FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Improving Real-Time Rendering Quality and Efficiency using Variable Rate Shading on Modern Hardware

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Visual Computing

eingereicht von

Stefan Stappen, BSc.
Matrikelnummer 1329020

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Associate Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer
Mitwirkung: Univ.Ass. Dipl.-Ing. Johannes Unterguggenberger

Wien, 11. November 2019

Stefan Stappen	Michael Wimmer

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Improving Real-Time Rendering Quality and Efficiency using Variable Rate Shading on Modern Hardware

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Visual Computing

by

Stefan Stappen, BSc.
Registration Number 1329020

to the Faculty of Informatics

at the TU Wien

Advisor: Associate Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer
Assistance: Univ.Ass. Dipl.-Ing. Johannes Unterguggenberger

Vienna, 11th November, 2019

Stefan Stappen | Michael Wimmer

Erklärung zur Verfassung der Arbeit

Stefan Stappen, BSc.

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 11. November 2019

Stefan Stappen

Danksagung

Besonders bedanken möchte ich mich bei Johannes Unterguggenberger, für das durchgängig konstruktive und motivierende Feedback während der Entwicklung der neuen Algorithmen, für die gemeinsame Ideenfindung und die Hilfe bei deren Umsetzung. Außerdem gebührt meinem Betreuer Michael Wimmer großer Dank für das effiziente und akademisch wertvolle Feedback.

Außerdem bedanke ich mich beim Dekanat Informatik für das Förderungsstipendium, welches einen Großteil der nötigen Hardware für diese Diplomarbeit finanziert hat.

Ebenso gebührt meiner Mutter besonderer Dank für ihre Geduld während dieser anstrengenden Phase meines Studiums. Ihrem Verständnis ist es zu verdanken, dass ich meinen Fokus durchwegs auf diese Arbeit richten konnte. Ich möchte mich auch noch bei meinen Kollegen aus der Firma, Vereinen und Freunden bedanken, die mich immer wieder mit ihren Unterbrechungen aus meinem Tunnelblick herausgerissen haben und mir somit die Möglichkeit gaben, neue Perspektiven auf die neuen Herausforderungen der Arbeit zu gewinnen. Auch bei meiner Firma, und besonders meinem Vorgesetzten, bedanke ich mich, dass sie mir mit den flexiblen Arbeitszeiten und Homeoffice die Absolvierung meines Studiums ohne unnötige Aufschübe ermöglichten.

Acknowledgements

Special thanks goes to Johannes Unterguggenberger for the continuously constructive and motivating feedback during the development of the new algorithms, new ideas and implementation of this thesis. In addition, I want to thank my supervisor Michael Wimmer for the efficient and academically valuable feedback.

For the financing of a big part of the necessary hardware, I want to thank the Office of the Dean of TU Wien Informatics, which granted me a funding scholarship.

I thank my mother for her patience during this exhausting phase of my studies. Only due to her efforts I could fully concentrate on my studies. I want to thank my colleges in my company, clubs, and friends, for their continuous interruptions that helped me out of routine-blinded phases. These interruptions allowed me to regain new perspectives and create new approaches. Additionally, I have to thank my company and especially my superiors for the flexible working hours and the allowance of home office, which enabled the possibility to complete my studies without unnecessary deferral.

Kurzfassung

Mit der neuen Grafikkartenmikroarchitektur NVIDIA Turing wurde nicht nur die Leistung erhöht, sondern es wurden auch neue Hardwarefunktionen eingeführt, wie Variable Rate Shading (VRS). VRS ermöglicht die Fokussierung der Rechenleistung durch dynamische Anpassung der Shading-Auflösung pro 16x16 Pixel-Bereich des Framebuffers. Die Shading-Auflösung entspricht der Anzahl der Pixel, die schattiert werden, wobei in Bereichen mit geringem visuellen Detailgrad weniger Pixel und in Bereichen mit höherem Detailgrad mehr Pixel schattiert werden können. Es gibt diverse Strategien, um den nötigen Detailgrad pro Bereich festzustellen. Darunter fallen die Techniken "Content-Adaptive Shadin" und "Motion-Adaptive Shading", die von NVIDIA vorgeschlagen wurden. Content-Adaptive Shading variiert die Shading-Rate auf Basis des Inhalts des vorhergehenden Frames. Motion-Adaptive Shading passt die Shading-Rate an die Bewegungen von Kamera und Objekten aufeinanderfolgender Frames an. Es gibt unterschiedliche Ansätze, welche Parameter zur Bestimmung der Shading-Rate herangezogen werden. Ein weiterer Ansatz zur Bestimmung der Shading-Rate ist die Anpassung der Auflösung an den Fokus des Betrachters. Das Zentrum des Fokus kann dabei höher aufgelöst werden, während Bereiche im Umfeld niedriger aufgelöst werden können. Dies kann in Kombination mit einem Eye-Tracking Gerät erfolgen und wird "Foveated Rendering"genannt. Neben diesen Vorgehensweisen zur Bestimmung der Shading-Rate beschreibt diese Diplomarbeit eine weitere, neue Strategie, nämlich „TAA-Adaptive Shading". Diese verwendet Daten von Temporal Anti-Aliasing Techniken zur Erkennung von Regionen, die zu niedrig oder zu hoch abgetastet wurden und wählt auf Basis dieser die Shading-Rate aus. Fünf Algorithmen wurden von uns entwickelt: Content-Adaptive Shading basierend auf Kanten und auf Texeldifferentialen, Motion-Adaptive Shading unter Berücksichtigung der Bewegungen aufeinanderfolgender Frames, Single-Pass Foveated Rendering und TAA-Adaptive Shading. Die Anwendbarkeit jedes Algorithmus auf moderne Renderer-Architekturen mit Forward Shading, Deferred Shading sowie Anti-Aliasing Post-Processing wurde evaluiert. Einige der VRS-Techniken, die in dieser Diplomarbeit beschrieben werden, sind in der Lage, eine Szene in vierfacher Auflösung bei gleichbleibender Performance oder mit vierfacher Performance bei gleicher Auflösung zu rendern.

Abstract

With the NVIDIA Turing graphics card micro-architecture released in 2018, not only performance in terms of operations per second is increased but also new hardware features are introduced, like Variable Rate Shading (VRS). VRS allows focussing the processing power by dividing the framebuffer into tiles and dynamically controlling the resolution of each tile. To be precise, the screen is partitioned into tiles of 16x16 pixels and for each tile, it can be specified how often the fragment shader shall be executed. It is both possible, to have fewer fragment shader invocations than there are fragments, or more fragment shader invocations than there are fragments. This allows individually defining lower sampling rates or supersampling for regions of the screen. Regions of less interest or with less visual details can be assigned less computational power in terms of shader executions while regions that should provide high fidelity can be supersampled. The challenges here are to find and distinguish these regions in a dynamic scene, like it is the case for games, and how this technique integrates with commonly used techniques in the industry, like deferred shading. NVIDIA already proposed some strategies on how these regions can be distinguished and how the shading rate can be selected. Among these strategies are Content-Adaptive Shading and Motion-Adaptive Shading. Content-Adaptive Shading varies the shading rate according to the current content of a frame and does not take temporal coherence into account. Motion-Adaptive Shading adapts the shading rate according to the changes in the scene. Stable regions, like for example the horizon and the car in a driving simulation, will be rendered with higher quality. In contrast, moving regions like the street will be rendered more coarsely because the viewer cannot focus on these regions anyway. Another approach for selecting the shading rate is to adapt the resolution to the viewer's focus. This can be done in combination with an eye-tracking device and is called foveated rendering. We invented a novel approach that utilizes data from temporal anti-aliasing techniques to detect under- and oversampled regions and select the appropriate shading rate for these regions. We developed five algorithms, edge-based and texel-differential based Content-Adaptive Shading, Motion-Adaptive Shading integrating the motion over multiple frames, single-pass foveated rendering and TAA-Adaptive Shading. The applicability of each algorithm to modern renderer architectures with forward and deferred shading and anti-aliasing post-processing has been evaluated. The major advantage of our VRS techniques is that some of them enable up to 4x higher rendering resolution with the same performance or up to 4x better performance at the same resolution.

Contents

CHAPTER 1

Introduction

Real-time 3D rendering applications strive to deliver visually appealing and often realistic-looking images while maintaining high frame rates of 60 Hz for desktop applications, 90 Hz for VR applications or even more frames per second for other high-responsive applications. Furthermore, with the increase in screen resolutions, higher quality images can be presented to the viewers, which comes at increased performance costs when rendering in their native resolutions. In order to meet these requirements of performance and quality, many different measures are taken. For example, one could focus on parts that are perceived stronger by a viewer and therefore neglect parts not directly visible to a viewer. NVIDIA's Turing Architecture [NVI18] provides us with a new tool putting us into the position to achieve exactly this, namely: Variable Rate Shading (VRS).

In computer graphics, triangles and other primitives are rasterized into so-called fragments by the rasterizer. Assuming that depth testing is enabled and blending is disabled in the graphics pipeline, the fragment which is closest to the camera will determine the final pixel color for the corresponding screen location. There are also techniques where multiple fragments are evaluated to determine the final pixel color, which is the fundamental idea of anti-aliasing techniques like supersampling, a well-researched anti-aliasing technique [AMHH18]. VRS, on the other hand, not only allows multiple fragments to determine the final color of one screen pixel, but it allows also the opposite: one fragment can determine the color of multiple screen pixels. This is called *Coarse Shading* while the former, where multiple fragments determine the color of one screen pixel, is called *Supersampling*. VRS, for example, enables us to compute twice as many fragments for a single screen pixel and average their results for the final pixel color value. Full viewport coarse shading can be achieved by simply rendering into a smaller framebuffer and up-scaling the image. In the same way, full viewport supersampling can be achieved by rendering into a higher resolution framebuffer and down-scale the image. VRS provides more fine-grained control over the type of sampling for individual parts of the framebuffer. This is achieved by dividing the image into a grid and specifying a shading rate for each cell of the grid. The

grid data is provided with an image, the shading-rate image, to the hardware. The so specified shading rate is then looked up in a table, the *shading rate palette*, to find the real shading rate, e.g., one invocation for 4x4 pixels, or 16x invocations per pixel.

With this new hardware feature, there are two fundamental possibilities, namely increasing the performance by coarse shading and increasing the quality by supersampling. We can improve performance by increasing the size of a multitude of fragments and perform coarse shading in suitable regions. This has a negative impact on the quality and may result in visible fragment edges if the color is not monotone across fragments, as it can be viewed in Figure-1.1. Depending on the scene, quality can be improved by increasing the invocations per fragment and therefore do supersampling. This has a negative impact on the performance as we do more work per pixel. Therefore, the main targets to be able to utilize this new hardware feature are to detect regions that would be beneficial to render with a better quality, regions that could be rendered with lower quality and how to mitigate the effects of rendering with lower quality.

This thesis addresses four main approaches to detect these regions, namely Content-Adaptive Shading, Motion-Adaptive Shading, foveated rendering and TAA-Adaptive Shading. The goal of Content-Adaptive Shading is to align the shading rate with the content of the scene captured in the current viewport. Depending on the scene, different properties could be used for selecting the right shading rate, and it can be a challenging task to select appropriate properties. Yang et al. [YZK+19], for example, uses luma. However, other properties are suitable for defining the shading rate, too. We have investigated two properties different from Yang et al. and present two different algorithms for Content-Adaptive Shading. Strategies for selecting the shading rate, especially Content-Adaptive Shading approaches, may introduce temporal artefacts into the rendered images by oscillating shading rates. Measures to reduce these artefacts and their root causes should be taken because these artefacts can be very noticeable to viewers. We propose an algorithm to counter these temporal artefacts.

Motion-Adaptive Shading utilizes the effect that motion is perceived blurred and aligns the shading rate with the motion of the scene, e.g., moving parts are rendered with lower resolution while stable parts are rendered with higher resolution. Foveated rendering exploits the physical nature of human perception by increasing the resolution within the focus of the viewer and decreasing the resolution in the peripheral vision, as described by Guenter et al. [GFD+12].

In real-time 3D rendering applications, anti-aliasing techniques are used to reduce aliasing artefacts and render images of higher visual quality. A special sub-category is represented by temporal anti-aliasing (TAA) techniques. Temporal anti-aliasing techniques use the information of previous frames to perform anti-aliasing. The data from the history frames used by TAA can also be useful for determining the shading rate. Based on this finding, we have developed a novel approach for determining the shading rate, TAA-Adaptive Shading. TAA-Adaptive Shading utilizes data from temporal anti-aliasing techniques to detect under- and oversampled regions and selects the appropriate shading rate for these regions.

These approaches lead to the following contributions of this thesis:

- We propose two novel algorithms for Content-Adaptive Shading operating on generic scenes: By deriving and using properties of the contents of the previous frame, namely edges in our first algorithm and texel differentials in our second algorithm, the shading rate is defined for the next frame using forward projection. Texel differential-based Content-Adaptive Shading utilizes the effective shading rate to be less impacted by changing shading rates.
- We describe a novel approach for controlling the shading rate based on data derived during TAA: TAA reduces the effects of wrong or missing temporal information, e.g., ghosting, by clipping color values of historic frames which deviate heavily from the current color value, for a specific fragment. High differences of historic color values indicate undersampled regions and could, therefore, benefit from a higher shading rate. If the differences are low, the historic information can be assumed to have a high probability of being correct and the shading rate can be lowered.
- We propose a technique, which is independent of the shading rate selection mechanism, that stabilizes the shading rates over time to mitigate temporal artefacts introduced by changes in the shading rate. Our technique can be described as a variant of temporal anti-aliasing that is specifically tailored to be applied to the shading-rate image of VRS.
- We present an algorithm and a concrete implementation for Motion-Adaptive Shading and an integration of VRS with a deferred shading pipeline. By utilizing motion vectors, for example, computed for motion blur post-processing techniques, we control the shading rate of regions of still parts of the scene and parts in motion.

The remainder of this thesis is structured the following way: In chapter 2, we review related work concerning dynamic resolution approaches before VRS existed, deferred shading and how it can be incorporated with multi-sampling approaches, eye-tracking and foveated rendering, and anti-aliasing techniques for mitigating the inevitable aliasing of coarse shading. In chapter 3, we describe the capabilities of VRS in detail. We will discuss the possibilities and also its limitations. In addition, we describe how to integrate our new VRS techniques into an existing rendering engine. In chapter 4, we present our developed algorithms and go into technical details of the implementation. First, each algorithm is described in detail with all the necessary steps. The main steps, like the computation of the shading-rate image, are further divided into sub-steps, explained and reasoned. Afterwards, we discuss how to incorporate eye tracking with VRS and an existing engine. To evaluate our approaches, we will compare the frame times of each method in a forward and a deferred renderer with and without VRS in chapter 5.

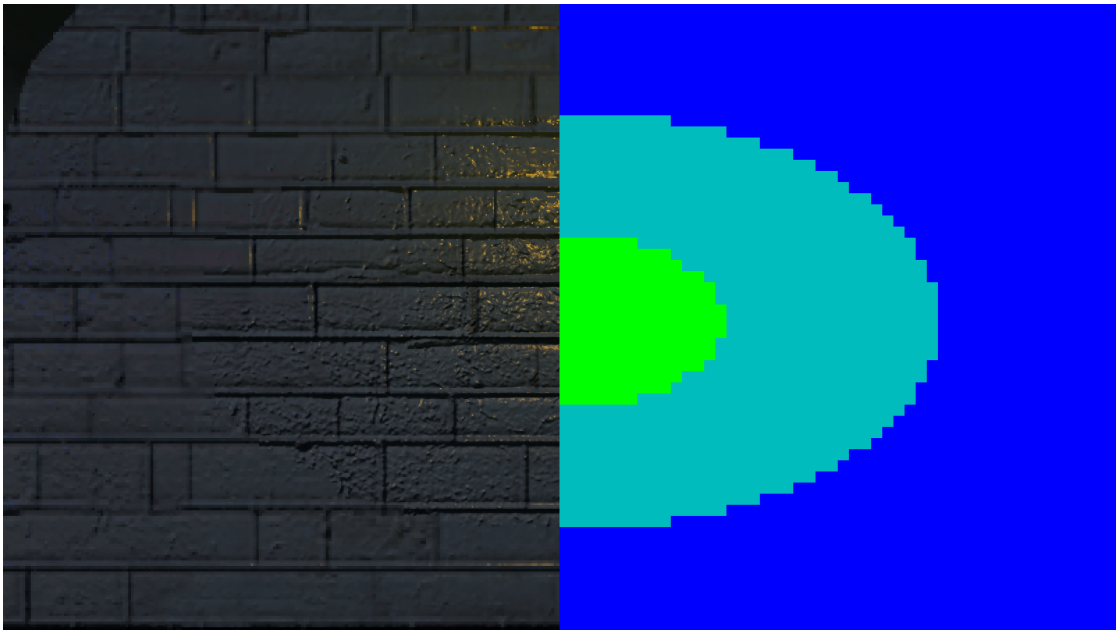


Figure 1.1: A visible edge at the border of different shading rates in the rendered image at the left-hand side. The right-hand side shows the shading-rate image with 1x1 shading in the green circle, 2x2 in the cyan circle and 4x4 in the blue circle.

CHAPTER 2

Related Work

2.1 Dynamic Resolution Approaches

He et al. [HGF14] presented with their multi-rate shading a concept that is pretty close to VRS. They state that fragment shading computations use most of the computational power for rendering a frame in real-time graphics. For mobile applications, the amount of time spent in the fragment shader compared to everything else goes up to 95%. This makes clear why techniques like VRS are advantageous for the field of real-time computer graphics: It enables us with better control of the area where we have most to gain. The extended graphics pipeline He et al. proposed, therefore reduces the fragment shader computations per-pixel to a factor of two to three. They evaluated their concept with a CPU implementation simulating heavy SIMD executions. Though, their concept is designed to be supported natively in graphics cards. In comparison to VRS, multi-rate shading performs rasterization on coarse fragments and then decides in coarse fragment shading if a fine per-pixel fragment shading is necessary. This leads to jagged images and more aliasing as edges are effectively rasterized at a lower resolution.

Ragan-Kelley et al. [RKLC+11] proposed extensions for modern graphics pipeline architectures which decouple visibility samples from shading samples. This is achieved by applying a hash function after each visibility sample which maps to a shading sample. The shading sample is cached and reused for further visibility samples. With this hash function, a many-to-one relationship between visibility and shading is defined. This extension would allow efficient supersampling for motion blur, defocus blur and also enables VRS. For example, with the hash function, one could reduce the shading rate by mapping four visibility samples to the same shading sample, or apply supersampling for visibility while keeping the shading rate at fragment size, as Multisample anti-aliasing (MSAA) does. The authors describe it as a generalized form of MSAA, though, the option of specifying a hash-mapping function is much more flexible. For example, coarse shading can have dynamic morphologies of dynamic sizes and does not have to be quadratic

tiles of predefined size. This flexibility comes with a disadvantage. The hash function has to be evaluated for each visibility sample. While this cost might not be excessive and as simple as a 3D matrix multiplication, another disadvantage might have more impact. The size of the framebuffer has to be the full supersampling resolution. For this reason, a manual resolve which scales the framebuffer to the screen resolution has to be implemented and added to the pipeline. In contrast, VRS allows supersampling only on a region basis, though utilizes current MSAA hardware support for the additional samples needed and the resolve function.

Very close to VRS is the proposed adaptation of the graphics pipeline by Vaidyanathan et al. [VST+14]. Like the extensions developed by Ragan-Kelley et al. [RKLC+11], their method decouples the visibility samples from the shading. Though, they already suggested to divide the screen into tiles and define the shading rate for each tile. They state, that the simplifications of a tiled-based approach together with fixed, predefined coarse shading rates would allow efficient implementation in current graphics hardware. The shading rate is specified as coarse pixel size x and y and can be controlled either as interpolated values of vertex shader outputs, set as a constant value for the whole frame or as a radial function in screen coordinates for foveated rendering. The so defined coarse pixel size is then quantized to the closest predefined shading rate and the minimal size of all defined shading rates is taken as size for a tile to meet the required shading rate. With VRS one cannot select the desired coarse pixel size freely but choose from a predefined set of shading rates. Additionally, the shading rate is controlled by a shading-rate image and not by a radial function or vertex-attributes. Vaidyanathan et al. [VST+14] also compensate for visible shading rate changes by scaling the texture LOD with the coarse pixel size. This is handled similarly with VRS: The hardware automatically chooses the right texture LOD level based on the shading rate.

Hillesland and Yang [HY16] proposed a method called Texel Shading which transfers the shading into the texture space, effectively decoupling the shading from the fragment size. Their method consists of three stages: Shade Queuing, Shade Evaluation, and Shade Gathering. In the first stage, Shade Queuing, the tiles of texels that require shading are computed by a geometry pass and added to a queue. They apply a caching algorithm to not add already shaded tiles to the queue. The second stage, Shade Evaluation, a compute pass shades the texels in texture space. In the final stage, Shade Gathering, a second geometry pass, the shaded texels are retrieved and the screen space image is computed. The whole algorithm works on a texel basis at a certain mipmap level. With this mipmap level, the shading rate can be effectively controlled. Their method requires that each mesh has an object-space parametrization. Additionally, this parametrization has to be non-overlapping in texture space and static. These requirements are very restrictive if one takes into account modern high-fidelity scenes. For example, tessellated displacement mapping cannot be easily used in combination with this method. Additionally, it is hard to integrate such an approach into an existing engine because it changes the whole stages setup.

Recently, several patents held by Microsoft were published, concerning variable rate

shading, e.g., [GGNB18], [NFGG18]. Both patents are concerned with abilities to dynamically control the shading rate. The patent held by Nevraev et al. [NFGG18] describes how the shading rate can be controlled by a map. It is described how map coordinates can be derived by coarse scanning a primitive. These map coordinates are then used to query a sampling rate parameter (SRP) map. By identifying a lookup value a fragment variable is calculated which is then used to determine the shading rate. The patent held by Grossman et al. [GGNB18] describes a method of how the shading rate parameter can be controlled on a per primitive basis. For example, they describe the definition of a shading rate per-vertex. This shading rate is then interpolated between the vertices, e.g., at the intersection of a tile and a primitive, in order to compute the final shading rate for the tile. This control mechanism was not mentioned by NVIDIA at the release of the Turing Architecture [Bho18]. Though, it might be supported in the future, as the patent of Nevraev et al. [NFGG18] utilizes the control mechanism of a per-primitive basis indirectly to control the shading rate via the shading-rate image.

2.2 Eye-Tracking and Foveated Rendering

Microsoft holds a patent for a foveated rendering method published by Guenter et al.[GFS+17]. This method is based on eye-tracking the gaze point of a user and computing eccentricity layers. These eccentricity layers are then anti-aliased and used to render a foveated image. This is the common way how foveated rendering was done before hardware support for different shading rates was present. Guenter et al. [GFD+12] describe why foveated 3D graphics are needed due to the falloff of perception in the human visual system. Their method uses three layers of rendered images at different resolutions hence with different sampling rates combined with an anti-aliasing strategy. It can be viewed in figure 2.1. The layer with the highest resolution is defined by tracking the user's gaze point. Around this point, the falloff for the different layers is computed and the layers are rendered and combined. Of course, this leads to unnecessary fragment shader executions around the center. To tackle this problem, they reduce the temporal resolution of the outer layers. With VRS unnecessary fragment shader executions and blending computations can be avoided.

