Amortized Supersampling

Lei Yang[1] Diego Nehab[2] Pedro V. Sander[1] Pitchaya Sitthi-amorn[3] Jason Lawrence[3] Hugues Hoppe[2]

[1]Hong Kong UST [2]Microsoft Research [3]University of Virginia

Abstract

We present a real-time rendering scheme that reuses shading samples from earlier time frames to achieve practical antialiasing of procedural shaders. Using a reprojection strategy, we maintain several sets of shading estimates at subpixel precision, and incrementally update these such that for most pixels only one new shaded sample is evaluated per frame. The key difficulty is to prevent accumulated blurring during successive reprojections. We present a theoretical analysis of the blur introduced by reprojection methods. Based on this analysis, we introduce a nonuniform spatial filter, an adaptive recursive temporal filter, and a principled scheme for locally estimating the spatial blur. Our scheme is appropriate for antialiasing shading attributes that vary slowly over time. It works in a single rendering pass on commodity graphics hardware, and offers results that surpass 4×4 stratified supersampling in quality, at a fraction of the cost.

1 Introduction

The use of antialiasing to remove sampling artifacts is an important and well studied area in computer graphics. In real-time rasterization, antialiasing typically involves two hardware-supported techniques: mipmapped textures for prefiltered surface content, and framebuffer multisampling to remove jaggies at surface silhouettes.

With the increasing programmability of graphics hardware, many functionalities initially developed for offline rendering are now feasible in real-time. These include procedural materials and complex shading functions. Unlike prefiltered textures, procedurally defined signals are not usually bandlimited [Ebert et al. 2003], and producing a bandlimited version of a procedural shader is a difficult and ad-hoc process [Apodaca and Gritz 2000].

To reduce aliasing artifacts in procedurally shaded surfaces, a common approach is to increase the spatial sampling rate using supersampling (Figure 1c). However, it can be prohibitively expensive to execute a complex procedural shader multiple times at each pixel. Fortunately, it is often the case that at any given surface point, expensive elements of the surface shading (such as albedo) vary slowly or are constant over time. A number of techniques can automatically factor a procedural shader into static and dynamic layers [Guenter 1994; Jones et al. 2000; Sitthi-amorn et al. 2008]. Our idea is to sample the static and weakly dynamic layers at a lower temporal rate to achieve a higher spatial sampling rate for the same computational budget. The strong dynamic layers can be either sampled at the native resolution or supersampled using existing techniques.

We present a real-time scheme, *amortized supersampling*, that evaluates the static and weak dynamic components of the shading function only once for the majority of pixels, and reuses samples computed in prior framebuffers to achieve good spatial antialiasing. The general idea of reusing shading information across frames has been studied extensively, as reviewed in Section 2. Our approach builds on the specific strategy of *real-time reprojection*, whereby the GPU pixel shader "pulls" information associated with the same surface point in earlier frames. Recent work on reprojection has focused on reusing expensive intermediate shading computations across frames [Nehab et al. 2007] and temporally smoothing shadow map boundaries [Scherzer et al. 2007]. In contrast, our goal is effective supersampling of more general shading functions.

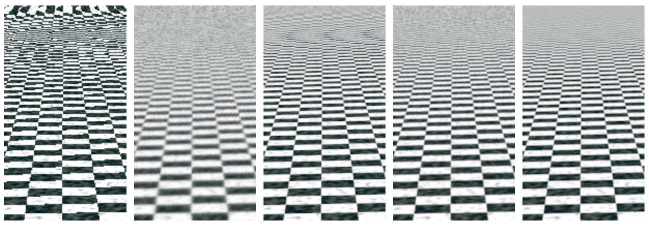

(a) No AA 140fps *(b) Jit. reproj. 88fps* *(c) 4×4 SS 11fps* *(d) Ours 63fps* *(e) Reference*

Figure 1: For a moving scene with a procedural shader (top row of Figure 14), comparison of (a) no antialiasing, (b) jittered reprojection, (c) 4×4 stratified supersampling, (d) our amortized supersampling, and (e) ground truth reference image.

Amortized supersampling faces many challenges not present in ordinary supersampling. Due to scene motion, the set of samples computed in earlier frames form an irregular pattern when reprojected into the current frame. Moreover, some samples become invalid due to occlusion. Thus the set of spatio-temporal samples available for reconstruction has much less structure than a typical grid of stratified stochastic samples.

We build on the jittered sampling and recursive exponential smoothing introduced in prior reprojection work. An important contribution of this paper is a theoretical analysis of the spatio-temporal blur introduced by these techniques as a function of the relative scene motion and smoothing factor applied in the recursive filter. We show that by adjusting this smoothing factor adaptively, the basic reprojection algorithm can be made to converge to perfect reconstruction (infinite supersampling) for stationary views of static scenes (Section 4.1). Furthermore, we show that for moving surfaces, straightforward reprojection leads to excessive blurring (Figure 1b). Our scheme makes several contributions in addressing this key issue:

- Use of multiple subpixel buffers to maintain reprojection estimates at a higher spatial resolution;
- Irregular round-robin update of these subpixel buffers to improve reconstruction quality, while still only requiring one sample evaluation per pixel per frame;
- A principled approach to estimate and limit the amount of blur introduced during reprojection and exponential smoothing;
- Adaptive evaluation of additional samples in disoccluded pixels to reduce aliasing;
- A strategy to estimate and react to slow temporal changes in the shading.

Amortized supersampling is compatible with the modern rasterization pipeline implemented on commodity graphics hardware. It is lightweight, requiring no preprocessing, and thus provides a practical approach for antialiasing existing procedural shaders. Also, it requires only a single rendering pass, and can be used in conjunction with hardware multisampling for antialiasing geometry silhouettes. We show that it achieves results that are qualitatively comparable or superior to 4×4 stratified supersampling, but at a fraction of the rendering cost (Figure 1d).

2 Related work

Data caching and reuse Many offline and interactive ray-based rendering systems exploit the spatio-temporal coherence of animation sequences [Cook et al. 1987; Badt 1988; Chen and Williams 1993; Bishop et al. 1994; Adelson and Hodges 1995; Mark et al. 1997; Walter et al. 1999; Bala et al. 1999; Ward and Simmons 1999; Havran et al. 2003; Tawara et al. 2004]. The idea is also used in hybrid systems that use some form of hardware acceleration [Simmons and Séquin 2000; Stamminger et al. 2000; Walter et al. 2002; Woolley et al. 2003; Gautron et al. 2005; Zhu et al. 2005; Dayal et al. 2005]. These systems focus primarily on reusing expensive global illumination or geometry calculations such as ray-scene intersections, indirect lighting estimates, and visibility queries. A related set of methods opportunistically reuse shading information to accelerate real-time rendering applications by *reprojecting* the contents of the previous frame into the current frame [Hasselgren and Akenine-Moller 2006; Nehab et al. 2007; Scherzer et al. 2007; Scherzer and Wimmer 2008]. Although we also apply a reprojection strategy to reuse shading information over multiple frames, we do so to achieve spatial antialiasing; this application was first noted by Bishop et al. [1994], but not pursued. Furthermore, we add to this area of research a rigorous theoretical analysis of the type of blur introduced by repeatedly resampling a framebuffer, a fundamental operation in these systems.

Antialiasing of procedural shaders There is a considerable body of research on antialiasing procedural shaders, recently reviewed by Brooks Van Horn III and Turk [2008]. Creating a bandlimited version of a procedural shader can be a difficult task because analytically integrating the signal is often infeasible. Several practical approaches are reviewed in the book by Ebert et al. [2003]. These include clamping the high-frequency components of a shader that is defined in the frequency domain [Norton et al. 1982], precomputing mipmaps for tabulated data such as lookup tables [Hart et al. 1999], and obtaining approximations using affine arithmetic [Heidrich et al. 1998]. However, the most general and common approach is still to numerically integrate the signal using supersampling [Apodaca and Gritz 2000]. Our technique brings the simplicity of supersampling to real-time applications at an acceptable increase in rendering cost.

Post-processing of rendered video The pixel tracing filter of Shinya [1993] is related to our approach. Starting with a rendered video sequence, the tracing filter tracks the screen-space positions of corresponding scene points, and combines color samples at these points to achieve spatial antialiasing in each frame. The filtering operation is applied as a post-process, and assumes that the full video sequence is accessible. In contrast, our approach is designed for real-time evaluation. We maintain only a small set of reprojection buffers in memory, and update these efficiently with adaptive recursive filtering.

3 Review of reprojection

Reprojection methods [Nehab et al. 2007; Scherzer et al. 2007] allow reusing values generated at the pixel level over consecutive frames. We next summarize the basic approach which has two main parts: reprojection and recursive exponential smoothing.

Reprojection The core idea is to let the rendering of the current frame gather and reuse shading information from surfaces visible in the previous frame. Conceptually, when rasterizing a surface at a given pixel, we determine the projection of the surface point into the previous framebuffer, and test if its depth matches the depth stored in the previous depth buffer. If so, the point was previously visible, and its attributes can be safely reused. Formally, let buffer f_t hold the cached pixel attributes at time t, and buffer d_t hold the pixel depths. Let $f_t[p]$ and $d_t[p]$ denote the buffer values at pixel $p \in \mathbb{Z}^2$, and let $f_t(\cdot)$ and $d_t(\cdot)$ denote bilinear sampling. For each pixel $p = (x, y)$ at time t, we determine the 3D clip-space position of its generating scene point at time t-1, denoted $(x', y', z') = \pi_{t\text{-}1}(p)$. The reprojection operation $\pi_{t\text{-}1}(p)$ is obtained using a simple computation in the vertex program and interpolator as described by Nehab et al. [2007]. If the reprojected depth z' lies within some tolerance of the bilinearly interpolated depth $d_{t\text{-}1}(x', y')$ we conclude that $f_t[p]$ has some correspondence with the interpolated value $f_{t\text{-}1}(x', y')$. If the depths do not match (due to occlusion), or if (x', y') lies outside the view frustum at time t-1, no correspondence exists and we denote this by $\pi_{t\text{-}1}(p) = \varnothing$.

