DOI: 10.1111/cgf.14018
EUROGRAPHICS 2020
R. Mantiuk and V. Sundstedt
(Guest Editors)

Volume 39 (2020), Number 2
STAR – State of The Art Report

A Survey of Temporal Antialiasing Techniques

Lei Yang, Shiqiu Liu, Marco Salvi

NVIDIA Corporation

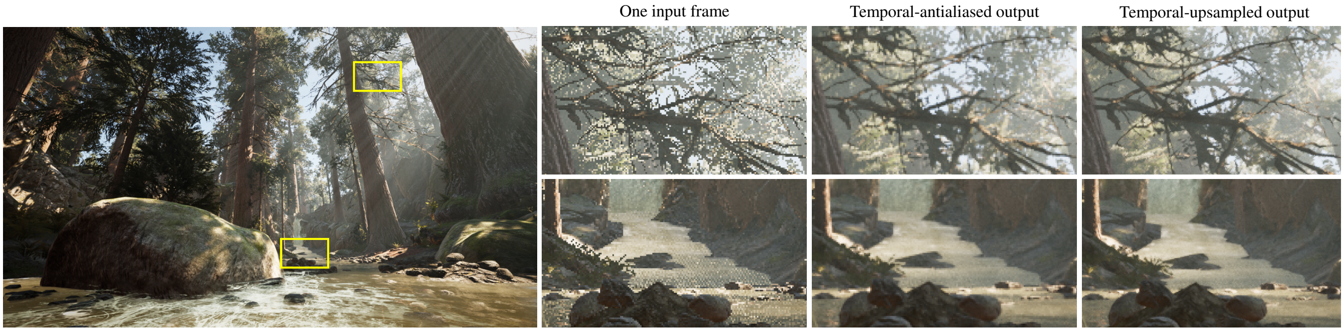


Figure 1: *Comparison of input, temporal-antialiasing and* 1.33× *temporal-upsampling results. Images are rendered by Unreal Engine 4.22.*

Abstract

Temporal Antialiasing (TAA), formally defined as temporally-amortized supersampling, is the most widely used antialiasing technique in today's real-time renderers and game engines. This survey provides a systematic overview of this technique. We first review the history of TAA, its development path and related work. We then identify the two main sub-components of TAA, sample accumulation and history validation, and discuss algorithmic and implementation options. As temporal upsampling is becoming increasingly relevant to today's game engines, we propose an extension of our TAA formulation to cover a variety of temporal upsampling techniques. Despite the popularity of TAA, there are still significant unresolved technical challenges that affect image quality in many scenarios. We provide an in-depth analysis of these challenges, and review existing techniques for improvements. Finally, we summarize popular algorithms and topics that are closely related to TAA. We believe the rapid advances in those areas may either benefit from or feedback into TAA research and development.

1. Introduction

Temporal Antialiasing (also known as Temporal AA, or TAA) is a family of techniques that perform spatial antialiasing using data gathered across multiple frames. Since its debut a decade ago, TAA and its variants have quickly become the de facto standard for antialiasing solutions in almost all video game engines and real-time 3D renderers. Traditionally, the name *temporal antialiasing* was used for techniques that aim to reduce temporal aliasing (also known as the wagon wheel effect) [KB83]. Nowadays, the term is somewhat of a misnomer as it is consistently used for spatial antialiasing (i.e. supersampling) using temporal samples. We choose to follow this new naming convention due to its wide acceptance in the real-time rendering community.

Prior to TAA, hardware-accelerated Multisample Antialiasing (MSAA) [Ake93] enjoyed the status of most used antialiasing technique in real-time engines. MSAA offers improved image quality at a small performance cost by limiting the rate of shading to a single per-primitive invocation for each pixel, effectively decoupling visibility determination from shading. Unfortunately, several challenges arise when MSAA is used in conjunction with deferred shading techniques [GPB04, Har04]. First, the hardware-accelerated mapping between visibility and shading is lost, effectively turning MSAA into costly supersampling antialiasing (SSAA). Second, even when such mapping between different sample types is recovered [SV12, KS14] the increased storage and memory bandwidth requirements of multisampled G-Buffers often negate any performance improvement due to running shaders at lower rate. Due to these difficulties various post-processing antialiasing techniques have been proposed to replace MSAA in deferred rendering (see Jimenez et al. [JGY*11] for an overview), but many of them suffer from temporal stability problems due to a lack of sufficient information to recover true pixel values.

TAA aims to resolve subpixel detail that is missing in single-sampled shading. Theoretically, this can be achieved by increasing per-pixel sampling rate using supersampling, which is often prohibitively expensive for real-time rendering with a brute-force implementation. By reprojecting shading results from previous frames, TAA effectively amortizes the cost of shading multiple samples per pixel over consecutive frames, and achieves supersampling at only a small cost on top of single-sample shading. From a quality standpoint, when compared to single-frame post-processing antialiasing techniques, TAA not only suppresses aliasing more effectively, but also generates more temporally stable results. TAA is also relatively easy to integrate into existing engines. It is usually implemented as a single post-processing pass, and requires only a feedback input of the output image from the previous frame.

TAA does face certain challenges that are not common in previous approaches. It is sometimes criticized for producing soft images, and can introduce artifacts such as ghosting and shimmering. We provide an in-depth analysis of such issues in existing TAA algorithms (Sec. 6), leading to future research directions in this area.

1.1. Algorithm overview

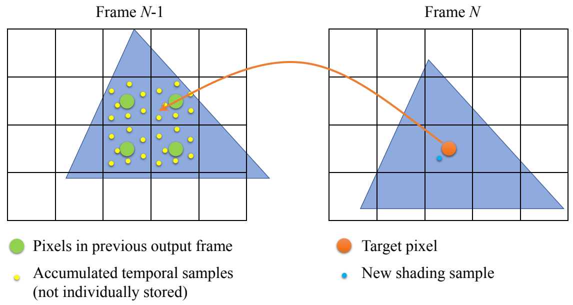

Frame N-1
Frame N
Pixels in previous output frame
Accumulated temporal samples (not individually stored)
Target pixel
New shading sample

Figure 2: *Conceptual illustration of how TAA amortizes spatial supersampling over multiple frames.*

The key idea behind TAA is to reuse subpixel samples accumulated from previous frames to effectively achieve supersampling. Figure 2 illustrates the conceptual process. Assuming a number of samples (yellow dots) were gathered for each pixel prior to frame N, and have been averaged and stored in the history buffer as a single-color value per pixel (green dot). For each pixel in frame N, we map its center location (orange dot) to the previous frame $N-1$ based on scene motion, and resample the history buffer at that location to obtain the history color for that pixel. With resampling, the history color represents the average of previously accumulated samples around that point. For the current frame N, we shade a new sample (blue dot) at a jittered location, and merge the result with the resampled history color. This produces the output pixel color of frame N, which then becomes the history for frame $N+1$.

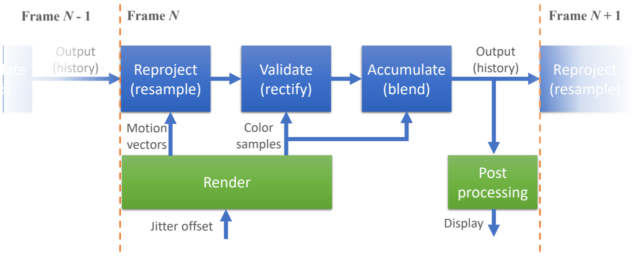


Figure 3: *Schematic diagram of a typical TAA implementation. Blue blocks are TAA components, and green blocks are stages of the render engine.*

Putting this concept into implementation, Figure 3 shows the components and data flow of a typical TAA algorithm. In order to evenly sample different locations within a pixel area, a sub-pixel jitter offset drawn from a sample sequence is used to shift the viewport each frame (Sec. 3.1). Temporally accumulated samples from the previous frame's output (history) are reprojected (resampled) using application-generated motion vectors to account for camera and object motion (Sec. 3.2). Due to change in occlusion, lighting or surface content, the fetched history data may be stale and inconsistent with the data from the current frame. Using stale data without scrutiny produces ghosting and image lag artifacts. History validation is a critical component of TAA algorithms as it identifies stale history data, and either rejects (Sec. 4.1) or rectifies (Sec. 4.2) the data to avoid introducing error to the current frame. This is a challenging and ill-posed task, as typically the only reliable information that we can use to validate previous data is the current sparsely-sampled frame. We review challenges associated with this problem in Sec. 6.

After history validation, the newly shaded sample is accumulated (blended) into the history color to obtain a new resolved color for each pixel (Sec. 3.3). This involves computing a weighted sum of the history color and the current frame sample, using either a fixed or adaptive blending weight. The blending weight has major influence over the tradeoff between antialiasing quality (convergence rate, temporal stability) and responsiveness (the speed to adapt to changes and reject errors in history). After being saved into a history buffer, the result can be further refined (e.g. image post-processing) and prepared for display.

There is a recent trend to reduce the input shading sample density to below one sample per pixel in TAA, which effectively makes it a spatial upscaling solution. Such techniques are commonly referred to as *temporal upsampling* (Sec. 5; also called temporal upscaling; notice that the name should not be confused with techniques that increase frame rate by using frame interpolation). Even though the scene is rasterized and shaded coarsely in each frame, samples that are gathered across multiple frames can jointly provide data of higher density and resolution. Temporal upsampling techniques leverage such data to recover fine details, and usually outperform pure spatial upsampling techniques in quality. Given an increasing demand in playing games in ultra-high-resolution today, temporal upsampling and its variants are on its way to becoming the new standard in modern game engines.

We review the performance characteristics of TAA and temporal upsampling in Sec. 7. There are also a few other techniques, closely related to TAA, that share similar building blocks as well as challenges. Examples include variable rate shading, temporal denoising, and machine learning-based reconstruction techniques. Sec. 8 gives an overview of these problems and solutions. All these techniques use temporally accumulated data to improve visual quality or to accelerate brute-force solutions.

2. A brief history of Temporal Antialiasing

Prior to the introduction of TAA, techniques that exploit temporal coherence (i.e. frame-to-frame coherence) to speed up rendering computation have been available for offline and interactive ray-tracing systems for decades. An overview to this family of techniques is given by Scherzer et al. [SYM*12]. Such methods typically aim at reusing shading results across multiple frames to save expensive computation, by using either proxy geometry warping or pixel scattering to reproject data across frames. Another related group of techniques, called motion-compensated filtering, use spatio-temporal filters to reduce noise in image (video) sequences [BKE*95]. Like TAA, such techniques also need to handle time-variant signal, using adaptive nonlinear filters [OST93] or signal decomposition methods [KLB95].