In order to study the effects of a decreasing resolution, Patney et al. [PSK+16] performed a user study utilizing foveated rendering images simulated by Gaussian blurs using a progressively increasing filter based on the distance to the focus of the user. The result of this study was that a large blur radius causes the user to feel a kind of tunnel vision. The reason for this effect is that the human vision indeed has reduced resolution in the periphery, though it has a high contrast sensibility. The needed contrast gets lost with a Gaussian blur and therefore they had to apply a contrast-preserving Gaussian filter. Additionally, they noticed a higher sensibility to motion in the periphery of the human vision. Patney et al. [PSK+16] designed a foveated rendering system that copes with these effects while still exploiting the reduced acuity in the peripheral vision for a performance gain. Their system consists of a pre-filtering step for some shading attributes, a software implementation of coarse shading from Vaidyanathan et al. [VST+14] and two

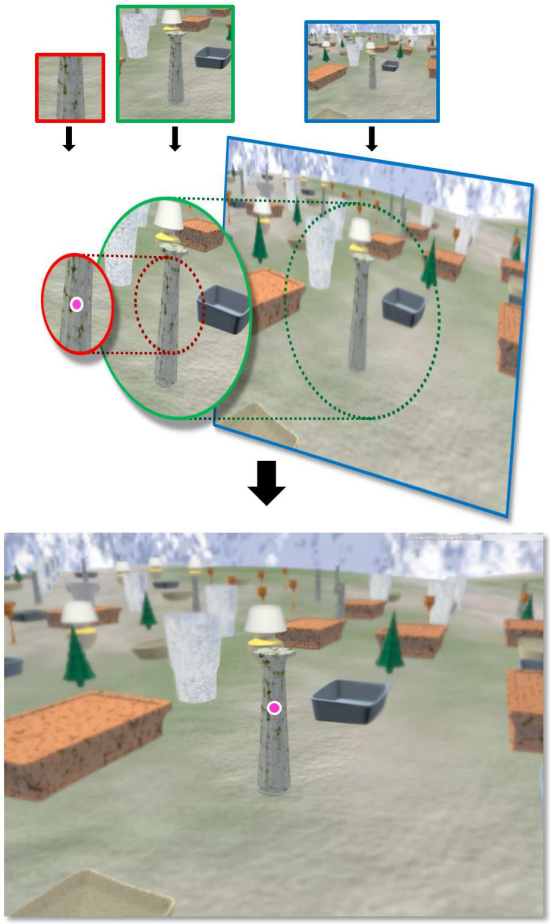


Figure 2.1: Foveated rendering method based on three layers at different resolutions by Guenter et al. [GFD+12].

post-processing steps, an adapted temporal antialiasing (TAA) algorithm and a contrast preservation based on Grundland et al. [GVWD06] work. The pre-filtering steps are to avoid aliasing from coarse shading. The coarse shading approach by Vaidyanathan et al. [VST+14] has already been discussed above. In order to avoid gaze-tracked aliasing, which results from changing shading rates following the gaze of the user, they introduce variance sampling, an adapted TAA algorithm. Variance sampling uses a mean and variance-based bounding box for the color of the previous frame. Additionally, to cope with fast saccade changes of the eye the alpha blend value of TAA is adapted to accelerate the convergence of the image. Contrast is reduced by alleviating aliasing with the pre-filtering and post-processing. For this reason, a contrast preserving post-processing step, a variant of unsharp masking, is applied after all other post-processing effects.

2.3 Deferred Rendering

Deferred shading is a technique that is applied to postpone the complex lighting computations and only execute them once per pixel. Applying VRS in a deferred shading renderer can lead to similar problems like with MSAA in a deferred shading renderer. The basic problem is the mismatch between the sampling rate in the geometry pass and the light pass. Therefore, it is of interest how this problem has been solved for MSAA. Some techniques use the already present information at a superresolution to perform post-processing anti-aliasing, like Chajdas et al. [CML11] do with Subpixel Reconstruction Anti Aliasing (SRAA). They use textures of the geometry buffer and normal buffer, which are present at a higher resolution to apply morphological post-processing anti-aliasing. The advantage compared to morphological anti-aliasing is that their technique has fixed runtimes independent of the scene complexity. Such techniques are not directly applicable to integrate VRS with deferred shading because they replace MSAA instead of integrating it.

2.4 Anti-Aliasing

Aliasing is the result of sampling a signal under the Nyquist rate. The lower the sampling the more this effect is prevalent in images. This undersampling happens especially with coarse shading in VRS as the sampling rate is further reduced. It not only results in a loss of quality but also in flickering during motion destroying the immersion in the scene. For this reason, we propose to apply and adapt anti-aliasing approaches in order to cope with the artefacts of shading rate-dependent aliasing.

The straightforward way to reduce aliasing is to sample at higher rates. This is called Supersampling Anti-aliasing (SSAA) and delivers very high-quality results, though at a high performance cost, Akenine-Moller et al. [AMHH18]. In order to improve performance while still maintaining a high anti-aliasing quality, multisampling anti-aliasing (MSAA) has been developed. With MSAA, visibility is sampled at a higher resolution, while shading samples are reused. For example, four samples are used for a single pixel, see Figure 2.2. When a triangle is rasterized, all samples covering the triangle for the pixel are computed. Then a single fragment shader execution is performed and the value is written in all samples covering the triangle of the pixel. After rendering all objects, a resolve pass averages all the samples and computes the final pixel value. With this technique, geometry aliasing is effectively reduced and at a level of SSAA. Texture aliasing is still present in the image and the performance impact is lower compared to SSAA but still too high, Akenine-Moller et al. [AMHH18].

Due to the lack of texture anti-aliasing of MSAA and because it does not integrate well with deferred shading, post-processing anti-aliasing techniques have been developed. Jimmer at el. [JGY+11] provide an overview of the main post-processing AA techniques used in modern game engines. FXAA, developed by Lotte/NVIDIA [Lot11], is one of the most performant post-processing anti-aliasing techniques which gives good results and

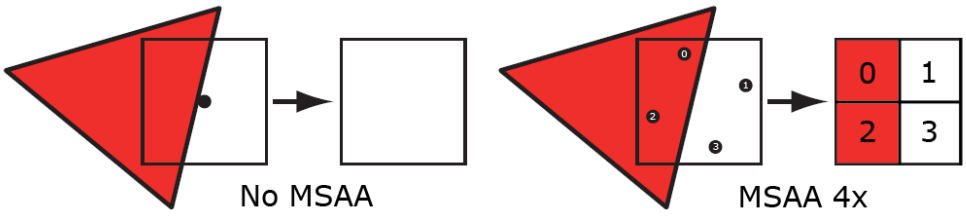


Figure 2.2: An example of drawing a triangle with MSAA disabled and with four MSAA samples. Reprinted from Pettineo [Pet12]

is therefore provided by most modern game engines. In order to reduce computational effort, FXAA operates only on edges.

The detailed process consists of the following steps and each step is visualized in Figure 2.3 numbered from left to right, top to bottom:

1. The input image as non-linear RGB color is internally converted into a scalar luminance value.
2. Edges (marked red) and sub-pixel aliasing (marked yellow) are detected based on local contrast.
3. The detected edges are classified as horizontal (gold) or vertical (blue).
4. Utilizing the edge orientation, the highest contrast pixel pair perpendicular to the edge is selected (blue/green).
5. By comparing the average luminance of the high contrast pixel pairs perpendicular to the edges the end-of-edge in both directions, negative (red) and positive (blue) is computed.
6. With the ends of the edge, the pixel position is transformed into a sub-pixel shift 90 degrees perpendicular to the edge in order to reduce aliasing, red/blue for minus/plus horizontal shifts, gold/skyblue for minus/plus vertical shifts.
7. With this sub-pixel offset, the input image is re-sampled.
8. In the last step, the amount of the detected sub-pixel aliasing is utilized for blending in a lowpass filter.

FXAA reduces the contrast of subpixel features or blurres them completely depending on the parameters. With coarse shading of VRS, this does not only apply to subpixel features but also to features spanning more than a single pixel, e.g., a coarse pixel. This would introduce a larger blur to the region coarsely shaded, which is perfectly fine as there is not more information present to reconstruct these features. FXAA and other edge finding algorithms are not temporally stable, e.g., they jitter under motion.

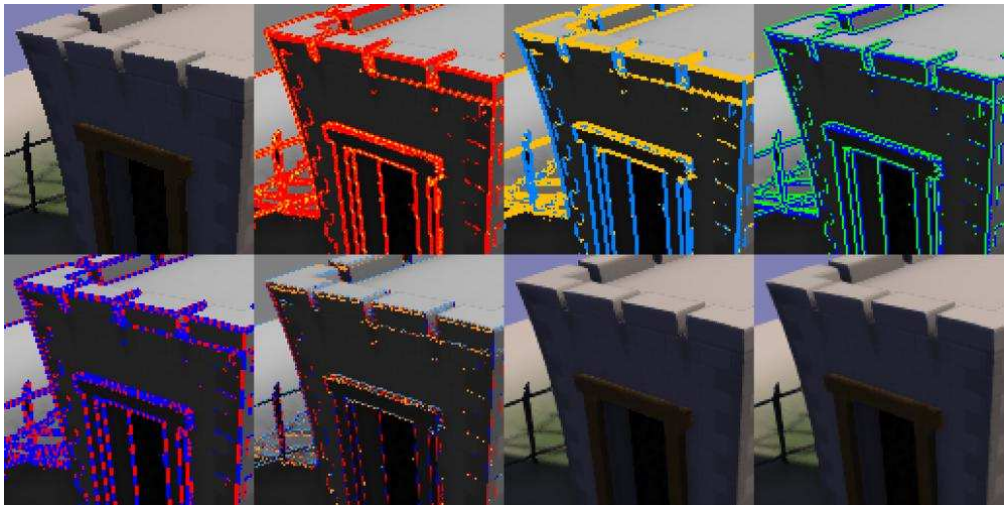


Figure 2.3: The steps of FXAA visualized. First, edges are computed based on the local contrast relative to the local maximal luminance. These edges form the basis for the anti-aliasing. Next, the edges are classified as horizontal or vertical. Then, the highest contrast pixel pair perpendicular to each edge is selected. With this high contrast pixel pair, both ends of an edge are searched by looking for a significant change in the average of the pair. These ends are then used to compute a subpixel offset perpendicular to the edge. Finally, the input texture is re-sampled with the computed subpixel offset and a low pass filter based on the detected subpixel aliasing is applied. Reprinted from Lotte/NVIDIA [Lot11].

Amortized Super Sampling by Yang et al. [YNS+09] utilizes the data from previous frames to exactly receive this missing information for reconstructing subpixel features. Their approach is a Monte Carlo integration utilizing importance sampling. Instead of rendering the scene multiple times they reuse the previous frames. Each frame is rendered with a subpixel offset moving it in one quadrant of the pixel and then saved into an individual subpixel buffer. The current image is computed as a weighted average of the four sub-pixel buffers. The process is visualized in Figure 2.4. Without motion, this would result in an image with double resolution SSAA. With motion, a re-projection is needed and the image gets blurry. Additionally, the re-projection might fail. Then they invoke the shader for this failed sample to get the correct value.

The current state-of-the-art for temporal methods, temporal supersampling anti-aliasing (TAA), was developed by Karis [Kar14]. In contrast to the method of Yang et al. [YNS+09] TAA only uses the previously resolved buffer for blending with the current frame, instead of multiple subpixel buffers. By utilizing a weighted average with large importance for the previous frame, one can approximate an average over multiple previous frames and therefore save the framebuffer memory and computational effort of sampling multiple textures. The detailed process can be viewed in Figure 2.5. In order to avoid clustering in space or time, a low discrepancy progressive sequence is used for the subpixel

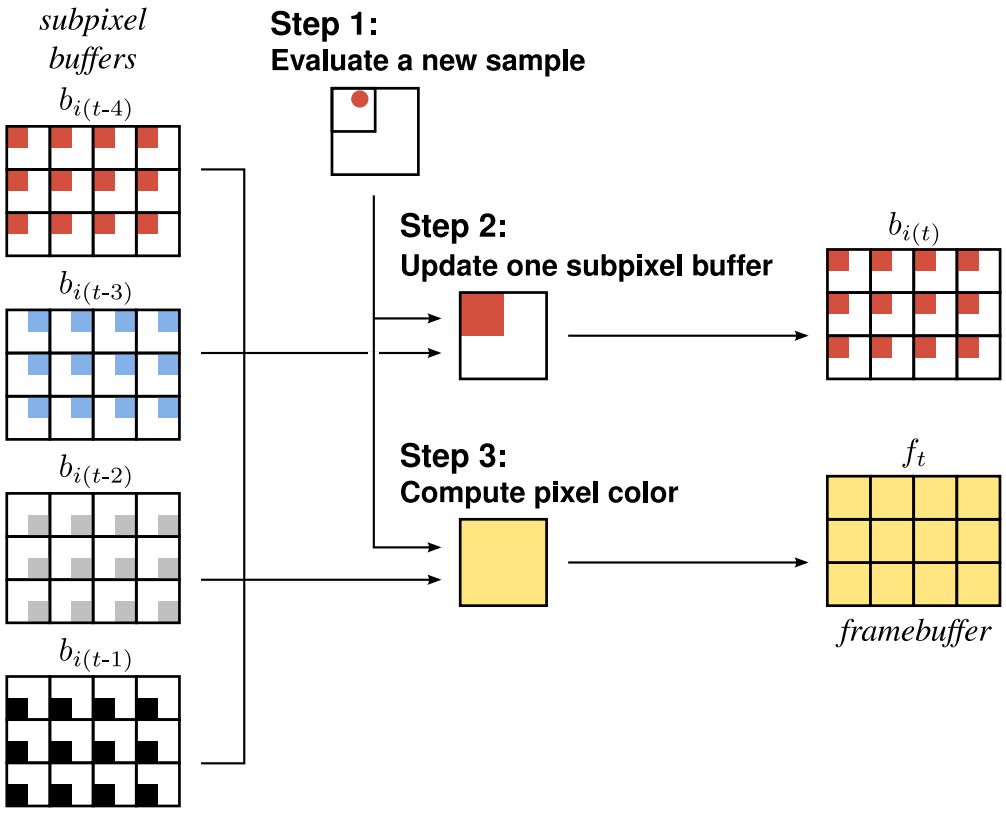


Figure 2.4: The main steps of amortized super sampling Yang et al. [YNS+09]. The four buffers on the left-hand side represent the four previous frames. In the current frame, at step 1, the scene is rendered with the next offset. In step 2., one of the subpixel buffers is updated with the result of this rendering. In step 3., the final image is computed by averaging the four subpixel buffers.

offsets, e.g, Halton sampling. Misses due to re-projection, which would result in ghosting, are handled by clipping the history value to the value of the current frame. This clipping is performed not in Chroma but Luma color space, as Luma has a high local contrast compared to Chroma. TAA delivers high-quality anti-aliasing with subpixel features at real-time ready performance rates. Nevertheless, it requires fine-tuning for the alpha and clamping parameters to avoid ghosting and other artefacts. Additionally, a static scene with no moving camera might result in flickering due to clipping.

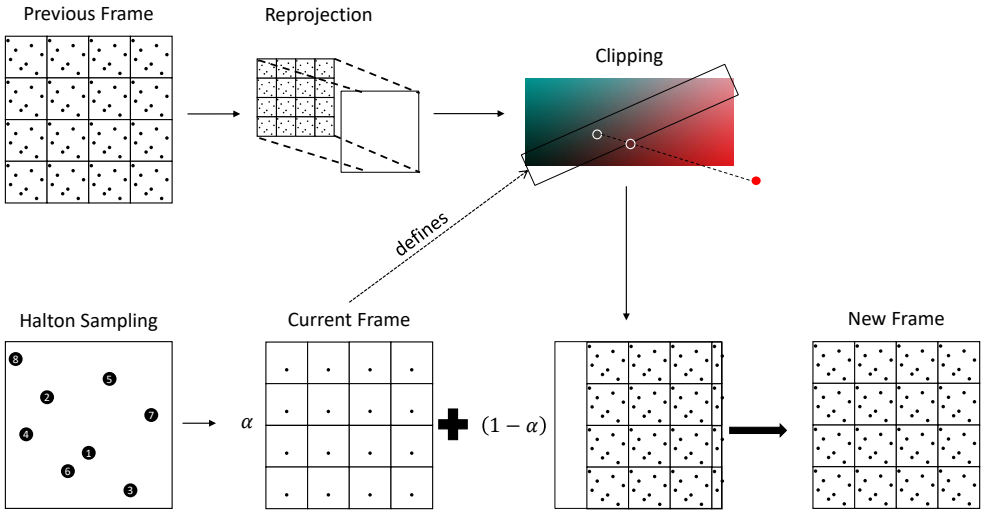


Figure 2.5: The main steps of TAA. Left in the bottom row, the current frame is sampled at the next Halton sample. At the top row, the previous frame is reprojected onto the current frame. Next, still in the top row, a clipping space based on the color values of the current frame is computed and used to clip the luma values of the reprojected frame. This image is then combined with the current frame by the weight α. The result of this weighted average then gives the anti-aliased new frame. Adapted from Karis [Kar14].

CHAPTER 3

Variable Rate Shading

Without VRS, the GPU hardware assigns each region of the framebuffer the same amount of fragment processing power. Disregarding overdraw and blending, there is exactly one fragment shader execution per pixel. In other words, the shading rate is one for each pixel. With NVIDIA's Turing architecture, this restriction has been removed and one can control the amount of processing power per framebuffer region, as described by Bhonde/NVIDIA [Bho18] and in the NVIDIA Turing architecture whitepaper [NVI18].

For example, a region spanning four pixels is only shaded once and the result is copied into all four pixels. As already discussed in chapter 2, there were already various proposals to extend the GPU architecture in order to be able to control the shading rate. He et al. [HGF14] proposed multi-rate shading, doing rasterization with a lower resolution and deciding per pixel if a higher shading rate is needed. Ragan-Kelley et al. [RKLC+11] proposed a hash-based shading-rate selection where already shaded samples are reused based on a computed hash for a visibility sample. The proposal of Vaidyanathan et al. [VST+14] already utilizes tiles for which a shading rate is defined. Many of these proposals have in common that the control of the shading rate resides between rasterization and fragment shader executions. For this reason, visibility is still computed at a higher rate, which avoids geometry aliasing. This means, VRS allows the reduction of fragment shader executions. This important fact has to be kept in mind while searching for various scenarios where we could utilize VRS. VRS not only allows to do lower sampling by taking regions spanning multiple pixels, called coarse shading, but also enables adaptive supersampling, e.g., shade a single pixel at multiple subpixels. The supported shading rates at the time of writing are:

- default sampling 1x1
- supersampling in invocations per pixel: 2x, 4x, 8x
- coarse shading in pixel regions: 1x2, 2x1, 2x2, 2x4, 4x2, 4x4

- no invocation

The coarse shading rates can be viewed in Figure 3.1. In addition to the shading rates displayed in Figure 3.1, shading rates for supersampling and a shading rate for no invocations per pixel can be used. While these are the shading rates which are currently supported by the hardware, one could imagine other shading rates. For example, the Vulkan specification [Khr19] already defines a supersampling shading rate for 16 invocations per pixel.

Figure 3.1: Supported coarse shading rates with an example of VRS in a racing scene. Reprinted from the NVIDIA Turing architecture whitepaper [NVI18].

For coarse shading, the center of the pixel region is used to interpolate the varying values for the fragment shader. If the center of the pixel regions is not inside the rasterized triangle, this can lead to the same aliasing artefacts as without coarse shading when the center of the pixel is not inside the rasterized triangle, though much more severe as now multiple pixels are affected. To reduce this effect, centroid sampling may be used, which guarantees that the varying values for the fragment shader are computed at a location where the fragment and the triangle intersect. In Figure 3.2 coarse pixels with coverage samples, the coarse pixel center and the centroid are exemplarily depicted.

The shading rate is controlled in a two-step fashion with one indirection step. First, a shading-rate image is created. This image specifies the desired shading rate for each 16x16 tile of the target viewport. The size (w_i, h_i) of the image should, therefore, be $w_i = \lfloor \frac{w+15}{16} \rfloor$ and $h_i = \lfloor \frac{h+15}{16} \rfloor$, where w and h are the dimensions of the viewport. For a pixel (x,y), the coordinates (u,v) in the shading-rate image are computed by $u = \lfloor \frac{x}{16} \rfloor$ and $v = \lfloor \frac{y}{16} \rfloor$. The shading rate at this location is used as an index in a lookup table to compute the real shading rate. This lookup table can be specified per viewport and allows changing the effective shading rate without the need to compute and bind a new shading-rate image. The process is visualized in Figure 3.3. While the tile size is currently fixed to 16x16 pixels by the NVIDIA hardware implementation, the Vulkan specification [Khr19] states that this value is implementation dependent and requires the user of the

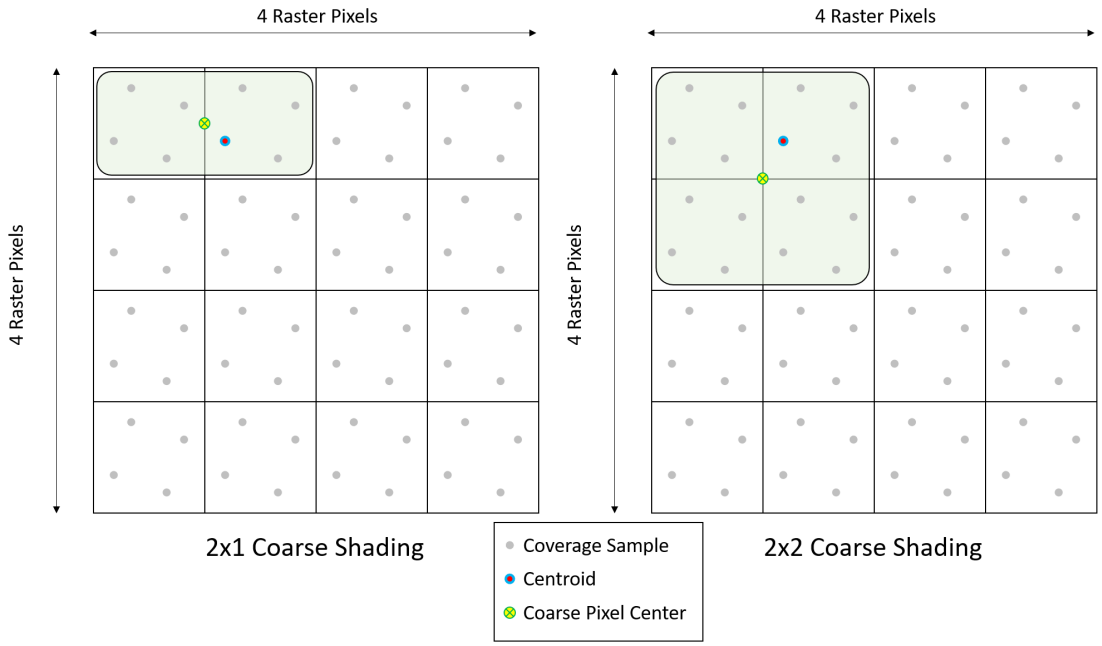


Figure 3.2: Coarse pixel center and example centroid for coarse shading rates 2x1 and 2x2. The centroid is only guaranteed to be in the intersection of the primitive, e.g., a triangle, and the coarse pixel. It can, therefore, be anywhere inside of this intersection. Reprinted from Bhonde/NVIDIA [Bho18].