Recursive exponential smoothing Both Nehab et al. [2007] and Scherzer et al. [2007] showed how this reprojection strategy can be combined with a recursive temporal filter for antialiasing. We first review this basic principle before extending it to a more general setting.

At each pixel p, the shader is evaluated at some jittered position to obtain a new sample $s_t[p]$. This sample is combined with a running estimate of the antialiased value maintained in $f_t[p]$ according to a recursive exponential smoothing filter:

$$f_t[p] \leftarrow (\alpha)s_t[p] + (1-\alpha)f_{t\text{-}1}\big(\pi_{t\text{-}1}(p)\big). \tag{1}$$

Note that the contribution a single sample makes to this estimate decreases exponentially in time, and the smoothing factor α regulates the tradeoff between variance reduction and responsiveness to changes in the scene. For example, a small value of α reduces the variance in the estimate and therefore produces a less aliased result, but introduces more lag if the shading changes between frames. If reprojection fails at any pixel (i.e. $\pi_{t\text{-}1}(p) = \varnothing$), then α is locally reset to 1 to give full weight to the current sample. This produces higher variance (greater aliasing) in recently disoccluded regions.

4 Amortized supersampling: theory

Spatial antialiasing is achieved by convolving the screen-space shading function S with a low-pass filter G [Mitchell and Netravali 1988]. We use a Monte Carlo algorithm with importance sampling [Robert and Casella 2004] to approximate this convolution $(S * G)(p)$ at each pixel p:

$$f_N[p] \leftarrow \frac{1}{N}\sum_{i=1}^{N} S\big(p + g_i[p]\big). \tag{2}$$

Here $g_i[p]$ plays the role of a per-pixel random jitter offset, distributed according to G. Our choice for G is a 2D Gaussian kernel with standard deviation $\sigma_G = 0.4737$ as this closely approximates

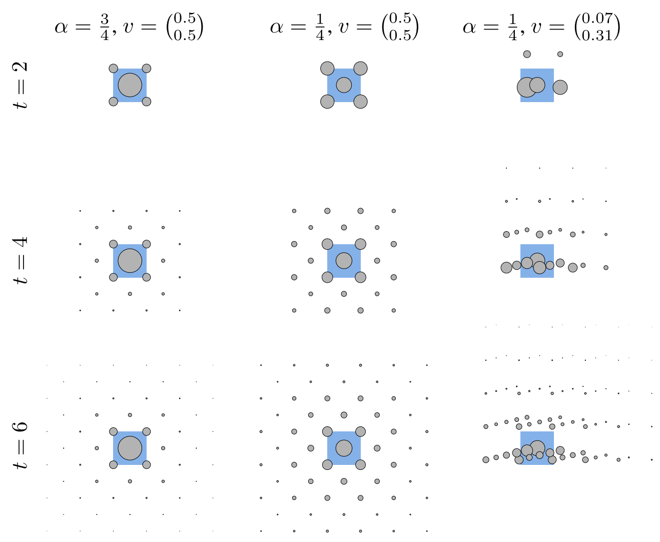

$\alpha = \frac{3}{4}, v = \binom{0.5}{0.5}$
$\alpha = \frac{1}{4}, v = \binom{0.5}{0.5}$
$\alpha = \frac{1}{4}, v = \binom{0.07}{0.31}$
$t = 2$
$t = 4$
$t = 6$

Figure 2: Illustration of the samples that contribute to the estimate at a single pixel $f_t[p]$ under a constant fractional velocity field v (and assuming no jitter for simplicity). Note that the sample points spread out over time, but that their weights (proportional to the radius of each dot) decreases at a rate determined by the smoothing factor α. We characterize the amount of blur at each pixel by the weighted spatial variance of these sample offsets.

the kernel of Mitchell and Netravali [1988] while avoiding negative lobes, which interfere with importance sampling [Ernst et al. 2006]. It is easily shown that the variance of the estimator is

$$\mathrm{Var}(f_N[p]) = \frac{1}{N}\mathrm{Var}\big(f_1[p]\big), \tag{3}$$

where $\mathrm{Var}\big(f_1[p]\big)$ is the per-pixel variance of the Monte Carlo estimator using just one sample. Using a recursive exponential smoothing filter, we can amortize the cost of evaluating the sum in (2) over multiple frames:

$$f_t[p] \leftarrow (\alpha_t[p])\, S\big(p + g_t[p]\big) + (1 - \alpha_t[p])\, f_{t\text{-}1}\big[\pi_{t\text{-}1}(p)\big]. \tag{4}$$

In words, a running estimate of (2) is maintained at each pixel p in the buffer f_t and is updated at each frame by combining a new jittered sample $S\big(p+g_t[p]\big)$ with the previous estimate according to the smoothing factor $\alpha_t[p]$. Note that this formulation allows $\alpha_t[p]$ to vary over time and with pixel location.

We first present an antialiasing scheme for stationary views and static scenes, and then consider the more general case of arbitrary scene and camera motion. Detailed derivations of key results in these sections are found in the appendix.

4.1 Stationary viewpoint and static scene

In the case of a stationary camera viewpoint and static scene, the reprojection map π is simply the identity. In this case, the smoothing factor can be gradually decreased over time as

$$\alpha_t[p] = \frac{1}{t}, \tag{5}$$

resulting in an ever-increasing accumulation of samples, all with uniform weights. This causes the estimates $f_t[p]$ to converge to perfect antialiasing (infinite supersampling) as $t \to \infty$, with variance decreasing as $\mathrm{Var}\big(f_1[p]\big)/t$.

4.2 Moving viewpoint and dynamic scene

The presence of relative scene motion (caused by a moving camera or moving/deforming scene objects) significantly complicates the task of amortizing (2) over multiple frames. Specifically, any algorithm must both respond to changes in visibility and avoid unwanted blurring due to successive resamplings. We next describe our approach for addressing these two issues.

4.2.1 Accounting for visibility changes

In dynamic scenes, the number of consecutive frames that a surface point has been visible varies across pixels. We therefore maintain a record of the effective number of samples that have contributed to the current estimate at each pixel. More precisely, we store the variance reduction factor $N_t[p]$ for each pixel, and use this value to set the per-pixel smoothing factor $\alpha_t[p]$ at each frame, with the goal of reproducing the stationary case in Section 4.1.

When a surface point becomes visible for the first time (either at start-up or due to a disocclusion) we initialize $\alpha_t[p] \leftarrow 1$ and $N_t[p] \leftarrow 1$. In subsequent updates, we apply the rules

$$\alpha_t[p] \leftarrow \frac{1}{N_{t\text{-}1}[p] + 1} \tag{6}$$

and

$$N_t[p] \leftarrow N_{t\text{-}1}[p] + 1. \tag{7}$$

As discussed later on, the presence of scene motion requires us to limit this indefinite accumulation of uniformly weighted samples — even for pixels that remain visible over many frames. To do so, Sections 5.2 and 6.2 prescribe lower bounds on the value of $\alpha_t[p]$ that override (6). This effectively changes the rate at which new samples are accumulated, leading to a new update rule for $N_t[p]$ (see the appendix for a derivation):

$$N_t[p] \leftarrow \left(\alpha_t[p]^2 + \frac{\big(1-\alpha_t[p]\big)^2}{N_{t\text{-}1}\big[\pi_{t\text{-}1}(p)\big]}\right)^{-1}. \tag{8}$$

Note that (8) reduces to (7) when $\alpha_t[p]$ is not overridden, so in practice we always use (8).

4.2.2 Modeling blur due to resampling

In general, the reprojected position $\pi_{t\text{-}1}(p)$ used in (4) lies somewhere between the set of discrete samples in buffer $f_{t\text{-}1}$ and thus some form of resampling is required. This resampling involves computing a weighted sum of the values in the vicinity of $\pi_{t\text{-}1}(p)$. Repeatedly resampling values at intermediate locations has the effect of progressively increasing the number of samples that contribute to the final estimate at each pixel. Moreover, the radius of this neighborhood of samples increases over time (Figure 2), leading to undesirable blurring (Figure 1b). Our goal is to limit this effect. We first model it mathematically.

The value stored at a single pixel is given by a weighted sum of a number of samples $n(t,p)$ evaluated at different positions:

$$f_t[p] = \sum_{i=1}^{n(t,p)} \omega_{t,i}\, S\big(p + \delta_{t,i}[p]\big) \quad \text{with} \quad \sum_{i=1}^{n(t,p)} \omega_{t,i} = 1. \tag{9}$$

The weights $\omega_{t,i}$ are a function of the particular resampling strategy employed and the sequence of weights α_t used in the recursive filter. The offsets $\delta_{t,i}$ denote the position of each contributing sample

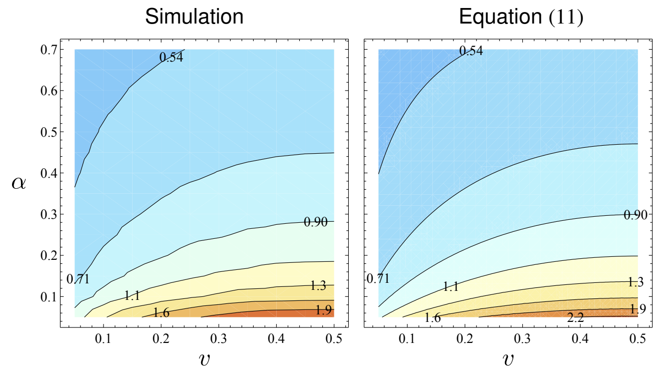


Figure 3: Experimental validation of Equation (11). For each velocity v and weight α we rendered a resolution chart until convergence using (4). We compared the rendered result to a set of images of the same resolution chart rendered with (2) using a low-pass filter G' and a range of standard deviations $\sigma_{G'}$. The left plot shows the $\sigma_{G'}$ that gives the best match (highest PSNR) as a function of v and α. The right plot shows the blur standard deviation predicted by (11). The RMSE between the observed and predicted blur standard deviations is only 0.0382 pixels.

