Spatio-Temporal Upsampling on the GPU

Robert Herzog* MPI Informatik
Elmar Eisemann† Saarland University / MPI / Télécom ParisTech
Karol Myszkowski‡ MPI Informatik
H.-P. Seidel MPI Informatik

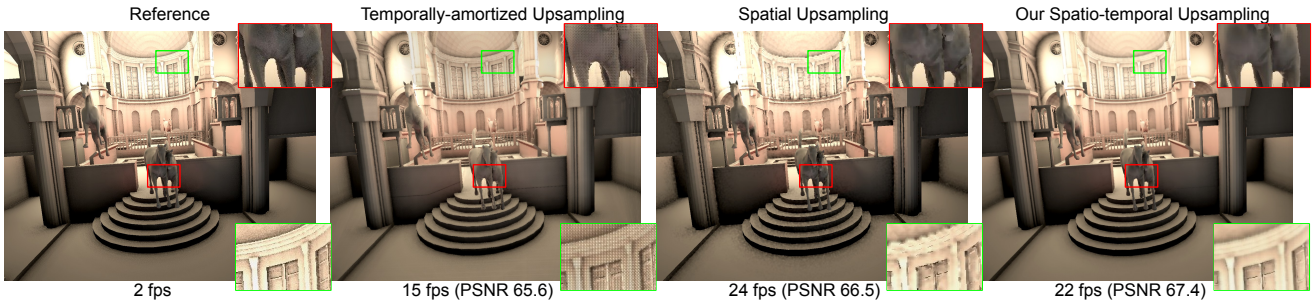

Figure 1: *Comparison of different upsampling schemes in a fully dynamic scene with complex shading (indirect light and ambient occlusion).*

Abstract

Pixel processing is becoming increasingly expensive for real-time applications due to the complexity of today's shaders and high-resolution framebuffers. However, most shading results are spatially or temporally coherent, which allows for sparse sampling and reuse of neighboring pixel values. This paper proposes a simple framework for spatio-temporal upsampling on modern GPUs. In contrast to previous work, which focuses either on temporal or spatial processing on the GPU, we exploit coherence in both. Our algorithm combines adaptive motion-compensated filtering over time and geometry-aware upsampling in image space. It is robust with respect to high-frequency temporal changes, and achieves substantial performance improvements by limiting the number of recomputed samples per frame. At the same time, we increase the quality of spatial upsampling by recovering missing information from previous frames. This temporal strategy also allows us to ensure that the image converges to a higher quality result.

1 Introduction

Shader units are a substantial element of modern graphics cards and lead to significant visual improvements in real-time applications. Despite the tendency towards more general processing units, pixel shading receives a constantly increasing workload. Much visual detail today, such as shadows, ambient occlusion, procedural materials, or depth-of-field, can be attributed to pixel processing and a faster execution often leads to a direct performance increase.

With the current trend towards enhancing the image resolution in modern High Definition (HD) and forthcoming Super High Definition (SHD) imaging pipelines, one can observe that neighboring pixels in spatial and temporal domains become more and more similar. Exploiting such spatio-temporal coherence between frames to reduce rendering costs, suppress aliasing, and popping artifacts becomes more and more attractive.

Our method is driven by the observation that high quality is most important for static elements, thus we can accept some loss if strong differences occur. This has been shown to be a good assumption, recently exploited for shadow computations [Scherzer et al. 2007]. To achieve our goal, we rely on a varying sampling pattern producing a low-resolution image and keep several such samples over time. Our idea is to integrate all these samples in a unified manner.

The heart of our method is a filtering strategy that combines samples in space and time, where the time and spatial kernel can be adapted according to the samples' coherence. For static configurations, the time window can be chosen to be large to produce a high-quality frame. When drastic changes occur, our method automatically favors consistent spatial samples. The result loses some of the visual accuracy, but maintains temporal consistency.

A significant property of our algorithm is locality, meaning that a good filtering strategy is chosen according to the actual image content. Here, we differ from other recent strategies, such as [Yang et al. 2008]. Although our method's overhead is small, we achieve higher quality. Our approach runs entirely on the GPU, leaving the CPU idle for other purposes which is important, e.g., for games.

2 Previous Work

There are several ways to reduce the pixel workload. It is possible to reduce the amount of shaded pixels, using techniques such as early-z/deferred shading, visibility tests [Koltun et al. 2000], or shader culling units [Hasselgren and Akenine-Möller 2007]. In this paper, we exploit coherence by reusing pixel values over space and time.