API to query it. An example scene with the shading rates as overlay for each tile can be viewed in Figure 3.1.

3.1 Possibilities

In the previous section, we have described the capabilities of VRS. In this section, we describe how this new hardware feature can be exploited to improve performance and quality. We have the ability to do selective supersampling and selective coarse shading. These two possibilities apply only to fragment-shader executions. Hence, the visibility resolution stays constant throughout the image and cannot be dynamically adapted with VRS. Only the shading resolution can be dynamically adapted. For this reason, the focus of the optimization lies on the shading resolution, e.g., by detecting regions that should be shaded with higher fidelity and regions that are not required to be shaded in full resolution. Finding such regions in real time can be expensive, and the cost for analyzing the scene and defining such regions with high and low details has to be weighed against the gain in performance achieved by lower sampling or the additional performance impact for higher quality. NVIDIA provides some examples of how to utilize VRS in the Turing architecture whitepaper, namely Content-Adaptive Shading, Motion-Adaptive Shading

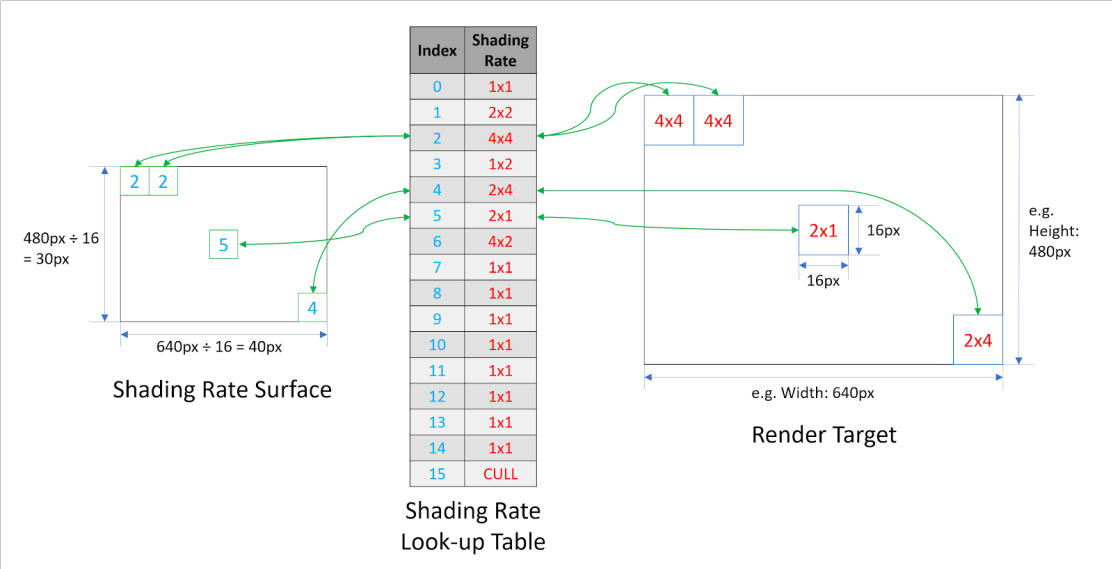


Figure 3.3: The process of the shading rate selection with the indirection step visualized. (u, v) coordinates in the shading-rate image are computed based on the (x, y) coordinates in the image. These coordinates are used to fetch an index. Subsequently, the fetched index is used to look up the actual shading rate in the Shading Rate Lookup Table. Reprinted from Bhonde/NVIDIA [Bho18].

and foveated rendering [NVI18].

In **Content-Adaptive Shading**, the shading rate is defined by the content of the scene. In the best case, homogeneous regions with only low-frequency content are shaded with a lower resolution and the perceived image stays the same. In practice, this is not always achievable. For example, there can be scenes where all regions have a high frequency or just some regions smaller than 16x16 pixels are homogeneous. Undersampling such regions would result in aliasing, which is more or less visible to the viewer. Another problem lies in the definition of the shading rate. The shading-rate image defining the shading rate must be computed and bound before the draw commands are submitted. Thus, information as to which regions are homogeneous needs to be known beforehand. NVIDIA suggests computing the shading rate after a frame is drawn for the next frame [NVI18]. The advantages of this approach are that little additional processing power is used for the computation of the shading-rate image and it can easily be integrated into an existing pipeline with only minor changes to the shaders. However, some points have to be considered for this approach: Due to changes in the scene, e.g., from animation or camera movement, a forward-projection to the next frame is necessary. The shading-rate image has to be computed before the frame is drawn, based on data from the previous frame and the current motion. The current motion can be used to compute the position in the new frame. The information the shading rate is based on can then be the spatial and temporal color coherence of the previous frames.

An alternative approach would be to pre-render the scene in a lower resolution. The resolution can be as low as $\frac{1}{16}$ in width and height of the screen size. This low-resolution pre-pass adds the computational effort of approximately $\frac{1}{256}$ of a full-resolution rendering in terms of fragment shader executions plus the duplicate geometry processing. However, one has to be very careful to pre-filter all required resources to avoid aliasing. Pre-filtering for resources like standard textures can be easily achieved with mipmapping. The pre-filtering of normal maps or procedurally generated resources can be harder to achieve or is not even possible. Depending on the screen resolution and the geometry of the scene, the lowered resolution can even cause geometric aliasing, e.g., visibility is sampled too low. In order to save additional GPU cycles, shaders directly computing the necessary results, e.g., edges for the shading-rate image, could be used. This kind of optimization could be taken even further, for example by pre-computing the edge images of textures and avoid real shading completely. Of course, this would not capture high frequencies introduced by lighting, e.g., on a rough surface. However, with a combined analysis of normals, geometry, and textures, even this problem can be addressed, and only dark spots with complex geometry or a rough surface would not be recognized as homogeneous areas. This approach requires a lot of pre-computation work, partially even on an artistic side, and existing pipelines would have to be adapted.

Results from psychophysical research suggest that motion is perceived blurred. Apthorp et al. [ASK+12] investigated the perception of random moving objects on a screen without motion blur artefacts and the connected neural activities. They discovered that blurred lines, or "motion streaks" as they call them, occur in the visual system due to temporal integration. These motion streaks are used by our perception to detect the direction of the motion. In computer animation, no real motion takes place but images are presented one after another, commonly in a 60 Hz update fashion. Due to temporal integration and objects jumping from one position on the screen to the next one during frame swaps, our eye combines both images for a short time. The object is not perceived in full detail but blurred. This perceived blurring of regions on the screen allows to improve the performance by reducing the rendered resolution. In Figure 3.4 an example of an object moving from left to right is shown. The multiple images seen by our eye are integrated. The blurred image at the bottom right of the figure is the result.

As suggested by the NVIDIA Turing architecture whitepaper [NVI18], it would be wasteful to render regions that are perceived blurred at full resolution. The resolution in these regions could be reduced without a noticeable visual impact. This effect is already simulated by motion blur, and **Motion-Adaptive Shading** takes advantage of this, too. NVIDIA [NVI18] suggests computing the shading rate directly from the magnitude and directions of the motion vectors, which are already computed for TAA. Furthermore, other blur effects like depth of field (DOF) could also be exploited with VRS in order to save performance.

As already discussed in Chapter 2, VRS can also be utilized for **foveated rendering**. Due to the physical composition of the human eye, the perceivable resolution falls off the more acute the viewing angle gets. The highest resolution is only achieved at the

Figure 3.4: An object moved from left to right, perceived blurred by our eyes. On the left, the object is displayed. Top right, the motion is schematically represented. The image bottom right shows the possible perceived result. Reprinted from NVIDIA Turing architecture whitepaper [NVI18].

center, the Fovea centralis. Hence, the name foveated rendering. With VRS we are able to directly use data from eye trackers to compute the shading-rate image and then set this image for the whole scene. The performance is expected to increase depending on how much time is spent computing the shading rate compared to the multi-pass setups or the emulations of VRS.

3.2 Limitations

While VRS offers great new possibilities to developers, there are also limitations. These limitations have to be considered when finding new application scenarios for VRS and should always be kept in mind during the implementation of new algorithms utilizing VRS. One core concept of VRS is that it separates the visibility sampling from the shading. We can dynamically configure the shading rate but not the visibility rate.

This leads us to an important limitation of VRS: We can only save performance or improve quality by controlling the number of fragment shader executions.

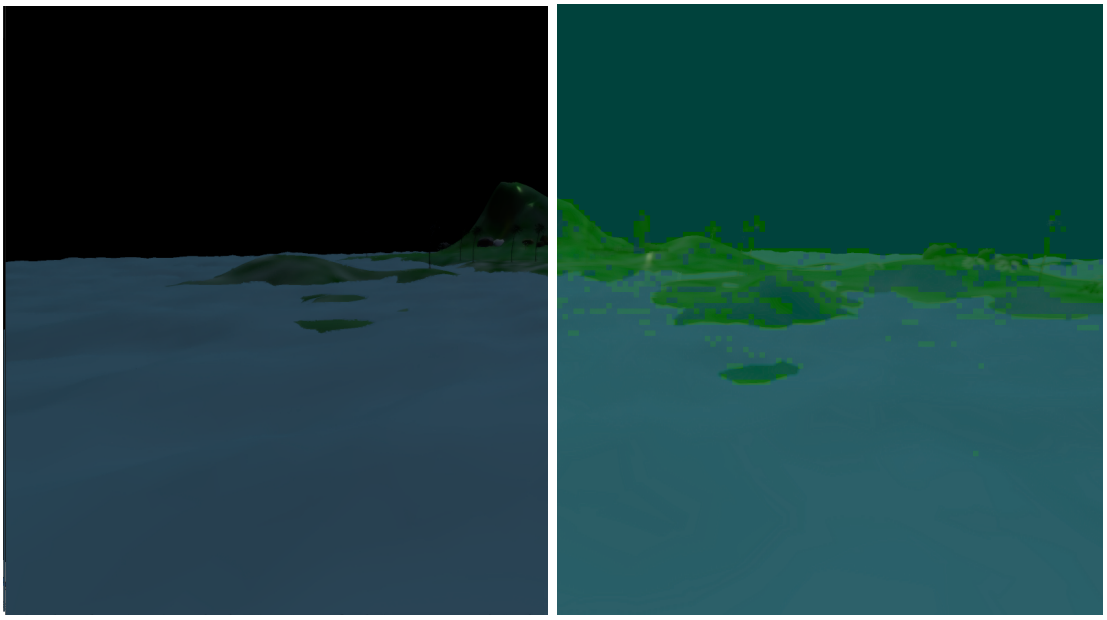
As an example of this limitation, we discuss how VRS seems to be applicable for cascaded shadow mapping. By creating better-fitting non-rectangular viewports, specifying the target frustum as shading-rate image and changing the shading rate based on the cascade, we expect a faster computation of the shadow maps. Cascades closer to the camera would be assigned higher shading rates than cascades further away from the camera. This

would allow a single-pass cascaded shadow mapping at different resolutions. However, VRS cannot be utilized in this way for cascaded shadow mapping as VRS separates the visibility sampling from the shading. For shadow mapping, we only need the visibility, e.g., the depth, and therefore no fragment shader is executed. Hence, VRS is not applicable to improve cascaded shadow mapping as described.

Another limitation that has to be considered during the development of VRS-based algorithms is the size of a tile controlling the shading rate. This tile size is limited to 16x16 pixels. At the time of writing, no other tile sizes are supported by VRS.

To illustrate this limitation, we present an example where we apply Content-Adaptive Shading on different types of scenes. The Content-Adaptive Shading implementation is based on edges. We extract an edge image from the previous frame and base our shading rate on these edges by doing a forward-projection. The amount of edges in a 16x16 tile specifies the shading rate. For example, in a region with no edges, 4x4 coarse shading is used, in a region with few edges, 2x2 coarse shading is used and in a region with many edges, the shading rate is specified as 1x1. This can be viewed in Figure 3.5a. In Figure 3.5b higher shading rates are colored green, while lower shading rates are colored cyan to blue. For this scene with many large homogeneous regions, like a large water surface and an island with grass in the distance, this approach works well. The shading rate is reduced for regions with few details, e.g., the water or the black sky and some parts of the green grass. Other parts like the palms and especially borders between the water and the island are rendered with high shading rates. In another scene, we have brick stone walls and a brick stone floor, as display in Figure 3.6a. The bricks are each too small to include a complete tile. While the regions inside of each brick have a homogeneous color, our shading rate is computed to be high for all the brick stone walls and floors. The reason for this is that each 16x16 tile of the image contains at least one junction of two bricks. A junction is visible as an edge and correctly computed as one. The shading rate is specified for a 16x16 tile, which contains these edges and is therefore kept at a level to preserve these edges. In Figure 3.6b we use an overlay to visualize the tile size. As a result, we have invested processing power in computing the edge image and the shading rate and did not gain anything, though there are many but very small homogeneous regions.

Another, not so obvious limitation of VRS has its roots in the hardware limitations and configuration of MSAA, specifically the MSAA samples. VRS supersampling utilizes the coverage samples of MSAA in the pipeline and textures. For this reason, the maximal amount of possible and configured MSAA samples defines the highest possible supersampling shading rate. If a higher shading rate is specified, it will automatically be reduced to the maximum possible with the current configuration. For example, if one specifies a shading rate of 16x supersampling and the hardware supports only 8x MSAA samples and 8x MSAA samples are configured for the pipeline and the framebuffer attachments, the shading rate will be automatically adjusted to 8x supersampling as it can be viewed in Figure 3.11. While this aspect might seem plausible as the existing samples for MSAA are reused, the configured MSAA sample count is not only limiting



(a) The island scene.

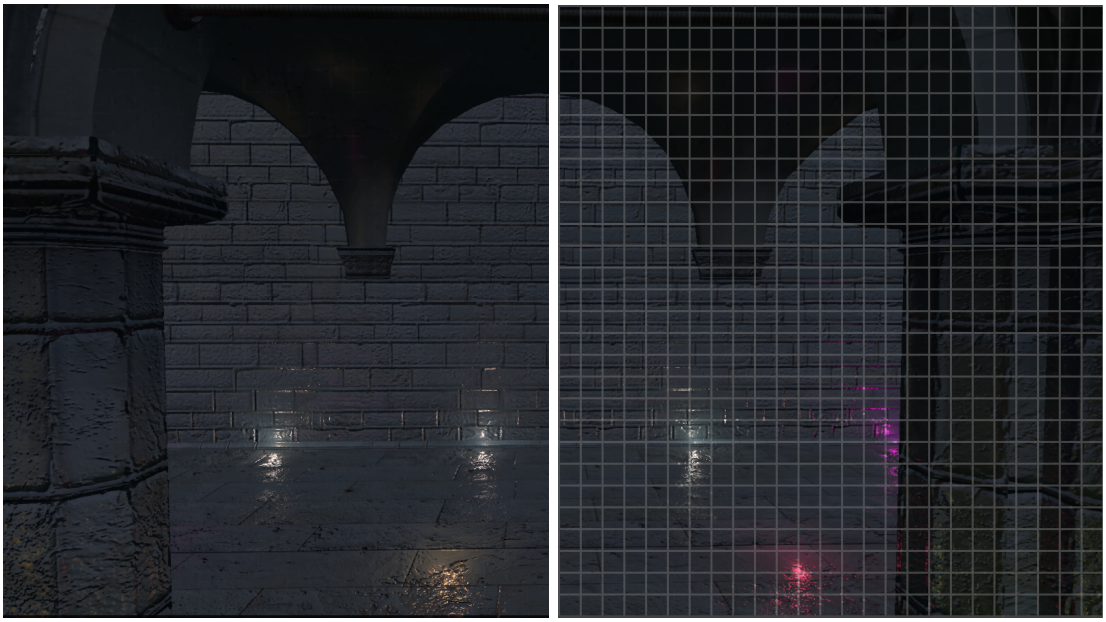
(b) The island scene with a shading rate overlay. Green marks high shading rates, blue low shading rates.

Figure 3.5: Example scene with an island in the sea. The island has more details and edges and is therefore shaded with a higher rate.

supersampling but coarse shading of VRS, too. In order to explain this limitation, we first have to describe a new implementation-dependent maximum introduced with VRS, namely shading rate max. coarse coverage samples. A single fragment, regardless of its size, only supports this maximum amount of coverage samples. Therefore, the hardware limits the number of coarse coverage samples for a single fragment, e.g., on a GTX2080 TI the number of possible coarse coverage samples is 16. For example, 4x4 fragment sizes are possible with one MSAA sample as there are 16 pixels multiplied with one coverage sample per pixel, and this number is lower or equal than the maximum of 16 coverage samples, as depicted in Figure 3.8. If we increase the coverage samples we want to use by configuring, for example, 2x MSAA samples, we now need two coverage samples per pixel and only fragment areas of pixel count $\frac{16}{2}$ are possible. Any shading rates specifying larger fragment sizes will be adjusted in order to not exceed the number of maximum coarse coverage samples. Figures 3.9, 3.10 and 3.11 show each possible configuration from 2x MSAA to 8x MSAA with the set and effective shading rates.

The Vulkan Specification [Khr19] describes this as follows:

> "Once a base shading rate has been established, it is adjusted to produce a final shading rate. First, if the base shading rate uses multiple pixels for each fragment, the implementation may reduce the fragment area to ensure



(a) The brick wall. (b) The brick wall with tile size overlay.

Figure 3.6: The tile size as a limitation: A brick wall with too small bricks for edge-based Content-Adaptive Shading.

> that the total number of coverage samples for all pixels in a fragment does not exceed an implementation-dependent maximum." (Vulkan Specification, Chapter 26.6. Shading Rate Image [Khr19])

How it is reduced is not specified and therefore open to the driver implementation. During our experiments, we have observed that the y-coordinate for coarse shading is given preference over the x-coordinate. In the above example, 2x MSAA and a 4x4 shading rate result in a 2x4 effective shading rate. An interesting side effect is that shading rates with a larger x-coordinate get reduced, too. For example, the shading rate 4x2 gets reduced to 2x2 as it is depicted in Figure 3.9. This is not compliant with the specification, as a fragment of size 4x2 needs the same amount of coverage samples as a fragment of size 2x4, and the latter is possible with the same configuration. Reducing the MSAA samples to one again enables the shading rate of 4x2. This preference of the y-coordinate can be observed for the 2x1 and 1x2 shading rates too, depicted in Figure 3.11. With 8x MSAA samples, every coarse fragment size larger than 1x2 gets reduced to 1x2, except the fragment size of 2x1, which gets reduced to 1x1.

In Figures 3.8, 3.9, 3.10 and 3.11 this limitation is visualized. For these images, the same shading-rate image, depicted in Figure 3.7, and the same shading rate palette are used. Only the MSAA sample count is altered between the values: 1x MSAA samples (disabling MSAA), 2x MSAA samples, 4x MSAA samples, and 8x MSAA samples.

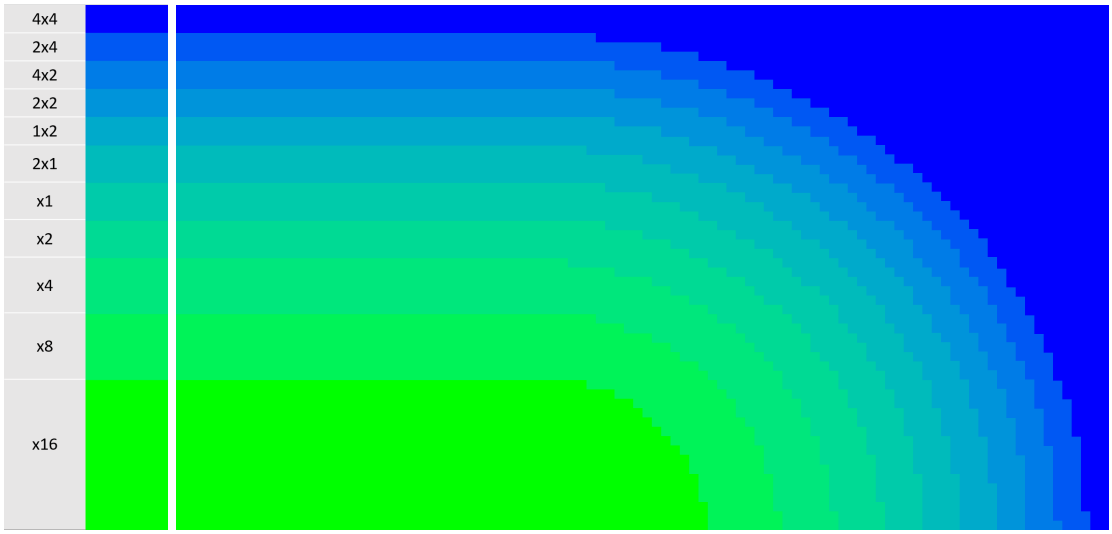


Figure 3.7: The MSAA sample count as a limitation: The shading-rate image with the corresponding palette values.

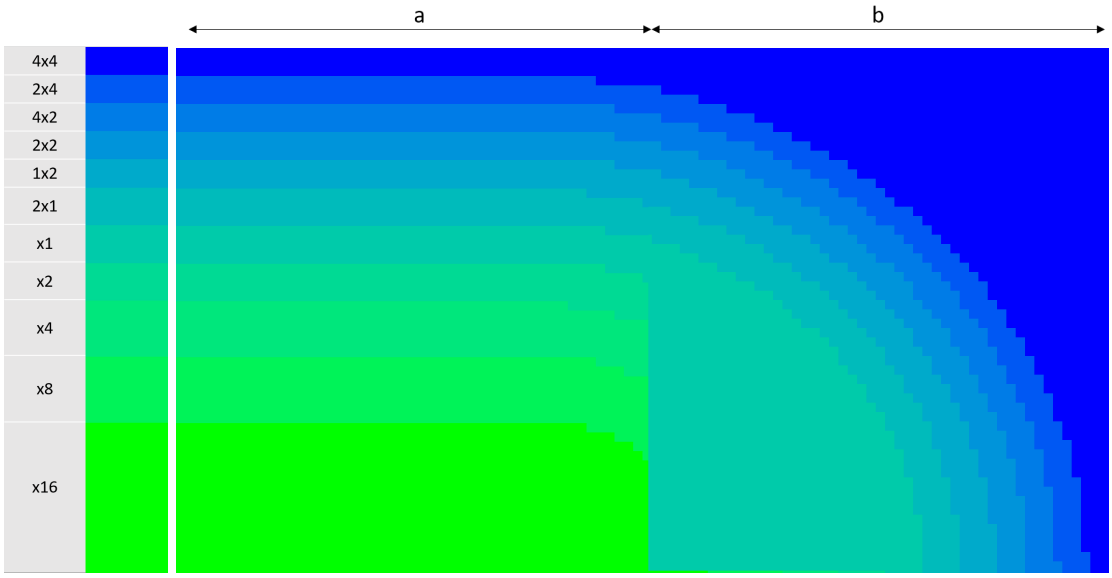


Figure 3.8: The MSAA sample count as a limitation: MSAA is disabled for this image. The shading rate image is distinguished as the a-section, the effective shading rate by the b-section. The full shading-rate image can be viewed in Figure 3.7. This image depicts that all coarse shading rates are possible but no supersampling shading rates can be used with no MSAA samples.