with respect to the center of the pixel p. Note that each displacement $\delta_{t,i}[p]$ is the result of a combination of offsets due to jitter and reprojection. Following the derivation in the appendix, the amount of blur at a pixel can be characterized by the average weighted spatial variance across both dimensions

$$\sigma_t^2[p] = \tfrac{1}{2}\mathrm{Var}_x\left(\{\delta_{t,i}[p]\}_{i=1}^{n(t,p)}\right) + \tfrac{1}{2}\mathrm{Var}_y\left(\{\delta_{t,i}[p]\}_{i=1}^{n(t,p)}\right). \quad (10)$$

Although obtaining a closed-form expression for $\sigma_t^2[p]$ is impractical for arbitrary scene motion, the case of constant panning motion is tractable and provides an insightful case that will serve as the basis of our approach for estimating (and limiting) unwanted blur. Moreover, other types of motion are well approximated locally by translations. This type of motion resamples each pixel at a constant offset given by the fractional velocity $v = \pi_{t\text{-}1}(p) - \lfloor \pi_{t\text{-}1}(p) \rfloor$. Furthermore, let us assume for now that standard bilinear interpolation is used to reconstruct intermediate values and that a constant smoothing factor α is used in (4). As shown in the appendix, under these assumptions the expected blur variance $E\big(\sigma_t^2[p]\big)$ converges (as $t \to \infty$) to

$$\sigma_v^2 = \sigma_G^2 + \frac{1-\alpha}{\alpha}\,\frac{v_x(1-v_x) + v_y(1-v_y)}{2}. \quad (11)$$

The simulation results shown in Figure 3 confirm the accuracy of this expression. Each factor in (11) suggests a different approach for reducing the amount of blur. The factor $\big(v_x(1-v_x) + v_y(1-v_y)\big)$ arises from the choice of bilinear interpolation. We encourage the fractional velocity v_x and v_y to concentrate around 0 or 1 by maintaining an estimate of the framebuffer at a higher resolution, and by avoiding resampling whenever possible (Section 5.1). In addition, we reduce the factor $\frac{1-\alpha}{\alpha}$ by using larger values of α, although this has the disadvantage of limiting the magnitude of antialiasing. We present a strategy for setting α that controls this tradeoff (Section 5.2). Together, these ideas form the backbone of our antialiasing algorithm.

5 Algorithm

As described in this section, our antialiasing algorithm uses multiple subpixel buffers to limit the amount of blur, adapts sample evaluation at disoccluded pixels, and adjusts the smoothing factor α to control the tradeoff between blurring and aliasing.

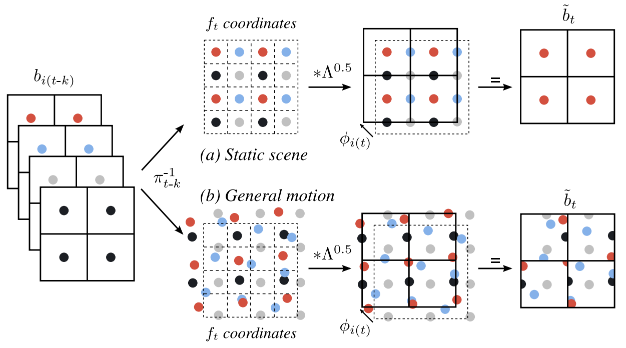


Figure 4: Sampling from multiple subpixel buffers. To limit the amount of blur, we use nonuniform blending weights defined by a tent function centered on the quadrant being updated. (a) In the absence of local motion, no resampling blur is introduced; (b) For a moving scene, our method selects those samples closest to the desired quadrant center to limit the amount of blur.

5.1 Subpixel buffers

We decrease unwanted blur by maintaining screen-space estimates f at twice the screen resolution, as this tends to reduce the terms $v_x(1-v_x)$ and $v_y(1-v_y)$ in (11) by half. We associate estimates with the four quadrants of each pixel, and update these in round-robin fashion. These 2×2 quadrant samples are deinterleaved to form $K = 4$ independent *subpixel buffers* $\{b_k\}$, $k \in \{0,1,2,3\}$, each at screen resolution. Each buffer b_k stores an estimate offset by $\phi_k \in \left\{\binom{-0.25}{-0.25}, \binom{-0.25}{0.25}, \binom{0.25}{-0.25}, \binom{0.25}{0.25}\right\}$ relative to the center of the pixel.

Note that in the absence of scene motion, these four subpixel buffers effectively form a higher-resolution framebuffer. However, under scene motion, the subpixel samples computed in earlier frames reproject to offset locations, as indicated in Figure 4.

At each frame, we compute one new sample per pixel, and update one of the subpixel buffers, $b_{i(t)}$, according to (4) using information gathered from all the subpixel buffers. (For now, let $i(t) = t \bmod K$.) We then compute the final pixel color as a weighted average of these subpixel buffers. Figure 5 illustrates the steps executed by the pixel shader at each frame t:

1. $s_t[p] \leftarrow \textsc{EvaluateSample}\big(p + \phi_{i(t)} + g_t[p]\big)$
   Evaluate the procedural shader at a new sample position;

2. $b_{i(t)}[p] \leftarrow \textsc{UpdateSubpixel}\big(\{b_k[p]\}, s_t[p], t\big)$
   Update one subpixel buffer using all previous subpixel buffers and the new sample;

3. $f_t[p] \leftarrow \textsc{ComputePixelColor}\big(\{b_k[p]\}, s_t[p]\big)$
   Compute the output pixel color given the new sample and the subpixel buffers;

4. $N_t[p] \leftarrow \textsc{UpdateEffectiveNumberOfSamples}\big(N_t[p]\big)$
   Update the new per-pixel effective number of samples.

These steps are implemented in the real-time graphics pipeline. The vertex shader computes the reprojection coordinates needed to access the four subpixel buffers (as with traditional reprojection [Nehab et al. 2007]), and the fragment shader outputs three render targets: the framebuffer f_t, an updated version $b_{i(t)}$ of one of the subpixel buffers, and updated values for the per-pixel number N_t of samples. All steps are performed together in a single GPU rendering pass, as described below.

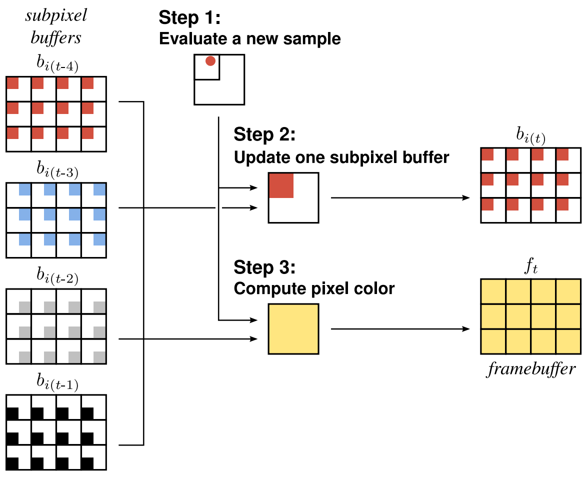


Figure 5: The main steps of our fragment shader algorithm. All steps are performed together in the main rendering pass.

Step 1: Evaluate a new sample The first step is to evaluate the procedural shader at a new sample position. The offset $\phi_{i(t)}$ centers the jitter distribution on the appropriate quadrant. The jitter value $g_t[p]$ is drawn from a Gaussian distribution with standard deviation $\sigma_G = 0.2575$. When the values from all the subpixel buffers are combined (and in the absence of motion), this standard deviation gives the best fit to the wider Gaussian kernel ($\sigma_G = 0.4737$). We jitter each pixel independently by offsetting the interpolated values received from the rasterizer before they are sent to the fragment shader.

Step 2: Update one subpixel buffer During this step, we access all subpixel buffers using reprojection to reconstruct $\tilde{b}(p + \phi_{i(t)})$, which is then combined with the sample $s_t[p]$ from step 1:

$$b_{i(t)}[p] \leftarrow (\alpha)\, s_t[p] + (1-\alpha)\, \tilde{b}(p + \phi_{i(t)}). \qquad (12)$$

To minimize the amount of blur, $\tilde{b}$ should favor samples nearest the center of the quadrant being updated. As illustrated in Figure 4, our approach is to define a bilinear tent function Λ_r that encompasses the current quadrant in the framebuffer (with $r = 0.5$):

$$\Lambda_r(v) = \text{clamp}(1 - \tfrac{|v_x|}{r}, 0, 1)\ \text{clamp}(1 - \tfrac{|v_y|}{r}, 0, 1). \qquad (13)$$

Ideally, we would like to compute the weighted sum of all the samples that fall under the support of this function, after these are *forward*-projected into the current frame. However, certain approximations are required to make this operation efficient on graphics hardware. Instead of using forward reprojection, we compute the positions p_k of the centers of these tents reprojected back into each subpixel buffer (reverse reprojection), properly accounting for the differences in quadrant offsets:

$$p_k = \pi_{t\text{-}k\text{-}1}(p + \phi_{i(t)}) - \phi_{i(t\text{-}k\text{-}1)}, \quad k \in \{0,1,2,3\}. \qquad (14)$$

We then approximate the footprint of the projected tent function by an axis-aligned square of radius

$$r_k = \left\| J_{\pi_{t\text{-}k\text{-}1}}[p] \begin{pmatrix}0.5\\0.5\end{pmatrix} \right\|_\infty, \qquad (15)$$

where $J_{\pi_{t\text{-}k\text{-}1}}$ is the Jacobian of each reprojection map, which is directly available using the `ddx`/`ddy` shader instructions. This is to account for changes in scale during minification or magnification.

