 2, 1, Article 6 (May 2019), 18 pages. https://doi.org/10.1145/3320287

Authors' address: Lei Yang; Dmitry Zhdan; Emmett Kilgariff; Eric B. Lum; Yubo Zhang; Matthew Johnson; Henrik Rydgård, NVIDIA Corporation, 2788 San Tomas Expressway, Santa Clara, California, 95051.

2577-6193/2019/5-ART6 $15.00
https://doi.org/10.1145/3320287

1 INTRODUCTION

The desire for richer, more realistic rendering is dramatically increasing the per-pixel shading cost in modern video game contents. This compounded with the trend toward higher resolution and increased framerate, satisfying the shading demand is increasingly difficult, even with rapid growth in modern GPU computing capabilities. Additionally, on mobile gaming platforms, power saving through shading reduction is increasingly necessary to deliver increased battery life.

Today's GPUs support various mechanisms to reduce shading cost. Examples include multi-sample antialiasing, mixed resolution shading, decoupled and texture-space shading, checkerboard rendering, and variable-rate (coarse-pixel) shading. Sec. 2.1 gives an overview. Some of these methods can be used to adaptively redistribute computation among different components of scene content or shading, in order to improve performance without sacrificing visual quality.

We present an algorithm that can adaptively reduce shading rate per fixed-sized screen tile by leveraging Variable Rate Shading (VRS), a new GPU feature. We focus on deriving a quality loss estimator and mechanisms to determine at runtime when shading rate can be reduced without leading to noticeable artifacts.

Our algorithm can be efficiently implemented in a compute shader that runs immediately before the main scene rasterization and lighting passes. The algorithm can essentially be divided into three stages:

(1) Analyzing the pixels from the previously rendered frame using the proposed loss estimator;
(2) Predicting the error per screen tile in the current frame under different reduced shading rates and the motion velocity;
(3) Using a perceptually corrected visibility threshold to decide which shading rate to use per screen tile.

To achieve high efficiency we compute only one loss estimator per screen tile (Sec. 3.1), and reuse it to estimate quality loss in that tile with various shading rates and the motion velocity. We achieve the reuse by transferring the loss energy to frequency domain (Sec. 3.2), and scale the loss energy under a different shading rate (Sec. 3.3) and motion (Sec. 4.1). We also present an optional simplification to achieve motion adaptive shading without computing the loss estimator (Sec. 4.2). Finally, we discuss our implementation of the algorithm in modern games (Sec. 5), and demonstrate the resulting performance improvements and quality preservation (Sec. 6).

To summarize, our main contributions are:

- A quantitative analysis of perceptual quality loss associated with variable rate shading and motion blur;
- A simple loss estimator, plus scaling functions for choosing per-screen-tile shading rate;
- An efficient implementation for computing adaptive shading rate with very low overhead.

2 BACKGROUND AND RELATED WORK

2.1 Variable-Rate Shading

There are a wide range of techniques that target lowering the amount of shading computation per display pixel. Multi-resolution or mixed resolution shading [Shopf 2009; Yang et al. 2008] renders expensive and low-frequency shading components in low resolution and then bilateral upsamples the results to full resolution. Decoupled shading [Clarberg et al. 2013; Liktor and Dachsbacher 2012; Ragan-Kelley et al. 2011] separates the shading rate from geometry or visibility sampling rate by establishing a mapping between the visibility samples and shading samples, and sharing or reusing shading samples when possible. A variant of decoupled shading is texture-space shading [Andersson et al. 2014; Burns et al. 2010; Clarberg et al. 2014; Hillesland and Yang 2016] that

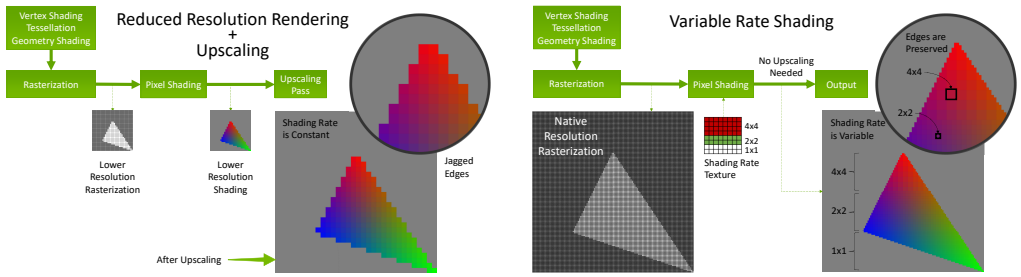


Fig. 2. Comparison of fixed low-resolution rendering (left) and variable rate shading (right). Diagrams courtesy of Bhonde [2018].

computes shading in texture or patch space in an appropriate density controlled by the mip level. These techniques require a significant change in the rendering pipeline.

Coarse Pixel Shading [He et al. 2014; Vaidyanathan et al. 2014; Xiao et al. 2018] or Variable Rate Shading (VRS) [Bhonde 2018; NVIDIA 2018] is a feature of today's GPUs, like NVIDIA's Turing and Intel's Gen11 architectures. VRS can be thought of as an extension of multi-sample antialiasing [Akeley 1993], by allowing shading to be computed once and used by multiple pixels. It preserves visibility resolution while allowing for shading rate to vary across the rendered frame on a granularity of a fixed tile size (e.g. 16×16). Perhaps most importantly, taking advantage of VRS has little impact on the rendering pipeline.

We implemented our adaptive shading algorithm on NVIDIA's Turing GPUs. In Fig. 2 we illustrate how VRS works on Turing. For each 16x16 tile of screen-space samples, the shading rate can be selected from 1×1, 1×2, 2×1, 2×2, 2×4, 4×2 and 4×4 samples per shade. Rate is specified using a shading rate texture that stores a byte per 16×16 sample tile to specify the shading rate. The lower half of the byte is used to lookup a shading rate from a programmable table. As rasterization proceeds the shading rate texture is read to determine the shading frequency for each 16×16 tile.

2.2 Perception Guided Adaptive Shading

The idea of adjusting shading or sampling rate adaptively based on content has a long history, particularly in the context of ray-tracing [Zwicker et al. 2015], where the sampling rate can be adjusted easily. Mitchell [1987] was the first to propose using contrast in a region to measure perceived image variation, which is used to guide whether the region needs a higher than moderate sampling rate. Bolin and Meyer [1998] and Farrugia and Péroche [2004] proposed a perceptually-based image error predictor model for guiding adaptive sampling decision. Both techniques employ a comprehensive model of the human vision system to detect image differences, but they are not suitable for real-time applications due to complexity. Besides, most ray-tracing based adaptation methods focus on estimating the per-pixel variance resulted from Monte-Carlo integral. Although related, this is different from what we need for variable rate shading applications, where the error between a coarse- and fine-shaded image needs to be estimated.

Since the mechanisms to control shading rate for real-time, rasterization-based renderers are only available recently, the problem is relatively new. Most recent advances are focusing on virtual reality applications, where there is a more crucial need for efficient shading. Multi-projection-based methods [Toth et al. 2016], such as Multi-Res Shading [NVIDIA 2015] and Lens-Matched Shading [NVIDIA 2016], employ a non-uniform shading rate on the shading buffer to approximate a 1:1 mapping between shading and display pixels, due to the non-linear lens warp compensation.