Temporal methods accelerate rendering by updating only a certain amount of pixels per frame [Bishop et al. 1994], but are susceptible to artifacts arising from changes in view or geometry. An improvement can be achieved by adaptively sampling and reprojecting the frame content [Dayal et al. 2005], but this is most efficient for ray-tracing. In such a context, some solutions suggested 3D warping to accelerate display and global illumination computations [Adelson and Hughes 1995; Mark et al. 1997; Walter et al. 1999] (we refer to [Sitthi-amorn et al. 2008a] for a more complete list of references). These sample-based reprojections can lead to significant fluctuations of sample density in the derived frames. Better quality is obtained by applying per object 4D radiance interpolants with guaranteed error bounds [Bala et al. 1999]. The reduced sampling

*e-mail: rherzog@mpi-sb.mpg.de
†e-mail: eisemann@mpi-sb.mpg.de
‡e-mail: karol@mpi-sb.mpg.de

coherence, or the involvement of ray tracing (although steps in this direction exist) make such solutions less GPU-adapted.

Reprojection caches [Nehab et al. 2007; Scherzer et al. 2007; Sitthi-amorn et al. 2008a] are more GPU-friendly. Here, supplementary buffers encode fragment movements. In contrast to image analysis, for geometry, it is relatively cheap to obtain displacement information by evaluation in the vertex shader and storage in the framebuffer. Given a current fragment, one can easily verify its presence in the previous frame and, if possible, reuse its value. This recovered information is not necessarily a final color, but can be an intermediate shader result, the so-called *payload*. The underlying assumption is that, for a fixed surface point, such value should remain constant over time. Such an assumption gave rise to the idea of integrating samples over time for static scene antialiasing [Nehab et al. 2007], where static also excludes illumination variations and changes introduce significant artifacts.

In concurrent work Yang et al. [Yang et al. 2009] extend this idea and propose an adaptive method for antialiasing with reprojection caching with a profound error analysis. They adapt the exponential smoothing factor that determines the decay of previously computed samples over time to temporal changes in shading and to blur due to the bilinearly filtered reprojection cache. Similar to our method they temporally interleave frames to virtually increase the spatial resolution. They target antialiasing and, thus, use a 2×2 sub-pixel buffers which is sufficient for their purposes. In contrast, our upsampling method uses larger windows and we propose a multiresolution approach depending on the payload. We further rely on a geometry-aware upsampling which is important for our context as it allows for a larger spatial support. We propose an efficient recovery of the concerned supplementary information and exploit it in the filtering process. This technique allows us to handle missing pixels and therefore we do not need to resort to a hole-filling per-pixel rendering and we can deal with silhouettes that are recovered from a higher resolution image, whereas Yang et al. [Yang et al. 2009] do not exploit this possibility. We further propose a way to adapt the filter kernel depending on the confidence of spatial or temporal data and use a different filtering scheme for temporal antialiasing to estimate the gradient for varying shading.

To detect almost constant shader components, one can use a learning stage with particular objects [Sitthi-amorn et al. 2008b]. This requires a long preprocessing time and such a setup cannot exploit coherence that might arise from the application itself. E.g., if the shader is light-position dependent, but the light is stationary during the execution. Our use of changing sample patterns enables a detection of inconsistent payloads. Nevertheless, our method would still benefit from a good shader decomposition, but we see such strategies as orthogonal to our goal.

Yang et al. [Yang et al. 2008] deal with dynamic changes and reduce the shading workload by producing low resolution frames that are upsampled to produce a complete frame. The downside of such a solution is that high-frequency detail might not be captured in the low resolution frame, and hence it is not always possible to deduce the information needed for the current frame. Super-Resolution techniques, e.g., [Calle and Montanvert 1998], deal with this problem and attempt to add new frequencies recovered through de-interlacing, image content analysis, or edge preserving interpolation. In our solution, we work with low resolution information to ensure a higher efficiency, and recover better high frequency estimates with a temporal strategy.

We involve samples from previous frames by compensating for motion and perform spatio-temporal filtering to exploit temporal coherence. Such filtering is commonly used in video restoration [Tekalp 1995; Bennett and McMillan 2005], and has been successful in suppressing aliasing artifacts in ray tracing [Shinya 1993]. Temporal coherence has been used to greatest advantage in a number of global illumination solutions discussed in the survey paper by Damez et al. [Damez et al. 2003]. Many of the presented techniques are off-line or even require knowledge of subsequent keyframes which is unacceptable for interactive rendering purposes. Other approaches exploit temporal coherence at a very low level, e.g., single photon paths. Low-level coherence usually gives more flexibility and enables the exchange of information between many frames at once. However, it is difficult to be efficiently exploited on current GPUs. In our solution, we rely on more GPU-compatible strategies that relate to interleaved sampling [Wald et al. 2002; Segovia et al. 2006] which has roughly similar goals in its CPU and CPU/GPU incarnation.