Rather than accumulating all of the samples in the resulting footprint (which would be too expensive), we consider only the four nearest samples in each subpixel buffer (altogether sixteen samples), located at $\lfloor p_k \rfloor + \Delta$ with $\Delta \in \left\{ \binom{0}{0}, \binom{1}{0}, \binom{0}{1}, \binom{1}{1} \right\}$:

$$\tilde{b}(p + \phi_{i(t)}) = \frac{\sum_{k,\Delta} w_{k,\Delta}\, b_{i(t\text{-}k\text{-}1)}\left[\lfloor p_k \rfloor + \Delta\right]}{\sum_{k,\Delta} w_{k,\Delta}}, \qquad (16)$$

weighting these nearest samples according to the tent function as

$$w_{k,\Delta} = \Lambda_{r_k}\left(p_k - (\lfloor p_k \rfloor + \Delta)\right). \qquad (17)$$

As proved in the appendix, this is *equivalent* to taking the weighted average of the values of each subpixel buffer resampled at positions $\lfloor p_k \rfloor + o_k$ using standard bilinear interpolation

$$\tilde{b}(p + \phi_{i(t)}) = \frac{\sum_k w_k\, b_{i(t\text{-}k\text{-}1)}\left(\lfloor p_k \rfloor + o_k\right)}{\sum_k w_k} \qquad (18)$$

where $w_k = \sum_\Delta w_{k,\Delta}$ and

$$o_k = \left(\frac{w_{k,\binom{0}{1}} + w_{k,\binom{1}{1}}}{w_k} \quad \frac{w_{k,\binom{1}{0}} + w_{k,\binom{1}{1}}}{w_k} \right)^{\mathrm{T}}. \qquad (19)$$

This formulation leverages hardware support for bilinear interpolation and accelerates the resampling process. Finally, note that for the special case of a static scene, the weight $w_{i(t\text{-}4)}$ is exactly one while the others are zero and the offset vector $v_{i(t\text{-}4)}$ is zero, so the previous point estimate is retrieved unaltered. Thus, no blur is introduced and the running estimate converges exactly. Also, in the special case of constant panning motion, each subpixel buffer contributes exactly one sample to the sum in (16) due to the fact that the tent function $\Lambda_{0.5}$ has unit diameter. Thus, each subpixel buffer effectively undergoes nearest-sampling.

Step 3: Compute pixel color The computation of the final pixel color follows a process very similar to that of step 2. We access all subpixel buffers using reprojection to form an estimated value $\tilde{f}(p)$ at the center of the pixel, and combine this estimate with the fresh sample $s_t[p]$:

$$f_t[p] \leftarrow \left(\frac{\alpha}{K}\right) s_t[p] + \left(1 - \frac{\alpha}{K}\right) \tilde{f}(p). \qquad (20)$$

The estimate $\tilde{f}(p)$ is obtained just as in (18), but using position $p'_k = \pi_{t\text{-}k\text{-}1}(p) - \phi_{i(t\text{-}k\text{-}1)}$ reprojected from the center of the pixel rather than the center of one quadrant, and using a tent function with radius $r'_k = 2r_k$ that is twice as large. The current sample $s_t[p]$ appears in this formula because the estimate $\tilde{f}(p)$ is computed using the old contents of buffer $b_{i(t)}$, before it is updated in step 2 in the same rendering pass. It is weighted by α/K because it would contribute to only one of the K subpixel buffers.

Step 4: Update effective number of samples Buffer N_t is updated from N_{t-1} and α_t using formula (8). The algorithm uses a pair of ping-pong buffers to maintain N_t and N_{t-1}.

5.2 Limiting the amount of blur

Given a threshold τ_b on the amount of blur (variance of the sample distribution) and the velocity v due to constant panning motion, we would like to compute the smallest smoothing factor $\alpha_{\tau_b}(v)$ that provides the greatest degree of antialiasing without exceeding this blur threshold. Unlike in the case of traditional bilinear reprojection, which admits a bound on α by inverting (11), our more

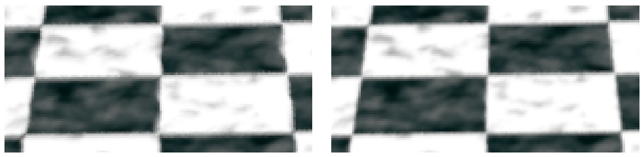


Figure 6: Comparison of signal drift for a simple round-robin subpixel update sequence (left) and our irregular sequence (right).

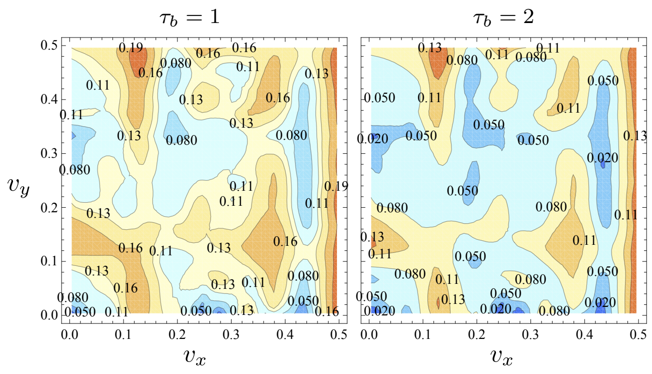


Figure 7: 2D plot of the smallest smoothing factor $\alpha_{\tau_b}(v)$ that respects a blur threshold τ_b, as function of the 2D velocity vector v.

complex algorithm does not lend itself to a similar analysis. The appendix provides a more detailed explanation of why extending these earlier results is impractical. In particular, the mean μ of the sample distribution is no longer guaranteed to be zero. This is due to the fact that the reconstruction kernel $\Lambda_{0.5}$ in (17) effectively reduces to nearest-sampling during panning motion. This can cause the signal to drift spatially and may lead to visible artifacts (Figure 6). We combine two strategies to address this problem:

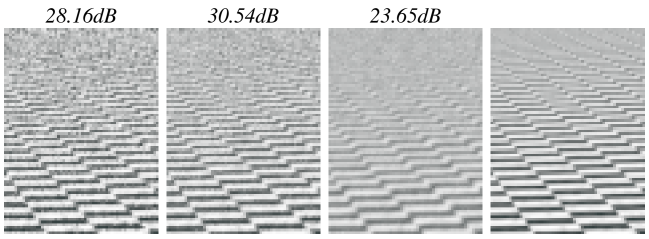
1. Rather than updating the subpixel buffers in simple round-robin order $\big(i(t) = t \mod K\big)$, we use an irregular update sequence that breaks the drift coherence. We found that the following update sequence works well in practice: (0, 1, 2, 3, 0, 2, 1, 3, 1, 0, 3, 2, 1, 3, 0, 2, 2, 3, 0, 1, 2, 0, 3, 1, 3, 2, 1, 0, 3, 1, 2, 0).

2. Rather than bound the variance σ^2 of the sample distribution (second moment about the mean μ), we bound the quantity $\sigma^2 + \mu^2$ (second moment about zero, the pixel center). This simultaneously limits the degree of *blur* and *drift*.

The irregular update sequence makes it difficult to derive a closed-form expression for the second moment of the sample distribution. Instead, we compute $\sigma^2 + \mu^2$ numerically in an off-line simulation. Specifically, we compute the moments of the sample distribution in each subpixel buffer over a range of values of α and v. We then average the moments over an entire period of the update sequence, over the x and y directions according to (10), and over all subpixel buffers. These results are finally inverted to produce a table $\alpha_{\tau_b}(v)$. At runtime, this table is accessed to retrieve the value of α that meets the desired threshold for the measured per-pixel velocity. We found that a 64^3 table is sufficient to capture the variation in $\alpha_{\tau_b}(v)$. Figure 7 shows slices of this function for two different values of τ_b.

Putting everything together, we apply the following update rule in place of (6):

$$\alpha_t[p] \leftarrow K \max\left(\frac{1}{N_{t\text{-}1}+1},\ \alpha_{\tau_b}(v)\right). \tag{21}$$



(a) Small τ_b *(b) Good τ_b* *(c) Large τ_b* *(d) Reference*

Figure 8: Effect of the blur tolerance τ_b. Values of τ_b that are too small aggressively discard earlier estimates and can lead to aliasing, while values that are too large may combine too much resampled data and cause blurring, as can be seen in this close-up view (from top row of Figure 14).

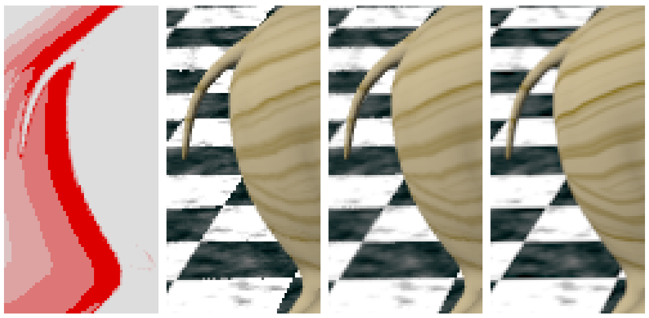

(a) Eval map *(b) One eval* *(c) Adaptive eval* *(d) Reference*

Figure 9: As shown in this close-up view (from Figure 14), we reduce aliasing in newly disoccluded regions by adaptively evaluating additional samples when reprojection fails. The shades of red in (a) indicate the number (1–4) of additional samples.

The factor $K = 4$ is due to the fact that we update each subpixel buffer every fourth frame. At pixels where the scene is stationary, the effect of (21) is to progressively reduce the value of α just as in (6). In the case of a moving scene, however, the value of α is set to give the greatest reduction in aliasing at an acceptable amount of blur as shown in Figure 8.

5.3 Adaptive evaluation when reprojection fails

For surfaces that recently became visible, the reprojection from one or more subpixel buffers may fail. In the worst case, only the new sample $s_t[p]$ is available. In these cases, the shading is prone to aliasing as seen in Figure 9b.