Foveated rendering [Patney et al. 2016; Stengel et al. 2016] adjusts shading rate in order to take advantage of the visual acuity fall off present in the human visual system. Stengel et al. [2016] further adapts shading rate by considering visual detail, including influences of texture, material properties, specular highlights, and brightness. This is similar to our goal, except that we seek a formal derivation of the error estimate specifically for the shading sample pattern available on hardware. With a closed form estimate, we are also able to introduce motion influence into the adaptation.

2.3 Perceptual Image Quality in Motion

Most modern LCDs show a frame of pixels continuously until the next frame is ready to replace the current. This sample and hold phenomenon allows for constant brightness with varying refresh rates, and avoids visible screen flicker. However, when objects are in motion, a sample and hold monitor introduces blurriness. Blur Buster's Motion Tests [TestUFO.com 2017] has a scrolling "UFO" test that illustrates this phenomenon. For a moving object, the eye tracks the object with a linear, continuous motion, while the displayed object is stationary for a frame time before jumping to the new position. This leads to the object being smeared across the retina, creating a blurred appearance. The effect is very apparent at refresh rates of 30 and 60Hz. It is also present on higher refresh rate displays such as 120 and 160Hz.

This type of display persistence blur has been well studied [e.g. Chan and Nguyen 2011; Watson et al. 1986]. The effect of the blur is equivalent to applying to the image a 1D box filter that is oriented to the motion direction. The width of the filter is equal to the distance that the displayed object travels within the lit time of a frame. A related effect is post-processing motion blur added by many high-end game engines to create a cinematic feel of rendered animation [Akenine-Moller et al. 2018]. This effect is commonly created with a similar box filter. Either of the effects offers an opportunity to lower the shading rate in a way that would normally cause a visual difference in a static image. Vaidyanathan et al. [2012] proposed a frequency content analysis of motion blur that is used to guide anisotropic adaptive sampling in decouple shading for stochastic rasterization pipelines. Our motion adaptive method is also based on frequency analysis, although targeting a different context. We combine the frequency response of the blurring filter with the detected signal spectrum, so that the method works in conjunction with content adaptive shading to better exploit optimization opportunities.

3 CONTENT ADAPTIVE SHADING

We propose a loss estimator that gauges the image quality degradation when shading rate is reduced. The estimator is simple and efficient to compute in the spatial domain, and it can be extended to handle different shading rates through an analysis in the frequency domain. The computed loss is compared against a perceptually-corrected Just-Noticeable-Difference (JND) threshold to determine if reduced shading rate may be used. The following analysis assumes a 1D image slice, although it can be trivially extended to 2D by computing the horizontal and vertical error estimates and shading rates separately.

3.1 Image Error with Half-Rate Shading

Suppose a single channel luma image tile in full-resolution shading is I, and its individual pixels values in the tile are $I_i, i \in [1, N]$. As illustrated in Fig. 3, with half-rate shading, the resulting pixel

values are:

$$I_i^H = \begin{cases} \dfrac{I_{i-1} + I_i}{2} & \text{if } i \text{ is even} \\ \dfrac{I_i + I_{i+1}}{2} & \text{if } i \text{ is odd.} \end{cases} \tag{1}$$

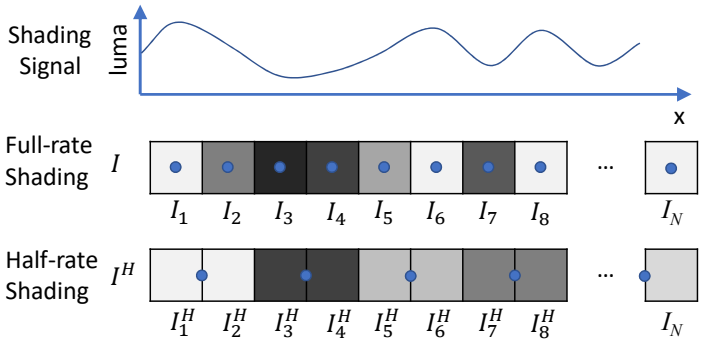


Fig. 3. 1D image slices in full-rate and half rate shading. Blue dots are locations where shading is computed (shading signal is assumed to be band-limited with proper antialiasing techniques applied).

We define an error term between I and I^H across the entire image tile as the L_2 norm[1] of the per-pixel difference:

$$\begin{aligned} \mathcal{E}(I, I^H) &= \left\| I - I^H \right\|_2 \\ &= \sqrt{\frac{1}{N} \sum_{j=1}^{N/2} \left(I_{2j-1} - I_{2j-1}^H \right)^2 + \left(I_{2j} - I_{2j}^H \right)^2} \\ &= \sqrt{\frac{1}{N} \sum_{j=1}^{N/2} \left(I_{2j-1} - \frac{I_{2j-1} + I_{2j}}{2} \right)^2 + \left(I_{2j} - \frac{I_{2j-1} + I_{2j}}{2} \right)^2} \\ &= \sqrt{\frac{1}{N} \sum_{j=1}^{N/2} 2 \left(\frac{I_{2j} - I_{2j-1}}{2} \right)^2} \\ &\approx \sqrt{\frac{1}{N-1} \sum_{i=2}^{N} \left(\frac{I_i - I_{i-1}}{2} \right)^2}. \end{aligned} \tag{2}$$

For the error term, we also have the option to use L_1 norm or L_∞ norm. An L_1 norm computes the average absolute error over the image tile, and an L_∞ norm computes the maximum or worst pixel error over the image tile. To simplify the following derivation we use L_2 norm, which is an average like L_1, but responds to larger errors more like L_∞.

We refer to this term, $\mathcal{E}(I, I^H)$, as the *error estimator*. The last step in Eq.2 approximates a spatially varying filter with a spatially invariant one. This simplifies the following frequency domain analysis, and ensures a stable filter response when the image is offset by an odd number of pixels between successive frames.

[1] L_2 norm is the square root of the commonly used Mean-Squared Error (MSE).

3.2 Frequency Domain Analysis

The term $\frac{I_i - I_{i-1}}{2}$ in the last step of Eq.2 is a linear Finite Impulse Response (FIR) filter $D = [-0.5, 0.5]$ that is convolved with the target image tile. We refer to D as the *differencing filter*. It is a high pass filter that extracts the high frequency contents of the image. With that in mind, we can write Eq.2 as:

$$\mathcal{E}(I, I^H) = \|I * D\|_2 , \tag{3}$$

which computes the energy of the resulting image after high-pass filtering. According to the Convolution and Parseval theorems [Oppenheim and Schafer 2009], this energy term can be computed in frequency domain as:

$$\begin{aligned} \mathcal{E}(I, I^H) &= \|I * D\|_2 \\ &= \|\mathcal{F}(I) \cdot \mathcal{F}(D)\|_2 , \end{aligned} \tag{4}$$

where $\mathcal{F}(\cdot)$ is the discrete Fourier transform, $*$ denotes convolution, and $\cdot$ denotes element-wise multiplication.

We also observe that the differencing filter D can be represented as the difference between an identity filter (single impulse) $H = [1]$ and a box filter $B_2 = [0.5, 0.5]$ that is two-pixel wide:

$$D = H - B_2 . \tag{5}$$

When both sides are convolved with an image, this filter computes the difference between the original image and the image filtered by the box filter B_2:

$$I * D = I - I * B_2 . \tag{6}$$

Note that the box filter B_2 is exactly the one used in image formation with half-rate shading. Transforming Eq.6 to the frequency domain and substituting into Eq.4, we have:

$$\mathcal{E}(I, I^H) = \|\mathcal{F}(I) \cdot (\mathbb{1} - \mathcal{F}(B_2))\|_2 . \tag{7}$$

Eq.7 is important because it establishes a connection between the error estimator in spatial domain and the frequency response of the box filter implied by half-rate shading.