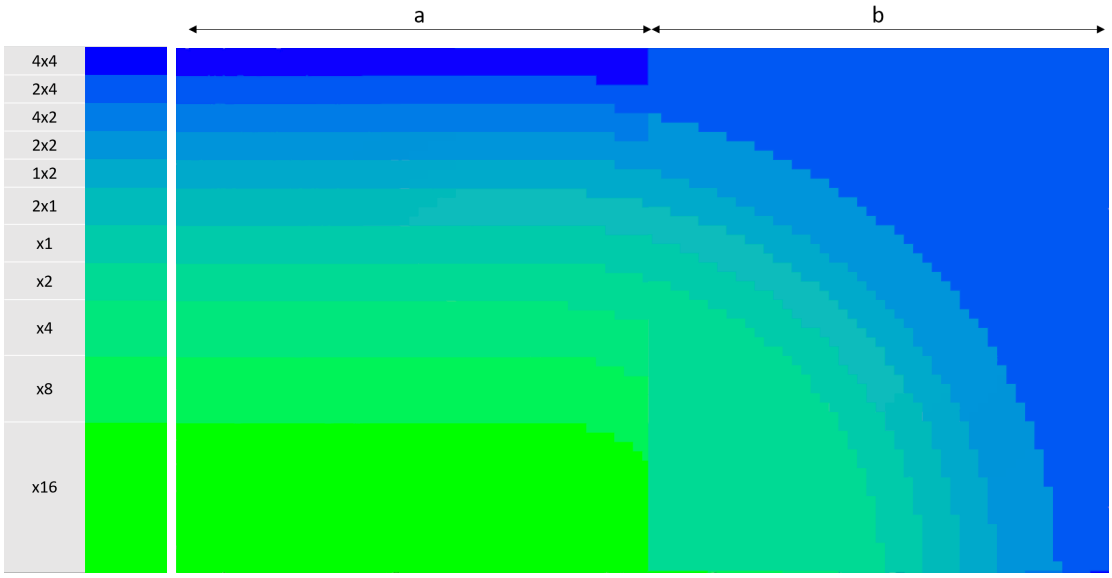


Figure 3.9: The MSAA sample count as a limitation: 2x MSAA samples are configured for this image. The shading-rate image is distinguished as the a-section, the effective shading rate by the b-section. The full shading-rate image can be viewed in Figure 3.7. With 2x MSAA samples x2 supersampling is possible. This image also shows how the coarse pixel size is reduced to the maximum possible with a preference for the y-coordinate. 4x2 coarse shading is not possible and reduced to 2x2 while 4x4 and 2x4 are both effectively have the 2x4 coarse pixel size.

MSAA Samples	Coarse Shading Rates	Supersampling Shading Rates
1x	1x2, 2x1, 2x2, 2x4, 4x2, 4x4	-
2x	1x2, 2x1, 2x2, 2x4	2x
4x	1x2, 2x1, 2x2	2x, 4x
8x	1x2	2x, 4x, 8x

Table 3.1: Possible shading rates per MSAA sample configuration.

Only a subset of the available shading rates can be used based on the configured MSAA level. By disabling MSAA, supersampling shading rates cannot be used but all coarse shading rates can be utilized. On the other hand, with 8x MSAA, only the 1x2 coarse shading rate can be used, but up to 8x supersampling shading rates are enabled. All intermediate cases aside from 1x and 8x MSAA samples function as discussed above. An overview of all possible configurations is given in Table 3.1.

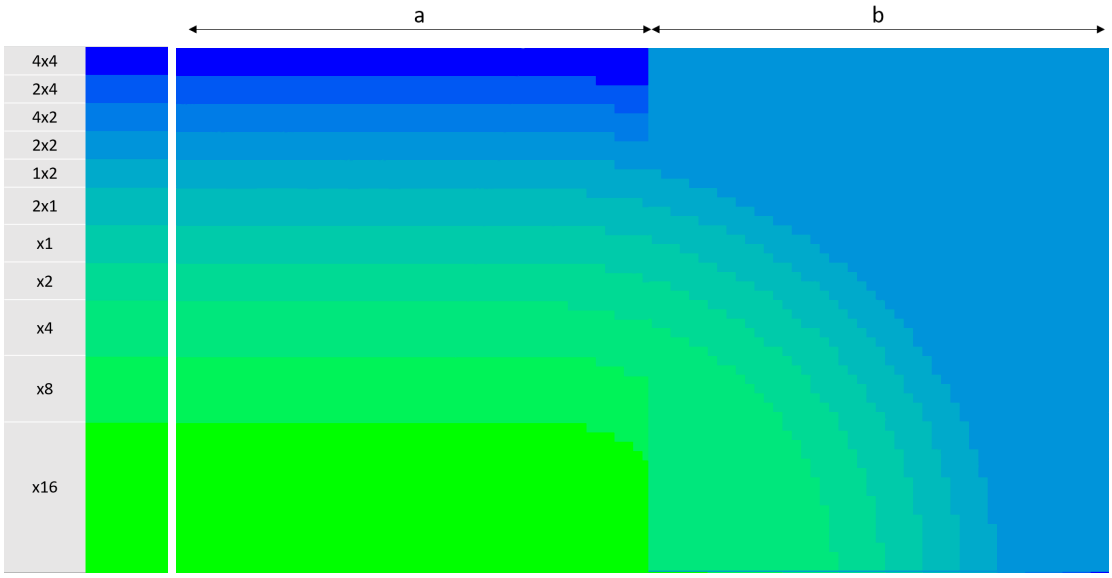


Figure 3.10: The MSAA sample count as a limitation: 4x MSAA samples are configured for this image. The shading-rate image is distinguished as the a-section, the effective shading rate by the b-section. The full shading-rate image can be viewed in Figure 3.7. With 4x MSAA samples up to x4 supersampling is possible. Coarse shading is limited to 2x2 coarse pixel sizes.

3.3 Integration with a Rendering Engine

Aside from additional properties, needed for the computation of the shading rate, the integration of VRS with an existing rendering engine is straightforward. The following steps are necessary:

1. Enable the VRS extension in the graphics API.
2. Configure a shading rate palette for each existing pipeline where VRS should be used.
3. Create an image with usage flags for a shading-rate image. This image has to be a one-component integer image.
4. Fill this image, for example, with a compute shader.
5. Set the image before executing a draw command which should use VRS.

These steps are based on the steps necessary for the Vulkan graphics API. However, they were abstracted and can be used as a guide for OpenGL and Direct3D, too. During the

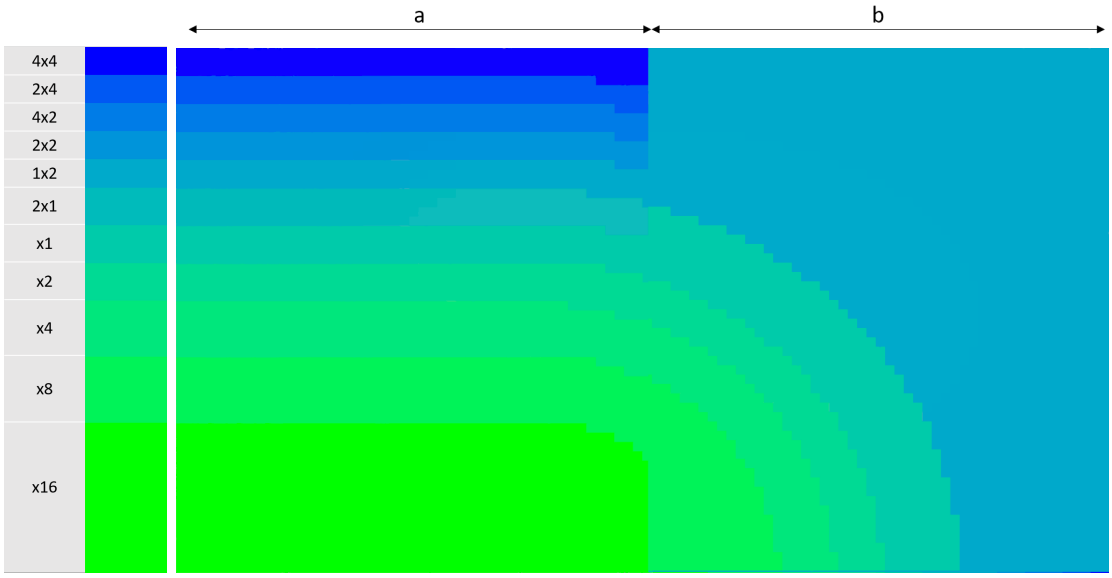


Figure 3.11: The MSAA sample count as a limitation: 8x MSAA samples are configured for this image. The shading-rate image is distinguished as the a-section, the effective shading rate by the b-section. The full shading-rate image can be viewed in Figure 3.7. With 8x MSAA samples up to x8 supersampling is possible. This image again shows the preference for the y-coordinate during hardware adaptation of the effective shading rate. 2x1 coarse shading is not possible and reduced to 1x1 while all other coarse shading rates are reduced to 1x2.

initialization phase, the only structures which need to be changed are the pipelines. All other steps are additional steps that can be integrated easily. For the run-time phase, there are only two additional steps needed: computing and setting the shading-rate image. If additional properties are needed to compute the shading-rate image, steps to compute these have to be integrated into the existing pipeline, e.g., the shaders, attachments, etc..

3.4 Hardware Support and Usage Scenarios

At the time of writing, VRS is only supported by hardware with NVIDIA's Turing architecture. However, AMD already filed a patent by Saleh et al. [SBPW19] that describes a very similar technique to VRS. In the meantime, Microsoft has standardized for DirectX 12 a version of VRS with an additional control mechanism for the shading rate: per-primitive [1].

[1] https://devblogs.microsoft.com/directx/variable-rate-shading-a-scalpel-in-a-world-of-sledgehammers/

Many real-time computer graphics applications can potentially profit from the reduced fragment shader load when using VRS. Yang et al. [YZK+19], for example, used it successfully to improve the rendering performance of the game *Wolfenstein II: The New Colossus* and Epic Game's *Infiltrator* demo in the *Unreal Engine 4* by 5% to 20%. Virtual reality applications, with their high-resolution head-mounted displays and high refresh-rate targets, could benefit from VRS even more, e.g., by foveated rendering, which potentially allows reducing the shading resolution additionally in the regions of a user's peripheral vision [PSK+16]. For advanced foveated rendering, additional hardware, namely an eye tracker, is needed. Eye trackers enable us to follow the focus of the user and adapt the scene resolution based on this data. This is an advancement compared to early foveated rendering methods, which assume the focus of the eye at the center of the screen. This assumption may often be correct, especially depending on the type of application, e.g., for ego-shooters, where the user can be expected to look to the crosshair at the center of the screen most of the time. However, it is an assumption that might be wrong as well, and when it is wrong, it breaks the immersion due to visible aliasing artefacts at the corners. By utilizing eye trackers, we are able to deliver a continuous adaptation of the resolution to the focus of the viewer. In the last decade, eye trackers have evolved from scientific gadgets to full-fledged devices appropriate for the mass market. As such, these devices are affordable and deliver accurate data. For this work, we used the TOBII eye-tracker 4C, which captures and provides the focus of the eye at a rate of 90 Hz, for details see the specification [Tob]. The licence of this eye tracker only permits "interaction us", while analytical use is prohibited. Our use case is to adapt the shading rate based on the current user focus, e.g., the user interaction.

CHAPTER 4

Techniques and Algorithms for Variable Rate Shading

In this chapter the algorithms that were developed in the context of this thesis are described and how they can be integrated into existing rendering engines. First, our algorithms for the three strategies proposed in the NVIDIA Turing architecture whitepaper [NVI18], Content-Adaptive Shading, Motion-Adaptive Shading and foveated rendering, discussed in Chapter 3, are described. For Content-Adaptive Shading, we present two algorithms, one based on texel differentials and one on edges. In Chapter 4.1, we will describe temporal artefacts, which arise for many situations when using VRS. To counter these artefacts, a method that generally reduces these artefacts by stabilizing the shading rate over time is proposed. For Motion-Adaptive Shading and foveated rendering, we present concrete implementations and discuss which details are important to get useful results with these techniques. Subsequently, we present our novel approach, TAA-Adaptive Shading.

4.1 Content-Adaptive Shading

The target of Content-Adaptive Shading is to adapt the shading rate to the content of the scene. This means the content of the scene needs to be analyzed in order to find regions where it is appropriate to lower the shading rate and regions where a higher shading rate is required to preserve more visual details. The scene's content offers many properties. There are primary properties, which are available due to the need for shading, e.g., depth, normals or the final color, and secondary properties which are derived from the primary properties, e.g., edges. Choosing suitable properties for a meaningful shading rate selection can be challenging. Primary properties could be directly used to control the shading rate. For example, color offers direct information of brighter or darker spots. Under the assumption that brighter spots have more often the focus of the viewer and

offer more visual details, for example on rough surfaces where light introduces higher frequencies due to the angle, the shading rate of darker spots could be lowered. However, this assumption might not hold for many scenes. Another example would be to directly use the depth information. Objects further away often have lower resolution and could, therefore, be shaded with a lower shading rate. However, in real scenes, objects in the distance often are objects with high-frequency surfaces just displayed smaller. Blur effects like DOF get the high-frequency image and do correct downsampling. With VRS, the signal is already sampled at a lower resolution. If high-frequency signals are sampled too low, the result is not a proper blur but aliasing.

Our Content-Adaptive Shading algorithm strives to be generally applicable to many scenes while not introducing unnecessary aliasing. For this reason, we developed two approaches based on secondary properties:

- Edges
- Texel differentials

While the two algorithms differ in details, like properties and how the shading rate is computed, the base algorithm is very similar and shown in Figure 4.1. At the beginning of the render loop, the shading-rate image is cleared. This is necessary to guarantee a default shading rate for all regions not covered by the shading-rate image computation. In the next step, we downscale the properties image computed in the previous iteration to $\frac{1}{16}(w, h)$, where w and h denote the width and height of the image, to derive a shading rate from the computed properties image. Downscaling uses bilinear interpolation, averaging each of the 16x16 tiles and its neighbourhood. Downscaling is necessary as the shading rate is specified for 16x16 tiles and an average of the properties can be suitable to describe the region. How this downscaling affects the individual properties is discussed for each property. In a subsequent step, the computation of the shading rate based on the downscaled properties image is performed. In a final step, the scene is drawn and the required properties are computed. Depending on the type of properties, the computation can be performed during the drawing of the scene, e.g., texel differentials, or as a post-processing step, e.g., edges.

A high-level overview of the common steps shared by all our shading-rate computation algorithms, is shown in Figure 4.2. First, our algorithm samples the properties image and derives the shading rate from it. The subsequent steps are to compute the coordinates in the next frame by a forward projection and sample the previous shading rate at the new coordinates for our shading-rate stabilization algorithm. Finally, the computed shading rate is stored at the new coordinates.

The result of the shading-rate computation is used as an index into the shading-rate palette. Our configuration for the shading-rate palette consists of eleven different values in increasing order, where index zero denotes a coarse shading rate of 4x4 and index ten a supersampling shading rate of x8. This configuration allows a simple adaptation of

Figure 4.1: The basic steps of our algorithm for Content-Adaptive Shading. These steps are executed inside the render loop in each iteration. At the start, the shading-rate image is cleared. Then, the image containing the properties computed during the previous iteration is downscaled and used to compute the shading rate for each 16x16 pixel region. Finally, the scene is drawn using the computed shading-rate image and the properties for the next iteration are computed.

the MSAA sample count without the need to change anything else in the setup. VRS automatically reduces the sample count to the maximum possible if a higher shading rate is specified than there are MSAA samples configured. The same applies if there are not enough coverage samples for a coarse pixel, the coarse pixel size is reduced automatically. For more details about the possible configurations, refer to Chapter 3. Table 3.1 shows all the configurations that are possible on a NVIDIA Turing RTX 2080 Ti.

In order to compute the shading-rate image, we would already need the properties of the frame before it is drawn. Since we do not have them for the current frame, we try to derive them from the previous frame. By forward projecting the screen position based on the motion of the camera, we base the new shading rate upon the data from the previous frame. The motion of the camera from the last to the current frame is usually available and therefore the position in the current frame can be computed in the same way as a backward-projection, for example during TAA. During a backward-projection,

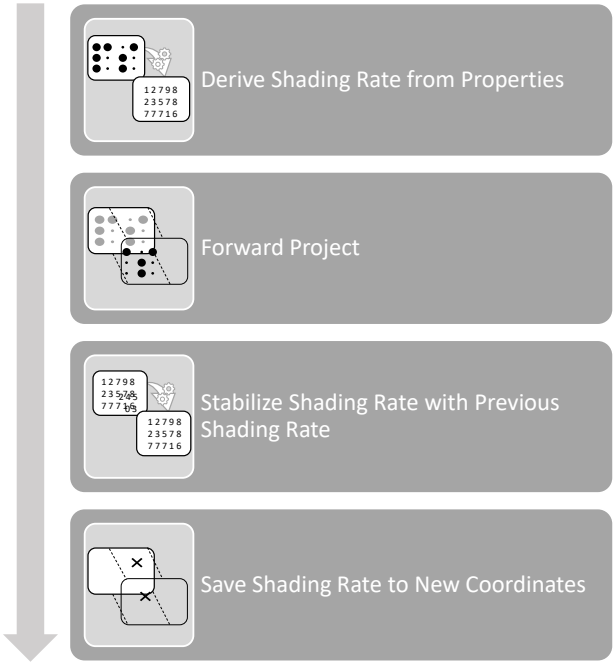


Figure 4.2: The common steps for the computation of the shading-rate image. First, the shading rate is derived from the properties. This process depends on the used properties. In the next step, the coordinates in the next frame are computed by a forward projection. Finally, the shading rate is stabilized by our TAA-based approach and saved to the new coordinates.

the screen-space coordinates of the previous frame are computed. These coordinates can be used to sample the corresponding data from the previous frame's framebuffer. The data can then be combined with the current frame data and written at the current screen-space coordinates.

For a forward projection, the process to compute the new screen-space coordinates is similar: First, the world position of the previous frame is computed using its depth and x,y screen-space coordinates. Then, the computed world position is used to compute the current screen-space coordinates. However, there is a big difference in those two processes: For a backward projection, this process yields a value for each pixel in the image. This value might be wrong, as the current position might be occluded or not inside in the last frame and therefore handled by texture border strategies. For a forward projection, the coordinates written to are computed inside the process. This computation has two problems, first, some target coordinates might not be written at all and second, others might be written multiple times. To address the first problem, we clear the shading-rate image with a default shading rate before computing the new values as it happens in the first step of the base algorithm shown in Figure 4.1.

The second problem requires further investigation. Multiple points fall into one point

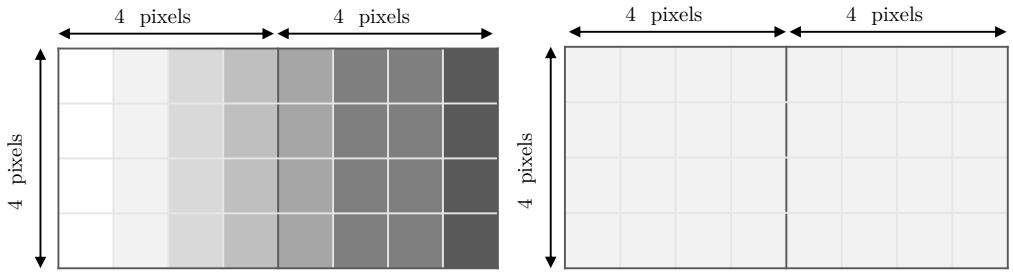
inside the next frame because they are projected into it. For example, the camera has zoomed out, four points further away may fall into a single one. If this case is not considered, the value written to the new coordinate would be one of the four, the last one written by the shader. A more correct way would be to average all of the points falling into this single point. However, as the values of the shading rate are already based on properties derived from an average of 16x16 pixels, there is not much difference between neighbouring values and the last written value is sufficient for all cases except if two neighbouring tiles are sampled. These cases were neglected to avoid expensive synchronizations inside shaders during the computation of the shading rate, which would become necessary to compute the average of multiple shader executions.

For animated objects, additional computations can become necessary to track their positions in screen space. Each object may be identified in screen space to apply the correct model matrices during a forward projection. This can become bothersome with many objects, and pipeline changes would probably become necessary to identify the objects in screen space.

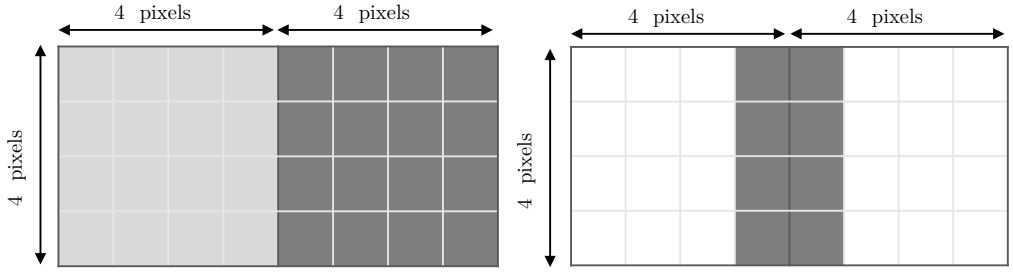
One challenge of basing Content-Adaptive Shading on secondary properties directly derived from the rendered frame is inherent in the technique itself: the shading rate might change the final color in the rendered frame and therefore the computed properties as a direct consequence. That means, the next frame's shading rate is a function of the previous frame and indirectly of the shading rate from the previous frame. This may result in the flipping of shading rates.

An example for shading rate flipping is shown in 4.3. Edges are used as properties for the example. Two neighbouring tiles of size 4x4 pixels initially shaded at full resolution have a low difference in color. The other 14 4x4 pixel tiles are homogeneous. For this reason, the shading rate is lowered to 4x4 for the complete 16x16 tile. Now, the difference between the two aforementioned 4x4 pixel tiles is suddenly slightly above the threshold, an edge is detected. This results in an increase in the shading rate for the whole tile and we get an oscillating behaviour. The visual result of this artefact is a floating or flickering texture.

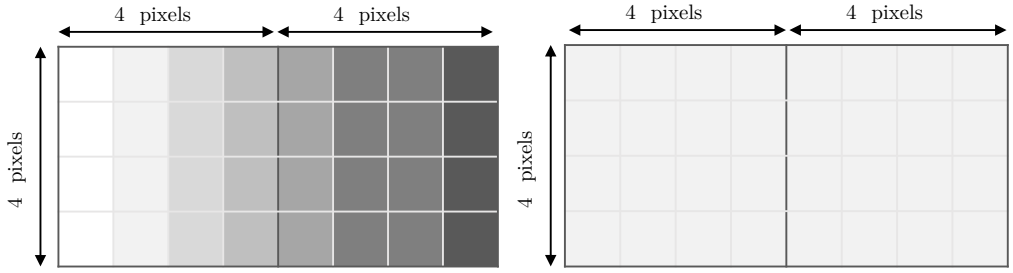
We tried various approaches to compensate for this behavior and to stabilize the shading rate. In one of our approaches, we reduced the resolution of the rendered scene image before the computation of the properties takes place. This approach achieved a stable shading rate, but at the cost of quality. Properties get lost and, therefore, undersampled with a low shading rate. In order to reduce the effect of the shading-rate flips, we applied TAA. TAA utilizes information from previous frames and therefore alleviates the flickering of the parts were the shading-rate flips. However, the effect is only reduced and the textures seem to slowly float. TAA was introduced by Karis [Kar14] in order to reduce spatial as well as temporal aliasing. The human visual perception is especially sensitive to motion like it is perceived if there is temporal incoherence, e.g., the same object in a scene changes color over time, or some small parts change color like it is the case if the shading rate oscillates.