To reduce these artifacts, for each reprojection that fails when computing $\tilde{f}(p)$ in (20) (i.e., $\pi_{t\text{-}k\text{-}1}(p + \phi_{i(t)}) = \varnothing$) we invoke the procedural shader at the appropriate quadrant offset $p + \phi_{i(t)}$. Consequently, the shader may be evaluated from one to five times (Figure 9a) to improve rendering quality (Figure 9c). Fortunately these troublesome regions tend to be spatially contiguous and thus map well to the SIMD architecture of modern GPUs.

6 Accounting for signal changes

The preceding analysis and algorithm assume that the input signal S does not change over time. However, it is often the case that the surface shading does vary temporally due to, for example, light and view-dependent effects such as cast shadows and specular high-

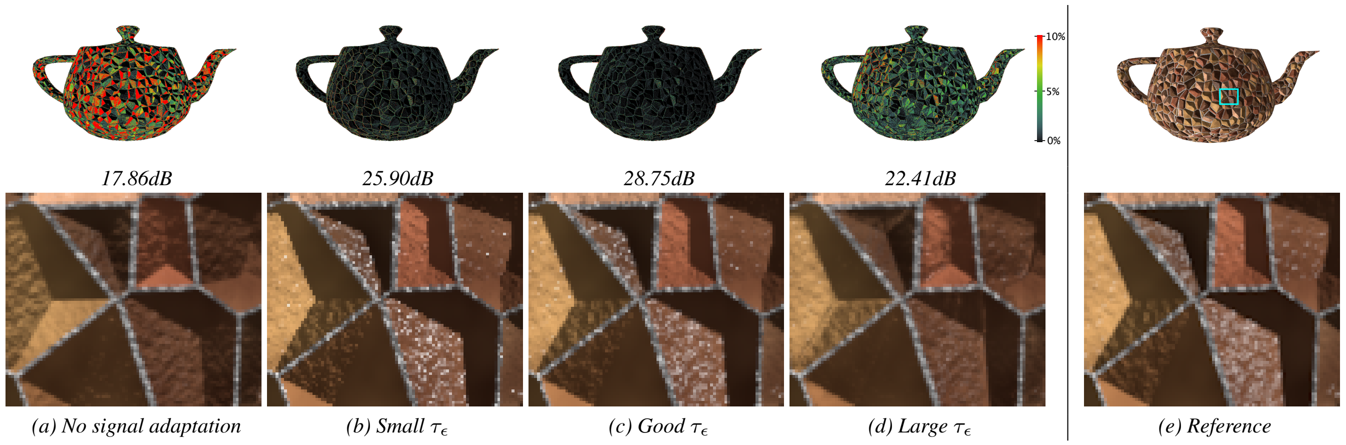


Figure 10: Effect of the residual tolerance τ_ϵ on the teapot scene, which has bump-mapped specular highlights and moving lights. Small values of τ_ϵ give too little weight to earlier estimates and lead to aliasing. Large values result in excessive temporal blurring when the shading function varies over time, as evident in this close-up view. The full-scene difference images are shown at the top.

lights. In these cases it is possible to apply our supersampling technique to only the constant view- and light-independent layers and evaluate the remaining portions of the shading at the native screen resolution or with an alternative antialiasing technique. However, providing a unified framework for antialiasing time-varying surface effects is a worthy goal and we present a preliminary solution to this problem in this section. We describe how to compute a lower bound on α to avoid unwanted temporal blur in the case of shading changes.

6.1 Estimating the residual

Detecting temporal changes in the shading requires estimating the residual between our current shading estimate and its true value:

$$\epsilon_t[p] = (S * G)_t(p) - f_t[p]. \tag{22}$$

Since the correct value of $(S * G)_t$ is naturally unavailable, we would like to use the most current information s_t to estimate this residual $\hat{\epsilon}_t$. However, since we expect s_t to be aliased (otherwise we would not need supersampling), we must smooth this value in both space and time. This corresponds to our assumption that although $S_t(p)$ may contain high-frequency spatial information, its partial derivative with respect to time $\partial S_t(p)/\partial t$ is smooth over the surface. In other words, we assume that temporal changes in the signal affect contiguous regions of the surface evenly. When this is not the case, our strategy for setting α will fail as discussed in Section 8.

Let the buffers e_k store the differences between the recent samples $s_{t\text{-}k}[p]$ and the values reconstructed from the previous contents of the subpixel buffers $b_{i(t\text{-}k)}[p]$ over the last K frames

$$e_k[p] = s_{t\text{-}k}[p] - b_{i(t\text{-}k)}[p], \; k \in \{0,1,2,3\}. \tag{23}$$

We temporally smooth these values by retaining at each pixel the difference with the smallest magnitude

$$e_{\text{smin}}[p] = e_j[p] \quad \text{where} \quad j = \arg\min_k \big|e_k[p]\big|, \tag{24}$$

and obtain our final estimate of the residual by spatially smoothing these using a box filter B_r of radius $r = 3$:

$$\hat{\epsilon}_t[p] = \big(B_3 * e_{\text{smin}}\big)[p] \tag{25}$$

Note that this approach risks underestimating the residual. In other words, when presented with the choice between aliasing or a slower response to signal changes, our policy is to choose the latter.

6.2 Limiting the residual

Similar to our approach for limiting the degree of spatial blur, we would like to establish a lower bound on α such that the residual $\hat{\epsilon}_{t+1}$ in the next frame remains within a threshold τ_ϵ. The choice of τ_ϵ controls the tradeoff between the degree of antialiasing and the responsiveness of the system to temporal changes in the shading.

Following the derivation in the appendix, our strategy to adapt to temporal changes is to replace (21) with

$$\alpha_t[p] \leftarrow K \max\left(\frac{1}{N_{t\text{-}1}+1},\; \alpha_{\tau_b}(v),\; \alpha_{\tau_\epsilon}\right), \tag{26}$$

where

$$\alpha_{\tau_\epsilon} = 1 - \frac{\tau_\epsilon}{\big|\hat{\epsilon}_t[p]\big|}. \tag{27}$$

At pixels where the shading is constant, $\hat{\epsilon}_t$ is less than τ_ϵ and α_t progresses according to the previous rules. When the residual increases, the value of α also increases, shrinking the temporal window over which samples are aggregated and producing a more accurate estimate of the shading. Figure 10 illustrates this tradeoff between aliasing and temporal lag.

The selection of τ_ϵ is closely related to the characteristic of the shading signal. In our experiments, we simply select τ_ϵ that achieves the best PSNR. Alternatively, other numerical or visual metrics can be used to limit both temporal lag and aliasing to acceptable amounts.

7 Results

Scenes We tested our algorithm using several expensive procedural shaders with high-frequency spatial details that are prone to aliasing. The *brick* scene combines a random-color brick pattern with noisy mortar and pits. Bump mapping is used for the lighting. The *horse* scene includes an animated wooden horse galloping over a marble checkered floor. The *teapot* consists of a procedural Voronoi cell pattern modulating both the color and the height field. The added rough detail is bump-mapped with specular highlights.

In addition to the basic scenes above, we show results for an *indoor* scene that has a higher variety of shaders, dynamic elements, and more geometric complexity. The scene consists of several procedurally shaded objects that altogether have over 100,000 triangles.

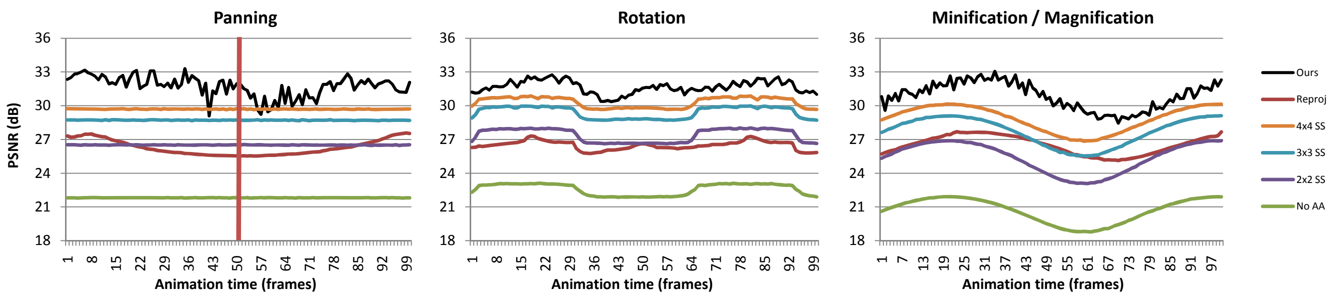


Figure 11: PSNR comparison of our approach with traditional supersampling and jittered reprojection for the brick scene using real-time animation sequences exhibiting different types of motion. The red line indicates the frame used in Figure 14.

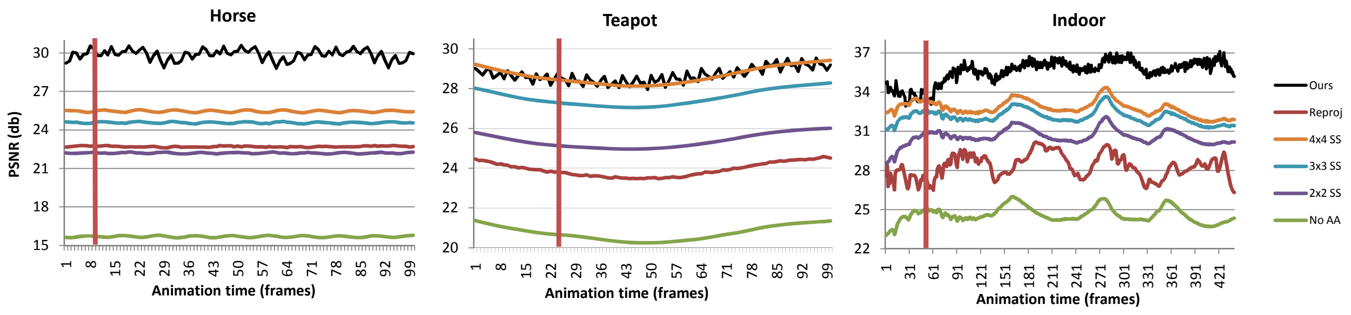


Figure 12: Additional PSNR comparisons of our approach with traditional supersampling and jittered reprojection. The horse is animated, the teapot is dynamically lit, and the indoor scene is animated and dynamically lit. The red lines indicate the frames used in Figures 10 and 14.