3.3 Quarter-Rate Shading

To extend this analysis to quarter-rate shading, we replace the filter B_2 with a four-pixel wide box filter $B_4 = [0.25, 0.25, 0.25, 0.25]$. However, for run-time efficiency, we prefer to reuse the differencing filter D instead of computing a new one. To establish a connection between them, we start by aligning the error term between half-rate and quarter-rate shading in the form of Eq.7 :

$$\mathcal{E}(I, I^Q) = \|\mathcal{F}(I) \cdot (\mathbb{1} - \mathcal{F}(B_4))\|_2 \tag{8}$$

$$= \|\mathcal{F}(I) \cdot (\mathbb{1} - \mathcal{F}(B_2)) \cdot \mathcal{K}\|_2 . \tag{9}$$

Here $\mathcal{K}$ is a constant vector in the frequency domain that encapsulates the difference between $(\mathbb{1} - \mathcal{F}(B_4))$ and $(\mathbb{1} - \mathcal{F}(B_2))$ in all frequency bins. Considering the image tile I as random variables, we take the expectation of both Eq.8 and Eq.9:

$$E\left(\|\mathcal{F}(I) \cdot (\mathbb{1} - \mathcal{F}(B_4))\|_2\right) = E\left(\|\mathcal{F}(I) \cdot (\mathbb{1} - \mathcal{F}(B_2)) \cdot \mathcal{K}\|_2\right) . \tag{10}$$

By using the Delta method [Casella and Berger 2002], we can move the expectation operator into the norm under approximation:

$$\|E\left(\mathcal{F}(I)\right) \cdot (\mathbb{1} - \mathcal{F}(B_4))\|_2 = \|E\left(\mathcal{F}(I)\right) \cdot (\mathbb{1} - \mathcal{F}(B_2)) \cdot \mathcal{K}\|_2 . \tag{11}$$

Vision studies conclude that natural images have an average Fourier amplitude slope of -1 in log space, meaning that the amplitude of Fourier transform decreases with increasing spatial frequency

approximately as $1/f$ [Field 1987; Penacchio and Wilkins 2015]. With rendered images we expect similar statistics. Therefore we substitute $E(\mathcal{F}(I))$ by an image-agnostic term:

$$\|\mathcal{S} \cdot (\mathbb{1} - \mathcal{F}(B_4))\|_2 = \|\mathcal{S} \cdot (\mathbb{1} - \mathcal{F}(B_2)) \cdot \mathcal{K}\|_2 . \tag{12}$$

where $\mathcal{S} = 1/f$. Since the equation no longer depends on the input image, it is now possible to reduce $\mathcal{K}$ to a scalar and move it out of the norm.

$$\|\mathcal{S} \cdot (\mathbb{1} - \mathcal{F}(B_4))\|_2 = k \cdot \|\mathcal{S} \cdot (\mathbb{1} - \mathcal{F}(B_2))\|_2 . \tag{13}$$

By solving this equation numerically, we get $k = 2.13$. Note that for contents that deviate significantly from natural images, k can be adjusted or refitted based on the spatial frequency distribution of the images. However in common cases we do not expect it to deviate very far from this value.

Eq.13 leads us to an image error estimate for quarter-rate shading that reuses the existing half-rate error estimator:

$$\mathcal{E}(I, I^Q) \approx k \cdot \mathcal{E}(I, I^H). \tag{14}$$

3.4 Shading Rate Adaptation with a Perceptually-Corrected Threshold

The term $\mathcal{E}$ is the L_2 norm of the pixel (luma) value error. Our goal is to find a threshold to $\mathcal{E}$ so that errors lower than the threshold are unlikely to be detected by the viewer. Based on Weber's law [Tan and Gan 2015], we define a Just-Noticeable Difference (JND) threshold as:

$$\tau_I = t \cdot (I_{\text{avg}} + l), \tag{15}$$

where I_{avg} is the average (background) luma in the image tile, t is a user selected sensitivity threshold, and l is the environment luminance that affects the sensitivity especially on dark ranges.

Given the detection filter response and the JND threshold, the shading rate of the tile can then be determined as:

$$S_I = \begin{cases} \text{Full}, & \text{if } \mathcal{E}(I, I^H) \geq \tau_I; \\ \text{Quarter}, & \text{if } k \cdot \mathcal{E}(I, I^H) < \tau_I; \\ \text{Half}, & \text{otherwise.} \end{cases} \tag{16}$$

For 2D image tiles, we compute the detection filter in both horizontal and vertical directions, and determine the X and Y shading rate independently. As a default setting in our current implementation, we do not use of quarter rate in X and Y simultaneously (e.g. 4×4 shading regions size), because in most cases it provides little additional performance benefit.

4 MOTION ADAPTIVE SHADING

LCD persistence induced motion blur has the same effect as applying a directional box filter to the displayed image. The width of box filter is equal to the velocity in units of pixels/frame. Similarly, most in-engine motion blur effects, if present, are computed with a directional post-processing blurring filter. Both effects exhibit low-pass filtering properties, and can hide image errors from reduced shading rate. In this section we provide a frequency-space analysis of this effect, and derive a method to compute a motion-adjusted error estimate. As an option, we also show how to approximate motion adaptive shading without evaluating the error estimator.

4.1 Diminished Error Under Motion Blur

Following Eq.4 and Eq.7, we define a motion-compensated error term in the frequency domain as:

$$\begin{aligned} \mathcal{E}(I, I^H, v) &= \|\mathcal{F}(I) \cdot \mathcal{F}(B_v) - \mathcal{F}(I)\mathcal{F}(B_2) \cdot \mathcal{F}(B_v)\|_2 \\ &= \|\mathcal{F}(I) \cdot (\mathbb{1} - \mathcal{F}(B_2)) \cdot \mathcal{F}(B_v)\|_2 , \end{aligned} \tag{17}$$

where $\mathcal{F}(B_v)$ is the Fourier transform of the motion blur filter. $\mathcal{F}(B_v)$ depends solely on the velocity v. Fig. 4 shows a plot of the magnitude of $\mathcal{F}(B_v)$ at different velocities. Intuitively speaking, the faster the motion is, the more attenuation in high-frequency content is observed.

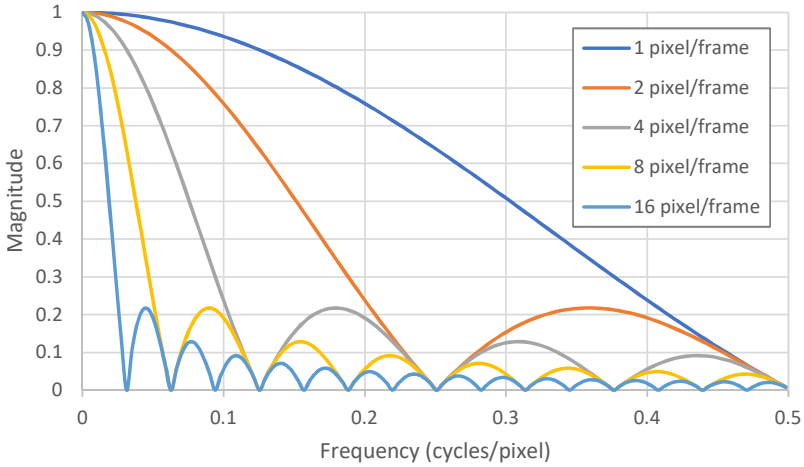


Fig. 4. Frequency response amplitude of the motion blur filter under different speeds.