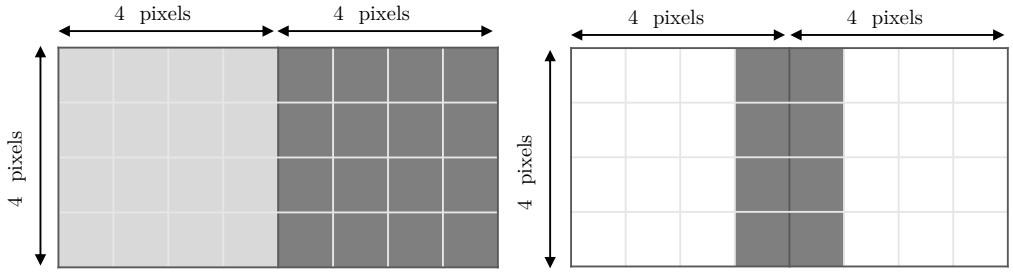
(a) With the full resolution at the beginning (left) the edge image (right) shows no edges.



(b) The shading rate is reduced based on the previous edge image and a bigger color difference arises (left). The new edge image (right) shows clearly an edge.



(c) This edge then leads to an increase in the shading rate. We get again a low local change in color. No edge is detected.



(d) This again leads to a decrease in the shading rate. Again, an edge is detected. We observe an oscillating behavior.

Figure 4.3: Changes in the shading rate lead to changes in the color output and therefore to flipping shading rates.

Our novel approach to mitigate this oscillation is based on TAA. We apply a modified version of TAA on the shading-rate image itself. We know from the previous shading-rate image that a particular region was sampled at a high rate. We can conclude that the property in this region indicates a high rate, as it was detected in the previous frame. Therefore, we can sample it again at a high rate. In contrast to color values, shading rates are discrete numbers and there are only ten different shading rates, from which a maximum of six can be used at the same time, depending on the MSAA configuration, for details see Chapter 3. Taking the rounded average of such small discrete numbers does not lead to the desired shading rate in most cases, e.g., a weighted average of both shading rates. For example, there is no shading rate between 2x2 and 2x4. It is rounded to one of them and the previous or the current shading rate wins. If the previous shading rate was high, our assumption is that there must have been a property indicating a high rate. We take the maximum of the previous and the current shading rate and are sure to miss no property indicating a high rate. The problem is, if once such a property was seen, a high shading rate will be used for this tile and if the camera moves, the shading rate stays the same: The result is shading-rate ghosting.

In TAA, ghosting is mostly resolved by clipping the color values of the previous frame to locally averaged color values of the current frame, details are described in Chapter 2. Similarly to TAA clipping, we clamp the shading rate value of the previous frame to the current shading rate value before computing the maximum of both. A fast reduction of the shading rate may introduce motion due to flipping colors. To reduce this effect, we based the clamping distance on the current shading rate and favor higher shading rates. For example, if we have eleven values in our shading-rate table with ascending shading rates, the current shading rate is the number six in the table, the clamping region will be $-1 + 6 < x < 6 + 2$. This enables a slower reduction of the shading rate and therefore a slower change in the scene, e.g., there is less motion if a property indicating a high shading rate diminishes and the viewer does not recognize the reduction in resolution.

In the next two subsections, the properties we selected to control shading rate, are described and the selection is reasoned. Subsequently, the algorithms are discussed in a step-by-step fashion with supporting pseudo-code.

4.1.1 Edges

Edges represent frequency changes and are therefore well suited to divide the scene into homogeneous and inhomogeneous regions. Edges enable us to sample each part of the scene at the needed frequency. By detecting sharp discontinuities in the depth, object borders can be distinguished. This allows a lower sampling of inner object parts while preserving the contrast between objects. On the other hand, object borders could be sampled at a higher rate. This would be an advancement of MSAA where only the visibility is sampled at a higher rate at object borders. With our approach, not only the visibility but also the shading is supersampled. Furthermore, computing edges of the final image color enables us to detect high frequencies due to texture, normal mapping or any other effects introduced to the scene. We have decided to use the same edge

detection as presented by the post-processing anti-aliasing technique FXAA developed by Lotte/NVIDIA [Lot11] because it is tailored for real-time applications in terms of performance and quality.

Algorithm 4.1 gives a high-level view of the steps which are necessary on the CPU part. The edge image is initialized with white on line 1, which is equivalent to an image consisting only of edges, so the first frame is not coarse-shaded. On line 9, the edge image is computed based on the final color image after all post-processing effects. To derive a shading rate from the computed edge image, on line 4 we downscale the edge image to $\frac{1}{16}(w, h)$, where w and h denote the width and height of the image. As discussed before, downscaling uses bilinear interpolation, averaging each of the 16x16 tiles and its neighbourhood. The result is that regions with more edges produce higher values and regions with fewer edges produce lower values. As the number of edges is directly correlated with the frequency, the shading rate is increased the more edges are detected. In the next step on line 5, the shading-rate image is computed with a compute shader and on line 6 set for subsequent draw calls like on line 7.

The pseudo-code of the compute shader for calculating the shading rate is presented in Algorithm 4.2. On line 2 of Algorithm 4.2, the forward projection used for computing the coordinates in the next frame begins. The world position of the previous frame is computed by using the depth and x,y screen-space coordinates received as input in the compute shader. In line 3, the world position to compute the current screen-space coordinates is used. After computing the target coordinates, the next step is to compute the shading rate. On line 5, the computation of the index in the aforementioned shading-rate palette starts. First, the values of the downscaled edge image are multiplied with the highest index in the shading-rate palette, e.g., ten in our basic configuration. The next two steps on lines 6 and 7 are our shading-rate stabilization technique, based on TAA. So, on line 6 of Algorithm 4.2, the shading-rate value of the previous frame is clamped to the current shading-rate value before computing the maximum of both on line 7.

4.1.2 Texel Differentials

Our second approach for computing the shading rate on secondary properties of the content of the scene utilizes texel differentials. Texel differentials are the partial derivatives in x and y direction of the texture space, e.g., the UV coordinates. This technique is more invasive, as it is not possible to compute texel differentials in a post-processing step, and existing fragment shaders have to be adapted to store the required values. However, texel differentials are very valuable. They convey the information of the used mipmap-level of a texture. Larger values of texel differentials result in higher mipmap levels being selected. Depending on the screen resolution and the texture resolution, with very large texel differentials whole textures fall into a single pixel. Hence, the shading rate can be reduced for regions with high texel differentials. Very small texel differentials mean a texture is oversampled and the shading rate can therefore again be reduced. Texel differential values between these two maxima can be sampled at the usual shading rate

Algorithm 4.1: Render Engine: Edge-based Content-Adaptive Shading

Input: Color image used for rendering the scene $\mathbf{C}$, depth image $\mathbf{D}$, edge image $\mathbf{E}$, edge image scaled down $\mathbf{E_s}$, shading-rate image $\mathbf{S}$, motion vector image $\mathbf{M}$

```
1  E ⇐ clear image with white;
2  while renderLoop do
3      S ⇐ clear image to default shading rate;
4      E_s ⇐ scale down E to 1/16 resolution;
5      S ⇐ computeShadingRateImage(E_s, S); // see Algorithm 4.2
6      set S for draw calls;
7      C, D, M ⇐ draw scene;
8      C ⇐ apply temporal anti-aliasing to C using M;
9      E ⇐ find edges in C;
10 end
```

Algorithm 4.2: computeShadingRateImage: Edge based Compute Shader

Input: edge image scaled down $\mathbf{E_s}$, shading-rate image $\mathbf{S}$, inverse projection matrix $\mathbf{P^{-1}_{prev}}$, inverse view matrix of previous frame $\mathbf{V^{-1}_{prev}}$, view projection matrix of next frame $\mathbf{V}, \mathbf{P}$

Output: shading-rate image $\mathbf{S}$

```
1  // forward projection: compute new NDC coordinates
2  vec3 pos_w ⇐ back project x, y, D[x, y] with P^-1_prev, V^-1_prev ;
3  x_c, y_c ⇐ forward project pos_w with V, P;
4  // compute shading rate
5  int sr ⇐ floor(E_s[x, y] * 11);
6  int prevSr ⇐ clamp(S[x_c, y_c], sr − 1, sr + 2);
7  sr ⇐ max(sr, prevSr);
8  S[x_c, y_c] ⇐ sr;
```

or higher shading rates. As with edge-based Content-Adaptive Shading, we use a shading rate palette with ascending values, the higher the index the higher the shading rate.

Algorithm 4.3 gives a high-level view of the steps which are necessary on the CPU part. These steps are similar as in Algorithm 4.1, with the difference that there is no edge image but a texel differential image. This texel differential image is computed during the drawing of the scene at line 7. The computation of this additional property during the draw is presented in Algorithm 4.5. We again scale the texel differential image down by bilinear interpolation at line 4 of Algorithm 4.3 to get the average texel differential for the region.

In Algorithm 4.4 the pseudo-code of the compute shader that is used to calculate the shading rate is presented. On lines 2 and 3, a forward projection to receive the new

position in the current frame is performed, as discussed previously. With the instructions on lines 5 and 6, the shading rate is computed. As discussed at the beginning of this section, high and low values of texel differentials should receive low shading rates. For this reason, we transform the texel differential on line 5. First, we shift the value from the range $[0, 1]$ to the range $[-1, 1]$. Next, we use a quadratic function so that originally low and high values are both high. Finally, we subtract the result from 1, with the result that low and high values now both have low values and therefore get low shading rates assigned. Intermediate values will receive intermediate to high shading rates. On line 6 the value is then again scaled with the highest index in the shading rate palette to receive the final shading rate index.

Content-Adaptive Shading based on texel differentials suffers, similarly as the method based on edges, from temporal artefacts introduced by unstable shading rates. Texel differentials get bigger if the shading rate is reduced and smaller if it is increased. At border cases, the shading rate oscillates. As texel differentials directly correlate with the shading rate they can be scaled by it as we do in Algorithm 4.5 on line 4. So, the texel differentials are additionally scaled by the inverse fragment size for coarse shading or multiplied by the sample count for supersampling. This helps to reduce the flipping in addition to our TAA-based approach for the shading-rate image.

Algorithm 4.3: Render Engine: Texel Differential-based Content-Adaptive Shading

Input: Color image used for rendering the scene $\mathbf{C}$, depth image $\mathbf{D}$, texel differential image $\mathbf{T}$, texel differential image scaled down $\mathbf{T_s}$, shading-rate image $\mathbf{S}$, motion vector image $\mathbf{M}$

1 $\mathbf{T} \Leftarrow$ clear image with white;
2 **while** *renderLoop* **do**
3 $\quad \mathbf{S} \Leftarrow$ clear image to default shading rate;
4 $\quad \mathbf{T_s} \Leftarrow$ scale down $\mathbf{T}$ to $\frac{1}{16}resolution$;
5 $\quad \mathbf{S} \Leftarrow computeShadingRateImage(\mathbf{T_s}, \mathbf{S})$; // see Algorithm 4.4
6 $\quad$ set $\mathbf{S}$ for draw calls;
7 $\quad \mathbf{C}, \mathbf{D}, \mathbf{M}, \mathbf{T} \Leftarrow$ draw scene; // see Algorithm 4.5
8 $\quad \mathbf{C} \Leftarrow$ apply temporal anti-aliasing to $\mathbf{C}$ using $\mathbf{M}$;
9 **end**

4.2 Motion-Adaptive Shading

In Chapter 3 we discussed that Motion-Adaptive Shading utilizes the fact that objects in motion appear blurry to the human visual system. Rendering them at lower resolution should have little to no impact on their appearance, as perceived by the human visual system.

The base algorithm, shown in Figure 4.4, is very similar to Content-Adaptive Shading as

Algorithm 4.4: computeShadingRateImage: Texel Differential based Compute Shader

Input: texel differential image scaled down $\mathbf{T_s}$, shading-rate image $\mathbf{S}$, inverse projection matrix $\mathbf{P^{-1}_{prev}}$, inverse view matrix of previous frame $\mathbf{V^{-1}_{prev}}$, view projection matrix of next frame $\mathbf{V}, \mathbf{P}$

Output: shading-rate image $\mathbf{S}$

```
// forward projection: compute new NDC coordinates
vec3 pos_w ⇐ back project x, y, D[x, y] with P^-1_prev, V^-1_prev ;
x_c, y_c ⇐ forward project pos_w with V, P;
// compute shading rate
float td ⇐ 1 − (T_s[x, y] ∗ 2 − 1)^2;
int sr ⇐ floor(td ∗ 11);
int prevSr ⇐ clamp(S[x_c, y_c], sr − 1, sr + 2);
sr ⇐ max(sr, prevSr);
S[x_c, y_c] ⇐ sr;
```

Algorithm 4.5: draw scene: additional properties Texel Differentials

Input: resources used for shading, UV coordinates **uv**, effective shading rate **sr** (sr.x/sr.y = fragment size, sr.z = sample count)

Output: Color image $\mathbf{C}$, depth image $\mathbf{D}$, motion vector image $\mathbf{M}$, texel differentials image **TD/td**

```
// light/shade scene, compute motion vectors
...
// compute texel differentials
td = fwdith(uv/sr.xy) ∗ sr.z;
```

discussed in Section 4.1 and shown in Figure 4.1. The main difference is the computation of the properties. For Motion-Adaptive Shading, regions in motion need to be detected. The straightforward approach would be to directly use the extents and directions from the motion vectors, e.g., computed during motion blur or TAA, which point from the previous to the current frame, to derive the shading rate for the next frame.

This assumes that the motion at a point leading from the previous frame to the current frame is equal to the motion leading from the current frame to the next frame at the same point. This assumption of a homogeneous motion can be true, e.g., for a homogeneously forward-moving camera, or wrong, e.g., for abrupt camera stops or objects changing motion direction. As motion is a process usually over multiple frames, the basic idea of our algorithm is to utilize these motion vectors and combine them with new motion vectors, pointing from the current frame to the next frame. The new motion vectors can be computed during the shading-rate computation itself, as the coordinates of the next frame are already computed by the forward projection to receive the target coordinates for writing the shading rate. This leads to our Motion-Adaptive Shading algorithm,

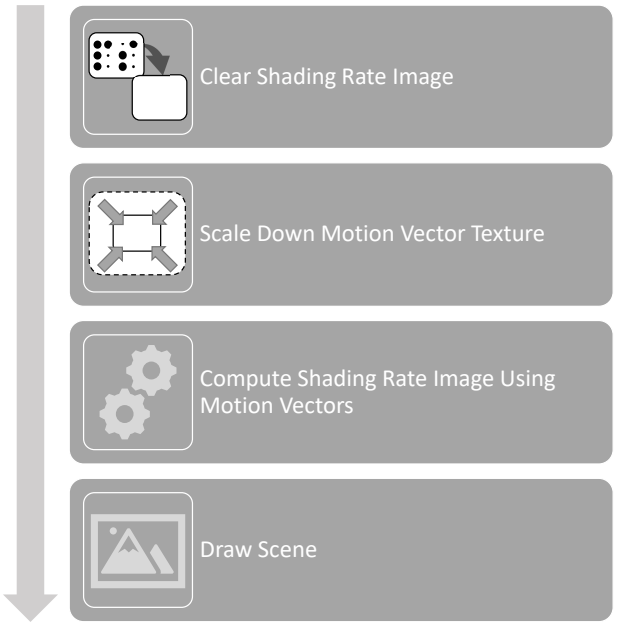


Figure 4.4: The basic steps of our algorithm for Motion-Adaptive Shading. These steps are executed in each iteration of the render loop. In the beginning, the shading-rate image is cleared. Then, the motion vector image is averaged for each tile and used to compute the shading rate. Finally, the scene is drawn using the computed shading-rate image. The motion vectors are computed by other effects, like motion blur or TAA.

utilizing motion paths to grasp the real motion of the scene. However, what we cannot do is compute the motion from the next frame to the frame after the next because we only have the camera matrices for the next frame. Additionally, this approach does not work with independently moving objects and animations. For this kind of objects, identification of objects and forward projection per object is needed as discussed in Section 4.1.

The CPU steps of our Motion-Adaptive Shading are shown in Algorithm 4.6. In contrast to the CPU steps of the previous Algorithms 4.1, 4.3, we do not need to compute additional properties, as we reuse the motion vectors from TAA. The motion-vector image still has to be scaled down on line 4 in order to use the average motion for the computation of the shading rate. Additionally, on line 1, we need to clear the motion-vector image to a zero motion as it is now used before it would be written. As the forward projection does not necessarily cover all pixels, we have to set a default shading rate for uncovered pixels on line 3 of Algorithm 4.6.

In Algorithm 4.7, under the assumption of a homogeneous motion, we forward project as discussed in Section 4.1 on the lines 2 and 3 and sample the motion pointing from the previous to the current frame on line 5. For homogeneous motion, we would already get the correct results. As we do a forward projection, we can also compute the new motion vectors leading to the new location and utilize them for specifying the shading rate, this

is done on line 6 in Algorithm 4.7. On line 7, we now combine the lengths of the previous motion vector and the new one and so integrate over the motion of the last two frames leading to the new position. Afterwards, we subtract the result from 1, because larger distances mean more motion and hence should result in a lower shading rate. Finally, the shading rate is again computed by upscaling the value with the maximum index of the shading rate palette to receive a shading rate index.

Algorithm 4.6: Render Engine: Motion-Adaptive Shading

Input: Color image used for rendering the scene $\mathbf{C}$, depth image $\mathbf{D}$, shading-rate image $\mathbf{S}$, motion vector image $\mathbf{M}$, motion image scaled down $\mathbf{M_s}$

```
1 M ⇐ clear image with zero, e.g., no motion;
2 while renderLoop do
3 |   S ⇐ clear image to default shading rate;
4 |   M_s ⇐ scale down M to 1/16 resolution;
5 |   S ⇐ computeShadingRateImage(M_s, S); // see Algorithm 4.7
6 |   set S for draw calls;
7 |   C, D, M ⇐ draw scene;
8 |   C ⇐ apply temporal anti-aliasing to C using M;
9 end
```

Algorithm 4.7: computeShadingRateImage: Motion based Compute Shader

Input: motion image scaled down $\mathbf{M_s}$, shading-rate image $\mathbf{S}$, inverse projection matrix $\mathbf{P^{-1}_{prev}}$, inverse view matrix of previous frame $\mathbf{V^{-1}_{prev}}$, view projection matrix of next frame $\mathbf{V}, \mathbf{P}$

Output: shading-rate image $\mathbf{S}$

```
1  // forward projection: compute new NDC coordinates
2  vec3 pos_w ⇐ back project x, y, D[x, y] with P^-1_prev, V^-1_prev ;
3  x_c, y_c ⇐ forward project pos_w with V, P;
4  // compute shading rate
5  float m_p ⇐ length(M_s[x, y]);
6  float m_c ⇐ length(x_c − x, y_c − y);
7  float m ⇐ 1 − avg(m_c, m_p);
8  int sr ⇐ floor(m ∗ 11);
9  int prevSr ⇐ clamp(S[x_c, y_c], sr − 1, sr + 2);
10 sr ⇐ max(sr, prevSr);
11 S[x_c, y_c] ⇐ sr;
```

4.3 Eye-Tracking and Foveated Rendering

As discussed in Chapter 3, eye trackers are well researched and working devices. Eye tracking is a form of user input and should therefore not interfere with rendering at

all. For this reason, we introduce an interface where the application can fetch the last seen eye position as a two-dimensional vector with values in a range between [0,1] for the x and y coordinate. The internals of the eye tracking run in a parallel thread, only updating the data returned by the interface with a single atomic operation. Querying the last eye data runs almost without blocking, e.g., the application receives the last known position or awaits a single atomic address copy and then receives the last known position.

For the integration of foveated rendering with VRS, we focused on the part of how the shading-rate image is generated. The shading-rate image basically stays the same and only a different viewport is needed to shift the center around. An example of such a shading-rate image can be viewed in Figure 4.5. In Algorithm 4.8 the steps of our foveated rendering are shown. The base shading-rate image computed at line 1 is double the size in width and height and can look like Figure 4.5 without the overlay highlighting the excerpts. Before the draw calls are submitted on line 6, at line 4 the right excerpt of the shading-rate image is copied into the actual shading-rate image and this image is then set on line 5. As such copy processes are quite fast on modern hardware there is not much overhead. In order to define the excerpt, as the required width and height are known, only the offsets need to be computed. The offset is exactly the width and height times the position of the eye in the range [0,1].

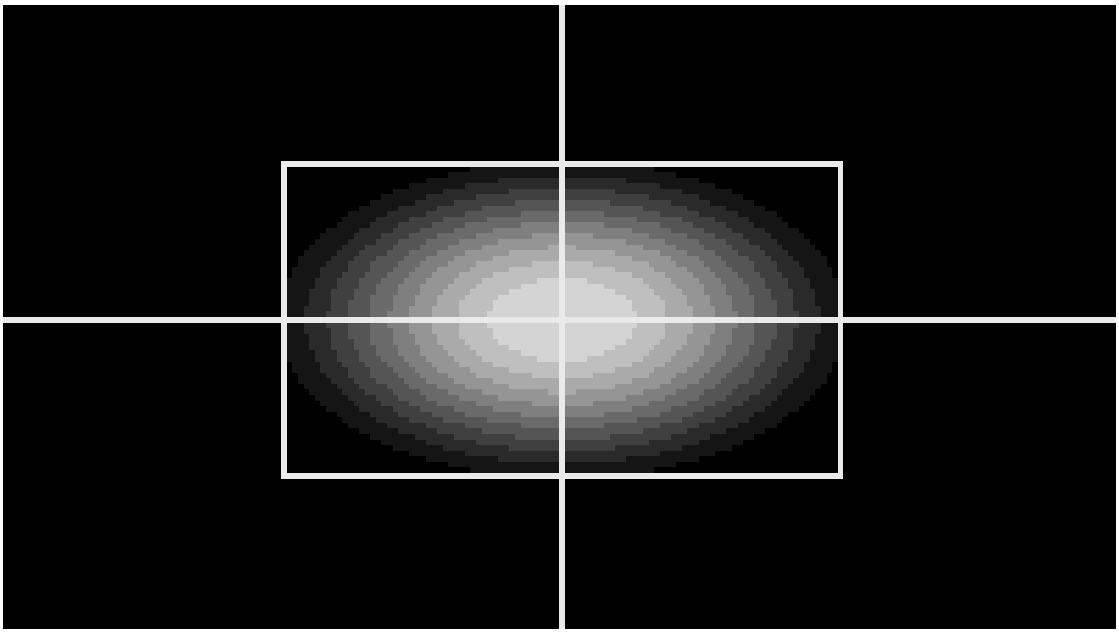


Figure 4.5: The base shading-rate image used for foveated rendering. Each frame, the eye focus is captured via an eye-tracking device and the fitting excerpt of this base shading-rate image is copied into the final shading-rate image. The four quadrants that are used if the eye focus lies at a corner and the center quadrant if the eye focus lies at the center are highlighted with a border.