These objects include bump-mapped brick walls, a shiny colored bumpy sphere, a glistening uneven marble floor, a reflective stone-bricked teapot, a fine-carved wooden box and a rusty metal exotic creature. The animated scene from the accompanying video combines several types of fast animation and camera motion, exhibiting similar complexity to what one would expect in a typical game scene. It also includes a rotating masked gobo light, which sheds a procedurally generated pattern onto the scene objects. The fast moving light and significant disocclusion caused by the fast motion and complex geometry details makes the scene extremely difficult to handle with traditional reprojection techniques. However, with multiple subpixel buffers and adaptive evaluation, our method produces satisfying results.

Memory usage The depth values required by reprojection are stored in the alpha channel of each subpixel buffer, which are 16-bit RGBA. We store the variance reduction factors $N_t[p]$ in an 8-bit auxiliary buffer. For the teapot and indoor scenes, residuals for the four subpixel buffers are stored in the four channels of one additional 8-bit RGBA buffer. A 1024×768 backbuffer consumes about 3MB, and our scheme uses an additional 27MB to supersample all shaders in a scene.

Comparisons All results are generated using an Intel Core2 2.13GHz PC with 2GB RAM and an AMD HD4870 graphics board at a screen resolution of 1024×768. We measure image quality using the peak signal-to-noise ratio (PSNR) with respect to a ground truth image generated with 256 samples/pixel weighted as in Section 4.1. We show comparisons between conventional rendering (no antialiasing), our algorithm, and traditional 2×2, 3×3 and 4×4 stratified supersampling (performed on the GPU by rendering to a larger framebuffer and then downsampling). In addition, we compare to the basic reprojection method [Nehab et al. 2007; Scherzer et al. 2007] that uses uniform jittered sampling, a single cached buffer, and a value of α chosen to maximize PSNR.

Figure 14 compares our algorithm to these alternatives. Note that standard rasterization is very efficient, but produces significant aliasing, especially under motion (see accompanying video). Jittered reprojection is also faster than our technique, but has inferior quality because it lacks high-resolution estimates and does not adapt α to limit blur. Our technique is also superior to traditional 2×2 and 3×3 stratified supersampling in terms of both rendering speed and quality. Finally, note that our technique gives higher PSNR when compared to 4×4 stratified supersampling in the majority of cases. The teapot from Figure 10 is the most challenging scene due to the fact that it contains a procedural shader with a high-frequency yet fast changing dynamic lighting component. As a result, our method limits the effective number of samples it aggregates to avoid temporal blurring, and this reduces the degree of antialiasing. For the indoor scene in motion, we chose not to antialias the gobo lighting computation to avoid excessive blurring of the pattern (see Section 8). Note, however, that the moving specular highlights over the shiny objects (such as the teapot scene, the sphere, floor and teapot in the indoor scene) at different scales are still properly antialiased without introducing noticeable temporal blur. This demonstrates that our technique can handle signals that change with moderate speed and in a spatially correlated manner. The indoor scene also shows the ability of our approach to preserve antialiased details in the presence of complex and very dynamic changes in occlusion. Overall, note that our algorithm is significantly faster than 4×4 supersampling on all scenes (about 5–10× faster depending on the relative amount of vertex and pixel processing).

Figure 11 and 12 graph the rendering quality (in PSNR) of our scenes for each of the techniques using different animation sequences. The red vertical lines denote the images shown in Figures 10 and 14. For the brick scene, Figure 11 demonstrates the superior quality of our technique under different types of motion: panning, rotation, and repeated magnification and minification. The small oscillations in PSNR do not visibly affect rendering quality as can be verified in the accompanying video. Figure 12 shows that similar results are achieved for the other three test scenes, which include various different types of motion. Again, the accompanying video shows animated versions of these results.

8 Limitations and future work

As with previous reprojection methods, our supersampling algorithm results in a increase in the amount of vertex processing and raster interpolation, as well as pixel shader computation. However, this cost is negligible if the bottleneck lies in the pixel shader, which is increasingly the case.

The work of Méndez-Feliu et al. [2006] makes the interesting point that due to differences in the portion of the scene that is seen below each pixel in adjacent frames, a technique based on multiple importance sampling is required to obtain an unbiased estimator for reprojected color. However, they note that the difference is negligible when reused frames are close to each other, as in our case.

The main limitation of our technique is that it cannot properly detect and antialias arbitrary temporal changes in the shading such as very fast moving sharp specular highlights and shadow boundaries, as well as parallax occlusion mapping. These types of effects cannot be accurately predicted by our reprojection framework because they involve a signal that rapidly moves relative to the object surface. However, we have provided numerous examples of our technique applied to the constant components in complex real-world shaders (i.e., those that are independent of the view and light directions) as well as slowly varying signals, such as the lighting in the teapot and indoor scenes. Note, however, that any strongly dynamic "lighting" effects can be evaluated at the native screen resolution independent of our technique. In the indoor scene, for example, the fast moving gobo light computation was deferred to avoid being severely blurred. Figure 13 shows a side-by-side comparison between amortized and native-resolution gobo lighting, clearly demonstrating that temporally varying effects that result in such extremely fast changes in surface color cannot be properly handled by our framework.

Extending our technique to handle a broader range of temporal effects is a clear direction of future work. We would also like to investigate extending our algorithm to allow supersampling geometry silhouettes in addition to the surface shading, and to allow the efficient rendering of motion blur. Finally, we believe it would be possible to modify the number of subpixel buffers over time in order to achieve a target render quality or framerate.

9 Conclusion

Amortized supersampling is a practical scheme for antialiasing procedural shaders. A key challenge this paper addresses is to characterize the spatial blur introduced by reprojection schemes that involve repeatedly reconstructing intermediate values in a discrete framebuffer. Based on this analysis, we introduced a principled approach for setting the smoothing factor in a recursive temporal filter and demonstrated how the shading can be maintained at a higher spatial resolution while still requiring roughly one shader invocation per pixel. We also introduced a method for handling temporally varying shaders that is appropriate when the temporal derivative has low spatio-temporal frequencies. We demonstrated the efficacy of our approach compared to fixed stratified supersampling and naive extensions of prior reprojection techniques.

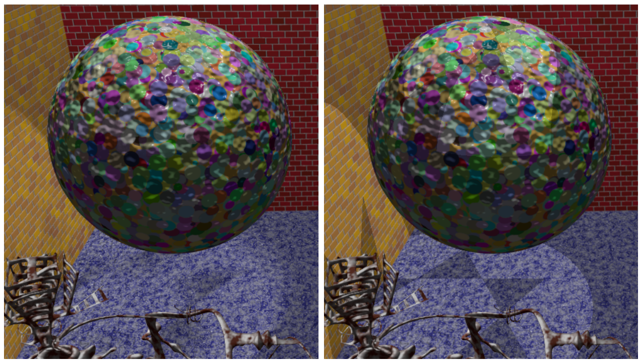


Figure 13: Comparison between performing the gobo computation (left) within our amortized framework, or (right) separately at the native resolution. Note that it is necessary to defer computation of such signals that move rapidly across the object surface.

References

Adelson, S. J. and Hodges, L. F. 1995. Generating exact ray-traced animation frames by reprojection. *IEEE Computer Graphics and Applications*, 15(3):43–52.

Apodaca, A. and Gritz, L. 2000. *Advanced RenderMan: Creating CGI for Motion Pictures*. Morgan Kaufmann.

Badt, S. 1988. Two algorithms for taking advantage of temporal coherence in ray tracing. *The Visual Computer*, 4(3):123–132.

Bala, K., Dorsey, J., and Teller, S. 1999. Radiance interpolants for accelerated bounded-error ray tracing. *ACM Transactions on Graphics*, 18(3):213–256.

Bishop, G., Fuchs, H., McMillan, L., and Zagier, E. J. S. 1994. Frameless rendering: Double buffering considered harmful. In *Proceedings of ACM SIGGRAPH 94*, pages 175–176.

Brooks Van Horn III, R. and Turk, G. 2008. Antialiasing procedural shaders with reduction maps. *IEEE Transactions on Visualization and Computer Graphics*, 14(3):539–550.

Chen, S. E. and Williams, L. 1993. View interpolation for image synthesis. In *Proceedings of ACM SIGGRAPH 93*, pages 279–288.

Cook, R. L., Carpenter, L., and Catmull, E. 1987. The REYES image rendering architecture. In *Computer Graphics (Proceedings of ACM SIGGRAPH 87)*, volume 21, pages 95–102.

Dayal, A., Woolley, C., Watson, B., and Luebke, D. 2005. Adaptive frameless rendering. In *Eurographics Symposium on Rendering*, pages 265–275.

Ebert, D. S., Musgrave, F. K., Peachey, D., Perlin, K., and Worley, S. 2003. *Texturing and Modeling: A Procedural Approach*. Morgan Kaufmann, 3rd edition.

Ernst, M., Stamminger, M., and Greiner, G. 2006. Filter importance sampling. In *IEEE Symposium on Interactive Ray Tracing*, pages 125–132.

Gautron, P., Křivánek, J., Bouatouch, K., and Pattanaik, S. 2005. Radiance cache splatting: A GPU-friendly global illumination algorithm. In *Eurographics Symposium on Rendering*, pages 55–64.

Guenter, B. 1994. Motion compensated noise reduction. Technical Report MSR-TR-94-05, Microsoft Research.