3 Upsampling

In this section, we will explain our upsampling strategy. As it is inspired by previous work, we will first review spatial upsampling (Section 3.1) then temporally-amortized spatial upsampling (Section 3.2), also referred to as *reprojection caching*. Step by step, we will describe our modifications before presenting our spatio-temporal solution (Section 3.3).

3.1 Spatial Upsampling

Yang et al. [2008] assume that expensive shader computations are spatially slowly varying and can be reconstructed by sparse sampling followed by interpolation. This is true for many low-frequency shaders, e.g., lighting computation. The authors apply a joint-bilateral filter [Eisemann and Durand 2004; Petschnigg et al. 2004] to perform an image upsampling [Kopf et al. 2007] where the filter weights are steered by geometric similarity encoded in a high-resolution *geometry buffer*. This means that samples which are close in world space and have similar surface orientation are better interpolation candidates. For simplicity, we will use 1D pixel indices i or j in the following derivations. Given the high-resolution geometry buffer and the low-resolution shading result l, the upsampled payload $h(i)$ can be computed as

$$h(i) = \frac{1}{\sum w_s} \sum_{j \in N\{i\}} w_s(i,j) \cdot l(\hat{j}), \tag{1}$$

where $N\{i\}$ is a neighborhood around i, $\hat{j}$ is the index of the nearest pixel in the low-resolution image and $w_s(i,j)$ is a spatial geometry-aware pixel weight defined as:

$$\begin{aligned} w_s(i,j) \quad = \quad & n(\max(0, 1 - (\vec{n}_i \bullet \vec{n}_j))^2, \sigma_n^2) \cdot \\ & d(|z_i - z_j|^2, \sigma_z^2) \cdot k(i,j) \end{aligned} \tag{2}$$

The weight consists of a geometric weighting function involving orientation n, linear depth d, and an image space filter k. For simple spatial upsampling, k can be chosen arbitrarily, e.g., a linear hat function [Yang et al. 2008]. The σ terms are user-defined variables. Whereas we kept $\sigma_n = 0.2$, σ_z depends on the scene. Throughout the paper, we used 3% of the difference between the near and far frustum plane.

For higher efficiency, we choose N to cover only the four nearest pixels in the low-resolution image l. One can choose n and d freely as long as it favors similarity, but falls off quickly. The method in [Yang et al. 2008] uses Gaussian filters, which can be relatively expensive. We use a simpler yet similar function for both n and d:

$$g(x, \sigma, \gamma) = (\max(\epsilon, 1 - (x/\sigma)))^\gamma . \tag{3}$$

σ represents the filter width, and γ controls the fall-off. We choose $\gamma = 3$, which corresponds to the *tri-weight* kernel $g(x^2, \sigma^2, 3)$. A plot of the function for various σ is shown in Figure 4(left). It has finite support and is clamped at a small epsilon which avoids zero weights.

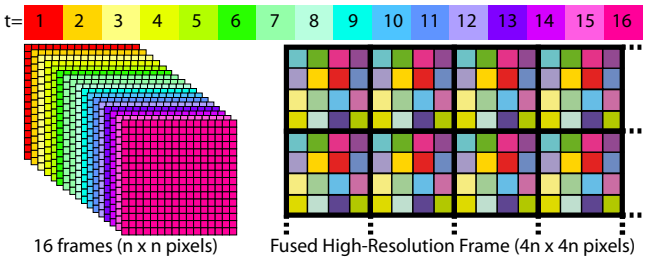

Figure 2: *Left: A regularly sampled low-resolution shaded image is produced for each frame. The high-resolution output is then fused from the previous frames.*

3.2 Temporally-Amortized Spatial Upsampling

To increase the final image quality, one can also look back in time. Instead of interpolating within the current frame, missing pixels are resurrected from past frames. This is most beneficial if frames correspond to differing pixel subsets. Otherwise, for the static case, no new information is gained over time and accurate convergence becomes impossible.

A random pattern was proposed in [Nehab et al. 2007], but it costs some efficiency because it implies a supplementary rendering pass to fill in the missing information, slightly accelerated via early-z termination. For our method, we want to avoid the computation of extra samples in order to ensure a relatively constant cost per frame, just like [Yang et al. 2008]. To make updates efficient on the GPU, we use a spatially regular pattern as shown in Figure 2. To render a sample set, we apply translations in post perspective space which can be encoded in the projection matrix and come at no supplementary rendering cost. The regularity will also be helpful for the spatio-temporal filtering (Section 3.3). We will concentrate on a 4×4 window for the rest of this section, but discuss other sizes in Section 4. The process is illustrated in Figure 2.

If pixels were simply reused from previous frames, visible distortions are likely to appear. Any camera movement or scene changes result in ghosting trails following the objects. One can compensate for this effect by computing pixel displacements between frames, referred to as motion flow. Motion flow is inexpensive because all the necessary 3D information is available [Sitthi-amorn et al. 2008a]. To improve quality, and because it is cheap, we compute the 2D motion flow in a high-resolution texture, always considering only two successive frames. For brevity we will refer to the q-frames motion-compensated pixel i at time t as $i(t-q)$ (i.e., $i(t) = i$).