Algorithm 4.8: Render Engine: Foveated Rendering

Input: Color image used for rendering the scene **C**, depth image **D**, shading-rate image **S**,base shading-rate image **B**, motion vector image **M**

```
1 B ⇐ compute base shading-rate image;
2 while renderLoop do
3 |   float e_x, e_y ⇐ get eye position;
4 |   S ⇐ shift B with offsets (width * e_x, height * e_y);
5 |   set S for draw calls;
6 |   C, D, M ⇐ draw scene;
7 |   C ⇐ apply temporal anti-aliasing to C using M;
8 end
```

4.4 TAA-Adaptive Shading

In this section we present TAA-Adaptive Shading, our completely novel approach for defining the shading rate. TAA-Adaptive Shading utilizes information generated by TAA as a side-product to reduce aliasing artefacts.

4.4.1 Derivation

TAA, developed by Karis [Kar14] , uses the information from previous frames for spatial as well as temporal anti-aliasing. As described with the aid of Figure 2.5 in Chapter 2, TAA is basically supersampling over time. Each frame, another subpixel is sampled utilizing a low-discrepancy progressive sequence, e.g., Halton sampling. This newly sampled subpixel is combined with the previously sampled subpixels by blending them with an adaptive blend factor. The blend factor defines how strong the history data should be weighted compared to the current frame's data. To take motion into account, TAA utilizes a backward projection to receive the correct history values. Sampled history values may not always be correct, as parts of the scene may have been invisible previously due to perspective or moving objects. Wrong history values result in artefacts called ghosting.

To reduce ghosting artefacts, Karis [Kar14] proposes a clipping technique. The historic color value of a specific pixel is clipped to a range computed by the local neighbourhood of the corresponding pixel in the current frame. In the original algorithm, the minimum and maximum color values of the local neighbourhood of the corresponding pixel are combined with a configurable parameter to define the clipping distance and thresholds, the clipping factors.

This approach works well for reducing ghosting. However, it introduces another artefact, namely flickering. Flickering occurs with a static camera and missing subpixel information as a consequence of undersampling. As shown in Figure 4.6, the history frame's corresponding pixel color is clipped to the current frame's pixel color which has divergent subpixel information. The final value is near this divergent subpixel information. In

the next frame, another subpixel is sampled which diverges from the previous frame's corresponding pixel color and the color changes again. This results in constantly changing color values for one pixel. The whole image has flickering regions. The workaround proposed by Karis for this problem is to reduce the blend factor in favor for the current frame when the history values are near the clipping factors [Kar14]. This solution introduces more ghosting and blurring back into the scene. Coarse shading rates of VRS amplify the problem as the subpixel regions are as large as one or multiple pixels.

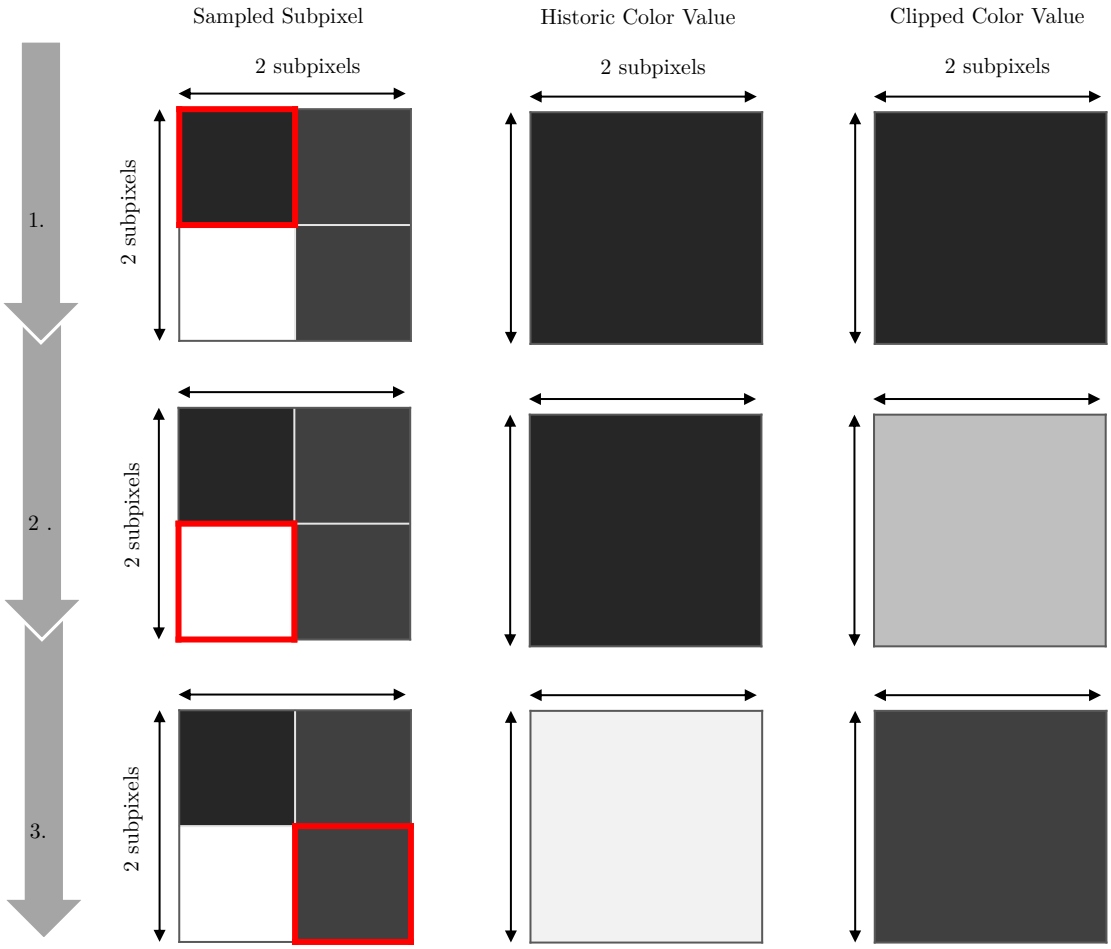


Figure 4.6: For pixels with non-homogeneous subpixel information, e.g., a higher frequency than resolution, depending on the sampled subpixel, a different color is computed. In the first row, the top-left subpixel, marked red, is sampled. The color is dark, the historic color is dark and, therefore, inside the clipping range. In the second row, a bright subpixel is sampled. The historic color is dark and therefore clipped to the bright color value. In the third row, the sampled subpixel is dark, and the bright historic color value is clipped to the darker value. Whenever a subpixel with a diverging color value is sampled, the effective color changes and the pixel flickers.

In the case of color-value clipping, flickering emerges as the result of undersampling a signal. This core understanding of flickering leads us to our novel approach for controlling the shading rate: TAA-Adaptive Shading. The fundamental strategy is to increase the shading rate for higher color differences between a pixel in the current frame and the corresponding pixel in the history frame. Conversely, the shading rate can be reduced if the difference gets smaller, as it can be assumed that the history values have a high probability of being correct and not undersampled.

Another problem introduced by applying TAA on coarsely shaded images has its roots in the computation of the clipping factors. With coarse shading, the defined local neighbourhood used to compute the clipping range might carry the same color values, which restrains the clipping region to a single color. As such, the defined local neighbourhood is not representative of the pixel and in fact, no anti-aliasing happens as the current frame's color value is used. To solve the coarse shading clipping-range restriction, the size of the neighbourhood should be scaled with the effective shading rate of a pixel. A lower shading rate and therefore a larger coarse pixel size results in a larger neighbourhood.

The problem of a non-representative neighbourhood due to coarse shading has already been addressed by Patney et al. [PSK+16], by introducing variance sampling to TAA. Variance sampling utilizes statistical moments to define a clipping distance and with it the clipping factors. Instead of computing the min and max values of the local neighbourhood for every pixel, they compute the first two raw moments of the local color distribution and derive the mean and the variance. The clipping distance is defined by the standard deviation computed with the variance. The clipping factors are the mean plus/minus the standard deviation. Since these statistical moments are linear quantities, they can be pre-filtered with mipmapping. The shading rate is used to select the right mipmap level.

We use these clipping factors sampled at the right mipmap level computed by variance sampling during TAA to define our shading rate. Color values inside the so defined clipping region are assumed to be historically correct and are therefore coarsely shaded. Color values outside of this clipping region are assumed to be undersampled or wrong and are therefore supersampled. This approach of utilizing the clipping distance could even be utilized for other application scenarios where a division in undersampled and oversamples regions in real time is necessary.

4.4.2 Algorithm

The steps of the base algorithm are shown in Figure 4.7 and are similar as for Content-Adaptive Shading described in Section 4.1, shown in Figure 4.1. However, the computation of the shading-rate properties is done during the post-processing step of TAA. For the computation of the shading-rate image, the same common steps as used by our Content-Adaptive Shading algorithm, shown in Figure 4.2., are performed. However, the computation of the shading rate differs as it depends on the clipping factors of TAA.

Algorithm 4.9 gives an overview of the steps and commands issued on the CPU. We start by clearing the clipping distance image to white on line 1, which is equal to setting the

Figure 4.7: The basic steps of our algorithm for TAA-Adaptive Shading. These steps are executed inside the render loop in each iteration. At the start, the shading-rate image is cleared. Then, the image containing the properties computed during the previous iteration is downscaled and used to compute the shading rate for each 16x16 pixel region. Subsequently, the scene is drawn using the computed shading-rate image. Finally, the drawn image is anti-aliased using TAA and the clipping distance for the next iteration is computed.

whole image to the maximum shading rate as all regions are initially undersampled. On line 3, the shading-rate image is again reset to a default shading rate due to pixels not covered by the forward projection, as discussed in Section 4.1. Afterwards, on line 4, the image is scaled down to $\frac{1}{16}(w, h)$, where w and h denote the width and height of the image, to receive an average color difference for each 16x16 tile. As discussed before, downscaling uses bilinear interpolation, averaging each of the 16x16 tiles. In the next step on line 5, the shading-rate image is computed with a compute shader utilizing the historic color differences and on line 6 set for subsequent draw calls like on line 7. For variance sampling, the mean, the variance and the corresponding mipmaps are computed at line 8. These images are then used on line 9 to perform TAA.

In Algorithm 4.10 the shading rate is computed based upon the previously saved color difference. The data from the previous frame is forward projected on lines 2 and 3, as

Algorithm 4.9: Render Engine: TAA-Adaptive Shading

Input: Color image used for rendering the scene $\mathbf{C}$, depth image $\mathbf{D}$, clipping distance image $\mathbf{T}$, clipping distance image scaled down $\mathbf{T_s}$, shading-rate image $\mathbf{S}$, motion vector image $\mathbf{M}$, color average image $\mathbf{C_a}$ and color variance image $\mathbf{C_v}$

```
1  T ⇐ clear image with white, e.g., max. distance;
2  while renderLoop do
3      S ⇐ clear image to default shading rate;
4      T_s ⇐ scale down T to 1/16 resolution;
5      S ⇐ computeShadingRateImage(T_s, S); // see Algorithm 4.10
6      set S for draw calls;
7      C, D, M, T ⇐ draw scene;
8      C_a, C_v ⇐ compute color average and variance of C with mipmaps;
9      C ⇐ apply temporal anti-aliasing to C using M, C_a, C_v; // see
         Algorithm 4.11
10 end
```

discussed in Section 4.1. The final color difference value is multiplied with the highest index of an ascending ordered shading-rate palette on line 5. The result is the shading rate, an index in this palette. On lines 6 and 7, our shading-rate stabilization technique is applied. Finally, on line 8, the shading rate is saved into the shading-rate image.

Algorithm 4.10: computeShadingRateImage: TAA based Compute Shader

Input: clipping distance image scaled down $\mathbf{T_s}$, shading-rate image $\mathbf{S}$, inverse projection matrix $\mathbf{P^{-1}_{prev}}$, inverse view matrix of previous frame $\mathbf{V^{-1}_{prev}}$, view projection matrix of next frame $\mathbf{V}, \mathbf{P}$

Output: shading-rate image $\mathbf{S}$

```
1  // forward projection: compute new NDC coordinates
2  vec3 pos_w ⇐ back project x, y, D[x, y] with P^-1_prev, V^-1_prev ;
3  x_c, y_c ⇐ forward project pos_w with V, P;
4  // compute shading rate
5  int sr ⇐ floor(T_s[x, y] * 11);
6  int prevSr ⇐ clamp(S[x_c, y_c], sr − 1, sr + 2);
7  sr ⇐ max(sr, prevSr);
8  S[x_c, y_c] ⇐ sr;
```

Algorithm 4.11 shows our adaptation of TAA developed by Karis[Kar14] including variance shading, as described by Patney et al. [PSK+16]. Utilizing the mipmaps computed for variance sampling, on line 4 the shading rate specifies the mipmap level to fetch the correct mean and variance. For example, a 2x2 shading rate can use the second mipmap level and a shading rate of 4x4 the third. Therefore, variance sampling allows us to have an efficient multi-scale TAA implementation. Additionally, it reduces ghosting

better than min-max neighbourhood clipping as outliers have less impact. In order to compute the shading-rate image, we extend this TAA post-processing program on line 6 to compute the absolute difference between the mean color value of the current frame at the mipmap level of the current shading rate and the value of the history frame. This value is then divided by the clipping distance computed during TAA on line 5. With this division, values larger than the clipping distance are transformed to values larger than one and values inside the clipping distance are transformed into values smaller than one. By dividing by two, the value range $[0, 0.5]$ belongs to coarse shading rates and the region $[0.5, 1]$ above the threshold corresponds to supersampling shading rates. The final value is saved in an additional image.

Algorithm 4.11: TAA: additional property clipping distance

Input: Color image $\mathbf{C}$, color average image $\mathbf{C_a}$, color variance image $\mathbf{C_v}$, previous color image used for rendering the scene $\mathbf{C_p}$, depth image $\mathbf{D}$, inverse projection matrix $\mathbf{P_{prev}^{-1}}$, inverse view matrix of previous frame $\mathbf{V_{prev}^{-1}}$, view projection matrix of next frame $\mathbf{V}, \mathbf{P}$, effective shading rate $\mathbf{sr}$ (sr.x/sr.y = fragment size, sr.z = sample count)

Output: Anti-aliased color image $\mathbf{C}$, clipping distance image $\mathbf{T/t}$

1 // back-project and sample correct history value
2 ...
3 $c_p \Leftarrow \mathbf{C_p}$ at back-projected position;
4 $c_a, c_v \Leftarrow$ color average and variance of $\mathbf{C_a}$ at mipmap level $\frac{\mathbf{sr.x}+\mathbf{sr.y}}{4} + 1$;
5 $s \Leftarrow$ compute clipping distance with c_a and c_v;
6 $\mathbf{t} = length(\frac{abs(c_p - c_a)}{2s})$;
7 // anti-alias $\mathbf{C}$ with c_p, c_a and s
8 ...

CHAPTER 5

Evaluation

The evaluation of our implemented VRS techniques consists of a performance and a quality analysis. For the performance analysis, we captured the frame times during an animation in various test configurations. The quality comparison of the different VRS approaches is based on frames which where captured at fixed animation-times and compared in terms of Peak Signal to Noise Ratio (PSNR) and Structure Similarity Index (SSIM). These metrics have also been used by Vaidyanathan et al. [VST+14] to evaluate their software implementation of a VRS approach, and by Hillesland and Yang [HY16], who used it to evaluate their approach to VRS, texel shading.

The test configurations vary in two scenes, forward and deferred shading, four resolutions and four MSAA sample configurations, which control the possible shading rates, making a total of 64 configurations for five runs, one per algorithm and one with VRS disabled. The first scene, displayed in Figure 5.1, consists of Crytek's Sponza[1], with detailed normal maps for each surface lit by 100 point lights and one directional light. The second scene, displayed in Figure 5.2, is an island in the sea consisting of large flat surfaces. Details are introduced by small tiles of grass and palms with geometry-only leaves to evaluate geometric aliasing. As for the shading model, physically based shading was used, which introduces heavy fragment shader loads (970 SPIR-V instructions).

We compare the performance of forward rendering and deferred rendering for all VRS techniques we implemented. For each of the following four resolutions, we used 1x,2x,4x and 8x MSAA samples:

- 1920x1080 (Full HD)
- 2560x1440 (WQHD)

[1] created by Frank Meinl, modified by Morgan McGuire in 2014 and by Johannes Unterguggenberger in 2018

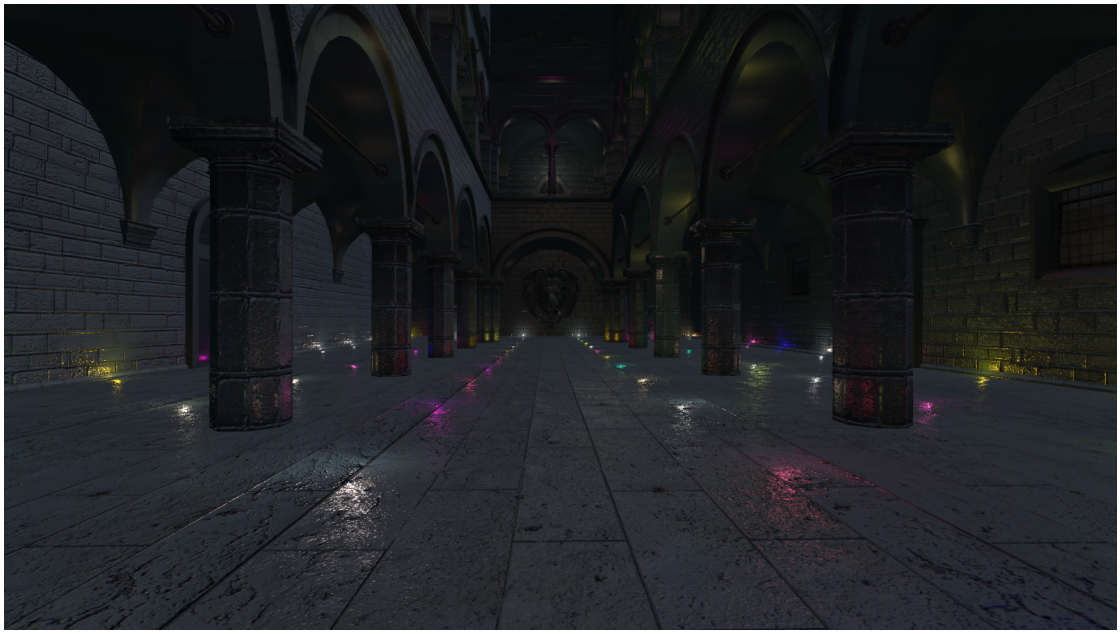


Figure 5.1: The Sponza scene: Frank Meinl's Crytek Sponza model with detailed normal maps for each surface lit by 100 point lights and one directional light.

- 3840x2160 (UHD-I/4k)
- 7680x4320 (UHD-II)

The animation has a duration of ten seconds and its camera path transitions from an overview shot over the complete scene to various close-up views.

The hardware which served for the evaluation consists of an Intel Core i7-8700k at a clock speed of 5.0 GHz, 16 GiB DDR4 3200 MHz memory and RTX 2080 Ti at a clock speed of 1650 MHz featuring 11 GiB GDDR5 1750 MHz memory.

5.1 Quality and Performance

As VRS allows both a higher performance by coarse shading and higher quality by supersampling, we evaluate by varying the MSAA sample configurations described above and discuss which trade-offs can be considered. In Figure 5.3 the performance for all configurations for the Sponza scene is visualized. The SSIM and PSNR values for all configurations computed in the Sponza scene are shown in Figure 5.4 and Figure 5.5, respectively. The ground truth used for the SSIM and PSNR computations is represented by images rendered at a resolution of 7680x4320 with 8x MSAA with VRS disabled. In all figures, from 5.3 to 5.9, "NONE" will be the configuration with VRS disabled, "VRS CAS TEXEL" stands for Content-Adaptive Shading based on texel differentials,

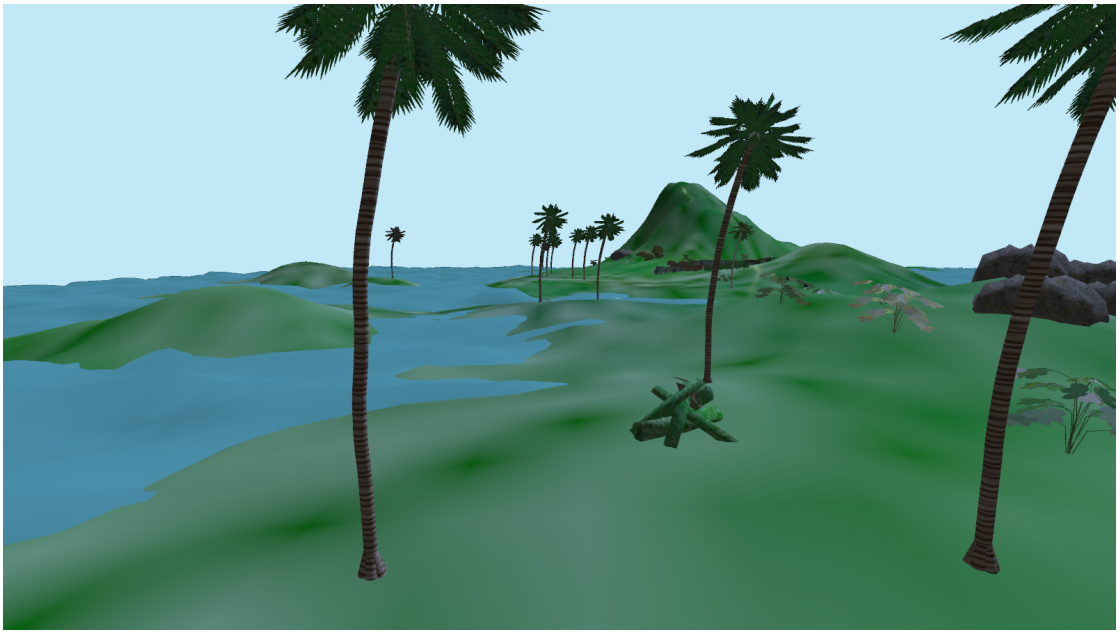


Figure 5.2: The Island scene: An island in the sea with large flat surfaces and palm leaves consisting of geometry to evaluate geometric aliasing.