Although the idea of using data across multiple frames for evaluating per-pixel integrals can be traced back to the *Accumulation Buffer* [HA90], it was not until the development of the *Reverse Reprojection Cache* (also known as history buffer) that this idea became practical for real-time rendering. Nehab et al. [NSL*07] and Scherzer et al. [SJW07] both discovered that by computing a per-pixel reprojection vector in the forward rendering pass, the idea of accumulation buffer can be extended to handle scene motion. More precisely, reprojection vectors track object and camera motion between frames, and allows each pixel in the current frame to retrieve data accumulated at the exact same surface point from previous frames. Both works proposed to use this technique to improve the quality of shadow-mapping.

The *Amortized Supersampling* paper [YNS*09] first proposed to use data reprojection as a general solution for shading antialiasing. The authors identified two major difficulties in TAA to produce high-quality output: excessive spatio-temporal blur due to resampling error, and slow adaptation to underlying signal changes. Through a theoretical analysis, they proposed to store the history buffer in 2× the target resolution to reduce blur, and introduced an adaptive blending scheme to identify and limit error caused by blur and signal changes.

An important technique that makes TAA more robust in applications is the idea of using current frame samples to rectify reprojected history data, first introduced by Lottes [Lot11]. Later refined and referred to as neighborhood clamping (or clipping) [Kar14, Sal16], this technique relies solely on the current-frame input color buffer to reduce artifacts caused by stale history data. Therefore, it significantly lessens the need for other history validation heuristics and additional input data, making TAA more robust and easy to integrate.

Since 2010, TAA quickly became popular in game engines and newly released titles. *Halo: Reach* [Lea10] uses alternate sampling patterns across frames and then adaptively blend the last two frames to achieve quality similar to 2× supersampling. *Crysis2* [Sou11] adds reprojection to allow accumulating multiple frames, and uses motion and depth-based consistency check to minimize artifacts. Starting in 2012, NVIDIA offers TXAA [NVI12] as their TAA solution on Kepler+ architectures, and enabled it in nearly 20 game titles. Major commercial game engines added TAA either as a standard antialiasing approach (e.g. Unreal Engine 4 [Kar14], Unity [Uni16]), or in combination with other spatial antialiasing solutions such as morphological antialiasing [Res09] (e.g., SMAA in CryEngine 3 [JESG12], and TSCMAA [Kim18]). Other notable titles and engines that implemented and enhanced TAA include *Dust 514* [Mal12], the *Decima* engine [Val14, dCI17], *Far Cry 4* [Dro14], *Quantum Break* [Aal16], *Rainbow Six Siege* [EM16], *Inside* [Ped16], and *Uncharted 4* [Xu16].

TAA has been coupled with the idea of temporal upsampling since its inception. Yang et al. [YNS*09] proposed a reconstruction algorithm to accumulate details at subpixel level in TAA. Herzog et al. [HEMS10] introduced a joint spatio-temporal filter to perform upsampling in image space. Similar ideas have been used in game engines to meet quickly evolving demand of high-resolution displays [Mal12, Val14, Aal16, Epi18]. More recently, new techniques that extend MSAA to shade less than once per pixel (e.g. checkerboard rendering [Lea16, EM16, Wih17], variable rate shading [PSK*16, XLV18]) have come to rely on TAA to filter or fill in missing shading details. These methods that decouple the shading rate from the visibility sampling rate can reduce shading costs while resolving geometry edges at a dense resolution, but often at the cost of significant changes to the engine rendering pipeline.

3. Accumulating temporal samples

From a theoretical standpoint, spatial antialiasing requires convolving the continuous shading signal with a low-pass pixel filter to suppress excessive high-frequency components before sampling the signal. In practice, we supersample the continuous signal before applying the low-pass filter on the discrete samples of the signal. TAA amortizes the cost of supersampling by generating, aligning, and accumulating these spatial samples over multiple frames. This section gives an overview of these procedures.

3.1. Jittering samples

For practical reasons, most TAA implementations render a single sample per pixel in each frame. A common approach to generate a different sample in each frame for all pixels is to add a viewport sub-pixel jitter offset to the camera projection matrix. The per-frame jitter offset is usually drawn from a well-distributed sample sequence, so that every pixel is evenly covered by samples generated over multiple frames.

Since an object can appear or become disoccluded at any time, the first sample of a pixel beginning accumulation can start from any index in the sequence. To allow fast convergence, an ideal sequence therefore must have the property that any subsequence of any length must be somewhat evenly distributed in the pixel domain. Certain low-discrepancy sequence such as Halton or Sobol have this property (see Christensen et al. [CKK18] for an overview). Depending on the targeted quality and the effective accumulated sample count, the length of the sequence can often be limited to a relatively small number. For example, Unreal Engine 4 [Epi15] uses a 8-sample sequence from Halton(2, 3) by default, Inside [Ped16] uses a 16-sample sequence from Halton(2, 3), SMAA T2x [Jim16] uses Quincunx, and Quantum Break [Aal16] uses rotated grid offsets. When the target antialiasing filter kernel is any non-negative kernel other than a common box filter, importance sampling can be used to efficiently sample the domain and avoid explicit weighting of the individual samples [YNS*09].

With GPU devices that support Direct3D *programmable sample position* feature tier 2, developers can specify a sample pattern across a 2×2 pixel grid, such that adjacent pixels do not have to share the same set of samples. This helps to reduce certain aliasing artifacts caused by the interference between the sample pattern repetition frequency and the underlying signal frequency. NVIDIA utilized this feature in the Multi-Frame Anti-Aliasing (MFAA) technique since their Maxwell architecture [Gru15]. Drobot [Dro14] also uses this feature to sample in FLIPQUAD pattern over two frames using $2\times$ MSAA.

Motion in the scene may disrupt a well-designed sample pattern by displacing samples accumulated over frames. With a short recurrent sequence, certain motion speed may cause sample locations from multiple frames to cluster in world space, leading to biased results (blurry or aliased). This is one of the causes of TAA quality degradation under motion (the others are discussed in Sec. 6). By randomizing the jittering pattern, we can break regular cycles in the pattern and lower the risk of running into those pathological scenarios.

With TAA, sampling and integrating over pixel area provides antialiasing for both geometry and texture. Since textures are usually filtered using mipmapping, they may risk being overblurred by TAA in the output. Therefore, typically a mipmap bias is applied to the forward pass where textures are sampled. This is particularly important for temporal upsampling [Epi18]. The level of mipmap bias should compensate for the ratio between the effective sample density and the input pixel density. For example, if the effective sample count per-input-pixel is expected to be 4 (see Sec. 3.3 and Figure 4 for how to estimate effective sample density), then the mipmap bias is calculated as $-\frac{1}{2}\log_2 4 = -1.0$. In practice, a less aggressive bias between this value and 0 is sometimes preferred to avoid hurting temporal stability as well as texture cache efficiency.

Since TAA is essentially an amortized sampling process, it can also be applied to other effects that require integration of samples, such as ambient occlusion, shadow, order-independent transparency, reflection, and diffuse global illumination. Providing randomized input to these effects (preferably with high-dimensional low-discrepancy sequences [CKK18]) allows TAA to integrate them along with screen-space antialiasing samples. For example, Unreal Engine 4 relies on TAA to denoise many effects that use stochastic sampling and dithering [Kar14].

3.2. Data reprojection between frames

With scene motion between frames, each pixel needs to compute its corresponding location in the previous frame to fetch history data. Reverse reprojection [NSL*07, SJW07] is widely used to perform this step. During scene rasterization, the geometry is transformed twice, once using previous frame's data, and once using current frame's data. The offset between the current and the previous location of every pixel is then stored into a motion vector texture (also called velocity in some engines), which is later used by the TAA algorithm to obtain the reprojected history buffer coordinate of every target pixel.

In order to save framebuffer bandwidth, some engines only explicitly compute and store motion vectors for animated and moving objects, with all affected pixels tagged in a stencil texture. For pixels that are not tagged, their location in the previous frame can be determined on the fly in the TAA pass by reconstructing the 3D clip-space coordinates of each pixel using the depth buffer, and projecting it to the previous frame using the camera matrices of the previous and current frame:

$$p_{n-1} = M_{n-1} M_n^{-1} p_n, \tag{1}$$

where $p_n = (\frac{2x}{w} - 1, \frac{2y}{h} - 1, z, 1)$ is 3D clip-space location of the current frame pixel in homogeneous coordinates, and M_{n-1} and M_n are previous and current frames' view-projection matrices, respectively. The resulting position is obtained after a perspective division. Unreal Engine 4 [Epi15] uses this approach by default.

The reprojected coordinates in the history buffer often contains a subpixel offset, and there is no longer a 1 : 1 pixel mapping between the source and the target frames. A resampling step is needed for obtaining history at each pixel, so that no distortion (fractional offset snapping) artifact is introduced. Typically, a hardware accelerated bilinear texture fetch or a bicubic texture filtering is used for this purpose. Section 6.1 discusses commonly used resampling filters and how they affect image quality.

Since motion vectors cannot be antialiased, reprojection using motion vectors may reintroduce aliasing artifacts to smooth, antialiased edges along object boundaries of moving objects. A simple approach to avoid such artifacts is to dilate the foreground objects when sampling motion vectors, so that all boundary pixels touched by the edge are reprojected along with these objects [Kar14]. Typically, a small 4-tap dilation window is used. An adaptive scheme has also been proposed by Wihlidal [Wih17] to only fetch and compare depth value when motion vectors diverge.

3.3. Sample accumulation

It is impractical to store all samples accumulated for each pixel over previous frames. In most TAA implementations, the accumulated samples for each pixel are averaged and stored as a single color, which is both the output of the current frame, and the history input of the following frame. This iterative accumulation process can be written as [YNS*09]:

$$f_n(p) = \alpha \cdot s_n(p) + (1 - \alpha) \cdot f_{n-1}(\pi(p)), \tag{2}$$

where $f_n(p)$ is frame n's color output at pixel p, α is the blending factor, $s_n(p)$ is frame n's new sample color at pixel p, and $f_{n-1}(\pi(p))$ is the reprojected output (history color) from previous frame, using the reprojection operator $\pi(\cdot)$ and resampling.