In standard temporally-amortized upsampling one can keep the payload of the updated pixel set in the final output and let the other pixels search in the previous frame:

$$h(i,t) = (1 - w_t(t,i))\; h((i(t-1), t-1) + w_t(t,i)\; l(\hat{i}, t), \quad (4)$$

where $h(i,t)$ is the high resolution image payload for the current pixel index i at time t, which is composed of the currently computed low-resolution image payload $l(\hat{i}, t)$, $\hat{i}$ is the nearest low-res. pixel index of i, and the previous image payload $h(i(t-1), t-1)$ at the motion compensated position $i(t-1)$. Special care has to be taken if $i(t-1)$ is pointing to a pixel outside the screen or if the 3D point corresponding to the pixel $i(t-1)$ is disoccluded (i.e. has not been visible in the previous frame). Those pixels cannot be fetched from the previous frame $h(\cdot, t-1)$ and, for temporally-amortized upsampling, are usually recomputed. To identify disocclusions we compare our warped depth values with the depth values from previous frame [Nehab et al. 2007]. Comparing only depth values we may miss certain disoccluded pixels located at contact points (see Fig. 3). To clear the ambiguity we also compare material IDs, which we store anyway in the geometry pass of our deferred renderer. For temporally-amortized upsampling, pixels are taken from

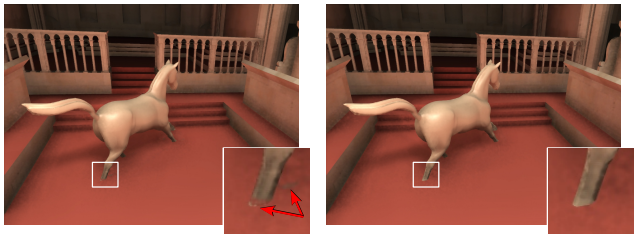

Figure 3: *Dealing with disocclusions in temporal reprojection: reprojection based on z-Buffer comparison may fail at contact points (left). Additional criteria (here material IDs) help to reduce ambiguities (right).*

the previous payload if they have not been computed in the current frame. The binary weight w_t determines for a given pixel whether to fetch the payload from the current or the previous frame(s), by setting $w_t(t,i) = 1$ if the pixel i has been computed in the current frame at time t, and zero respectively.

3.3 Spatio-Temporal Upsampling

The previously described upsampling schemes are not always well suited. Constructing the image only spatially is very efficient, but prone to undersampling and blurring of sharp image features. On the other hand, temporal caching [Sitthi-amorn et al. 2008a; Nehab et al. 2007] is sensitive to temporal changes, but it converges if the scene is nearly static. Consequently, we would like to combine the two approaches in a spatio-temporal upsampling framework:

$$h(i,t) = \frac{1}{\sum w_s w_t w_f} \sum_{q=0}^{T} \sum_{j \in N\{i(t-q)\}} w_s(i(t-q), j)\; w_t(t-q, j)\; w_f(q)\; l(\hat{j}, t-q), \quad (5)$$

where $w_f(q)$ is a temporal fadeout kernel to favor payloads that have been computed recently ($w_f \in [0..1]$). A good choice is an exponential falloff, e.g., $w_f(q) := 0.9^q$, but we improve upon this in Section 3.5. The other terms were explained in Equations 1 and 4. As indicated before, T is usually 16 (i.e. 4×4 upsampling), which is the amount of low-res textures needed to cover all samples of the high-resolution frame. Intuitively, we follow the sample i back over time. For each time step t, w_t ensures that only those payloads are considered that have been computed at time t. The contribution is then influenced by the weight w_s that measures the geometric difference and w_f, that penalizes age. The pixel filter k, computed in w_s, has to be chosen carefully. Precisely, we define $k(i,j) := g(||x(i) - x(j)||, r, \gamma)$, where $x(i)$ is the pixel position on the screen, and r the low-res pixel diagonal length. For $\gamma = 0$ filter k is constant and the results approach spatial upsampling. For $\gamma = \infty$, k is a dirac-like filter ($k(i,j) := 1$ if $(i = j)$, 0 otherwise) and, hence, is equivalent to temporally-amortized upsampling. Consequently, we want to use the function k to blend between the two extrema depending on the temporal coherence, see