"VRS CAS EDGE" for Content-Adaptive Shading based on edges, "VRS MAS" for our Motion-Adaptive Shading algorithm and "VRS TAA" for our novel method of TAA-Adaptive Shading. In Table 5.1 the used shading rates per MSAA sample configuration are given. With higher numbers of configured samples, more supersampling shading rates get enabled and fewer coarse shading rates can be used, as discussed in Chapter 3. As Figure 5.3 shows, all VRS techniques besides Motion-Adaptive Shading using the configuration with 1x MSAA sample are at least 25% faster compared to not using VRS in terms of frame times for our configuration. At low resolutions, the performance gain of Motion-Adaptive Shading is negligible, while at resolutions equal or above 7680x4320 the performance gain is 25%, too. VRS TAA and VRS CAS TEXEL show similar frame times for resolutions as high as 3840x2160 at 2x MSAA compared to rendering with a lower resolution of 1920x1080 with VRS disabled and are four times faster in terms of frame times compared to rendering at the same resolution with VRS disabled while preserving a similar quality. At a very high resolutions of 7680x4320 with 8x MSAA samples, these two techniques still outperform the other techniques and show frame times as low as two-thirds of the ground truth's. The quality for this configuration is very close to the ground truth with an SSIM of 98%, as can be seen in Figure 5.4. We noticed that, while rendering with VRS disabled, more MSAA samples are almost for free on modern hardware. In Figure 5.4 the correlation between configured MSAA samples and quality can be observed. This is expected, as more MSAA samples result in higher supersampling rates and less coarse shading, therefore result in better quality at the cost of performance. In terms of quality, aside from NONE, VRS MAS delivers the best

MSAA Samples	Coarse Shading Rates	Supersampling Shading Rates
1x	1x1, 2x2, 4x4	-
2x	1x1, 2x2, 2x4	2x
4x	2x2	2x, 4x
8x	1x2	2x, 4x, 8x

Table 5.1: Used shading rates per MSAA sample configuration.

results, which are always very close to the renderings with VRS disabled and sometimes even better, as VRS MAS supersamples still regions. However, the SSIM values of all methods are between 81% and 98% and therefore deliver reasonable results. VRS TAA and VRS CAS TEXEL perform similarly in terms of quality whereby VRS TAA performs slightly better for low MSAA samples like 1x or 2x. VRS CAS EDGE outperforms these two at lower resolutions, though it produces very similar results up from 3840x2160 with 4x MSAA. The PSNR values in Figure 5.5 support these discussed quality comparisons except for resolutions of 7680x4320 and above, where VRS MAS is the only technique that produces similar PSNR values and as such quality as renderings with VRS disabled. The PSNR value for the configuration with VRS disabled at the resolution 7680x4320 with 8x MSAA samples is equivalent to the ground truth and therefore infinite, which is the reason why it is not displayed in Figure 5.5.

The performance metrics for the Island model, displayed in Figure 5.6, are quite similar, despite one major difference: VRS CAS EDGE performs similarly to VRS CAS TEXEL and VRS TAA. The reason for this might be that the Island model has more flat surfaces while the Sponza model with its heavily normal-mapped surfaces does not offer many surfaces without edges to optimize. The value ranges of the SSIM metric displayed in Figure 5.7 is very close and highlights that quality is insignificantly reduced by our VRS techniques in such scenes, with a worst SSIM value of 99%. Similarly, the PSNR values displayed in Figure 5.8 have, for almost all VRS techniques, per configuration a maximum difference of three. Only for the highest resolution of 7680x4320 and at the highest sample rate of 8, VRS MAS shows higher quality as all the other techniques.

5.2 Forward and Deferred Shading Performance Comparison

With deferred shading, the MSAA resolve has to take place in the lighting pass, which makes it more expensive. This also has an impact on supersampling with VRS, as can be observed in Figure 5.9. The overlayed bars are frame times computed by deferred shading while the background bars are computed with forward shading. With only one MSAA sample, most of the techniques are faster or at least almost as fast as with deferred shading enabled. For higher MSAA sample counts, deferred shading falls behind forward shading, and at 8x MSAA frame times show a substantial overhead. VRS CAS TEXEL

and VRS TAA delivery the same or better performance for all configurations in the Sponza scene with forward shading and should, therefore, be preferred for scenes like Sponza. For the island scene, as it is more simple in terms of lighting, deferred shading is slower for all configurations. Additionally, VRS may already reduce the shader load with coarse shading and therefore reduces the advantage of deferred shading. However, for scenes that profit highly from deferred shading, performance improvements may be possible, at least with no MSAA samples.

5.3 Temporal Artifacts

In Chapter 4 we discussed the temporal instability of the shading-rate image for techniques that base the shading rate on properties of the previous frames. Due to the oscillation of the shading rate, regions of pixels change their color value continuously. This effect is reduced by temporal anti-aliasing, as the color-changing pixels are averaged with color values from the corresponding pixels of previous frames. Additionally, we applied our variant of temporal anti-aliasing directly to the shading-rate image to stabilize the shading rates. Both these measures are effective in reducing the number of shading rate changes and their visibility. However, they cannot eliminate this effect completely. In Figure 5.10 an example of these temporal artefacts can be seen. Three consecutive frames were captured at a frame time of about 5ms. The excerpt used for the comparison is highlighted with a white border in Figure 5.10a. Three differences are highlighted in Figures 5.10b, 5.10c, and 5.10d by red circles. These effects emerge especially in dark scenes with non-flat surfaces and near lights because there color differences of neighbouring pixels are high, as is the case for the Sponza test scene. In the Island test scene, such artefacts could not be observed. Additionally, these artefacts could only be observed for configurations focusing on performance over quality because these configurations heavily use the maximum coarse shading rates. With configurations that disable the maximum coarse shading rate, like 4x MSAA samples, such artefacts are drastically reduced and are almost not visible in the Sponza scene.

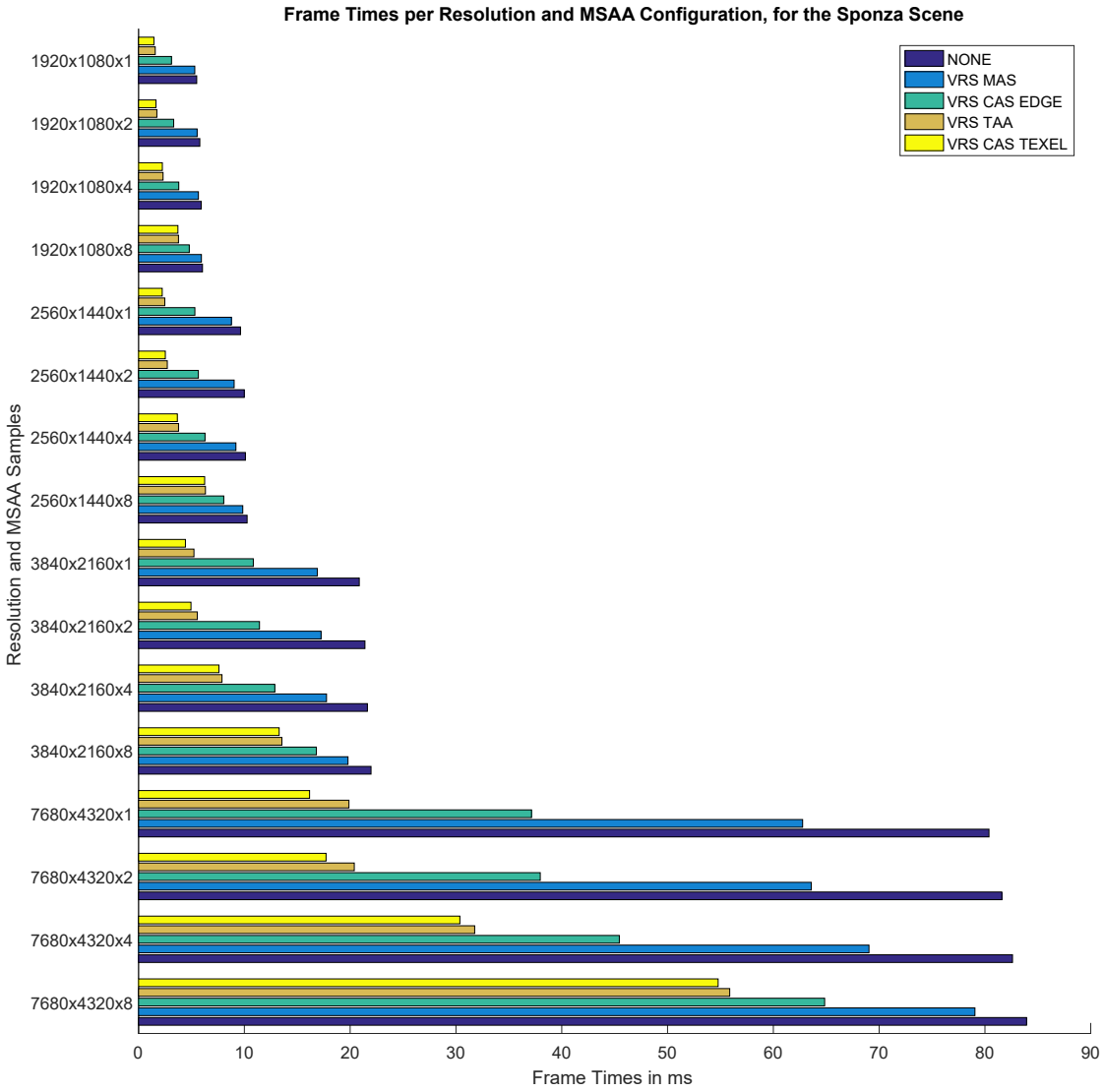


Figure 5.3: The bar chart shows how rendering at different resolution configurations affects performance of ordinary rendering (labeled “NONE”, which means VRS disabled, represented by the blue line) and of rendering with VRS enabled (orange, yellow, purple, and green lines, each representing a different VRS technique as described in Chapter 4). The resolution configurations vary in framebuffer resolution and the number of MSAA samples. The more MSAA samples are used, the more supersampling shading rates are enabled and, consequently, more coarse shading rates get disabled.

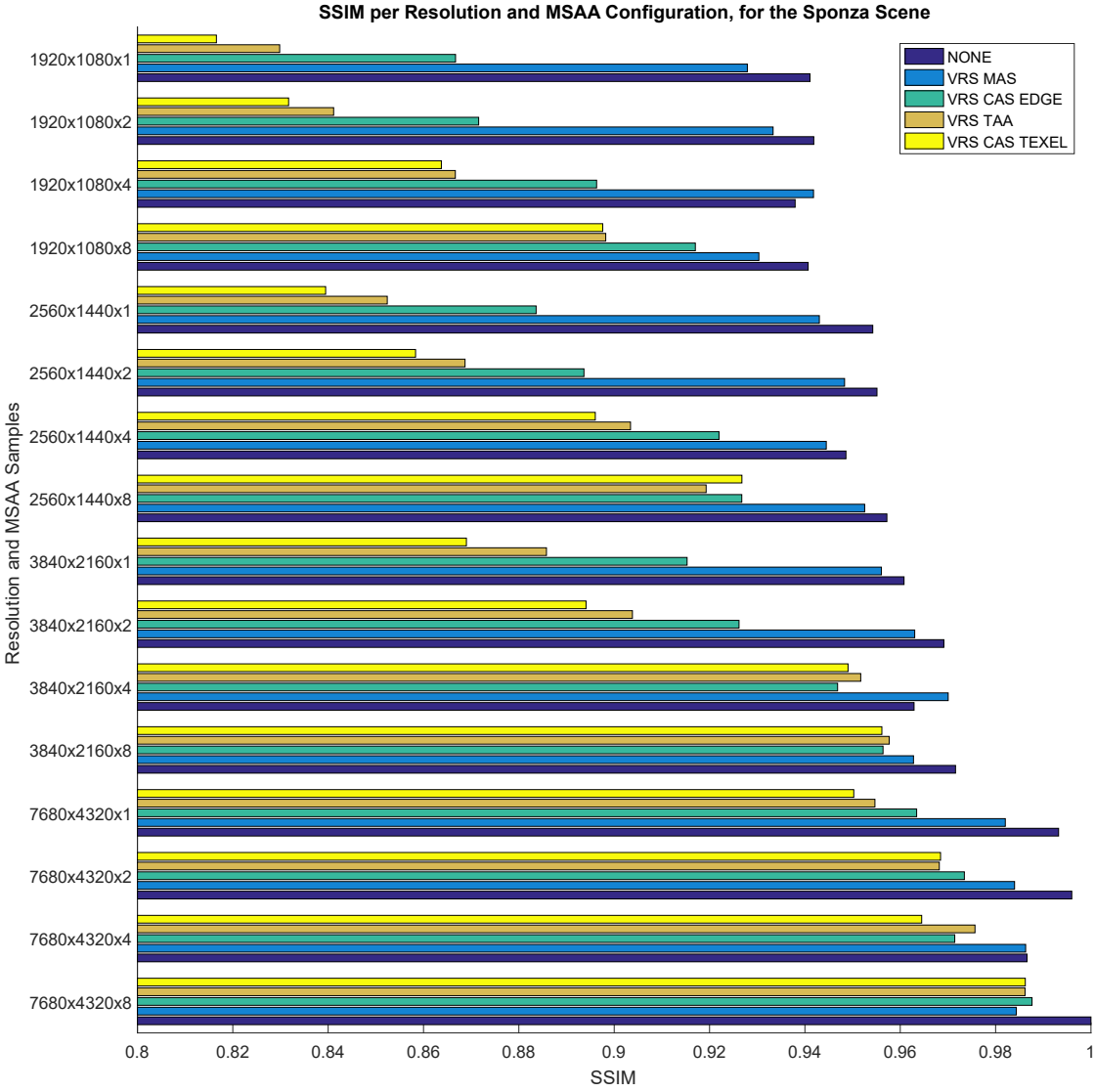


Figure 5.4: This bar chart displays the quality measured with SSIM for the Sponza scene for each configuration and VRS technique as described in Figure 5.3. It highlights that quality depends on the number of MSAA samples. Especially methods with better performance as shown in Figure 5.3, profit from higher sample counts. The ground truth was set by images rendered without VRS at a resolution of 7680x4320 with 8x MSAA.

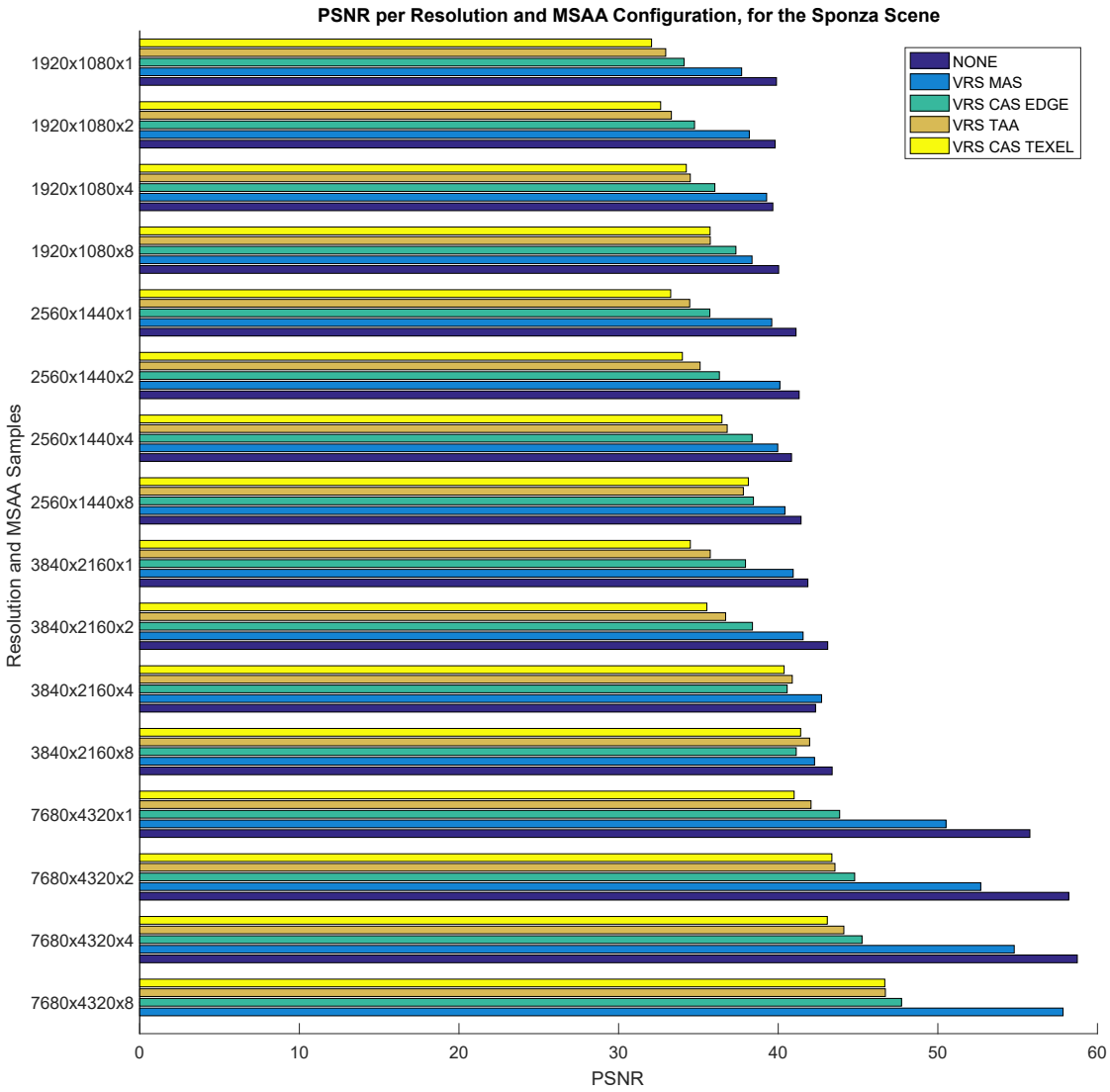


Figure 5.5: This bar chart shows the quality measured with PSNR for the Sponza scene for each configuration and VRS technique as described in Figure 5.3. For the computations, the same ground truth images as for the SSIM evaluation displayed in Figure 5.4 were used. In contrast to the SSIM evaluation, PSNR values of the faster techniques show a higher difference to the ground truth for larger resolutions like 7680x4320.

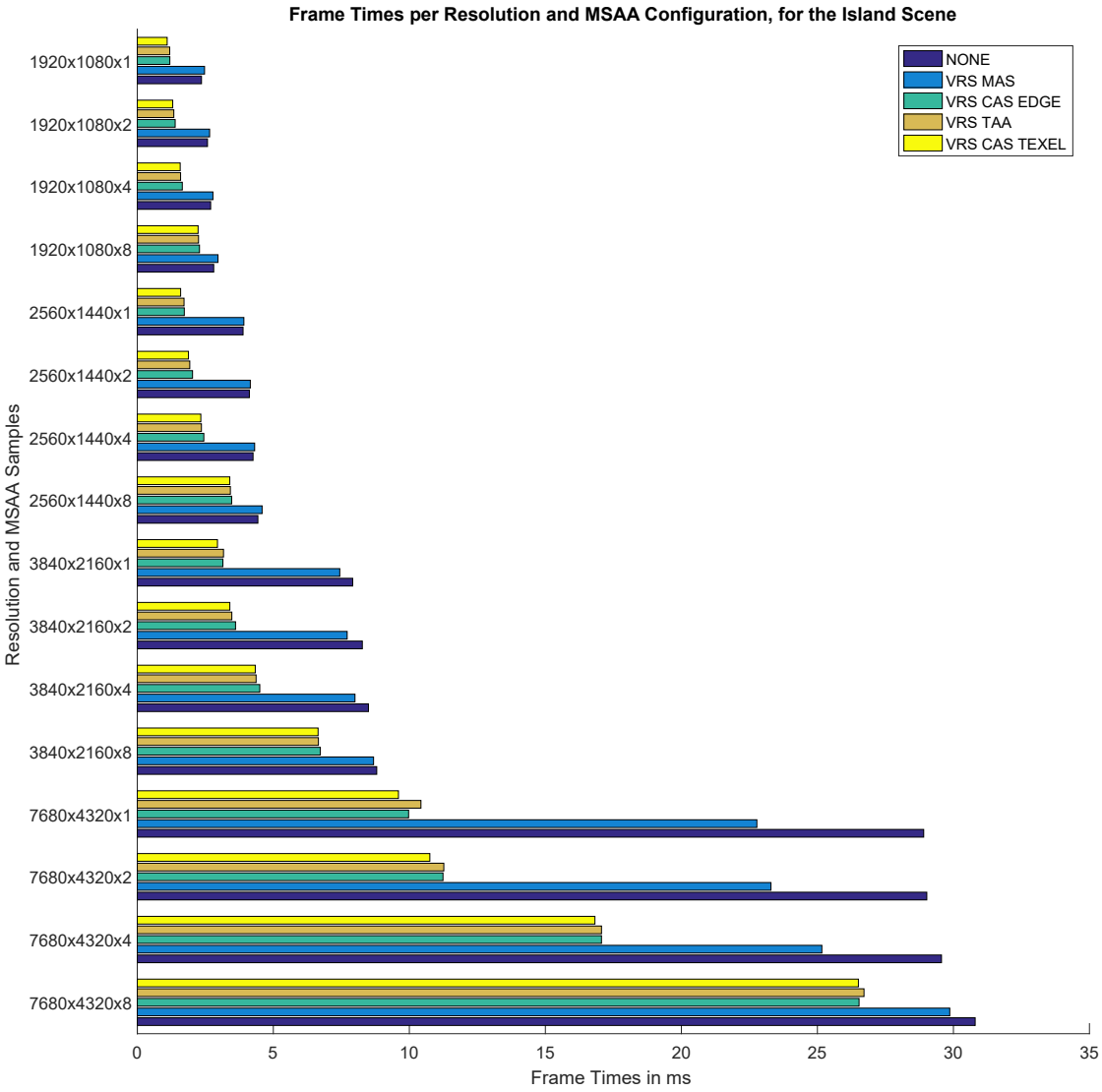


Figure 5.6: The performance for the Island scene in frame times for each configuration is visualized by this chart. The structure is described in and based on Figure 5.3 for the frame times of the Sponza scene. The results are very similar to the Sponza frame times, besides that VRS CAS EDGE performs similarly as VRS CAS TEXEL or VRS TAA.