Following a technique similar to that used in Section 3.3, we approximate Eq.17 by:

$$\begin{aligned}\mathcal{E}(I, I^H, v) &\approx b_H(v) \cdot \|\mathcal{F}(I) \cdot (\mathbb{1} - \mathcal{F}(B_2))\|_2 \\ &= b_H(v) \cdot \mathcal{E}(I, I^H).\end{aligned} \tag{18}$$

The function $b_H(v)$ scales down the perceived error of half-rate shading under velocity v. Its value is 1 when $v = 0$, reducing Eq.18 to Eq.7, and its value decreases as velocity grows. Similar to Eq.13, We compute the motion-influenced error under quarter-rate shading:

$$\begin{aligned}\mathcal{E}(I, I^Q, v) &= \|\mathcal{F}(I) \cdot (\mathbb{1} - \mathcal{F}(B_4)) \cdot \mathcal{F}(B_v)\|_2 \\ &= \|\mathcal{F}(I) \cdot (\mathbb{1} - \mathcal{F}(B_2)) \cdot \mathcal{K} \cdot \mathcal{F}(B_v)\|_2 \\ &\approx b_Q(v) \cdot \|\mathcal{F}(I) \cdot (\mathbb{1} - \mathcal{F}(B_2))\|_2 \\ &= b_Q(v) \cdot \mathcal{E}(I, I^H).\end{aligned} \tag{19}$$

Like Eq.14, we are using the same error estimator derived for half-rate shading. Therefore, the function $b_Q(v)$ maximizes at k when v approaches 0. We compute the values of $b_H(v)$ and $b_Q(v)$ numerically over a range of motion velocities, as plotted in Fig. 5.

For convenience of implementation, we fitted two closed-form functions to $b_H(v)$ and $b_Q(v)$:

$$\tilde{b}_H(v) = \left(\frac{1}{1 + (1.05v)^{3.10}}\right)^{0.35}, \tag{20}$$

$$\tilde{b}_Q(v) = 2.13\left(\frac{1}{1 + (0.55v)^{2.41}}\right)^{0.49}. \tag{21}$$

As shown in Fig. 5, they match closely with the numerical solutions and can be efficiently evaluated at run time.

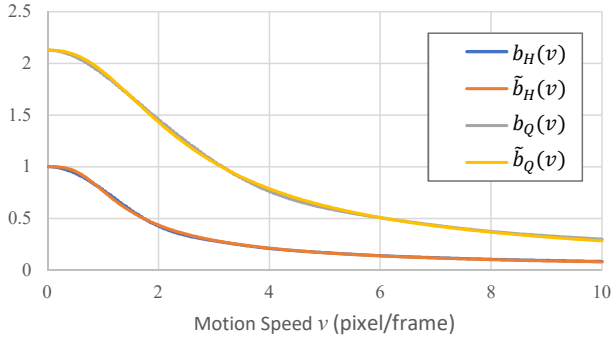


Fig. 5. Motion-influenced shading error scaling function under half-rate (H) and quarter-rate (Q) shading. $\tilde{b}_H(v)$ and $\tilde{b}_Q(v)$ are closed-form fitted functions of numerically calculated curves $b_H(v)$ and $b_Q(v)$.

Replacing the error terms in Eq.16 by the motion influenced ones, we then determine the shading rate using

$$S_I = \begin{cases} \text{Full}, & \text{if } \tilde{b}_H(v) \cdot \mathcal{E}(I, I^H) \geq \tau_I; \\ \text{Quarter}, & \text{if } \tilde{b}_Q(v) \cdot \mathcal{E}(I, I^H) < \tau_I; \\ \text{Half}, & \text{otherwise}. \end{cases} \tag{22}$$

The above analysis assumes a uniform velocity v in the entire image tile I. In practice this is rarely the case, as a tile can cross an object boundary with different velocities between foreground and background. As a conservative choice, we always take the minimum velocity of all pixels in the tile as v. This guarantees that we never lower the shading rate on even a tiny object that does not carry enough motion to hide the error.

4.2 Standalone Motion Adaptive Shading

We have shown how motion influence can augment content adaptive shading. In some applications, it is desirable to have motion adaptive shading implemented as a standalone optimization, without computing the error estimate. In such cases, we need to further simplify the rate decision criteria.

Considering Eq.15 and 22, the stopping threshold for full-rate shading can be written as:

$$b_H(v) < t \cdot \frac{I_{\text{avg}} + l}{\mathcal{E}(I, I^H)}. \tag{23}$$

The reciprocal of the fractional term on the right hand side has a psychophysical interpretation. $\mathcal{E}(I, I^H)$ is the average (L_2 mean) level of pixel error under half-rate shading. The quotient $\mathcal{E}(I, I^H)/(I_{\text{avg}} + l)$ normalizes that error against the background intensity, and gives us a relative error. This error is typically bounded in a given scene or game level, given the complexity of texture and material properties. Suppose the maximum relative error is e, Eq.23 is then conservatively simplified as:

$$b_H(v) < \frac{t}{e}. \tag{24}$$

Since the right-hand side is a run-time constant, the decision of enabling half-rate shading only depends on the velocity. For example, if sensitivity $t = 0.03$, and maximum relative error $e = 0.1$, then based on the $b_H(v)$ curve in Fig. 5, velocities exceeding 2.8 pixel/frame can enable half-rate shading without causing noticeable difference. The condition for enabling quarter-rate shading can be computed in a similar way. It should be noted that using a constant e to approximate relative

error can miss certain optimization opportunities. In other circumstances, a less than conservative estimate has to be used to reap enough performance benefit. Therefore, it should only be used when content adaptation is not an option.

5 IMPLEMENTATION

We describe a typical implementation of content and motion adaptive shading in modern game and rendering engines. The main algorithm can be implemented in one or a few compute passes that run at the start of the frame (right after a depth pre-pass, if it exists). It computes a shading-rate texture that is later used by the main scene rasterization and shading passes in a frame.

As the proposed techniques are targeting performance optimizations, the first principle of our design is "do no harm". This means that the performance overhead of running added compute passes should be negligible, even when running on high resolution targets like 4K UHD. In other words, it should not slow down the cases where there is little opportunity for optimization. We present a careful design that achieves this goal.

5.1 Content Adaptation

To analyze per-tile content variation, we bind the final image from the previous frame as a texture to the shading rate compute pass. Since there is typically scene motion between successive frames, we reproject the pixels from the previous frame to the current frame using texture fetches offset by per-pixel motion vectors [Nehab et al. 2007]. The details of obtaining the motion vectors are described in Sec. 5.2. Alternatively, we can forward reproject the pixels from the previous frame using iterative image warping [Bowles et al. 2012; Yang et al. 2011], which only requires motion vectors from the previous frame instead of the current frame. Objects that cannot be reprojected correctly (e.g. disoccluded regions) can be identified by depth-based rejection [Nehab et al. 2007], and are marked to be shaded in full rate.