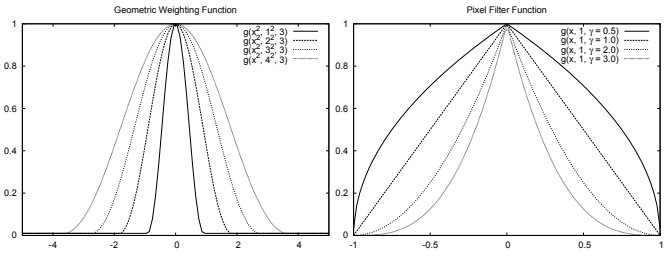

Figure 4: *Left: geometric weighting function n, d (triweight kernel), right: image space filter function k.*

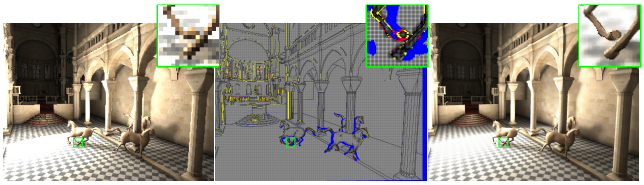

Figure 5: *Left: dynamic shading result, middle: sum of upsampling weights w_s (brightness represents confidence, yellow regions have insufficient spatial weight, blue regions are disoccluded and have no temporal weight, and red pixel are both), right: upsampled image*

Figure 4(right). A better convergence over time, is achieved with a larger γ, while a lower value improves the temporal response. We will present a temporal gradient-guided steering of γ in Section 3.5 and one can assume for the moment that $\gamma = 3$.

Figure 5(middle) visualizes the spatial upsampling weights w_s of 4×4 pixel regions. Pixels marked as blue are disoccluded. Consequently, these cannot be taken from previous frames. Yellow pixels indicate a small geometric weight for the current frame and are, thus, likely to be undersampled. Here, previous frames should be used. Pixels in red have neither spatial nor temporal confidence, making them candidates for recomputation. In general, such pixels are sparse and our filtering mechanism ensures that a plausible value is attributed, making it possible to avoid recomputation.

3.4 Exponential History Buffer

Our filtering in Equation 5 leads to better results than spatial or temporal filtering alone, but it's beyond question that it is computationally expensive because we need to perform many dependent texture lookups repeatedly to produce one pixel. To improve performance, we observe that the lookups are coherent over time. It is thus possible to cache these lookup chains through previous frames. This leads to the idea of an exponential history buffer. It stores the previous upsampling result h_r, and the previous spatio-temporal upsampling weight h_w. Ignoring the temporal component in Equation 5, we obtain a standard bilateral upsampling:

$$\tilde{h}(i,t) = \frac{1}{\sum w_s w_t} \sum_{j \in N\{i\}} w_s(i,j) w_t(t,j) l(\hat{j},t) \qquad (6)$$

with weight $\tilde{w}(i) = \sum w_s w_t$. Given $\tilde{h}$, and our history buffers h_r and h_w, we compute the actual output h via weight-dependent blending:

$$h(i,t) = \frac{\tilde{h}(i)\ \tilde{w}(i) + w_f(1)\ h_w(i(t-1))\ h_r(i(t-1))}{\tilde{w}(i) + w_f(1)\ h_w(i(t-1))}. \qquad (7)$$

The history buffer is updated for the next frame by storing $h_r(i) := h(i,t)$ and $h_w(i) := \tilde{w}(i) + w_f(1)\ h_w(i(t-1))$. If one has chosen w_f as an exponential falloff, Equations 5 and 7 are equivalent. Note that $w_f(0) = 1$. This reduces the performance penalty with respect to [Yang et al. 2008] to a single texture lookup and a few algebraic operations. Furthermore, other simplifications are possible: we no longer need 16 low resolution textures (one high-resolution history buffer and one low resolution frame are enough), time dependence vanishes in general, and in particular, for the function w_f.

3.5 Temporal Weighting Function

In the previous section, we have seen an efficient way to involve a longer temporal history. Nevertheless, in some cases this might not be wanted. Fast changes, induced by, e.g., shadows, would profit from the use of spatial upsampling instead of temporal. On the other hand, to improve the quality of the shaded output, it is necessary to integrate samples from previous frames. To this extent, we will dynamically adapt the temporal weight w_f and the image space filter k locally.

Precisely, we want to address the following points:

1. Temporal flickering due to low resolution sampling of high-frequency spatial signals;
2. Temporal ghosting artifacts due to fast temporal changes;
3. Higher-quality convergence for static elements.

To choose w_f and falloff of k appropriately, we examine the variance of the temporal signal. We do this by relying on a temporal gradient. A weak gradient implies that more confidence can be granted to the evaluation over time. Consequently, w_f should be large and k should reduce image blur by giving more weight to actually computed pixels in the history buffer. Contrarily, large gradients indicate that changes occurred and that the history is no longer reliable. Accordingly, w_f should fall off faster and k should favor spatial upsampling by introducing more image blur. We achieve this by scaling the exponential fall-off γ of our filter function k defined in Section 3.3 with our confidence value w_f, where a maximum of $\gamma \cdot w_f = 4$ is a good choice and was used in all our experiments.