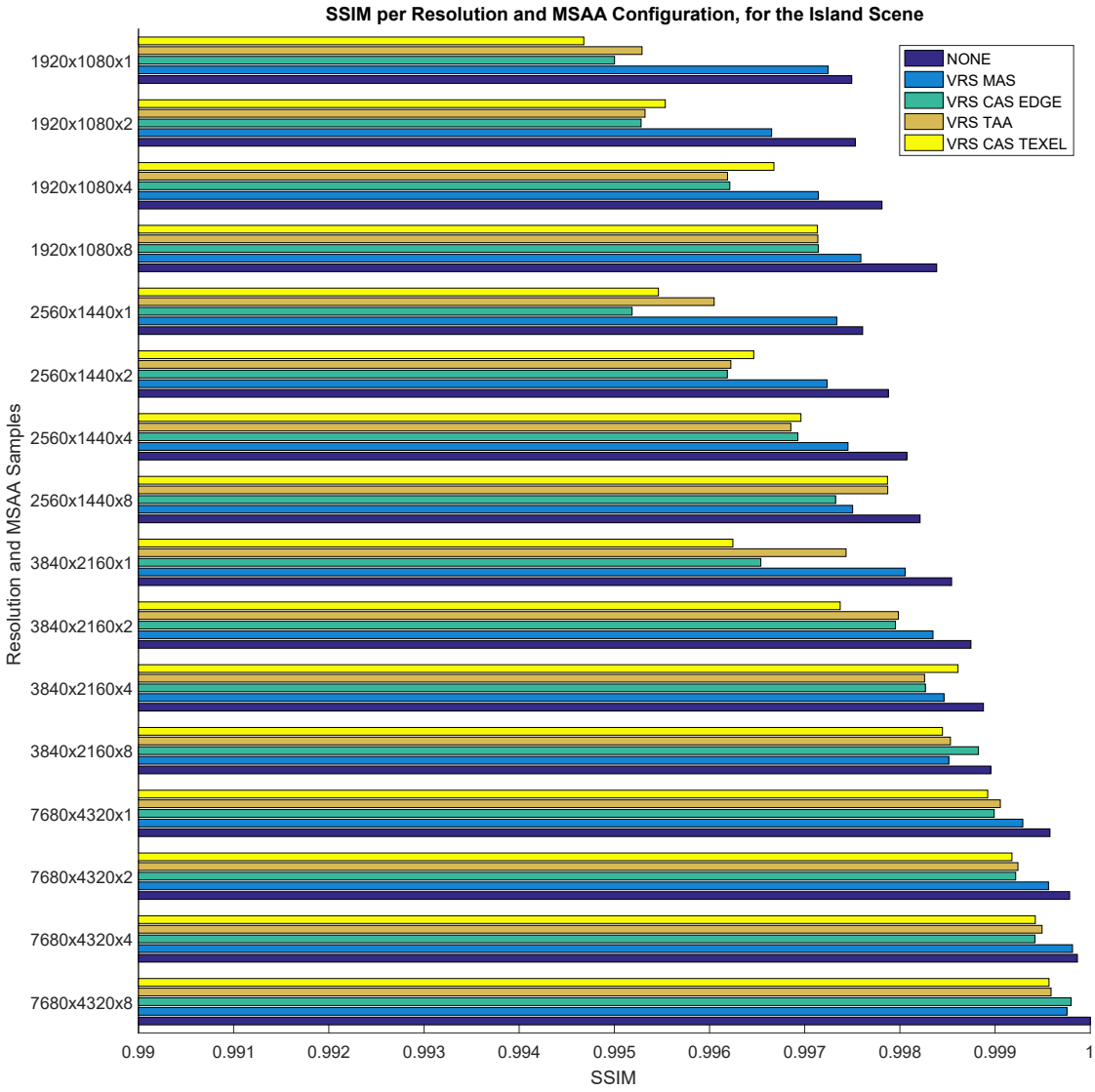


Figure 5.7: This bar chart displays the quality measured with SSIM for the Island scene for each configuration and VRS technique as described in Figure 5.3. It underpins the results measured for the Sponza scene shown in Figure 5.4, though the quality gap of the different techniques is smaller for the Island scene.

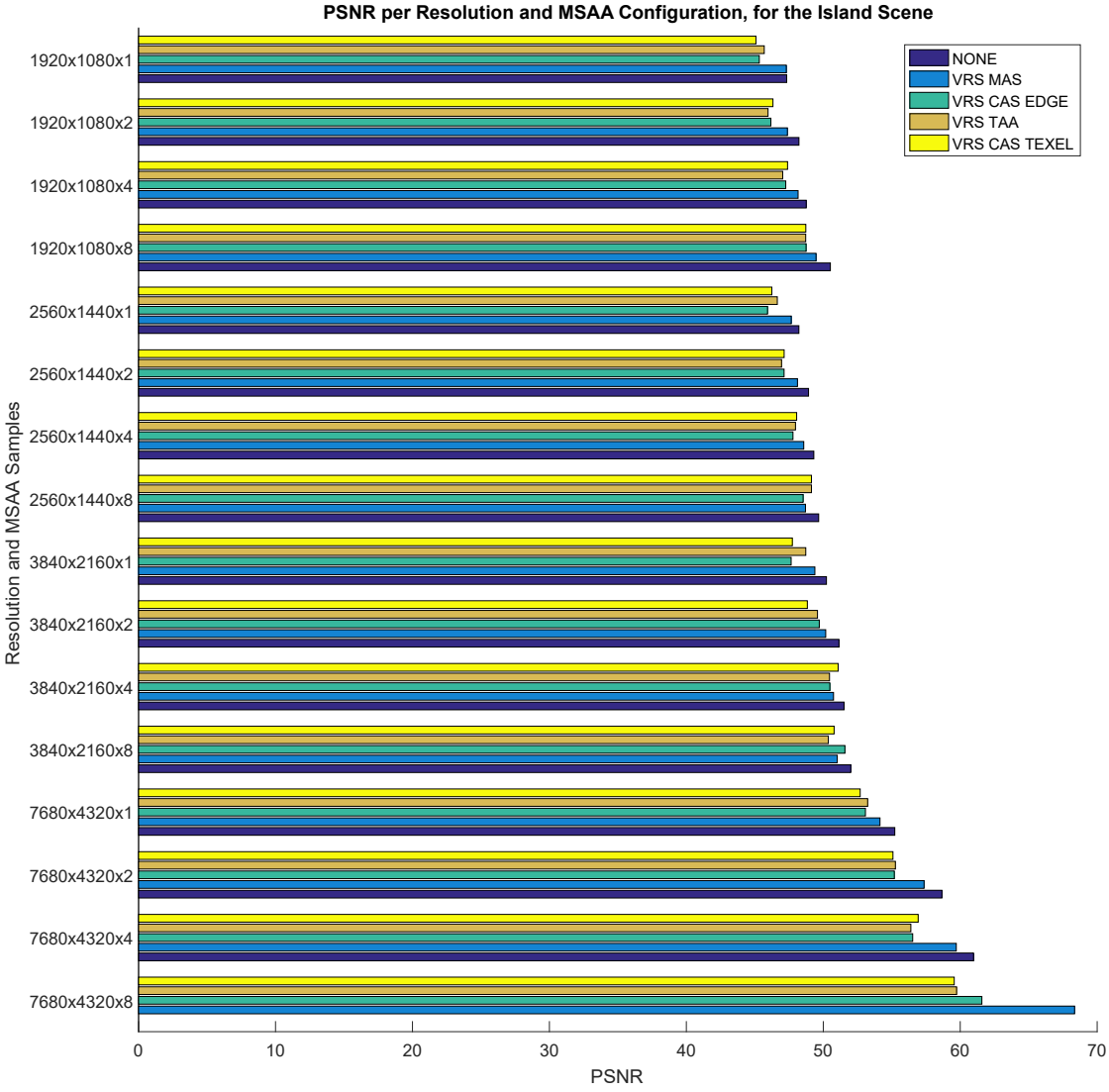


Figure 5.8: This chart shows the quality measured with PSNR for the Island scene for each configuration and VRS technique as described in Figure 5.3. The values show, compared to the PSNR values of the Sponza scene displayed in Figure 5.5, very small differences for each configuration and VRS technique.

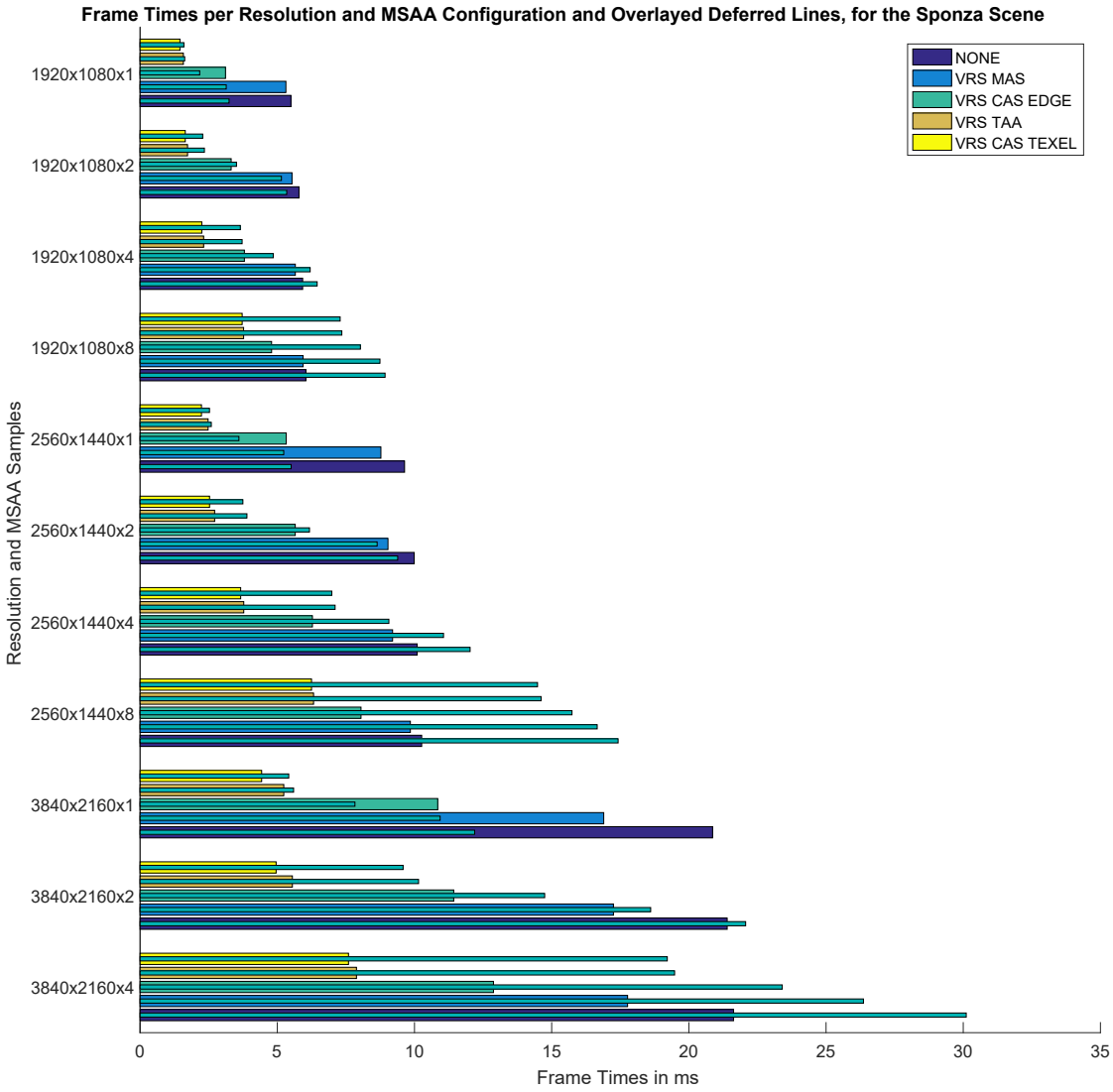
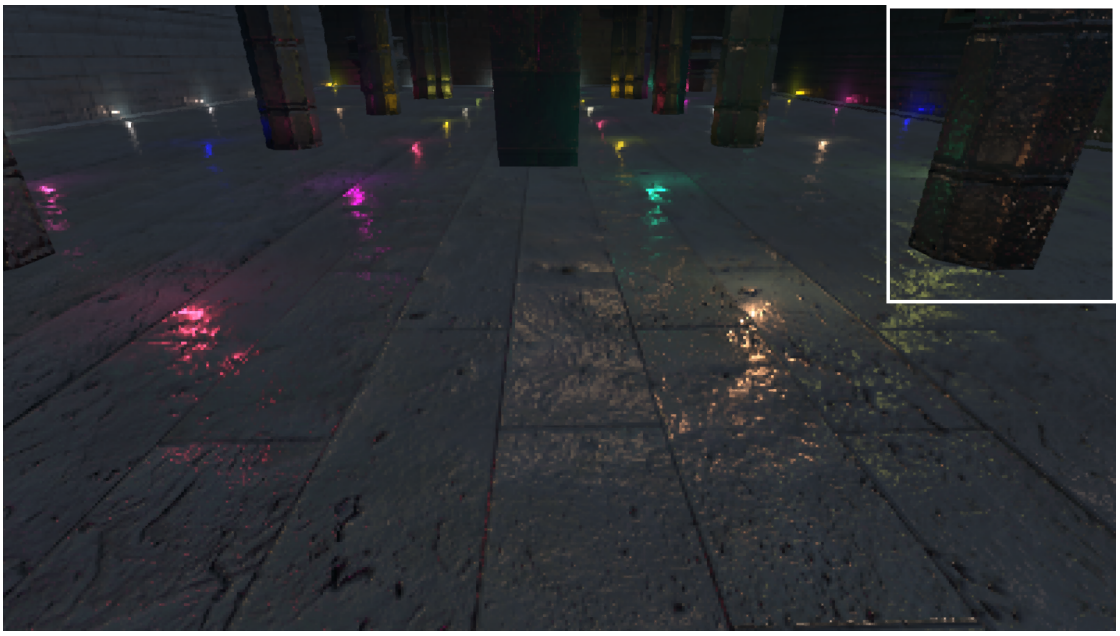


Figure 5.9: This chart visualizes the measured performance of forward shading and deferred shading (box overlays) for the Sponza scene in frame times for each configuration. Deferred shading can improve performance for low MSAA sample counts while with higher numbers of MSAA sample counts the performance demands increase significantly.



(a)

(b) (c) (d)

Figure 5.10: Example of temporal artefacts occurring in the Sponza Scene. The excerpt that is investigated is highlighted in (a) by a white border. In (b),(c) and (d), consecutive frames are displayed and three regions with differences marked by red circles.

CHAPTER 6

Conclusion and Future Work

In this thesis, we examined VRS, a new hardware feature introduced with the NVIDIA Turing micro-architecture. We gave a detailed explanation of what can be achieved with it, discussed its potential and which limitations for this new technology arise. Especially the limitations that not all shading rates can be used at the same time and that they are limited by the number of configured MSAA samples is an interesting finding.

Exploiting the capabilities of VRS, we developed various novel algorithms for selecting a shading rate per 16x16 pixels region. Content-Adaptive Shading and Motion-Adaptive Shading are two standard strategies for selecting appropriate shading rates. We propose two new approaches for Content-Adaptive Shading, namely edge-based and texel differential-based Content-Adaptive Shading. Depending on the scene content's properties used for computing the shading rate, the shading rate may be unstable which introduces temporal artefacts.

To counter the artefacts introduced by unstable shading rates, we apply shading rate-dependent TAA using variance sampling. Additionally, we propose a technique for stabilizing the shading rate, independent of the shading-rate selection mechanism. Our stabilization technique applies an adaptation of TAA on the shading-rate image itself and successfully stabilizes the shading rate in many cases. However, shading-rate instabilities are not eliminated completely and as such, temporal artefacts can arise depending on the concrete properties.

We also introduce a completely novel algorithm, TAA-Adaptive Shading, based on the idea of calculating the shading rate from different color samples over time. The core concept behind this algorithm is that color deviations of different subpixels inside the same pixel indicate additional subpixel information. In contrast, very similar colors of different subpixels inside the same pixel may indicate homogeneous regions. This information can easily be derived during TAA, which our algorithm is based on.

Additionally, we present concrete implementations for Motion-Adaptive Shading, based on the motion of multiple frames, and the integration with a deferred shading pipeline.

Our results show that some of our algorithms can increase the performance by a factor of four or allow the increase of the resolution by the same factor while not affecting performance. The quality of our algorithms in terms of PSNR and SSIM is high, with SSIM values of 82%, to very high at SSIM values of 98%. The integration of our best-performing algorithms into existing rendering pipelines is straightforward, requires few additional steps and few changes to the pipeline itself.

Further temporal stabilization of the shading rate could be an interesting challenge for future work. Besides temporal stabilization of the shading rate, temporal stabilization of the final color could hide the effects of unstable shading rates. For example, previous frames could be weighted stronger during TAA for static cameras to alleviate the effects of unstable shading rates even more. However, this approach does not work for moving objects in the scene as ghosting artefacts would be introduced by a higher weight for previous frames. Objects in motion could be tracked by their motion vectors and regions in motion could be assigned a lower weight for previous frames compared to the stable regions. An example of such regions would be the region where objects have moved away from.

Another possibility for future work is the creation of new shading rate control algorithms, as Microsoft has standardized a new version of VRS for DirectX 12 which allows to control the shading rate per primitive. [1]. This new control mechanism could enable algorithms that do not need to know information about the scene beforehand and could show more temporal stability. These new algorithms could use any information present in the vertex shader, like position or depth. Other properties could be precomputed in the vertex shader to define the shading rate. As the DirectX 12 specification defines a combination of the shading rate from the shading-rate image and the primitive, shading rates per primitive could be used to stabilize the shading rate and, therefore, reduce the temporal artefacts.

As discussed in Chapter 3, the Vulkan specification states that the tile size for the shading-rate image is implementation-dependent. The same applies to the number of coverage samples. In future hardware generations, these limitations could be removed or at least extended with more supported configurations. With smaller tile sizes, problems as present in the scene with the brick wall could be avoided at the cost of more expensive shading-rate image computations as the resolution needs to be increased. Maybe the tile size could even be configured dynamically and an additional shading-rate image specifies the different tile sizes. This concept could be taken further by a shading-rate pyramid, e.g., by mipmapped shading-rate images. This leads to another topic, the number of shading rates. By increasing the number of coverage samples and MSAA samples, more shading rates could be used at the same time. Additionally, more shading rates could be

[1] https://devblogs.microsoft.com/directx/variable-rate-shading-a-scalpel-in-a-world-of-sledgehammers/

easily imagined, as coarse shading rates like 8x8, 16x16, or supersampling shading rates like 16x, 32x. Especially higher coarse shading rates will be needed as screen resolutions increase. In conclusion, we expect many new algorithms based on VRS and its upcoming extensions, given its ease of integration and its potential.

List of Figures

List of Tables

List of Algorithms

Bibliography

[AMHH18] Tomas Akenine-Moller, Eric Haines, and Naty Hoffman. *Real-Time Rendering*. AK Peters/CRC Press, 4th edition edition, 2018.

[ASK+12] Deborah Apthorp, D Samuel Schwarzkopf, Christian Kaul, Bahador Bahrami, David Alais, and Geraint Rees. Direct evidence for encoding of motion streaks in human. *Proceedings of The Royal Society B: Biological Sciences*, 2012.

[Bho18] Swaroop Bhonde. Turing variable rate shading in vrworks. NVIDIA developer blog, https://devblogs.nvidia.com/turing-variable-rate-shading-vrworks/, September 2018. online, accessed 12.11.2019.

[CML11] Matthäus G Chajdas, Morgan McGuire, and David Luebke. Subpixel reconstruction antialiasing for deferred shading. In *Symposium on Interactive 3D Graphics and Games*, pages 15–22. Citeseer, 2011.

[GFD+12] Brian Guenter, Mark Finch, Steven Drucker, Desney Tan, and John Snyder. Foveated 3d graphics. *ACM Transactions on Graphics (TOG)*, 31(6):164, 2012.

[GFS+17] Brian K Guenter, Mark Finch, John Snyder, Steven Drucker, and Desney S Tan. Foveated image rendering, August 8 2017. US Patent 9,727,991.

[GGNB18] Mark S Grossman, Jason Matthew Gould, Alexander Nankervis, and Charles Neill Boyd. Variable rate shading, February 15 2018. US Patent App. 15/237,334.

[GVWD06] Mark Grundland, Rahul Vohra, Gareth P. Williams, and Neil A. Dodgson. Cross dissolve without cross fade: Preserving contrast, color and salience in image compositing. *Computer Graphics Forum*, 25(3):577–586, 2006.

[HGF14] Yong He, Yan Gu, and Kayvon Fatahalian. Extending the graphics pipeline with adaptive, multi-rate shading. *ACM Transactions on Graphics (TOG)*, 33(4):142, 2014.

[HY16] Karl E Hillesland and JC Yang. Texel shading. In *Proceedings of the 37th Annual Conference of the European Association for Computer Graphics: Short Papers*, pages 73–76. Eurographics Association, 2016.

[JGY+11] Jorge Jimenez, Diego Gutierrez, Jason Yang, Alexander Reshetov, Pete Demoreuille, Tobias Berghoff, Cedric Perthuis, Henry Yu, Morgan McGuire, Timothy Lottes, et al. Filtering approaches for real-time anti-aliasing. *ACM SIGGRAPH Courses*, 2(3):4, 2011.

[Kar14] Brian Karis. High-quality temporal supersampling. *Advances in Real-Time Rendering in Games, SIGGRAPH Courses*, 1:1–55, 2014.

[Khr19] The Khronos® Vulkan Working Group. *Vulkan® 1.1.113 - A Specification (with all registered Vulkan extensions)*, June 2019. https://www.khronos.org/registry/vulkan/specs/1.1-extensions/html/vkspec.html, from git branch: github-master commit: 8897c572ce5473486bb92e3ab9574a5f130794c3.

[Lot11] Timothy Lottes. Fxaa. *NVIDIA white paper*, 2:8, 2011.

[NFGG18] Ivan Nevraev, Martin JI Fuller, Mark S Grossman, and Jason M Gould. Variable rate shading, August 23 2018. US Patent App. 15/629,997.

[NVI18] NVIDIA Corporation. *NVIDIA Turing Architecture Whitepaper*, 2018. https://www.nvidia.com/content/dam/en-zz/Solutions/design-visualization/technologies/turing-architecture/NVIDIA-Turing-Architecture-Whitepaper.pdf, accessed on 12.11.2019.

[Pet12] Matt Pettineo. A quick overview of msaa. The Danger Zone, https://mynameismjp.wordpress.com/2012/10/24/msaa-overview/, October 2012. online, accessed 12.11.2019.

[PSK+16] Anjul Patney, Marco Salvi, Joohwan Kim, Anton Kaplanyan, Chris Wyman, Nir Benty, David Luebke, and Aaron Lefohn. Towards foveated rendering for gaze-tracked virtual reality. *ACM Transactions on Graphics (TOG)*, 35(6):179, 2016.

[RKLC+11] Jonathan Ragan-Kelley, Jaakko Lehtinen, Jiawen Chen, Michael Doggett, and Frédo Durand. Decoupled sampling for graphics pipelines. *ACM Transactions on Graphics (TOG)*, 30(3):17, 2011.

[SBPW19] Skyler Jonathon Saleh, Christopher J. Brennan, Andrew S. Pomianowski, and Ruijin Wu. Variable rate shading, February 2019. US Patent App. 15/687421.

[Tob] Tobii AB. *Specifications for the Tobii Eye Tracker 4C*. https://help.tobii.com/hc/en-us/articles/213414285-Specifications-for-the-Tobii-Eye-Tracker-4C.

[VST+14] Karthik Vaidyanathan, Marco Salvi, Robert Toth, Tim Foley, Tomas Akenine-Möller, Jim Nilsson, Jacob Munkberg, Jon Hasselgren, Masamichi Sugihara, Petrik Clarberg, et al. Coarse pixel shading. In *Proceedings of High Performance Graphics*, HPG '14, pages 9–18. Eurographics Association, 2014.

[YNS+09] Lei Yang, Diego Nehab, Pedro V. Sander, Pitchaya Sitthi-amorn, Jason Lawrence, and Hugues Hoppe. Amortized supersampling. *ACM Trans. Graph.*, 28(5):135:1–135:12, December 2009.

[YZK+19] Lei Yang, Dmitry Zhdan, Emmett Kilgariff, Eric B. Lum, Yubo Zhang, Matthew Johnson, and Henrik Rydgard. Visually lossless content and motion adaptive shading in games. *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, 2(1):6:1–6:18, 2019.