The luma component of the fetched linear RGB colors can be computed using standard YUV transform coefficients:

$$I = 0.299R + 0.587G + 0.114B. \tag{25}$$

While in theory it is more accurate to compute color differencing in RGB or a transformed perceptual color space, we found that using luma for content adaptation suffices and offers a performance benefit. For each 16×16 pixel tile, we compute both the horizontal and vertical luma differencing term (Eq.2) and the average (background) luma I_{avg} in the same compute shader.

To achieve the best efficiency, we launch a compute grid that matches the size of the shading-rate texture. Each thread group contains 32 threads, with each thread computing reprojection, error estimator differencing kernel and average luma for 8 pixels. Therefore, each thread group covers exactly one 16×16 pixel tile, which maps to one texel in the output shading rate texture. For adding the differencing and average luma results over the tile, each thread first computes the sum of kernel outputs from 8 pixels that it owns. All 32 threads then issue an atomic add, which can be translated to inter-warp communication instructions to achieve better efficiency [Luitjens 2016], since the group size is exactly the warp (wavefront) size of the architecture. Our optimized compute kernel runs within 0.2-0.3ms in our test setting (NVIDIA RTX 2080 GPU, rendering at 3840×2160 screen resolution).

5.2 Motion Adaptation

In order to obtain the minimum motion vector in a tile, we need to access the per-pixel motion vector, which is the screen-space coordinate difference between the current and previous locations of a surface point. Game engines typically compute this vector per-pixel for temporal antialiasing

and motion blur effects. If this is available as a texture after the depth pre-pass, we can directly bind it to the shading rate compute pass. Otherwise, for each pixel, we compute its location in previous frame by first reconstructing its clip-space coordinate using screen coordinate and depth, and then back-projecting it to the previous frame using the camera view-projection matrices from the previous and the current frame. Note that this computation only considers camera motion, which typically applies to the majority of the screen. Static objects such as hand weapons in a first-person-shooter game should be specially masked or faded out while computing the motion influence.

The per-pixel motion vectors are reduced to a minimum motion vector per-tile using similar method as in Sec. 5.1 for luminance counterparts. Then the leading thread in the group computes the motion-based error scalers $\tilde{b}_H(v)$ and $\tilde{b}_Q(v)$ (Eq.20, 21) for both horizontal and vertical directions.

5.3 Applying Adjusted Shading Rate

Once the per-tile values of $\mathcal{E}(I, I^H)$, $\tilde{b}_H(v)$, $\tilde{b}_Q(v)$ and I_{avg} are calculated, the leading thread of each thread group computes the shading rate in both horizontal and vertical directions using Eq.15 and 16. The computed horizontal and vertical shading rate of the tile is converted into one of the defined shading rate patterns (Sec. 2.1), and the result is saved into the shading rate texture. After that, and before launching the shading passes, we bind the shading rate texture to the pipeline and enable VRS. There is no need to change the shading program or any other pipeline state, as VRS is transparently enabled by the hardware. This greatly simplifies the engine integration effort.

5.4 Optional Quality Improvements

One undesirable effect sometimes observed is when the predicted shading error (e.g. $\tilde{b}_H(v) \cdot \mathcal{E}(I, I^H)$ for half rate shading) of a tile is around the threshold of τ_I, causing the shading rate to oscillate in successive frames. Another possible cause is the feedback loop that adaptive shading introduces, which happens when frame n shaded in lower rate causes shading error to increase, such that frame $n+1$ is reverted back to higher rate, drops the error and forms a cycle. This may cause temporal instability that becomes visible to the viewer, especially when an aggressive JND threshold is used, or when only motion adaptive shading is used. To mitigate this problem, we can optionally apply two thresholds $t_+ = t \cdot (1+\epsilon)$ and $t_- = t \cdot (1-\epsilon)$, used when the predicted error increase or decrease from the previous frame, respectively. This introduces hysteresis to shading rate transition to favor a more stationary state.

It is not uncommon to see two adjacent tiles have shading rates differ by more than a factor of 2. This in some cases can cause visible discontinuity in shading appearance and may exacerbate the feedback loop problem. In some implementations, we choose to limit the rate difference in adjacent tiles by 2×, by using a separate compute pass to filter the shading rate buffer. Boundary shading rates are increased from quarter to half resolution to smooth out the transition. The running time of this pass is negligible, since it computes on the shading rate texture, which is $1/16 \times 1/16$ screen resolution.

The error estimate terms are reasonably accurate in most cases. One exception is when there is heavy aliasing in shading, where unfiltered pixels can sometimes cause the differencing filter to underestimate the error. This is usually seen on rough surfaces, where thin and bright specular highlights can appear blocky due to misdetection of high-frequency details in coarse rate. We currently use surface material properties retrieved from the G-buffer of the previous frame to selectively control the adaptation influence in shading rate. Shiny metallic surfaces get reduced adaptation in shading rates, since they are more likely to appear blocky when specular energy dominates.

Fig. 6. Image quality comparison with adaptive shading on versus off (*Wolfenstein II* Roswell scene).

Fig. 7. Image quality comparison with adaptive shading on versus off (*Wolfenstein II* Submarine scene).

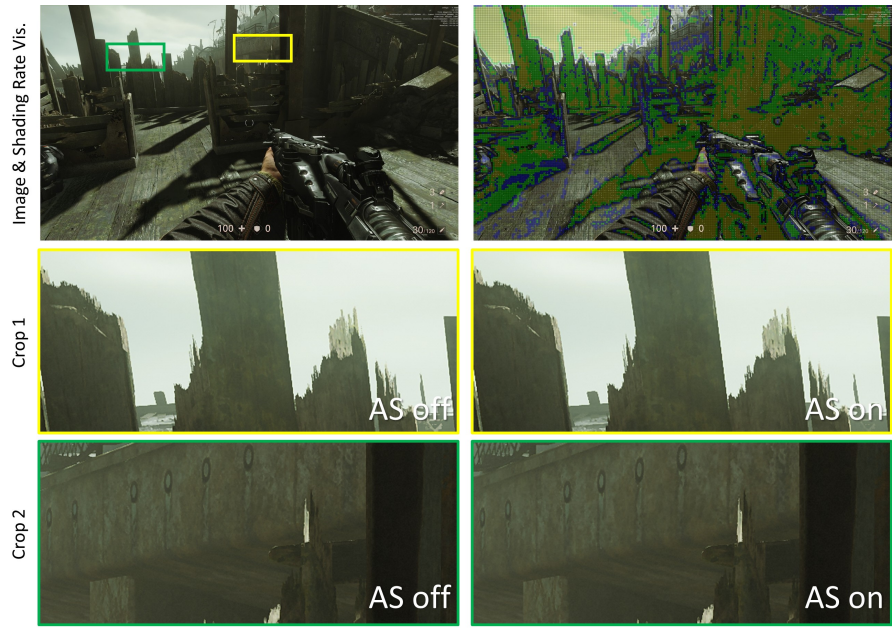


Fig. 8. Image quality comparison with adaptive shading on versus off (*Wolfenstein II* Manhattan scene).

6 RESULTS

We implemented content and motion adaptive shading in two high-end game engines: id Tech 6 and Unreal Engine 4.19, using Vulkan and DirectX 11 (NvAPI) access to the NVIDIA Variable-Rate Shading (VRS) feature on the latest Turing architecture. In November 2018, the content and motion adaptive shading feature was released by MachineGames as a patch to the highly successful id Tech 6 based game *Wolfenstein II: The New Colossus*, and the feature received critical acclaim from major gaming technology media. On Unreal Engine 4 we tested Epic Game's high-end technology demo *Infiltrator* using the engine's forward renderer.