3.5.1 Estimating Temporal Gradients

We will assume that the payload h is one dimensional. Otherwise, one could either compute a norm of the payload, or consider separate gradients. We rely on finite differences to define a gradient. To address the case where the value domain is unknown, we realized that it usually makes sense to compute relative gradients. In particular, for colors, such a definition tends to capture contrast (when reducing color to luminance).

$$\frac{\partial}{\partial t} h(i,t) = \frac{h(i,t) - h(i(t-1),t-1)}{max(|h(i,t)|, |h(i(t-1),t-1)|)}.$$

A more general definition according to time is not needed because the previous section reduced the time dependence to a single frame. We then define:

$$w_f := a \cdot \left(1 - |\frac{\partial}{\partial t} h|\right)^2, \qquad (8)$$

where a steers the sensitivity between spatial and temporally-amortized upsampling. Usually a is kept to 1 in our experiments. The square in Equation 8 suppresses w_f more aggressively for large temporal gradients than for small fluctuations. One may notice that w_f depends on $h(i,t)$, which is the result we currently want to

compute. Therefore, we approximate h based on $\tilde{h}$, the spatially upsampled result we computed to make use of the history buffer.

Because our spatial upsampling $h(i,t)$ may suffer from temporal flickering and finite differences are potentially always noisy, we decided to use a low-pass filtering in time *and* space. Simple spatial filtering, e.g., using mipmapping, would not be an option since it cannot detect spatial aliasing artifacts (which might result in low-frequency). However, spatial aliasing corresponds to a periodic high-frequency temporal signal, which is greatly reduced when summing a few consecutive frames. Therefore, we address this issue by making use of the current and the three previous finite differences. These are stored in a vector $\vec{\nabla} := \left(\frac{\partial}{\partial t}h(i(t),t), \ldots, \frac{\partial}{\partial t}h(i(t-3),t-3)\right)$. To filter the gradient temporally, we compute an absolute weighted sum $||\vec{\nabla}|| := |\vec{\nabla} \bullet (0.4, 0.3, 0.2, 0.1)^T|$. The sum, as well as the current and the two previous differences can be stored in a single RGBA texture.

While the finite differences will only be relevant for the next frame, $||\vec{\nabla}||$ is used to control the temporal-spatial upsampling. Though filtered over time, this value can still fluctuate spatially. To filter it, we want to use a kernel that is slightly larger than the distance between recomputed samples. Thus slightly larger than 4×4. In practice, we found that tri-linearly filtered mipmaps deliver good quality, when values are chosen from level 2.5. This leads to an improved estimate, in particular, in the presence of aliasing as can be seen in (Figure 7). This temporally and spatially filtered value $||\vec{\nabla}||$, is then used in Equation 8.

Care has to be taken in the special case where pixel regions are disoccluded and cannot be reprojected to the previous frame. Here, computing a temporal gradient is impossible. Simply ignoring the gradients for such pixels may lead to visible discontinuities at the transition boundaries between successfully reprojected pixels and disoccluded pixels. In order to suppress these discontinuities, we set the relative temporal gradient to its maximum for all disoccluded pixels. In consequence, the temporal gradient spreads into the neighboring pixels after the spatial low-pass filtering. This leads to a gradually diminishing gradient and, hence, a smooth transition between spatial and temporally-amortized upsampling (see Fig. 6).

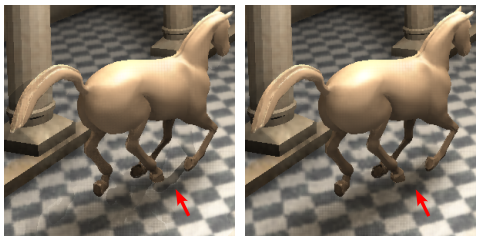

Figure 6: *Disoccluded pixels in this cropped screen-shot of a running horse can cause discontinuities along motion boundaries (left), which are smoothed when setting the maximum relative gradient for all disoccluded pixels (right) since it is spread to neighbor pixels during the spatial low-pass filtering.*

It has to be pointed out, that whenever the gradient has been falsely assumed to vary, our method does not *break* and instead falls back to spatial filtering. The adaptation of w_f only improves the quality of the results. Even though, our algorithm is not ensured to converge to the actual high-resolution solution, the increase of the temporal window improves quality significantly as illustrated in Figure 7 (left) and the accompanying video.

Alternatively, temporal gradients could be obtained via joint-bilateral spatial upsampling of the known temporal gradients based on the newly computed values. In practice, this resulted in more expensive and qualitatively less convincing results. Our solution filters space and time, for very little cost and better handles disocclusions. We investigated special cases, like zero motion vectors, but found that quality remained similar, making the supplementary memory load for storing these motion vectors unjustified.