The blending factor parameter α balances the relative contribution of new sample and the history color. Most TAA implementations today use a fixed α, which effectively leads to a recursive

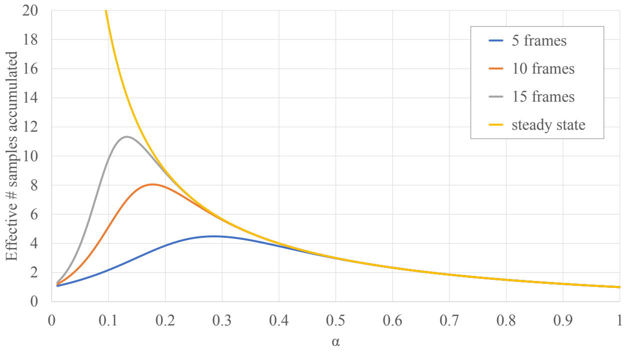


Figure 4: *Effective number of accumulated sample when using exponential smoothing with a fixed α. The curves are plotted based on Eq. 30 listed in the appendix of Yang et al. [YNS*09].*

exponential smoothing filter [NSL*07, SJW07, YNS*09]:

$$f_n(p) = \alpha \cdot \left(s_n(p) + (1-\alpha)s_{n-1}(p) + (1-\alpha)^2 s_{n-2}(p) + \dots \right). \tag{3}$$

This equation progressively assigns older samples lower weights, which in some cases is a desirable property, since older samples are more likely to become stale due to changes in the scene. Note that from a variance reduction perspective, this is suboptimal. The optimal variance reduction is achieved when all samples are weighted equally in the sum [YNS*09]. This is not the case when a fixed α is used, which may compromise the quality of the results at both low and high sample counts. Figure 4 shows the relation between α and the effective number of samples accumulated if uniform weights were used, with a matching degree of variance reduction. For example, with a commonly used $\alpha = 0.1$, a result from 5 accumulated frames is equivalent to 2.2 samples per pixel, 10 frames equivalent to 5.1 samples, and 15 frames equivalent to 9.8 samples. At steady state with an infinite number of input frames accumulated, $\alpha = 0.1$ results in 19 effective samples at its best.

As an alternative to weighting samples equally, we can store a per-pixel accumulated sample count $N_t(p)$ in the alpha channel of the history buffer [YNS*09]. This value is initialized to 1 whenever the pixel is refreshed, and is incremented every frame. By setting $\alpha = 1/N_t(p)$, Eq. 2 assigns the same weight to all history samples. It then enables optimal convergence rate at the cost of an additional storage channel. This method is used in some applications that require fast convergence after disocclusion [WMB19, KIM*19], particularly when target frame rate is low.

It should be noted that when α is small, the result becomes susceptible to resampling errors (Sec. 6.1) or temporal lag (Sec. 6.2). To avoid those artifacts, α is often clamped to a lower bound to ensure a minimum amount of history refresh. Yang et al. [YNS*09] propose a theory to derive an adaptive α lower bound based on fractional motion speed, such that accumulation of bilinear resampling error is prevented. CryEngine 3 [Sou11] and Unreal Engine 4 [Epi15] use simple heuristics to increase α with larger motion speeds. Also, when history rejection (Sec. 4.1) or rectification (Sec. 4.2) happens, effective accumulated sample count $N_t(p)$, if tracked, needs to be updated accordingly to match α [YNS*09].

3.3.1. Sample accumulation in HDR color space

In most rendering engines, physically-correct post processing effects are implemented in linear HDR (i.e. radiometrically linear) color space. It is desirable to place TAA before all these effects to avoid aliased high-energy color samples being exaggerated by effects such as bloom or lens flare. This requires TAA to take input samples in linear HDR space. On the other hand, since tonemapping is typically non-linear, filtering operations such as antialiasing are best applied in the post-tonemapped space to produce correct edge gradient on display. A workaround to solve this conflict is to tonemap the samples before applying TAA, and invert tonemap the output back to linear HDR space to feed the rest of the post processing chain [Pet15]. In practice this is typically implemented with an invertible tonemapping operator like the Reinhard operator $1/(1+x)$, which acts as a surrogate to the final tonemapping function.

To avoid desaturating colors when tonemapping, Karis [Kar13, Kar14] uses a luminance-adaptive weight when accumulating new samples to history:

$$w(c) = \frac{1}{1+L(c)}, \tag{4}$$

where $L(c)$ is the luma channel of color c, and $w(c)$ is applied as a bilateral weight when blending current frame samples and history color using Eq. 2. This effectively tonemaps the input samples and can avoid generating extremely high-energy color pixel in the output, sometimes referred to as "fireflies". A similar technique is implemented in *Call of Duty: Advanced Warfare* [Jim14] to stabilize bloom effect.

4. Validating history data

In reality, history pixel data reprojected from the previous frame should never be trivially reused without checking. They can either be invalid due to scene occlusion changes, or stale due to lighting and shading changes. Failure to properly handle such cases can result in strong temporal artifacts such as ghosting. In this section, we review commonly used techniques for validating history data.

4.1. History rejection

A straightforward approach to handle stale or invalid data is to reject the data when error is detected. Rejection is done by setting α to 1 in Eq. 2. A soft refresh is also possible by limiting α to a lower bound threshold, allowing a faster than regular update to the accumulated history color. In general, there are two types of sources that can be used to determine the confidence of the history: geometry data (depth, normal, object ID and motion vector), and color data.

Geometry data is typically used to identify invalid history data reprojected from mismatching surfaces due to occlusion changes. Figure 5 shows an example. By comparing reprojected depth of p and previous depth of $\pi_{t-1}(p)$ against a small error tolerance, we can identify disoccluded pixels like p_1 and avoid reusing history data there [NSL*07]. To achieve more robust matching, other geometry information such as surface normal and object ID can also

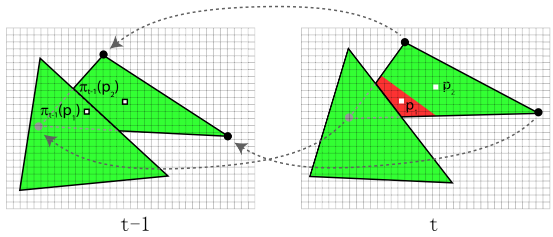


Figure 5: *Checking for disocclusion changes. Pixel coordinates p_1 in frame t is reprojected to coordinates $\pi_{t-1}(p_1)$ in frame $t-1$, where it is occluded by another piece of geometry. By checking geometry data (depth, surface normal, or object ID), the occlusion can be identified and properly handled. In contrast, pixel p_2 finds a match and can safely be reused. Diagram adapted from Nehab et al. [NSL*07]*

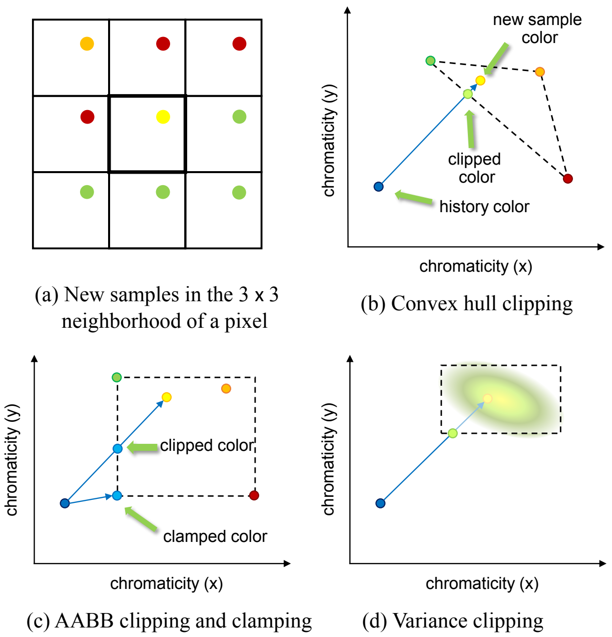
be used additional indicators of consistency. In the TAA implementation of *Crysis 2* [Sou11], Sousa also proposes to increase α when motion speed increases, in addition to a depth-based consistency check. Note that, although geometry information tends to correlate strongly with shading color, it is only a part of the shading inputs. Shading changes in effects like lighting, shadows and reflections cannot be detected using geometry information. The unfilterable nature of geometry data may re-introduce discontinuity into smooth, antialiased edges. Therefore, relying only on geometry information to reject history is often a brittle solution.

Comparing color between history buffer f_{n-1} and current frame samples s_n can provide a more direct indicator of the validity of history data. This is useful for detecting when history data is either stale due to visible lighting or shading change, or distorted due to resampling error and incorrect motion vectors. Since the current frame samples s_n are expected to be aliased (otherwise we would not need antialiasing), directly comparing $f_{n-1}(p)$ and $s_n(p)$ gives us biased estimates of the error, meaning that it can be both spatially and temporally unstable. Yang et al. [YNS*09] observe this and propose to filter the error estimate:

$$\varepsilon_n(p) = B_3 * (f_{n-1} - s_n)(p), \quad (5)$$

where B_3 is a box filter of radius 3. The error estimate is then used to set a lower limit on α to enforce a minimum amount of refresh of the history. Herzog et al. [HEMS10] compute the temporal gradient of the reconstructed color, and apply spatial and temporal smoothing to the gradient to improve stability. Malan [Mal12] computes the color extent (bounding box) of the neighborhood in current frame samples. The history color is kept if it is inside or close to the bounding box. The technique also biases α based on the extent of the color bounding box: a small bounding box (flat color) would increase α to avoid visible ghosting, whereas a large bounding box (sharp discontinuity) would reduce α to increase accumulated samples and suppress temporal artifacts. The use of color bounding box is related to history rectification techniques (Sec. 4.2).

One option to reliably detect temporal changes is to compare raw samples that are shaded at the exact same surface location across frames. In *Killzone: Shadow Fall* [Val14] and following *HRAA* technique [Dro14], an alternating odd-even frame sample pattern (e.g. temporal FLIPQUAD) is used, such that frame N and $N-2$ have matching sample locations. Samples in those frames can therefore be directly compared when no motion is present. For ray-tracing applications with a higher shading rate budget, Schied [SPD18] proposes to forward-project a subset of previous shading samples to the current frame, and shade new samples at exact same locations to reliably compute shading change (temporal gradient). The sparsely computed gradients are then upscaled with a joint-bilateral filter, and are used to control α to keep the history always up to date.



(a) New samples in the 3 x 3 neighborhood of a pixel

(b) Convex hull clipping

(c) AABB clipping and clamping

(d) Variance clipping

Figure 6: *Common history rectification techniques illustrated. For simplicity we visualize the color samples and their extent over a 2D chromaticity space. In practice they are defined in a 3D color space (e.g. RGB, or YCoCg).*

4.2. History rectification

Rejecting stale or invalid history data effectively resets the per-pixel integration process, and can lead to increased temporal artifacts. History rectification ameliorates this issue by making the otherwise rejected data more consistent with the new samples. As in most TAA implementations today, the rectified history is then blended with the current frame, leading to more visually acceptable results.