Image Quality. As stated earlier, the design goal of content and motion adaptive shading is to provide performance benefit with no loss of perceived image quality. We have carefully tested and improved the technique to ensure visual equivalence in different types of scenes, materials and lighting effects. Fig. 1, 6, 7 and 8 show a few examples of quality comparison, captured from different levels in *Wolfenstein II*. These include both outdoor scenes and indoor scenes. Generally speaking, regardless of scene types and lighting conditions, regions with no or only low-contrast high-frequency details receive the most significant shading rate reduction.

For motion adaptive shading, the quality is compared with an in-engine motion blur effect enabled. When the motion blur filter is chosen to match the per-frame pixel motion footprint, the result can also be seen as a simulation of perceived LCD persistence blur. Fig. 9 shows such a comparison using images captured from *Infiltrator*. The accompanying video also shows the shading rate pattern change between static and motion scenes.

The main parameter in our method is the error sensitivity threshold t, which can be adjusted to achieve a desired tradeoff between quality and performance. In Fig. 11 we show an example of how the threshold t affects image quality. Note that due to limited engine code access, we generated the images in this figure by post processing a screenshot with emulated VRS. The result may differ

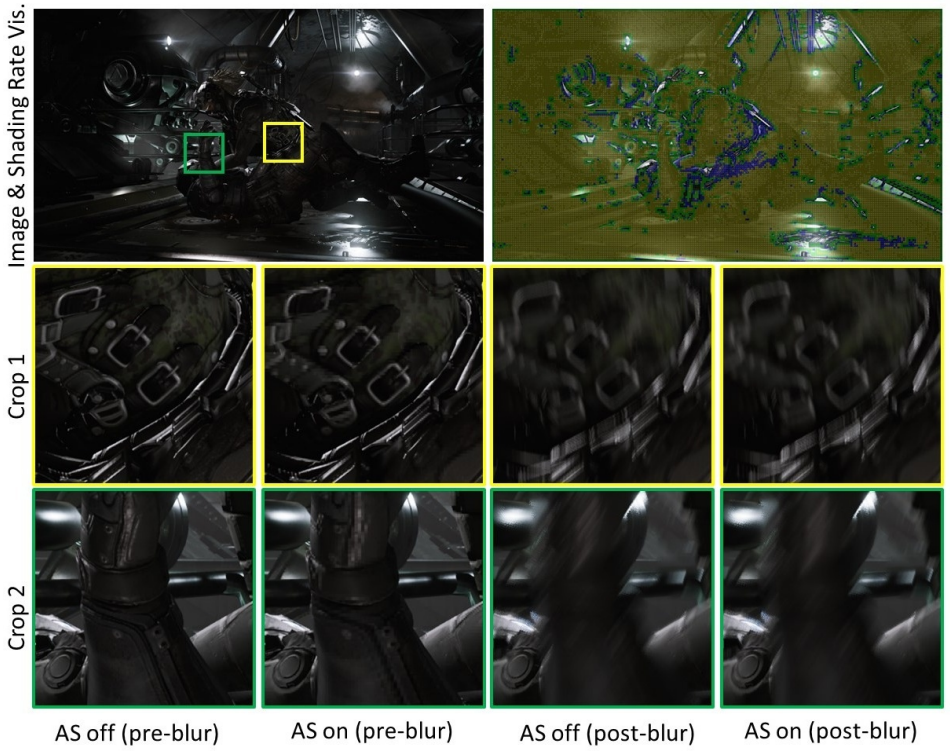


Fig. 9. Rendering *Infiltrator* with content and motion adaptive shading under motion blur. Motion allows most tiles to render with 8× reduced shading rate, with the exception of areas in extreme high contrast (crop 1), where the error could be visible after motion blur, due to leaked high frequency from the blurring filter. All results shaded in lower rate look identical to the native rate shading after motion blur.

slightly from the real ones, where geometry edges can avoid being undersampled, i.e., the effect shown in Fig. 2. We also plot the overall effect of t on the average shading rate and image quality in Fig. 10. The other parameter is the environment luminance l, which can be adjusted to adapt to ambient lighting and monitor black level. In the rest of this paper we use $t = 0.15$ and $l = 0.05$ in the results. This works well with our test environment (27 inch 4K display, 25 inch viewing distance, leading to 76.2 PPD or pixels-per-degree pixel density). With lower PPD settings, e.g. larger display, lower spatial resolution, closer viewing distance, or on VR HMDs, the threshold t may need to be reduced due to higher contrast sensitivity of lower frequency signals in the human visual system.

Performance. Adaptive shading is currently enabled on opaque, non-alpha tested forward shading passes in both engines. The performance gain of the technique is highly dependent on these three factors:

- The percentage of frame time taken by the VRS-enabled draw calls;
- The average primitive-to-pixel size ratio (over-tessellated pixel-sized primitives cannot benefit from VRS);
- The complexity of the pixel shader (draw calls with expensive shading benefit more from VRS).

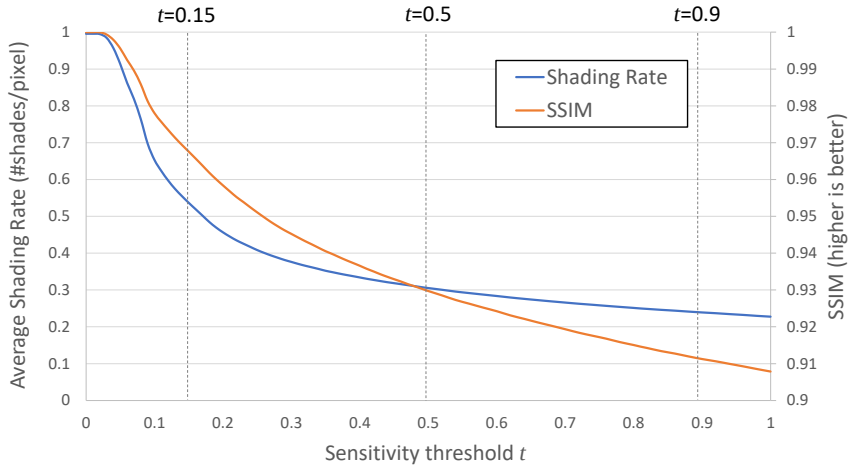


Fig. 10. The overall effect on average shading rate and image quality (SSIM) when sensitivity threshold t is adjusted. The curves are measured on the image shown in Fig. 11, with the three thresholds used marked in dashed lines.

Currently, in both titles, adaptive shading benefit is primarily bounded by the first item. VRS-enabled draw calls take about 15% – 40% of frame time, depending on the scene, level, and camera angle. Therefore the performance gain is scaled by that percentage. Overall we observe 20% – 60% average draw call time reduction, and 5% – 20% average frame time reduction in various levels and scenes in both engines, as detailed in Table 1. We consider these very encouraging performance gains. Given that both engines are already highly optimized, getting additional 1–3ms of free cycles out of a packed 16.7ms frame time budget (60FPS target) is valuable for improving or adding other effects. The reduction in frame time can also help greatly reduce stuttering caused by slow frames missing refresh on a fixed refresh-rate display.