Hart, J., Carr, N., Kameya, M., Tibbitts, S., and Cole-

MAN, T. 1999. Antialiased parameterized solid texturing simplified for consumer-level hardware implementation. In *Graphics Hardware*, pages 45–53.

HASSELGREN, J. and AKENINE-MOLLER, T. 2006. An efficient multi-view rasterization architecture. In *Eurographics Symposium on Rendering*, pages 61–72.

HAVRAN, V., DAMEZ, C., MYSZKOWSKI, K., and SEIDEL, H.-P. 2003. An efficient spatio-temporal architecture for animation rendering. In *Eurographics Symposium on Rendering*, pages 106–117.

HEIDRICH, W., SLUSALLEK, P., and SEIDEL, H.-P. 1998. Sampling procedural shaders using affine arithmetic. *ACM Transactions on Graphics*, 17(3):158–176.

HOGG, R. and TANIS, E. 2001. *Probability and Statistical Inference*. Prentice Hall, 6th edition.

JONES, T. R., PERRY, R. N., and CALLAHAN, M. 2000. Shadermaps: A method for accelerating procedural shading. Technical report, Mitsubishi Electric Research Laboratories.

MARK, W. R., MCMILLAN, L., and BISHOP, G. 1997. Post-rendering 3D warping. In *Proceedings of the Symposium on Interactive 3D Graphics*, pages 7–12.

MÉNDEZ-FELIU, À., SBERT, M., and SZIRMAY-KALOS, L. 2006. Reusing frames in camera animation. *Journal of WSCG*, 14.

MITCHELL, D. P. and NETRAVALI, A. N. 1988. Reconstruction filters in computer-graphics. In *Computer Graphics (Proceedings of ACM SIGGRAPH 88)*, volume 22, pages 221–228.

NEHAB, D., SANDER, P. V., LAWRENCE, J., TATARCHUK, N., and ISIDORO, J. R. 2007. Accelerating real-time shading with reverse reprojection caching. In *Graphics Hardware*, pages 25–35.

NORTON, A., ROCKWOOD, A. P., and SKOLMOSKI, P. T. 1982. Clamping: a method of antialiasing textured surfaces by bandwidth limiting in object space. In *Computer Graphics (Proceedings of ACM SIGGRAPH 82)*, volume 16, pages 1–8.

ROBERT, C. P. and CASELLA, G. 2004. *Monte Carlo Statistical Methods*. Springer, 2nd edition.

SCHERZER, D., JESCHKE, S., and WIMMER, M. 2007. Pixel-correct shadow maps with temporal reprojection and shadow test confidence. In *Eurographics Symposium on Rendering*, pages 45–50.

SCHERZER, D. and WIMMER, M. 2008. Frame sequential interpolation for discrete level-of-detail rendering. *Computer Graphics Forum (Proceedings of Eurographics Symposium on Rendering 2008)*, 27(4):1175–1181.

SHINYA, M. 1993. Spatial anti-aliasing for animation sequences with spatio-temporal filtering. In *Proceedings of ACM SIGGRAPH 93*, pages 289–296.

SIMMONS, M. and SÉQUIN, C. H. 2000. Tapestry: dynamic mesh-based display representation for interactive rendering. In *Eurographics Workshop on Rendering*, pages 329–340.

SITTHI-AMORN, P., LAWRENCE, J., YANG, L., SANDER, P., NEHAB, D., and XI, J. 2008. Automated reprojection-based pixel shader optimization. *ACM Transactions on Graphics*, 27(5):127.

STAMMINGER, M., HABER, J., SCHIRMACHER, H., and SEIDEL, H.-P. 2000. Walkthroughs with corrective texturing. In *Eurographics Workshop on Rendering*, pages 377–388.

TAWARA, T., MYSZKOWSKI, K., and SEIDEL, H.-P. 2004. Exploiting temporal coherence in final gathering for dynamic scenes. In *Proceedings of the Computer Graphics International*, pages 110–119.

WALTER, B., DRETTAKIS, G., and GREENBERG, D. P. 2002. Enhancing and optimizing the render cache. In *Eurographics Workshop on Rendering*, pages 37–42.

WALTER, B., DRETTAKIS, G., and PARKER, S. 1999. Interactive rendering using the render cache. In *Eurographics Workshop on Rendering*, pages 235–246.

WARD, G. and SIMMONS, M. 1999. The holodeck ray cache: an interactive rendering system for global illumination in nondiffuse environments. *ACM Transactions on Graphics*, 18(4):361–368.

WOOLLEY, C., LUEBKE, D., WATSON, B., and DAYAL, A. 2003. Interruptible rendering. In *Proceedings of the Symposium on Interactive 3D Graphics*, pages 143–151.

ZHU, T., WANG, R., and LUEBKE, D. 2005. A GPU accelerated render cache. In *Pacific Graphics*.

Appendix: Detailed derivations

Derivation of (8) According to our definition, N_t is the variance reduction factor that appears in (3), or intuitively the effective number of samples that have so far contributed to the estimate. Omitting the dependency on p for simplicity:

$$\frac{1}{N_t} = \frac{\mathrm{Var}(f_t)}{\mathrm{Var}(f_1)} = \frac{\mathrm{Var}\big(\alpha_t S + (1-\alpha_t) f_{t\text{-}1}\big)}{\mathrm{Var}(f_1)} \tag{28}$$

$$= \alpha_t^2 \frac{\mathrm{Var}(f_1)}{\mathrm{Var}(f_1)} + (1-\alpha_t)^2 \frac{\mathrm{Var}(f_{t\text{-}1})}{\mathrm{Var}(f_1)} \tag{29}$$

$$= \alpha_t^2 + (1-\alpha_t)^2 \frac{1}{N_{t\text{-}1}}. \tag{30}$$

Taking the reciprocal of both sides results in (8).

Derivation of (10) We choose to characterize the blur at each pixel by the covariance matrix of the collection of weighted samples that contribute to it:

$$V[p] = \begin{bmatrix} \mathrm{Var}_x\big(\delta_t[p]\big) & \mathrm{Cov}_{xy}\big(\delta_t[p]\big) \\ \mathrm{Cov}_{yx}\big(\delta_t[p]\big) & \mathrm{Var}_y\big(\delta_t[p]\big) \end{bmatrix}, \tag{31}$$

where $\delta_t[p] = \big\{\delta_{t,i}[p]\big\}_{i=1}^{n(t,p)}$. The two eigenvalues λ_1 and λ_2 of $V[p]$ correspond to the variances along the directions of minimum and maximum variance. We use their average as our estimate for the degree of blur:

$$\sigma_t^2[p] = \frac{\lambda_1 + \lambda_2}{2} = \tfrac{1}{2}\mathrm{Tr}\big(V[p]\big) \tag{32}$$

$$= \tfrac{1}{2}\mathrm{Var}_x\big(\delta_t[p]\big) + \tfrac{1}{2}\mathrm{Var}_y\big(\delta_t[p]\big). \tag{10}$$

Derivation of (11) We first perform the derivation in 1D and then extend it to the 2D case. At time t, the value of a pixel is given by the weighted sum of a set of samples. We can represent each sample in the set by a pair carrying its weight and displacement:

$$F_t = \big\{(\omega_{t,i}, \delta_{t,i})\big\}_{i=1}^{n(t)}. \tag{33}$$

Here, the displacements $\delta_{t,i}$ are relative to the pixel center and the weights $\omega_{t,i}$ sum to one. To simplify notation let us define two operations:

$$a{\cdot}F_t = \big\{(a\,\omega_{t,i}, \delta_{t,i})\big\}_{i=1}^{n(t)}, \quad F_t{+}b = \big\{(\omega_{t,i}, \delta_{t,i}{+}b)\big\}_{i=1}^{n(t)}, \tag{34}$$

which scale all of the weights or translate all of the displacements by an equal amount, respectively. If we ignore the effect of jittering and assume constant panning motion, then every pixel will contain the same set F_t. Furthermore, using (4) and assuming linear resampling, F_t can be written in terms of $F_{t\text{-}1}$ at neighboring pixels:

$$F_t = \alpha \cdot \{(1,0)\} \cup (1\text{-}\alpha)\cdot\big(a\cdot(F_{t\text{-}1}+b) \cup (c\cdot F_{t\text{-}1}+d)\big), \quad (35)$$

where

$$a = 1-v, \qquad b=-v, \qquad c=v, \qquad d=1-v. \quad (36)$$

Note that in (35) all samples are of the form

$$\big(\alpha\,(1\text{-}\alpha)^{j+k}a^j c^k,\ j\,b+k\,d\big). \quad (37)$$

Furthermore, for large enough t, each one appears $\binom{j+k}{j}$ times. We can now rephrase the problem using probability theory and take advantage of several mathematical tools [Hogg and Tanis 2001]. First, let X_t be a random variable that takes on values in the set $\{\delta_{t,i}\}_{i=1}^{n(t)}$, with a probability mass function (p.m.f.) proportional to the corresponding weights $\omega_{t,i}$. Our goal is to compute $\mathrm{Var}(X)$, where $X=\lim_{t\to\infty}X_t$. This can be accomplished by computing the first and second moments of X. The p.m.f. is given by

$$P(X=j\,b+k\,d) = \tbinom{j+k}{j}\alpha(1\text{-}\alpha)^{j+k}a^j c^k. \quad (38)$$

Let $M_X(u)$ denote the *moment-generating* function of X:

$$\begin{aligned}
M_X(u) &= E\big(\lim_{t\to\infty} e^{uX_t}\big) && (39)\\
&= \sum_{j=0}^{\infty}\sum_{k=0}^{\infty}\tbinom{j+k}{j}\alpha(1\text{-}\alpha)^{j+k}a^j c^k e^{u(jb+kd)} && (40)\\
&= \sum_{q=0}^{\infty}\sum_{j=0}^{q}\tbinom{q}{j}\alpha(1\text{-}\alpha)^{q}a^j c^{q\text{-}j} e^{u(jb+(q\text{-}j)d)} && (41)\\
&= \alpha\sum_{q=0}^{\infty}\big((1\text{-}\alpha)(a\,e^{ub}+c\,e^{ud})\big)^q && (42)\\
&= \frac{\alpha}{1-(1\text{-}\alpha)(a\,e^{ub}+c\,e^{ud})}. && (43)
\end{aligned}$$