Most history rectification algorithms assume that the current frame samples provide reliable information of the extent (or distribution) of the colors covered by each pixel. Since the current frame samples are often sparse and aliased, a 3×3 or larger neighborhood of each pixel is considered for pulling the color information (Figure 6(a)), with the assumption that more neighboring samples provide a better estimation of the distribution of local color variations. The convex hull of the neighborhood samples in color space represent the extent of the colors we expect around the center pixel. If the history color falls inside the convex hull, it is assumed to be consistent with the current frame data and can be reused safely. If it falls outside the convex hull, we need to rectify it to make it consistent with the current frame data. We do so by connecting history color with the current-frame sample color, and clip the line segment against the convex hull. The intersection point is the best estimate of history that is consistent with the current frame samples.

In practice, computing a convex hull and the ray-hull intersection per pixel can be prohibitively expensive. A common approximation to this is to compute an axis-aligned bounding box (AABB) of the colors using a min/max filter, and either clip or clamp the history color against the AABB (Figure 6(c)) [Lot11, Kar14]. This is related to Malan's rejection technique [Mal12], where the same AABB is used to either accept or reject history data (see Section 4.1). While an AABB is simple to compute, the approximation trades off temporal stability for an increased likelihood of ghosting artifacts. Karis [Kar14] further reduces ghosting by clipping to color AABB in YCoCg space. Compared to RGB, the YCoCg space typically leads to a tighter AABB, because it decorrelates chroma (Co, Cg) channels from the luma (Y) channel, in which local contrast usually dominates.

Note that min/max filtering of very dark or bright outliers can inflate the volume of the bounding box, leading to accepted history data that would otherwise be rectified. Variance clipping [Sal16] addresses outliers by using the local color mean and standard deviation to center and size the color extents used for rectification:

$$\begin{aligned} C_{\min} &= \mu - \gamma\sigma, \\ C_{\max} &= \mu + \gamma\sigma, \end{aligned} \tag{6}$$

where μ and σ are the mean and standard deviation of the color samples in the local neighborhood, and γ is a scalar parameter typically chosen between 0.75 and 1.25. The computed $C_{\min}$ and $C_{\max}$ are used in place of the AABB extent for history clipping (Figure 6(d)).

One special technique related to history rectification is the pixel history linear models by Iglesias-Guitian et al. [GMK*16]. Their method derives pixel color from a running "feature", which is accumulated, stable low frequency color information at each pixel. The feature is transformed to output pixel color by a per-pixel parameterized linear model, updated per frame to keep up with shading signal changes. During update, neighborhood clamping is also used to refine the color prediction model. The technique as a whole aims to find a superior balance between temporal coherence and responsiveness, at the cost of more per-pixel storage of history data.

5. Temporal upsampling

Temporal upsampling is a natural extension of TAA, by further reducing the effective sampling rate from one sample per pixel to a fraction of a sample per pixel. This is often desired by applications with heavy pixel workload, and is becoming increasingly popular in games targeting high-resolution displays. Temporal upsampling essentially accumulates lower-resolution shading results, and produces higher resolution images that often contain more details than pure spatial upsampling results. Although it shares many basic building blocks with TAA, such as sample jittering, reprojection and history validation techniques, the accumulation step requires special handling to achieve the best image quality. In this section, we propose a general framework that covers a variety of existing upsampling techniques [YNS*09, HEMS10, Mal12, Aal16, Epi18], and discuss some improvement strategies.

5.1. Scaling-aware sample accumulation

Temporal upsampling differs from TAA in that input samples are accumulated into a buffer of higher pixel density. Since there is no longer a 1 : 1 mapping between input samples and output pixels, we first upscale the input samples to the output resolution using:

$$\bar{s}_n(p) = \frac{1}{w(p)} \sum_{i \in \Omega(p)} \delta(o_i) s_i, \tag{7}$$

where $\Omega(p)$ is a collection of input samples in a fixed-sized neighborhood of output pixel p, s_i is the i-th sample in $\Omega(p)$, o_i is distance between s_i and p, and δ is a reconstruction filter kernel chosen for the target pixel. The normalization factor $w(p)$ is the sum of weights:

$$w(p) = \sum_{i \in \Omega(p)} \delta(o_i). \tag{8}$$

Essentially, Eq. 7 computes a weighted sum of input samples that fall under the reconstruction kernel δ of the target pixel. The reconstruction kernel δ can either be a Gaussian kernel [HEMS10, Epi18], a 1-pixel-wide bilinear tent (hat) function [YNS*09], or other filtering kernels with wider support than the input sample grid size. The fixed-sized neighborhood of $\Omega(p)$ is chosen to match the filter size.

While the upscaled input samples $\bar{s}_n(p)$ could be directly plugged into Eq. 2 for accumulation, doing so will not produce output images with desired sharpness. By nature of upscaling, the set of values $\bar{s}_n$ contain mixed quality output samples, where some with proximity to input samples receive direct copies of those (higher quality), and others are interpolated from more distant samples (lower quality). With a fair per-frame jittering pattern, an output pixel should receive a higher quality sample every few frames.

The key to producing consistent high quality in all pixels is to selectively blend input samples into history based on a confidence-of-quality factor $\beta(p)$ defined for each output pixel p. For that purpose, the recursive accumulation step (Eq. 2) can be rewritten as:

$$f_n(p) = \alpha \cdot \beta(p) \cdot \bar{s}_n(p) + (1 - \alpha \cdot \beta(p)) \cdot f_{n-1}(\pi(p)). \tag{9}$$

The confidence factor $\beta(p)$ is a value in $[0,1]$ that is designed to bias the accumulation towards retaining the history when the quality of $\bar{s}_n(p)$ is expected to be low. Unreal Engine 4 [Epi18] uses the following term to compute the confidence:

$$\beta(p) = \max_{i \in \Omega(p)} \delta(o_i), \tag{10}$$

where $\delta(o_i)$ is the same Gaussian kernel used in Eq.7. Most other temporal upsampling algorithms that scatter samples into the high-resolution accumulation buffer can be viewed as using a box kernel as $\delta(o_i)$ that matches the target pixel size [YNS*09, Mal12, Aal16].

As an example, consider the special case where the input samples are of 2× lower resolution in each dimension, and kernel $\delta(p)$ is a box of target pixel size. There is a fixed $1:4$ mapping between input samples and output pixels. Since samples are jittered inside the source pixels, in each frame only one out of four pixels in target 2×2 pixel block has a valid sample that fall into its covered area. In that pixel the box kernel in $\beta(p)$ evaluates to 1, and the sample is accumulated into $f_n(p)$. The other three pixels all have $\beta(p)$ evaluates to 0, leaving the history values unchanged. Notice that arbitrary upsampling ratio is supported by this scheme. Examples of different upsampling ratios are illustrated in Figure 7.

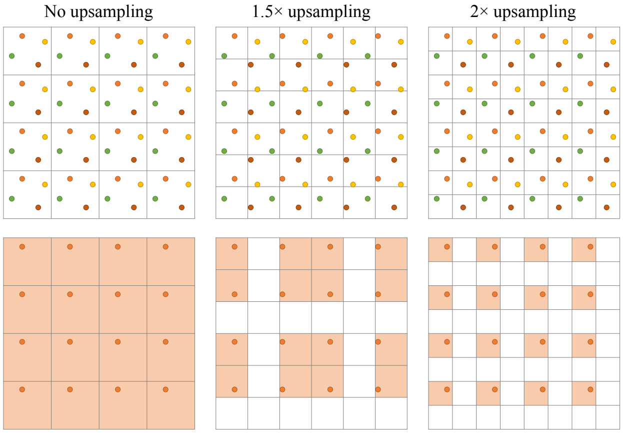


Figure 7: *Spatio-temporal sample accumulation in temporal upsampling. Upper: Input color samples from four frames (green, brown, orange, yellow) and how they overlap target pixel grid with 1× (no upsampling), 1.5×, and 2× upsampling rate per dimension. Lower: When using a target pixel-sized box kernel as $\delta(p)$, the updated pixels in one frame (highlighted) are those touched by the samples shaded in that frame.*

5.2. Miscellaneous improvements

History validation in temporal upsampling shares the same type of challenges with that in TAA, but it is more difficult because sparser input samples relative to output pixels provide less reliable information about the real content. As will be further discussed in Sec. 6.1.2, Sec. 6.2 and Sec.6.3, the quality of history rectification suffers when input samples are sparse, which would ultimately lead to worse trade-off between ghosting, temporal instability and blurriness artifacts.

During temporal upsampling, the typical 3×3 neighborhood used for color bounding box calculation in the input image now covers more than 3×3 pixel area in the target image, leading to an increased chance of overestimating the color gamut. To avoid that, Unreal Engine 4 [Epi18] computes a shrink-sized neighborhood based on subpixel offset for color bounding box calculation. Samples far away from the target pixel center are not included by the bounding box, leading to less ghosting artifacts. Similarly, Andersson et al. [ANS*19] propose to clamp each history pixel only to its immediate neighboring 2×2 samples.

To improve temporal stability and sharpness in temporal upsampling, *Quantum Break* [Aal16] choose to adaptively relax the color bounding box based on pixel motion speed. On objects that are close to stationary, accumulating samples is encouraged over aggressive clamping. This will lead to more stable and sharper images during static shots, but at the cost of potential ghosting on dynamic lighting and shadows, or animated textures, since these changes are not reflected by the motion vectors.

In the context of applying temporal upsampling to ray tracing, Andersson et al. [ANS*19] observe significant quality differences when the sampling order interacts with motion. With regular sampling patterns, they recognize two main sampling order modes, hourglass and bowtie. They propose an adaptive scheme to dynamically switch between the two modes per pixel based on the fractional part of motion speed and motion direction, resulting in reduced image error and visible artifacts.

5.3. Checkerboard rendering

Checkerboard rendering (CBR) is a special type of temporal upscaling technique. It became popular after game console platforms, such as PlayStation 4 Pro and XBox One X, use it as a tool to target 4K resolution [Lea16]. Unlike other temporal upscaling techniques that rely on random jittered samples to cover the target pixel grid, CBR deterministically shades two opposing pixels on the diagonal line in each 2×2 pixel quad (hence the name "checkerboard") in one frame, and flips that pattern next frame to shade the other two pixels in the quad. The technique to shade samples in a checkerboard pattern can be a 45° rotated framebuffer [dCI17], hardware accelerated 2× MSAA [EM16, ML18], programmable sample positions [Dro14], or variants of target-independent rasterization [Wih17]. Such techniques often leverage hardware dependent and are orthogonal to the topics of this paper.