	VRS passes			Whole frame		
Scene	off (ms)	on (ms)	↓ %	off (ms)	on (ms)	↓ %
Manhattan	3.1	1.6	48	10.5	8.6	18
Roswell	2.7	1.7	37	9.1	8.1	11
Submarine	2.3	1.5	35	9.0	7.9	12
Hallway	5.9	2.6	56	14.4	12.5	13
Engine	4.8	2.8	42	12.3	10.5	15

Table 1. Performance comparison in different scenes with content adaptive shading on versus off (stationary camera). The first three scenes are from different levels in *Wolfenstein II*, and the rest are from *Infiltrator*. All performance numbers in this paper are measured on an NVIDIA RTX 2080 GPU rendering at 3840 × 2160 full-screen resolution. Note that the time saved in VRS passes and in whole frame time do not necessarily match, since VRS may change the workload overlap pattern between the graphics and asynchronous compute channels.

Typically, higher performance gains are observed with scene or camera motion when motion adaptive shading is enabled. Table 2 shows the performance gain when motion is present. One benefit specific of motion adaptive shading is to compensate for the overhead of in-engine motion blur effects when heavy motion is present. The cost of motion blur is typically associated with

Scene	Stationary			Motion		
	off (ms)	on (ms)	↓ %	off (ms)	on (ms)	↓ %
Submarine	9.0	7.9	12	11.0	8.6	22
Engine	12.3	10.5	15	12.6	10.0	21

Table 2. Performance comparison with and without motion in the scene.

the blur kernel size. Traditionally, a fast camera motion can cause prolonged motion blur pass computation, leading to reduction in frame rate and stuttering. Motion adaptive shading can allow a significant shading rate reduction when heavy motion is present, saving shading time for use in motion blur.

7 CONCLUSION AND FUTURE WORK

We presented content and motion adaptive shading techniques for optimizing the performance of modern game and rendering engines. The proposed technique makes use of the variable-rate shading feature available on latest-generation GPUs, and achieves steady performance improvement with no perceived image quality loss. We demonstrated the techniques on two examples of high-end game contents, and we expect the technique to be widely adopted.

For future work, we consider applying adaptive shading to more types of shading and post-processing passes, including ray-tracing passes, full-screen post-processing passes, compute shader passes, and particle shading passes. There is no restriction in our theory to use adaptive shading on non-rasterization passes. Widening its applications will likely involve more engine integration effort, but will allow us to unlock the acceleration of the remaining workloads within a frame.

ACKNOWLEDGMENTS

We thank our colleagues Henry Moreton and Dale Kirkland for their kind help with the manuscript preparation, and the anonymous reviewers for their insightful and constructive feedback. We are also thankful to MachineGames for their help with integrating, testing and publishing the *Wolfenstein II* adaptive shading patch. The *Infiltrator* demo and Unreal Engine 4 are courtesy of Epic Games Inc.

REFERENCES

Kurt Akeley. 1993. Reality engine graphics. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*. ACM, 109–116.

Tomas Akenine-Moller, Eric Haines, and Naty Hoffman. 2018. *Real-time rendering*. AK Peters/CRC Press.

Magnus Andersson, Jon Hasselgren, Robert Toth, and Tomas Akenine-Möiler. 2014. Adaptive texture space shading for stochastic rendering. *Computer Graphics Forum* 33, 2 (2014), 341–350.

Swaroop Bhonde. 2018. Turing Variable Rate Shading in VRWorks. https://devblogs.nvidia.com/turing-variable-rate-shading-vrworks/.

Mark R Bolin and Gary W Meyer. 1998. A perceptually based adaptive sampling algorithm. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*. ACM, 299–309.

Huw Bowles, Kenny Mitchell, Robert W Sumner, Jeremy Moore, and Markus Gross. 2012. Iterative image warping. In *Computer graphics forum*, Vol. 31. Wiley Online Library, 237–246.

Christopher A Burns, Kayvon Fatahalian, and William R Mark. 2010. A lazy object-space shading architecture with decoupled sampling. In *Proceedings of the Conference on High Performance Graphics*. Eurographics Association, 19–28.

George Casella and Roger L Berger. 2002. *Statistical inference*. Vol. 2. Duxbury Pacific Grove, CA.

Stanley H Chan and Truong Q Nguyen. 2011. LCD motion blur: modeling, analysis, and algorithm. *IEEE Transactions on Image Processing* 20, 8 (2011), 2352–2365.

Petrik Clarberg, Robert Toth, Jon Hasselgren, Jim Nilsson, and Tomas Akenine-Möller. 2014. AMFS: adaptive multi-frequency shading for future graphics processors. *ACM Transactions on Graphics (TOG)* 33, 4 (2014), 141.

Petrik Clarberg, Robert Toth, and Jacob Munkberg. 2013. A sort-based deferred shading architecture for decoupled sampling. *ACM Transactions on Graphics (TOG)* 32, 4 (2013), 141.

Jean-Philippe Farrugia and Bernard Péroche. 2004. A progressive rendering algorithm using an adaptive perceptually based image metric. In *Computer Graphics Forum*, Vol. 23. Wiley Online Library, 605–614.

David J Field. 1987. Relations between the statistics of natural images and the response properties of cortical cells. *Josa a* 4, 12 (1987), 2379–2394.

Yong He, Yan Gu, and Kayvon Fatahalian. 2014. Extending the graphics pipeline with adaptive, multi-rate shading. *ACM Transactions on Graphics (TOG)* 33, 4 (2014), 142.

Karl E Hillesland and JC Yang. 2016. Texel shading. In *Proceedings of the 37th Annual Conference of the European Association for Computer Graphics: Short Papers*. Eurographics Association, 73–76.

Gábor Liktor and Carsten Dachsbacher. 2012. Decoupled deferred shading for hardware rasterization. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. ACM, 143–150.

Justin Luitjens. 2016. Faster Parallel Reductions on Kepler. https://devblogs.nvidia.com/faster-parallel-reductions-kepler.

Don P Mitchell. 1987. Generating antialiased images at low sampling densities. In *ACM SIGGRAPH Computer Graphics*, Vol. 21. ACM, 65–72.

Diego Nehab, Pedro V. Sander, Jason D. Lawrence, Natalya Tatarchuk, and John R. Isidoro. 2007. Accelerating Real-Time Shading with Reverse Reprojection Caching. In *ACM SIGGRAPH/Eurographics Symposium on Graphics Hardware*. 25–35.

NVIDIA. 2015. VRWorks - Multi-Res Shading. https://developer.nvidia.com/vrworks/graphics/multiresshading.

NVIDIA. 2016. VRWorks - Lens-Matched Shading. https://developer.nvidia.com/vrworks/graphics/lensmatchedshading.

NVIDIA. 2018. NVIDIA Turing GPU Architecture Whitepaper. https://www.nvidia.com/content/dam/en-zz/Solutions/design-visualization/technologies/turing-architecture/NVIDIA-Turing-Architecture-Whitepaper.pdf.

Alan V. Oppenheim and Ronald W. Schafer. 2009. *Discrete-Time Signal Processing* (3rd ed.). Prentice Hall Press, Upper Saddle River, NJ, USA.

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 Transactions on Graphics (TOG)* 35, 6 (2016), 179.

Olivier Penacchio and Arnold J Wilkins. 2015. Visual discomfort and the spatial distribution of Fourier energy. *Vision research* 108 (2015), 1–7.