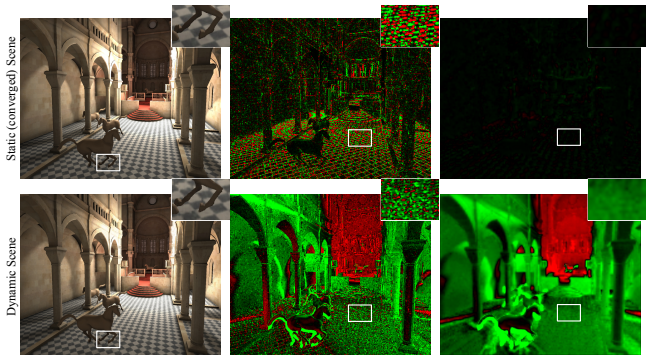

Figure 7: *Temporal gradients are important to detect changes in illumination. The figure shows gradients in a static (frozen) scene (top row) and the same frame in a fully dynamic scene (bottom row) (red = negative, green = positive gradients). Simple finite differences (center) are prone to aliasing and noise. Our filtering (right) regularizes the gradient estimation (e.g., for the static scene the gradient is almost zero (top right)). It eliminates flickering and achieves high quality. Some high-frequency gradients (bottom right) might be lost, but this is almost invisible in a dynamic context. Our upsampled result is shown on the left. (For demonstration purposes, we show signed gradients. Spatial smoothing relies on absolute values $||\vec{\nabla}||$.)*

4 Implementation Details and Overview

So far, we considered upsampling windows of size 4×4, this is a strong approximation and up to 16 frames are needed to produce an accurate result in a static scene, although high-quality convergence is often faster. For fluctuating payloads it can make sense to decrease the window size. The algorithm easily extends to various upsampling sizes, such as 2×2 or 8×8, but we found that for a screen resolution of 1280×1024, 2×2 windows of varying values are better addressed with simple temporally-amortized upsampling (not even spatial bilateral upsampling is needed) and bigger windows take too much time to converge to a high quality result. Nevertheless, for even higher resolutions, this might change. Further, complex shaders often consist of several independent shading terms, some of which can be of very low frequency and high computational complexity. In fact, most high-frequency components (such as textures, direct light shadows) are not expensive and could be efficiently computed for every pixel. Hence, following [Sitthi-amorn et al. 2008b], splitting the shader into individual components enables even higher gains. To illustrate the effectiveness of our approach, we refrained from such decompositions in our results.

Our spatio-temporal upsampling is integrated into a deferred shading approach. In the initial pass all necessary geometry and material information as well as the 2D motion vectors are written to the high-resolution G-buffer. In the next pass all expensive shading computations are performed at low resolution. The frustum is jittered in a coherent sampling pattern (see Figure 2) corresponding to a different subset of pixels. Next, we upsample the current shading result to high resolution taking into account the high-resolution G-buffer and motion flow and solely the previous upsampling result

	shade	temporal		spatial		spatio-t.		ref.
	ms	fps	ms	fps	ms	fps	ms	fps
Jeep (PCF)	1.6	244	1.0	239	1.1	191	2.1	171
stdev.	0.02	2.4		1.6		1.5		
Horses (AO)	11.3	47.5	5.3	62.5	1.1	55.9	2.2	7.6
stdev.	0.09	0.69		0.36		0.26		
Horses (AO+GI)	35.7	11.4	45.2	24.5	1.6	21.8	3.0	2.2
stdev.	0.27	3.2		0.56		0.78		
Elephant (AO+DO)	80.3	5.9	76.1	12.3	1.8	11.9	2.9	0.7
stdev.	39.7	4.5		5.1		4.9		

Table 1: *Final average fps and standard deviation (stdev.) for rendering a frame with spatial, temporally-amortized with disocclusion-hole filling, spatio-temporal (spatio-t.) upsampling and reference (ref.) for image resolution of* 1280×1024 *and a* 4×4 *upsampling window. The cost for the low-res input (shade), and the time spent on upsampling only is given in [ms].*

(thanks to Section 3.4) and the (mipmapped) temporal gradients $\vec{\nabla}$. Because the temporal gradient after filtering is assumed to be spatially coherent, we do not need to compute the upsampled gradients at a high resolution, but instead compute it at an intermediate resolution (e.g., at 2×2 pixels).