Since every single frame only shades half of the target pixels, temporal data reuse is needed to generate a complete image in each frame. The common TAA and temporal upscaling techniques such as reprojection, sample accumulation and history clamping are directly applicable to CBR. The distribution of new samples in a checkerboard pattern is also inherently more friendly to interpolation or neighborhood clamping steps than earlier temporal upsampling attempts like temporal interlaced rendering [Val14]. But a CBR implementation usually requires significant engine code changes, due to the non-standard sample layout and the large number of shading and postprocessing passes that interact with it. Like temporal upscaling, CBR techniques usually resolves upscaling and antialiasing in a single pass. A special post-processing pass can be used to avoid producing visible saw-tooth pattern in final rendering [EM16]. With support of special hardware, visibility information such as depth and object/primitive ID can be decoupled from shading and sampled at target resolution to improve the quality of final pixel resolve [Wih17]. Unlike temporal upsampling, CBR assumes a fixed $1:2$ input-to-output pixel ratio. Therefore, it is sometimes coupled with a traditional resolution scaling technique to support varying workloads [Wih17].

6. Challenges

Amortizing sampling and shading across multiple frames does sometimes lead to image quality defects. Many of these problems are either due to limited computation budget (e.g. imperfect resampling), or caused by the fundamental difficulty of lowering sampling rate on spatially complex, fast changing signals. In this section we review the common problems, their causes, and existing solutions.

6.1. Blurriness

There are two sources that accounts for the “soft” look of TAA results: the error introduced by resampling when history is reprojected, and the error introduced by history rectification where accumulated detail in history color is clipped or clamped.

6.1.1. Resampling blur

Resampling happens when motion causes target pixels to reproject to fractional pixel locations in the previous frame. Using a hardware-accelerated bilinear texture filtering can efficiently interpolate the color value of the four nearest pixels in the history buffer, around the reprojected pixel center ($\pi(p)$ in Eq. 2). However, bilinear filters soften the resampled image, since they perform poorly in retaining high-frequency contents. Since the history is reprojected in every frame, the filtering error accumulates and quickly result in an objectionable, blurry look. This blur can be quantified using a statistical analysis [YNS*09]. This analysis led to strategies to avoid the excessive blur by both clamping α to an adaptive lower threshold based on motion speed (because the amount of blur varies with fractional motion speed), and using a higher resolution history buffer (in order to relax the bandwidth of the bilinear filter). In applications that do not require regular sampled outputs (e.g. ray tracing [CSK*17]), forward reprojection can be used to avoid resampling, together with explicit hole filling steps.

Other than limiting α and increasing history resolution, advanced resampling filters are also commonly used to reduce the reprojection blur. Drobot [Dro14] uses Back and Forth Error Compensation and Correction (BFECC) [DL03, NABW12] to significantly reduce the error of bilinear sampling. Cubic interpolating splines like Catmull-Rom [CR74] are also commonly used as a resampling function in TAA to avoid blur accumulation. For example, TAA in Unreal Engine 4 [Kar14, Epi15] and SMAA T2x [Jim16] both use an approximate, optimized version of Catmull-Rom interpolator with as few as five bilinear texture fetches. As shown in Figure 8, both BFECC and Catmull-Rom offer significant improvement over bilinear results after 100 frames of repeated resampling of accumulated history. Recently, Nehab and Hoppe [NH14] study generalized filter kernels that demonstrate superior resampling quality over traditional convolutional filters. The latest filter of this kind [SN15] produces almost visually lossless results after 100 frames of resampling (Figure 8, last column shows the results using their cubic quasi-interpolator). The generalized filters are implemented as a traditional linear, quadratic or cubic resampling filter followed by a linear recursive filter pass, which can be efficiently implemented on the GPU using parallel recursive filtering [NM16].

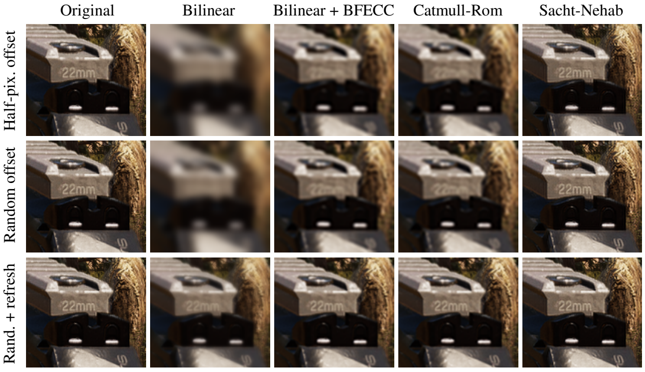


Figure 8: *Comparison of resampling blur using different filtering techniques. All images except the original are generated by repeated translating and resampling the image 100 times. The first row of images uses half-pixel translation in both horizontal and vertical directions to demonstrate worst-case resampling quality. The second row uses random translation offset to simulate the average case. The third row applies exponential smoothing (Eq.2) with* $\alpha = 0.1$ *after every resampling step, using ground-truth translated image as input* (s_n)*. It shows that the resampling blur can be ameliorated to some extent when new input is blended into the resampled history every frame.*

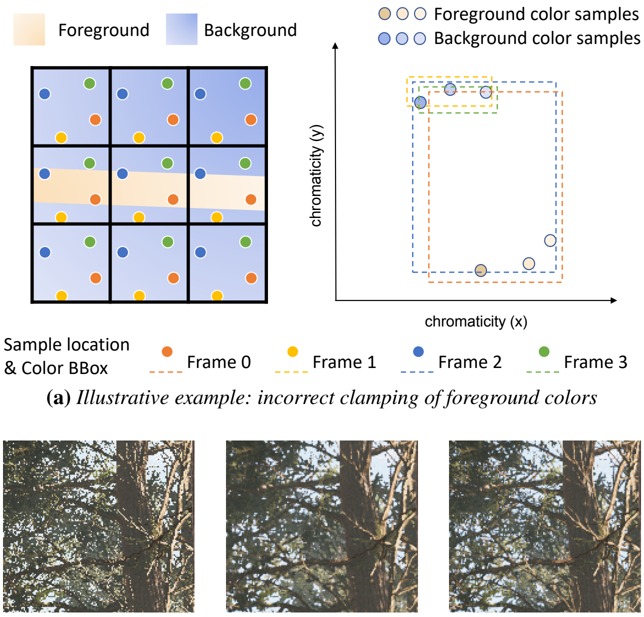
6.1.2. History rectification-induced blur

History rectification techniques (Sec. 4.2) are based on the assumption that the current frame samples in the neighborhood of each pixel contain the entire gamut of surface colors covered by that pixel. Since the current frame samples are sparse (≤ 1 sample per pixel), the hope is that any thin feature in geometry or shading is at least covered by one pixel in any 3×3 neighborhood it touches. Unfortunately, with highly detailed content, this assumption is often violated. Figure 9(a) demonstrates a common case where a subpixel wide thin line is missing in the input of certain frames, causing the underestimated color bounding box to clip or clamp away the line color from history. This happens commonly in highly detailed scenes, where small, sharp features are smoothed out in the output (Figure 9(b-d)).

Since history rectification is essential in most modern TAA implementations, and must be adjusted to avoid more severe artifacts like ghosting (Sec. 6.2), detail preservation is sometimes sacrificed. To regain some of the sharp, detailed appearance, some renderers apply a sharpening filter, such as a discrete Laplacian, to the result of TAA output [Dro14, Xu16, Wih17]. An adaptive sharpening filter [Kra19] can be used to avoid producing out-of-gamut colors in sharpening.

6.2. Ghosting and temporal lag

Ghosting is a common type of artifact seen in screen-space temporal data reuse techniques. Ghosting often appears in newly disoccluded regions caused by moving objects, where invalid history data is not completely removed from the current frame by history rejection or rectification techniques. That region can then appear as a second copy of the moving object, and it may be carried through



(a) *Illustrative example: incorrect clamping of foreground colors*

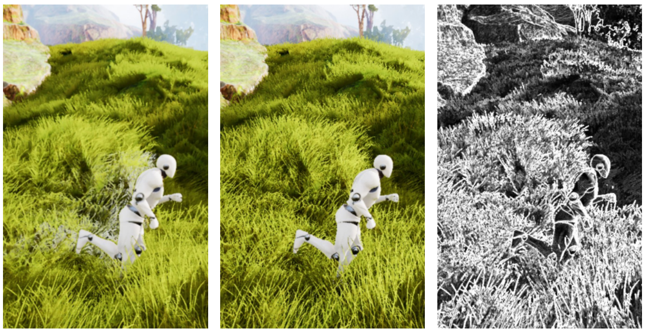
(b) *An input frame* **(c)** *TAA w/ clamping* **(d)** *TAA w/o clamping*

Figure 9: *History rectification-induced blur. In the illustrative example (a), a thin foreground object is sampled in frame 0 and 2, but is completely missed in frame 1 and 3. The foreground color is then removed from the accumulated history in frame 1 and 3, leading to biased results. Screenshots of a forest scene shows that a heavily undersampled input (b) causes missing tree branches and other details in a converged TAA image with clamping (c), when compared to the result without clamping (d).*

in subsequent frames. Other than disocclusion, incorrect motion vectors can also lead to invalid reprojected history in a false shape of a moving object. Common effects that produce to incorrect motion vectors are transparency, particles and reflection, which are known challenging cases for TAA to handle. When motion is absent, but the history data is stale due to fast changing shading (e.g. specular highlights or animated shadows), the result may simply appear as being blurred or blended across multiple frames. This is often referred to as temporal lag.

Both ghosting and temporal lag could be avoided if history rectification mechanisms are effective. However, the effect of history rectification techniques like history clipping or clamping is often compromised near high-contrast object boundaries. Figure 10 shows a common case in games, where the input samples from the current frame contain dense high contrast edges (b), which lead to large color bounding boxes in the area (c). The large bounding boxes are ineffective to represent the color gamut of the pixel colors, and thus allow the invalid history colors to pass during clipping or clamping.