This function can be used to compute the desired moments:

$$\mu_X = E(X) = M'_X(0) = \frac{(ab+cd)\alpha(1\text{-}\alpha)}{\big(1-(1\text{-}\alpha)(a+c)\big)^2} \quad (44)$$

$$\begin{aligned}
\sigma_X^2+\mu_X^2 &= \mathrm{Var}(X)+\mu_X^2 = M''_X(0)\\
&= \frac{(ab^2+cd^2)\alpha(1\text{-}\alpha)}{\big(1-(1\text{-}\alpha)(a+c)\big)^2}+\frac{2(ab+cd)^2\alpha(1\text{-}\alpha)^2}{\big(1-(1\text{-}\alpha)(a+c)\big)^3}. \quad (45)
\end{aligned}$$

The introduction of sample jittering simply adds a Gaussian random variable G to X. Since $\mathrm{Var}(X+G)=\mathrm{Var}(X)+\mathrm{Var}(G)$, we can substitute (36) into (44–45) to obtain

$$\mu_v = 0 \quad (46)$$

$$\sigma_v^2 = \sigma_G^2 + \frac{1-\alpha}{\alpha}v\,(1-v). \quad (47)$$

Extending this result to 2D requires two modifications. First, the fractional velocity v and offsets $\delta_{t,i}$ become 2D vectors. Second, (35) now contains four terms with $F_{t\text{-}1}$, corresponding to the four pixels involved in bilinear resampling. Because the bilinear weights are separable in v_x and v_y, we can consider the x and y components of these moments separately, and the respective sample sets reduce to the 1D case in (35). Therefore, we can use (47) and (10) to reach (11).

Beyond linear resampling The analysis leading to (44–45) applies without modification to the more sophisticated resampling scheme of Section 5.1, if we restrict ourselves to a simple round-robin update sequence in one dimension (with two subpixel buffers). The weights a, c and offsets b, d then become:

$$\begin{aligned}
b &= \mathrm{frac}(2v+1/2)-1/2 & d &= \mathrm{frac}(v)-1/2 && (48)\\
a &= \bar a/(\bar a+\bar c) & c &= \bar c/(\bar a+\bar c),\ \text{where} && (49)\\
\bar a &= \max(0,1-2|b|) & \bar c &= \max(0,1-2|d|), && (50)
\end{aligned}$$

with $\mathrm{frac}(x)=x-\lfloor x\rfloor$. Substituting (48–49) into (44–45) introduces three complications not previously present in (46–47): (i) the fact that (45) is a cubic makes it inconvenient to obtain an explicit solution for α given a bound τ_b on the variance of the sample distribution; (ii) since $ab+cd$ is generally nonzero, the mean of the sample distribution is also nonzero (and this is the source of the drift in Figure 6); (iii) the weights a, c are no longer separable, so the extension to 2D would result in expressions for the x and y components of the moments that depend on both components v_x and v_y of the velocity vector.

In addition, our use of an irregular update sequence (Section 5.2) invalidates the simple recurrence in (35) altogether. Therefore as explained in the text, we instead precompute a numerical table in an off-line preprocess.

Derivations of (18) and (19) For the sums in (16) and (18) to be equal, the weights associated to each value $b_{i(t\text{-}k\text{-}1)}\big[\lfloor p_k\rfloor+\Delta\big]$ must be the same. This leads to a system of equations for each o_k:

$$\left\{\begin{aligned}
(1-o_{k_x})(1-o_{k_y}) &= w_{k,\binom{0}{0}}/w_k = \beta_{k,0}\cdot\gamma_{k,0}\\
(1-o_{k_x})\ o_{k_y} &= w_{k,\binom{0}{1}}/w_k = \beta_{k,0}\cdot\gamma_{k,1}\\
o_{k_x}\ (1-o_{k_y}) &= w_{k,\binom{1}{0}}/w_k = \beta_{k,1}\cdot\gamma_{k,0}\\
o_{k_x}\ o_{k_y} &= w_{k,\binom{1}{1}}/w_k = \beta_{k,1}\cdot\gamma_{k,1}
\end{aligned}\right. . \quad (51)$$

Note that both sides add up to one, and recall that the tent filters are axis-aligned and separable, which allows us to factor the weights $w_{k,\Delta}$ into products $\beta\cdot\gamma$ as shown above. Therefore there exists a unique solution to each of these systems, given by (19).

Derivation of (27) The residual in the next frame is equal to

$$\begin{aligned}
\hat\epsilon_{t+1}[p] &\approx s_{t+1}[p]-f_{t+1}[p] && (52)\\
&= s_{t+1}[p]-\big((\alpha)s_{t+1}[p]+(1-\alpha)f_t[p]\big) && (53)\\
&\approx s_{t+1}[p]-(\alpha)s_{t+1}[p]-(1-\alpha)\big(s_t[p]-\hat\epsilon_t[p]\big) && (54)\\
&= (1-\alpha)\big(s_{t+1}[p]-s_t[p]\big)+(1-\alpha)\hat\epsilon_t[p]. && (55)
\end{aligned}$$

Here we do not attempt to predict the additional residual introduced due to future signal changes, so we set $s_{t+1}[p]=s_t[p]$ in (55). This leads to the relation

$$\hat\epsilon_{t+1}[p] \approx (1-\alpha)\hat\epsilon_t[p]. \quad (56)$$

Requiring $|\hat\epsilon_{t+1}|$ to be smaller than τ_ϵ, we reach

$$\alpha > \alpha_{\tau_\epsilon} = 1-\frac{\tau_\epsilon}{\big|\hat\epsilon_t[p]\big|}. \quad (27)$$

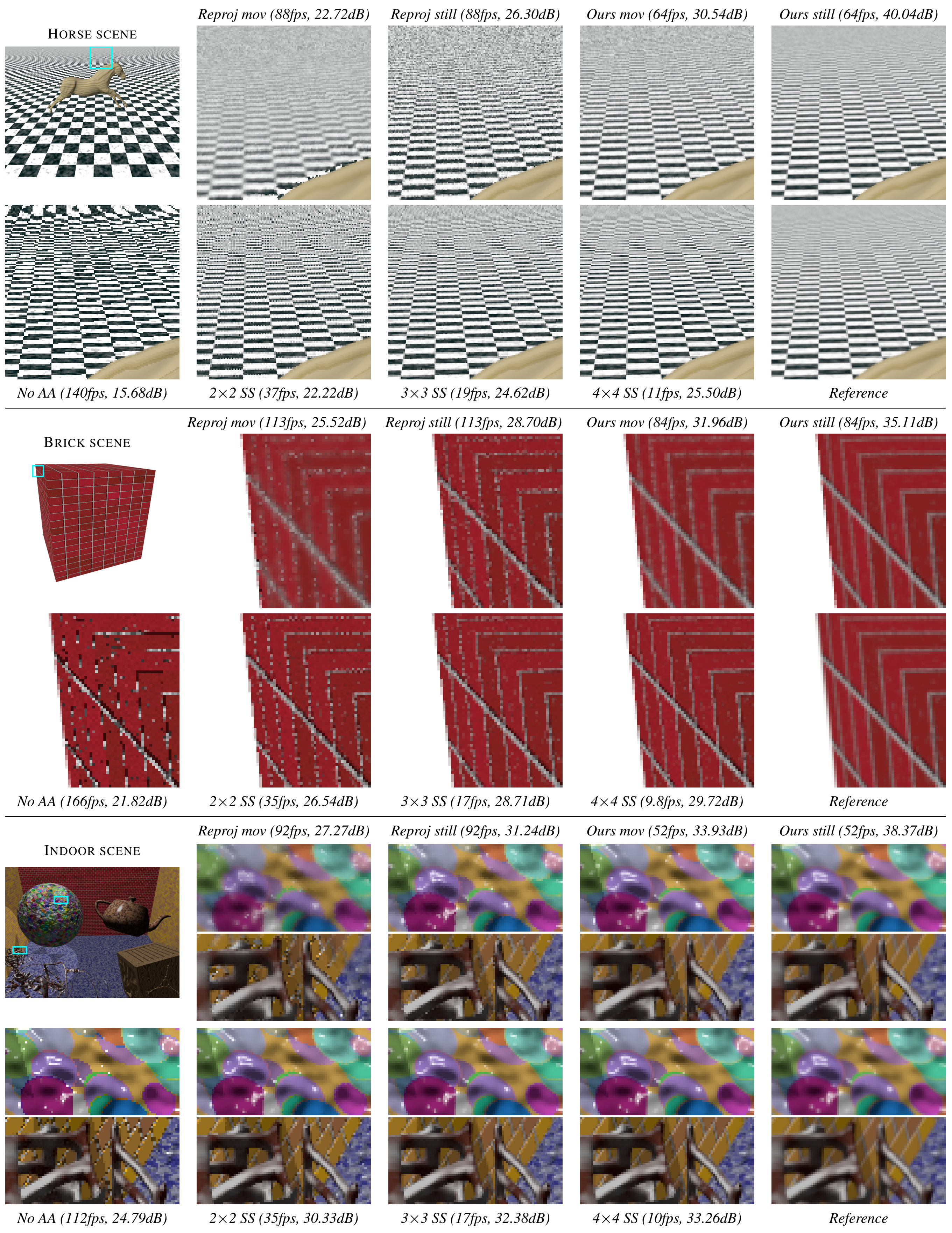


Figure 14: Comparison between our approach, no antialiasing, stratified supersampling, and jittered reprojection.