Jonathan Ragan-Kelley, Jaakko Lehtinen, Jiawen Chen, Michael Doggett, and Frédo Durand. 2011. Decoupled sampling for graphics pipelines. *ACM Transactions on Graphics (TOG)* 30, 3 (2011), 17.

Jeremy Shopf. 2009. Mixed resolution rendering. In *Game Developers Conference*.

Michael Stengel, Steve Grogorick, Martin Eisemann, and Marcus Magnor. 2016. Adaptive Image-Space Sampling for Gaze-Contingent Real-time Rendering. In *Computer Graphics Forum*, Vol. 35. Wiley Online Library, 129–139.

Ee-Leng Tan and Woon-Seng Gan. 2015. Computational Models for Just-Noticeable Differences. In *Perceptual Image Coding with Discrete Cosine Transform*. Springer, 3–19.

TestUFO.com. 2017. Blur Busters UFO Motion Tests. https://www.testufo.com/.

Robert Toth, Jim Nilsson, and Tomas Akenine-Möller. 2016. Comparison of projection methods for rendering virtual reality. In *Proceedings of High Performance Graphics*. Eurographics Association, 163–171.

Karthik Vaidyanathan, Marco Salvi, Robert Toth, Tim Foley, Tomas Akenine-Möller, Jim Nilsson, Jacob Munkberg, Jon Hasselgren, Masamichi Sugihara, Petrik Clarberg, et al. 2014. Coarse pixel shading. In *Proceedings of High Performance Graphics*. Eurographics Association, 9–18.

Karthik Vaidyanathan, Robert Toth, Marco Salvi, Solomon Boulos, and Aaron Lefohn. 2012. Adaptive image space shading for motion and defocus blur. In *Proceedings of the Fourth ACM SIGGRAPH/Eurographics conference on High-Performance Graphics*. Eurographics Association, 13–21.

Zhou Wang, Alan C Bovik, Hamid R Sheikh, Eero P Simoncelli, et al. 2004. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing* 13, 4 (2004), 600–612.

Andrew B Watson, Albert J Ahumada, and Joyce E Farrell. 1986. Window of visibility: a psychophysical theory of fidelity in time-sampled visual motion displays. *JOSA A* 3, 3 (1986), 300–307.

Kai Xiao, Gabor Liktor, and Karthik Vaidyanathan. 2018. Coarse pixel shading with temporal supersampling. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. ACM, 1.

Lei Yang, Pedro V. Sander, and Jason Lawrence. 2008. Geometry-Aware Framebuffer Level of Detail. *Computer Graphics Forum (Proc. of Eurographics Symposium on Rendering 2008)* 27, 4 (2008), 1183–1188.

Lei Yang, Yu-Chiu Tse, Pedro V Sander, Jason Lawrence, Diego Nehab, Hugues Hoppe, and Clara L Wilkins. 2011. Image-based bidirectional scene reprojection. In *ACM Transactions on Graphics (TOG)*, Vol. 30. ACM, 150.

Matthias Zwicker, Wojciech Jarosz, Jaakko Lehtinen, Bochang Moon, Ravi Ramamoorthi, Fabrice Rousselle, Pradeep Sen, Cyril Soler, and S-E Yoon. 2015. Recent advances in adaptive sampling and reconstruction for Monte Carlo rendering. In *Computer Graphics Forum*, Vol. 34. Wiley Online Library, 667–681.

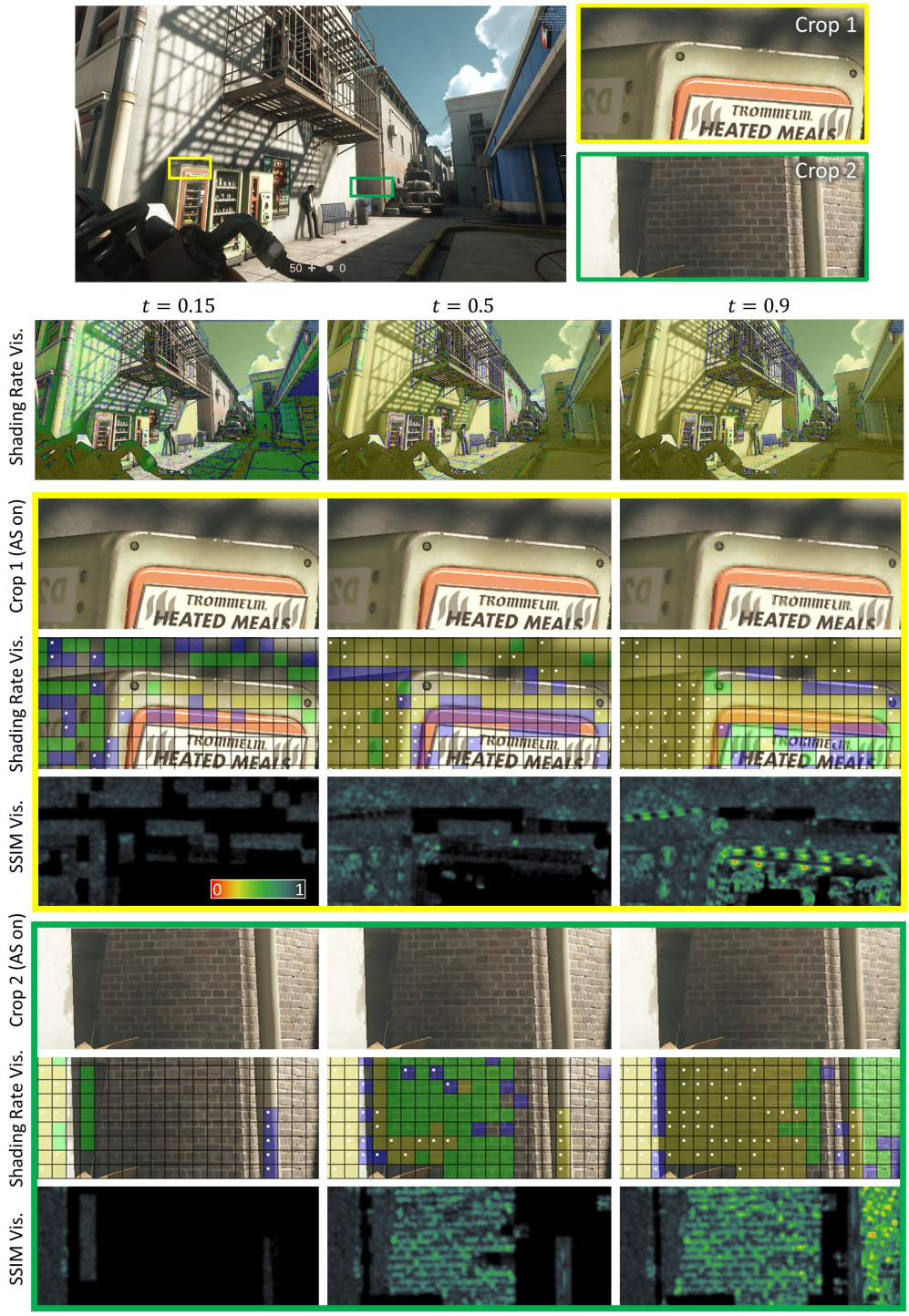


Fig. 11. Image quality comparison under different sensitivity threshold t. Image quality is measured and visualized using the SSIM index [Wang et al. 2004].