To avoid ghosting artifacts, TAA in Unreal Engine 4 [Kar14] adds a "responsive AA" flag on translucent materials, which is written to a stencil buffer and used by the TAA pass to increase α and allow faster refresh of incorrect history. A similar technique is used in Uncharted 4 [Xu16] to mask out materials that are prone to ghosting. TAA in Uncharted 4 also uses a dilated stencil mask to avoid ghosting trail near smooth, antialiased edges, since half of the pixels on the edge may not carry the mask. Jimenez [Jim16] combine history rejection and rectification, to force refresh history pixels that fails the depth comparison. The TAA technique used in *INSIDE* [Ped16] transitions TAA output to motion-blurred pixels with increasing motion velocity to hide some of the ghosting artifacts.



(a) *TAA result* **(b)** *Input samples* **(c)** *Color BBox Vis.*

Figure 10: *An example of ghosting artifact in TAA: (a) a running character leaves a ghosting trail over disoccluded grass; (b) the input samples have high luminance and chrominance complexity in the grass-covered regions; (c) visualization of the large luma extent of the per-pixel neighborhood color bounding box, which is used to clip the history color.*

To fix incorrect reprojection of reflections, Xu [Xu16] describe a method to obtain correct vector for planar reflective surfaces. There are also other related techniques that aim at warping refracted and reflected contents [LRR*14, ZRJ*15, LRBR16] and shadows [LSR17], by using additional data structures to track correspondence.

6.3. Temporal instability

One of the main goals of TAA is to reduce temporal instability of aliased frames, such as flickering or shimmering artifacts. Since a different jitter offset is applied to the viewport in each frame, the shading sample computed for a pixel changes every frame even when the camera stays stationary. Typically, the difference is absorbed by the sample accumulation step (Sec. 3.3) to provide the correctly filtered pixel. However, history rejection or rectification algorithms may mistakenly invalidate or clamp history due to jitter-induced sample color change. Figure 11 shows such an example where a low frequency Moiré pattern causes a complete gamut change in the neighborhood of some pixels, forcing the history color to be clipped or clamped to the new sample value in every frame, effectively disabling accumulation. Disabling or relaxing history rectification avoids the flickering (third row in Figure 11), but may expose the output to ghosting or temporal lag. In fact, avoiding ghosting and flickering are often contradicting goals in the design of history rejection or rectification algorithms.

The fundamental problem that causes temporal instability is sim-

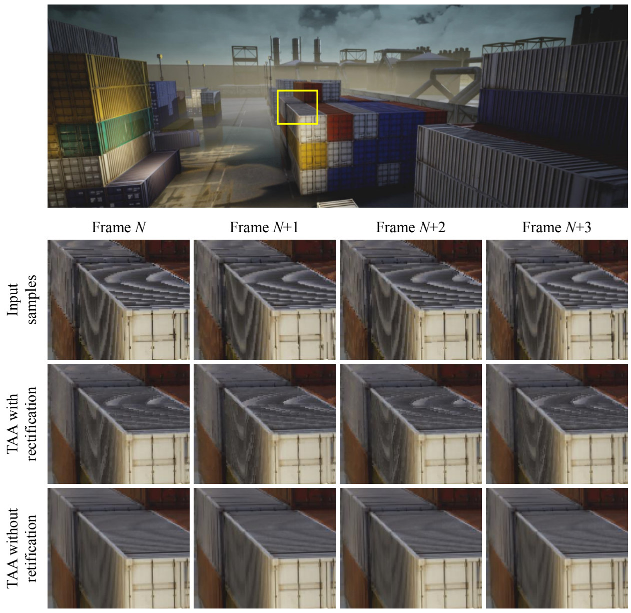


Figure 11: *An example of temporal instability introduced by history rectification using strongly aliased input. Low frequency Moiré pattern in the input images changes the local pixel gamut completely every frame (first row of close-up views), forcing history rectification to clip or clamp colors to the unstable input(second row). Results is stable and much less aliased if history rectification is disabled (third row).*

ilar to the cause of the problems introduced earlier: the undersampled input does not carry enough information to allow reliable detection of invalid or stale history. False positives in detection (valid history treated as invalid) causes blur and temporal instability, whereas false negatives (invalid history treated as valid) causes ghosting and temporal lag. In temporal upsampling algorithms, the artifacts are often more pronounced than in regular TAA, since the input is more heavily undersampled. The alternating sample pattern [Dro14] and selective reshading [SPD18] techniques discussed in Sec. 4.1 are aimed at mitigating this problem, but they are not compatible with mainstream TAA techniques discussed earlier.

There are several heuristic algorithms that can reduce temporal instability under certain scenarios. A test of motion coherency between the previous and the current frame [Sou11,Dro14,EM16] can provide extra input to bias the clamping towards preserving more history, but only if shading is known to be stable. Karis [Kar14] reduces blend factor α when history is near clamping, in order to soften temporal change after clamping happens. Extending that method, Jimenez [Jim16] proposes to track spatial contrast changes between frames as an indicator of flickering artifacts, and increase convergence time when flickering is detected.

6.4. Undersampling artifacts

Since TAA reduces aliasing by integrating samples over time, visible aliasing may exist in areas where accumulated sample count is low. This is commonly seen at camera cuts, where the first few frames after transition usually have an overly sharp and aliased appearance. Regions that are newly disoccluded or with rapid shading changes are also prone to contain undersampling artifacts.

A common technique to mitigate such undersampling artifacts is to increase the spatial reuse for these regions. Several TAA implementations [Epi15, Xu16] use a low pass filter on the new samples to reduce aliasing artifacts and the overly sharp look of newly shaded pixels that appear different from converged results. Alternatively, screen space spatial antialiasing like morphological antialiasing techniques [Res09, Lot09] can help to antialias the raw samples from the current frame [JESG12, Dro14, Jim16, dCI17,Kim18]. In addition to the approximation techniques, Adam et al. [MSG*18] adaptively use ray tracing to gather more samples in region where raster samples are sparse. Similarly, Yang et al. [YNS*09] adaptively supersample the shading of the disoccluded region to reduce shading aliasing where applicable. In practice, adaptive supersampling can often be used on inexpensive procedures inside pixel shaders, such as alpha testing [Wih17].

7. Performance

In modern rendering engines, TAA is typically implemented as a single compute shader or a full-screen pixel shader pass. An optimized TAA implementation runs reasonably fast on modern GPUs. Table 1 shows the typical cost of TAA in Unreal Engine 4 [Epi15]. As an image-space technique, the cost of TAA or temporal upsampling scales relative to the output image resolution (except for motion vector generation cost on dynamic objects). Therefore, it is usually stable and scene-independent, which is a desirable property for the engine to maintaining a constant frame rate. In contrast, the cost of MSAA, which is tied to the forward shading pass, varies based on factors such as geometry density, depth complexity, multi-sample count, and hardware implementation characteristics. That said, MSAA is often cheaper than TAA on lower-end platforms (especially tile-based mobile GPUs), and are sometimes a preferred option on forward renderers.

	Target Resolution		
Technique	1920 × 1080	2560 × 1440	3840 × 2160
TAA	0.14ms	0.24ms	0.52ms
TAAU	0.15ms	0.24ms	0.50ms

Table 1: *Performance of the TAA and the temporal upsampling (TAAU) pass in Unreal Engine 4 running on an NVIDIA RTX 2080 Ti GPU. All TAAU experiments use a* 2× *per dimension upsampling factor.*

Temporal upsampling may sometimes incur a performance overhead to the engine. Unlike conventional image-based upsampling filters, which are typically applied before the image sent to display, temporal upsampling pass usually precede certain postprocessing passes [Epi18]. This is because temporal upsampling doubles as an antialiasing technique, which is responsible for removing aliasing and temporal instability from the input to the postprocessing effects (Sec. 3.3.1). As a result, these postprocessing effects have to be computed at the upsampled resolution. The impact of this change varies by the cost of these effects and the upsampling factor.

8. Other related techniques

There are several techniques and applications that either leverage or extend TAA techniques to achieve better image quality. This section provides a brief overview of these areas.

8.1. Variable rate shading

Variable rate shading (VRS, also known as coarse pixel shading, or CPS) [VST*14, HGF14] decouples the rate of visibility and shading computations by extending multisampling antialiasing [Ake93] to image regions larger than a pixel. VRS has been adopted by graphics hardware and major graphics APIs [NVI18]. It is effective at improving performance by reducing the overall number of fragment shader invocations, but it can also introduce spatial and temporal artifacts, since all pixels belonging to a coarse pixel share a uniform color. To mitigate this problem, several techniques adapt and leverage the spatial-temporal filter in TAA to improve fine pixel reconstruction from coarsely shaded ones.

Patney et al. [PSK*16] accelerate foveated rendering by using coarse pixel shading in the periphery of the image, where undersampling artifacts caused by shading less than one fragment per pixel are addressed via variance clipping (see Section 4.2). The impact of calculating the mean and standard deviation of color over large image regions can be avoided by exploiting the linearity of raw moments, which are pre-integrated over tiles (e.g. 16×16 pixel) and reconstructed at intermediate scales using hardware trilinear filtering [Sal16, PSK*16]. Xiao et al. [XLV18] tailors TAA to coarse pixel shading by adopting a jittering sequence that generates a distribution of visibility and shading samples with blue noise properties. Furthermore, ghosting artifacts introduced by using variance clipping at coarse pixel scale are reduced by adaptively rescaling color variance around edges where a large number of shading samples are available from overlapping surfaces.

8.2. Temporal denoising

Stochastic algorithms in real time rendering often produce noisy results due to having a low sample budget. TAA-like temporal reuse algorithms can be used together with or in place of spatial filtering as a means to achieve variance reduction. Examples include soft shadows [NSL*07, SSMW09], screen-space ambient occlusion (SSAO) [MSW10, BA12], screen-space reflection (SSR) [Sta15], global illumination [Xu16], and other temporal dithering effects [WM17]. Temporal reuse of data allows spatial filtering to effectively achieve larger filter support at a fixed computation cost and output quality.

Reprojection-based temporal data reuse has also been widely applied in denoising ray-traced images. Many techniques rely on TAA-like temporal filter to denoise path-traced images rendered at very low sampling rates [SKW*17, MMBJ17, SPD18, KIM*19, LLCK19, WMB19, SA19, BBHW*19]. One major problem of reusing samples in path-tracing is the difficulty to obtain correct motion vectors for secondary shading effects like reflections and lighting. Zimmer et al. [ZRJ*15] use a generalized version of manifold exploration [JM12] to estimate motion vector for objects seen after specular bounces during rendering, and also use image-based optical flow for estimating the complex motion in irradiance components.

8.3. Machine learning-based methods

Salvi [Sal17] enhances TAA image quality by using stochastic gradient descent (SGD) to learn optimal convolutional weights for computing the color extents used with neighborhood clamping and clipping methods (see Section 4.2). Image quality can be further improved by abandoning engineered history rectification methods in favor of directly learning the rectification task. For instance, variance clipping can be replaced with a recurrent convolutional autoencoder which is jointly trained to hallucinate new samples and appropriately blend them with the history data [Sal17].

On the denoising side, Chaitanya et al. [CKS*17] introduce recurrent connections in a deep U-Net structure to improve temporal stability. Vogels et al. [VRM*18] propose a kernel-predicting network (KPN) based temporal filter, which uses both the previous and the future frames (reprojected to the current frame) as the input for temporal feature extraction and denoising.

9. Conclusion and future work

The past decade has witnessed great advances and success of temporal antialiasing and upsampling techniques in the gaming industry. We have presented a general overview of the existing techniques, and organized them in a framework that encompasses most commonly used building blocks. In particular, we discuss how spatial samples are gathered and accumulated temporally, how history data are validated, and how temporal samples can be leveraged to enhance spatial resolution. Most techniques discussed are built into shipped games and production game engines, and many of them are established as current best practices. The wide applications and adoptions make this topic extremely practical.

Antialiasing in real-time rendering has never been a solved problem, and likely will not be in the near future. TAA is not an exception. We have identified and discussed the main challenges in TAA we are facing today, and summarized existing solutions and workarounds. TAA still need to make a great leap to achieve true cinematic-level quality and robustness. Particularly, we think that the robustness and reliability of history rectification needs to be further improved to fully utilize the temporal data available. In the near future, we expect advances in both analytical and machine learning-based solutions to surpass today's best techniques in this area, and bring the image quality of TAA to the next level.

Acknowledgement

We thank Diego Nehab for providing help in testing the Sacht-Nehab filter for resampling quality comparison. We are also grateful to Kevin Shih, Robert Pottorff and the anonymous reviewers for their help in proofreading the manuscript and giving valuable suggestions.

References

[Aal16] AALTO T.: Towards cinematic quality anti-aliasing in quantum break. In *Game Developers Conference Europe* (2016). 3, 4, 7, 8

[Ake93] AKELEY K.: Reality engine graphics. In *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques* (1993), SIGGRAPH '93, pp. 109–116. 1, 12

[ANS*19] ANDERSSON P., NILSSON J., SALVI M., SPJUT J., AKENINE-MÖLLER T.: Temporally dense ray tracing. In *High-Performance Graphics* (2019). 8

[BA12] BAVOIL L., ANDERSSON J.: Stable SSAO in battlefield 3 with selective temporal filtering. In *Game Developers Conference 2012* (2012). 12

[BBHW*19] BARRÉ-BRISEBOIS C., HALÉN H., WIHLIDAL G., LAURITZEN A., BEKKERS J., STACHOWIAK T., ANDERSSON J.: Hybrid rendering for real-time ray tracing. In *Ray Tracing Gems: High-Quality and Real-Time Rendering with DXR and Other APIs*, Haines E., Akenine-Möller T., (Eds.). Apress, 2019, pp. 437–473. 12

[BKE*95] BRAILEAN J. C., KLEIHORST R. P., EFSTRATIADIS S., KATSAGGELOS A. K., LAGENDIJK R. L.: Noise reduction filters for dynamic image sequences: A review. *Proceedings of the IEEE 83*, 9 (1995), 1272–1292. 3

[CKK18] CHRISTENSEN P., KENSLER A., KILPATRICK C.: Progressive multi-jittered sample sequences. *Computer Graphics Forum 37*, 4 (2018), 21–33. 3, 4

[CKS*17] CHAITANYA C. R. A., KAPLANYAN A. S., SCHIED C., SALVI M., LEFOHN A., NOWROUZEZAHRAI D., AILA T.: Interactive reconstruction of monte carlo image sequences using a recurrent denoising autoencoder. *ACM Trans. Graph. 36*, 4 (July 2017), 98:1–98:12. 12

[CR74] CATMULL E., ROM R.: A class of local interpolating splines. In *Computer Aided Geometric Design*, Barnhill R. E., Riesenfild R. F., (Eds.). Academic Press, 1974, pp. 317 – 326. 9

[CSK*17] CORSO A. D., SALVI M., KOLB C., FRISVAD J. R., LEFOHN A., LUEBKE D.: Interactive stable ray tracing. In *Proceedings of High Performance Graphics* (2017), HPG '17, pp. 1:1–1:10. 9

[dCI17] DE CARPENTIER G., ISHIYAMA K.: Decima engine: Advances in lighting and aa. In *ACM SIGGRAPH Courses: Advances in Real-Time Rendering in Games* (2017). 3, 8, 11

[DL03] DUPONT T. F., LIU Y.: Back and forth error compensation and correction methods for removing errors induced by uneven gradients of the level set function. *Journal of Computational Physics 190*, 1 (2003), 311 – 324. 9

[Dro14] DROBOT M.: Hybrid reconstruction anti-aliasing. In *ACM SIGGRAPH Courses: Advances in Real-Time Rendering in Games* (2014). 3, 4, 6, 8, 9, 11

[EM16] EL MANSOURI J. E.: Rendering 'Rainbow Six | Siege'. In *Game Developers Conference* (2016). 3, 8, 11

[Epi15] EPIC GAMES: The Unreal Engine 4 source code. https://www.unrealengine.com/en-US/ue4-on-github, 2015. Accessed in August 2019. 4, 5, 9, 11

[Epi18] EPIC GAMES: Unreal Engine 4.19: Screen percentage with temporal upsample. https://docs.unrealengine.com/en-US/Engine/Rendering/ScreenPercentage/index.html, Mar. 2018. Accessed in August 2019. 3, 4, 7, 8, 11

[GMK*16] GUITIÁN J. A. I., MOON B., KONIARIS C., SMOLIKOWSKI E., MITCHELL K.: Pixel history linear models for real-time temporal filtering. *Comput. Graph. Forum 35* (2016), 363–372. 7

[GPB04] GELDREICH R., PRITCHARD M., BROOKS J.: Deferred lighting and shading. In *Game Developers Conference* (2004). 1

[Gru15] GRUEN H.: New GPU features of NVIDIA's Maxwell architecture. In *Game Developers Conference* (2015). 4

[HA90] HAEBERLI P., AKELEY K.: The accumulation buffer: Hardware support for high-quality rendering. *SIGGRAPH Comput. Graph. 24*, 4 (Sept. 1990), 309–318. 3

[Har04] HARGREAVES S.: Deferred shading. In *Game Developers Conference* (2004). 1

[HEMS10] HERZOG R., EISEMANN E., MYSZKOWSKI K., SEIDEL H.-P.: Spatio-temporal upsampling on the GPU. In *Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (2010), I3D '10, pp. 91–98. 3, 6, 7

[HGF14] HE Y., GU Y., FATAHALIAN K.: Extending the graphics pipeline with adaptive, multi-rate shading. *ACM Trans. Graph. 33*, 4 (July 2014), 142:1–142:12. 12

[JESG12] JIMENEZ J., ECHEVARRIA J. I., SOUSA T., GUTIERREZ D.: SMAA: Enhanced morphological antialiasing. *Computer Graphics Forum (Proc. EUROGRAPHICS 2012) 31*, 2 (2012). 3, 11

[JGY*11] JIMENEZ J., GUTIERREZ D., YANG J., RESHETOV A., DEMOREUILLE P., BERGHOFF T., PERTHUIS C., YU H., MCGUIRE M., LOTTES T., MALAN H., PERSSON E., ANDREEV D., SOUSA T.: Filtering approaches for real-time anti-aliasing. In *ACM SIGGRAPH Courses* (2011). 1

[Jim14] JIMENEZ J.: Next generation post processing in call of duty: Advanced warfare. In *ACM SIGGRAPH Courses: Advances in Real-Time Rendering in Games* (2014). 5

[Jim16] JIMENEZ J.: Filmic SMAA: Sharp morphological and temporal antialiasing. In *ACM SIGGRAPH Courses: Advances in Real-Time Rendering in Games* (2016). 4, 9, 10, 11

[JM12] JAKOB W., MARSCHNER S.: Manifold exploration: a markov chain monte carlo technique for rendering scenes with difficult specular transport. *ACM Transactions on Graphics (TOG) 31*, 4 (2012), 58. 12

[Kar13] KARIS B.: Tone mapping. http://graphicrants.blogspot.com/2013/12/tone-mapping.html, Dec. 2013. Accessed in August 2019. 5

[Kar14] KARIS B.: High quality temporal supersampling. In *ACM SIGGRAPH Courses: Advances in Real-Time Rendering in Games* (2014). 3, 4, 5, 7, 9, 10, 11

[KB83] KOREIN J., BADLER N.: Temporal anti-aliasing in computer generated animation. *SIGGRAPH Comput. Graph. 17*, 3 (July 1983), 377–388. 1

[Kim18] KIM S.: Temporally stable conservative morphological anti-aliasing (TSCMAA). https://software.intel.com/en-us/articles/temporally-stable-conservative-morphological-anti-aliasing-tscmaa, Jan. 2018. Accessed in January 2020. 3, 11

[KIM*19] KOSKELA M., IMMONEN K., MÄKITALO M., FOI A., VIITANEN T., JÄÄSKELÄINEN P., KULTALA H., TAKALA J.: Blockwise multi-order feature regression for real-time path-tracing reconstruction. *ACM Trans. Graph. 38*, 5 (June 2019), 138:1–138:14. 5, 12

[KLB95] KLEIHORST R. P., LAGENDIJK R. L., BIEMOND J.: Noise reduction of image sequences using motion compensation and signal decomposition. *IEEE Transactions on Image Processing 4*, 3 (1995), 274–284. 3

[Kra19] KRAMER L.: Fidelityfx cas. https://gpuopen.com/gaming-product/fidelityfx/, June 2019. Accessed in October 2019. 9

[KS14] KERZNER E., SALVI M.: Streaming g-buffer compression for multi-sample anti-aliasing. In *Proceedings of High Performance Graphics* (2014), HPG '14, pp. 1–7. 1

[Lea10] LEADBETTER R.: Tech analysis: Halo: Reach. http://www.eurogamer.net/articles/digitalfoundry-halo-reach-tech-analysis-article, Sept. 2010. Accessed in August 2019. 3

[Lea16] LEADBETTER R.: Inside playstation 4 pro: How sony made the first 4k games console. https://www.eurogamer.net/articles/digitalfoundry-2016-inside-playstation-4-pro-how-sony-made-a-4k-games-machine, Oct. 2016. Accessed in August 2019. 3, 8

[LLCK19] Liu E., Llamas I., Cañada J., Kelly P.: Cinematic rendering in ue4 with real-time ray tracing and denoising. In *Ray Tracing Gems: High-Quality and Real-Time Rendering with DXR and Other APIs*, Haines E., Akenine-Möller T., (Eds.). Apress, 2019, pp. 289–319. 12

[Lot09] Lottes T.: FXAA whitepaper. https://developer.download.nvidia.com/assets/gamedev/files/sdk/11/FXAA_WhitePaper.pdf, Feb. 2009. Accessed in August 2019. 11

[Lot11] Lottes T.: TSSAA: Temporal supersamping AA. http://timothylottes.blogspot.com/2011/04/tssaa-temporal-super-sampling-aa.html, Apr. 2011. Accessed in August 2019 via archive.org. 3, 7

[LRBR16] Lochmann G., Reinert B., Buchacher A., Ritschel T.: Real-time Novel-view Synthesis for Volume Rendering Using a Piecewise-analytic Representation. In *Vision, Modeling and Visualization* (2016), Hullin M., Stamminger M., Weinkauf T., (Eds.), The Eurographics Association. 10

[LRR*14] Lochmann G., Reinert B., Ritschel T., MÃijller S., Seidel H.-P.: Real-time Reflective and Refractive Novel-view Synthesis. In *Vision, Modeling and Visualization* (2014), Bender J., Kuijper A., von Landesberger T., Theisel H., Urban P., (Eds.), The Eurographics Association. 10

[LSR17] Leimkühler T., Seidel H.-P., Ritschel T.: Minimal warping: Planning incremental novel-view synthesis. In *Computer Graphics Forum* (2017), vol. 36, pp. 1–14. 10

[Mal12] Malan H.: Real-time global illumination and reflections in dust 514. In *ACM SIGGRAPH Courses: Advances in Real-Time Rendering in Games* (2012). 3, 6, 7

[ML18] Mcferron T., Lake A.: Checkerboard rendering for real-time upscaling on intel integrated graphics. https://software.intel.com/en-us/articles/checkerboard-rendering-for-real-time-upscaling-on-intel-integrated-graphics, Aug. 2018. 8

[MMBJ17] Mara M., McGuire M., Bitterli B., Jarosz W.: An efficient denoising algorithm for global illumination. In *Proceedings of High Performance Graphics* (July 2017), ACM. 12

[MSG*18] Marrs A., Spjut J., Gruen H., Sathe R., McGuire M.: Adaptive temporal antialiasing. In *Proceedings of the Conference on High-Performance Graphics* (2018), HPG '18, ACM, pp. 1:1–1:4. 11

[MSW10] Mattausch O., Scherzer D., Wimmer M.: High-quality screen-space ambient occlusion using temporal coherence. *Computer Graphics Forum 29*, 8 (Dec. 2010), 2492–2503. 12

[NABW12] Netzel R., Ament M., Burch M., Weiskopf D.: Spectral Analysis of Higher-Order and BFECC Texture Advection. In *International Symposium on Vision, Modeling and Visualization (VMV)* (2012). 9

[NH14] Nehab D., Hoppe H.: *A Fresh Look at Generalized Sampling*. Now Publishers Inc., Hanover, MA, USA, 2014. 9

[NM16] Nehab D., Maximo A.: Parallel recursive filtering of infinite input extensions. *ACM Trans. Graph. 35*, 6 (Nov. 2016), 204:1–204:13. 9

[NSL*07] Nehab D., Sander P. V., Lawrence J., Tatarchuk N., Isidoro J. R.: Accelerating real-time shading with reverse reprojection caching. In *Proceedings of the 22Nd ACM SIGGRAPH/EUROGRAPHICS Symposium on Graphics Hardware* (2007), GH '07, pp. 25–35. 3, 4, 5, 6, 12

[NVI12] NVIDIA: TXAA: Temporal anti-aliasing. https://www.geforce.com/hardware/technology/txaa, 2012. Accessed in August 2019. 3

[NVI18] NVIDIA: Variable rate shading. https://developer.nvidia.com/vrworks/graphics/variablerateshading, 2018. 12

[OST93] Ozkan M. K., Sezan M. I., Tekalp A. M.: Adaptive motion-compensated filtering of noisy image sequences. *IEEE transactions on circuits and systems for video technology 3*, 4 (1993), 277–290. 3

[Ped16] Pedersen L. J. F.: Temporal reprojection anti-aliasing in INSIDE. In *Game Developers Conference* (2016). 3, 4, 10

[Pet15] Pettineo M. J.: Rendering the alternate history of the order: 1886. In *ACM SIGGRAPH Courses: Advances in Real-time Rendering* (2015). 5

[PSK*16] Patney A., Salvi M., Kim J., Kaplanyan A., Wyman C., Benty N., Luebke D., Lefohn A.: Towards foveated rendering for gaze-tracked virtual reality. *ACM Trans. Graph. 35*, 6 (Nov. 2016), 179:1–179:12. 3, 12

[Res09] Reshetov A.: Morphological antialiasing. In *Proceedings of the Conference on High Performance Graphics 2009* (2009), HPG '09, pp. 109–116. 3, 11

[SA19] Smal N., Aizenshtein M.: Real-time global illumination with photon mapping. In *Ray Tracing Gems: High-Quality and Real-Time Rendering with DXR and Other APIs*, Haines E., Akenine-Möller T., (Eds.). Apress, 2019, pp. 409–436. 12

[Sal16] Salvi M.: An excursion in temporal supersampling. In *Game Developers Conference* (2016). 3, 7, 12

[Sal17] Salvi M.: Deep learning: The future of real-time rendering? In *ACM SIGGRAPH Courses: Open Problems in Real-Time Rendering* (2017). 12

[SJW07] Scherzer D., Jeschke S., Wimmer M.: Pixel-correct shadow maps with temporal reprojection and shadow test confidence. In *Proceedings of the 18th Eurographics Conference on Rendering Techniques* (2007), EGSR'07, pp. 45–50. 3, 4, 5

[SKW*17] Schied C., Kaplanyan A., Wyman C., Patney A., Chaitanya C. R. A., Burgess J., Liu S., Dachsbacher C., Lefohn A., Salvi M.: Spatiotemporal variance-guided filtering: Real-time reconstruction for path-traced global illumination. In *Proceedings of High Performance Graphics* (2017), HPG '17, pp. 2:1–2:12. 12

[SN15] Sacht L., Nehab D.: Optimized quasi-interpolators for image reconstruction. *IEEE Transactions on Image Processing 24*, 12 (2015), 5249–5259. 9

[Sou11] Sousa T.: Anti-aliasing methods in CryENGINE 3. In *ACM SIGGRAPH Courses: Filtering Approaches for Real-Time Anti-Aliasing* (2011). 3, 5, 6, 11

[SPD18] Schied C., Peters C., Dachsbacher C.: Gradient estimation for real-time adaptive temporal filtering. *Proc. ACM Comput. Graph. Interact. Tech. 1*, 2 (Aug. 2018), 24:1–24:16. 6, 11, 12

[SSMW09] Scherzer D., Schwärzler M., Mattausch O., Wimmer M.: Real-time soft shadows using temporal coherence. In *Advances in Visual Computing: 5th International Symposium on Visual Computing (ISVC 2009)* (Dec. 2009), Springer, pp. 13–24. 12

[Sta15] Stachowiak T.: Stochastic screen-space reflections. In *ACM SIGGRAPH Courses: Advances in Real-time Rendering* (2015). 12

[SV12] Salvi M., Vidimče K.: Surface based anti-aliasing. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (2012), I3D '12, pp. 159–164. 1

[SYM*12] Scherzer D., Yang L., Mattausch O., Nehab D., Sander P. V., Wimmer M., Eisemann E.: Temporal coherence methods in real-time rendering. *Computer Graphics Forum 31*, 8 (12 2012), 2378–2408. 3

[Uni16] Unity Technologies: Unity effects documentation: Anti-aliasing. https://docs.unity3d.com/Packages/com.unity.postprocessing@2.1/manual/Anti-aliasing.html, June 2016. Accessed in August 2019. Release note. 3

[Val14] Valient M.: Taking Killzone Shadow Fall image quality into the next generation. In *Game Developers Conference* (2014). 3, 6, 8

[VRM*18] Vogels T., Rousselle F., McWilliams B., Röthlin G., Harvill A., Adler D., Meyer M., Novák J.: Denoising with kernel prediction and asymmetric loss functions. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2018) 37*, 4 (2018), 124:1–124:15. 12

[VST*14] Vaidyanathan K., Salvi M., Toth R., Foley T., Akenine-Möller T., Nilsson J., Munkberg J., Hasselgren J., Sugihara M., Clarberg P., Janczak T., Lefohn A.: Coarse pixel shading. In *Proceedings of High Performance Graphics* (2014), HPG '14, pp. 9–18. 12

[Wih17] Wihlidal G.: 4k checkerboard in Battlefield 1 and Mass Effect Andromeda. In *Game Developers Conference* (2017). 3, 4, 8, 9, 11

[WM17] Wyman C., McGuire M.: Hashed alpha testing. In *Proceedings of the 21st ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (2017), I3D '17, pp. 7:1–7:9. 12

[WMB19] Willberger T., Musterle C., Bergmann S.: Deferred hybrid path tracing. In *Ray Tracing Gems: High-Quality and Real-Time Rendering with DXR and Other APIs*, Haines E., Akenine-Möller T., (Eds.). Apress, 2019, pp. 475–492. 5, 12

[XLV18] Xiao K., Liktor G., Vaidyanathan K.: Coarse pixel shading with temporal supersampling. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (2018), I3D '18, pp. 1:1–1:7. 3, 12

[Xu16] Xu K.: Temporal antialiasing in Uncharted 4. In *ACM SIGGRAPH Courses: Advances in Real-Time Rendering in Games* (2016). 3, 9, 10, 11, 12

[YNS*09] Yang L., Nehab D., Sander P. V., Sitthi-amorn P., Lawrence J., Hoppe H.: Amortized supersampling. *ACM Trans. Graph. 28*, 5 (Dec. 2009), 135:1–135:12. 3, 4, 5, 6, 7, 9, 11

[ZRJ*15] Zimmer H., Rousselle F., Jakob W., Wang O., Adler D., Jarosz W., Sorkine-Hornung O., Sorkine-Hornung A.: Path-space motion estimation and decomposition for robust animation filtering. *Computer Graphics Forum 34*, 4 (2015), 131–142. 10, 12