Antialiasing Our spatio-temporal upsampling is extendable to antialiasing in the spirit of [Yang et al. 2009]. A common way for antialiasing with deferred shading is to compute the results in a $m \times m$ higher resolution and then merge $m \times m$ pixels to obtain the average pixel color. Such scheme nicely fits to our proposed upsampling. We only need to compute the geometry and material buffer and the intermediate upsampling buffers at a higher resolution than the display buffer and down-sample the upsampled result to the display resolution. To some extent this follows the hardware-based antialiasing which computes shading values at a lower resolution. Nevertheless, the hardware antialiasing does cannot involve previous frames, nor is it compatible with deferred shading which results in a significant overhead and its quality depends on the shader's spatial frequency.

5 Results

For comparison, we implemented simple temporally-amortized upsampling as described in Section 3.2 with second-pass hole filling [Sitthi-amorn et al. 2008a] and spatial upsampling [Yang et al. 2008]. We used OpenGL with GLSL on a GeForce GT 280 and performed tests on various challenging scenes at a resolution of 1280×1024. The timings are giving in Table 1 and the visual results with statistical error measures (PSNR) are shown in Figure 8. The initial geometry pass of our deferred renderer deviates only slightly throughout the different methods and is not explicitly shown in Table 1. The overhead for computing the motion vectors was always less than 1 ms in our experiments.

The first scene is a jeep with little geometry, but a moving light source, challenging texture, percentage-closer filtering (PCF) [Reeves et al. 1987] with a 6×6 kernel, Figure 8(top). The second scene, contains animated horses, Figure 8(bottom), with a static camera and applies screen-space ambient occlusion (SSAO) [Bavoil and Sainz 2008] with 8×8 randomized horizon samples. In a different scenario, Figure 1, we added instant global illumination by evaluating a large number (2000) of virtual point lights (VPLs) without visibility generated with reflective shadow maps [Dachsbacher and Stamminger 2005]. In this setup the camera is moving and the indirect lighting is changing quickly. Our approach even suppresses the flickering due to temporal noise of the VPL sampling. Please refer also to the video material provided with this paper. Finally, the last scene shows a running elephant with a large animated body which is rendered offline at fixed frame rate (30 fps) with SSAO (32×24 samples), a spot-light with PCF, and screen-space directional occlusion (SSDO) [Ritschel et al. 2009] with direct light sampling (128 samples) from an environment map. This scene is challenging as the viewpoint changes very quickly and reveals large surfaces making it difficult to reproject old samples. Furthermore, the shading is of high-frequency and cannot be accurately reproduced by neither method. To favor spatial upsampling, we set $a = 0.7$ for this scene. Temporally-amortized upsampling completely fails in this scenario whereas our method, similar to spatial upsampling, still produces decent results.

Results generally benefit from temporally-amortized and spatial upsampling. The horse scene shows that in static regions the SSAO converges nicely whereas in dynamic regions, near the running horses, more values are taken from the current spatial upsampling, trading-off response for quality. The jeep's texture (even when not separating it from the payload) faithfully converges and the shadow shows only minor artifacts that are hidden by its motion. Temporally-amortized upsampling shows many ghosting artifacts and spatial upsampling does not converge to a high-quality solution. This is visible near geometric details, e.g., in the background of the cathedral where geometric discontinuities become smaller and aliasing artifacts emerge. Moreover, flickering appears when the camera moves because of high-frequency details that were undersampled by the low-resolution payload. Temporally-amortized upsampling cannot assure a constant framerate because the supplementary rendering to fill disocclusion holes can have very differing cost. Our approach is similar in performance to spatial upsampling, as illustrated in Table 1 while maintaining a good image quality overall. This makes our solution a good choice for dynamic and static scenes.

6 Conclusion

Although spatio-temporal processing has been explored in different contexts, it has received less attention in terms of GPU rendering. In fact, GPUs inherently exploit spatial coherence in the SIMD structure of the massively parallel processing pipeline. However, modern GPUs benefit very little from temporal coherence. In this work, we proposed a relatively simple framework for spatio-temporal rendering as a trade-off between efficiency and quality. Compared to joint-bilateral spatial upsampling our algorithm produces higher quality and introduces only a small performance overhead and keeps memory demands small. For dynamic scenes, our algorithm is more robust than temporal reprojection caching. It combines benefits from both methods. We demonstrated on relatively complex shaders that our upsampling technique can reduce the GPU's shading costs. Our solution is well suited for increasing screen resolutions and beneficial for various algorithms, e.g., global illumination, soft shadows, or procedural textures.

7 Discussion and Future Work

Although our algorithm produces generally better and more robust results than previous real-time upsampling techniques [Yang et al. 2008] for the GPU there are situations where it may also fail. First limitation is that our reprojection caching is solely based on fast computable "geometric" flow which might differ from *optical flow*. As a worst case scenario imagine a fast moving object with a camera attached to it moving at the same speed. A shadow cast by the object on a static floor is completely decorrelated with the world-space pixel positions and temporal reprojection can not help. In such case mainly geometry-aware spatial upsampling is influencing the final pixel color as temporal gradients become large.

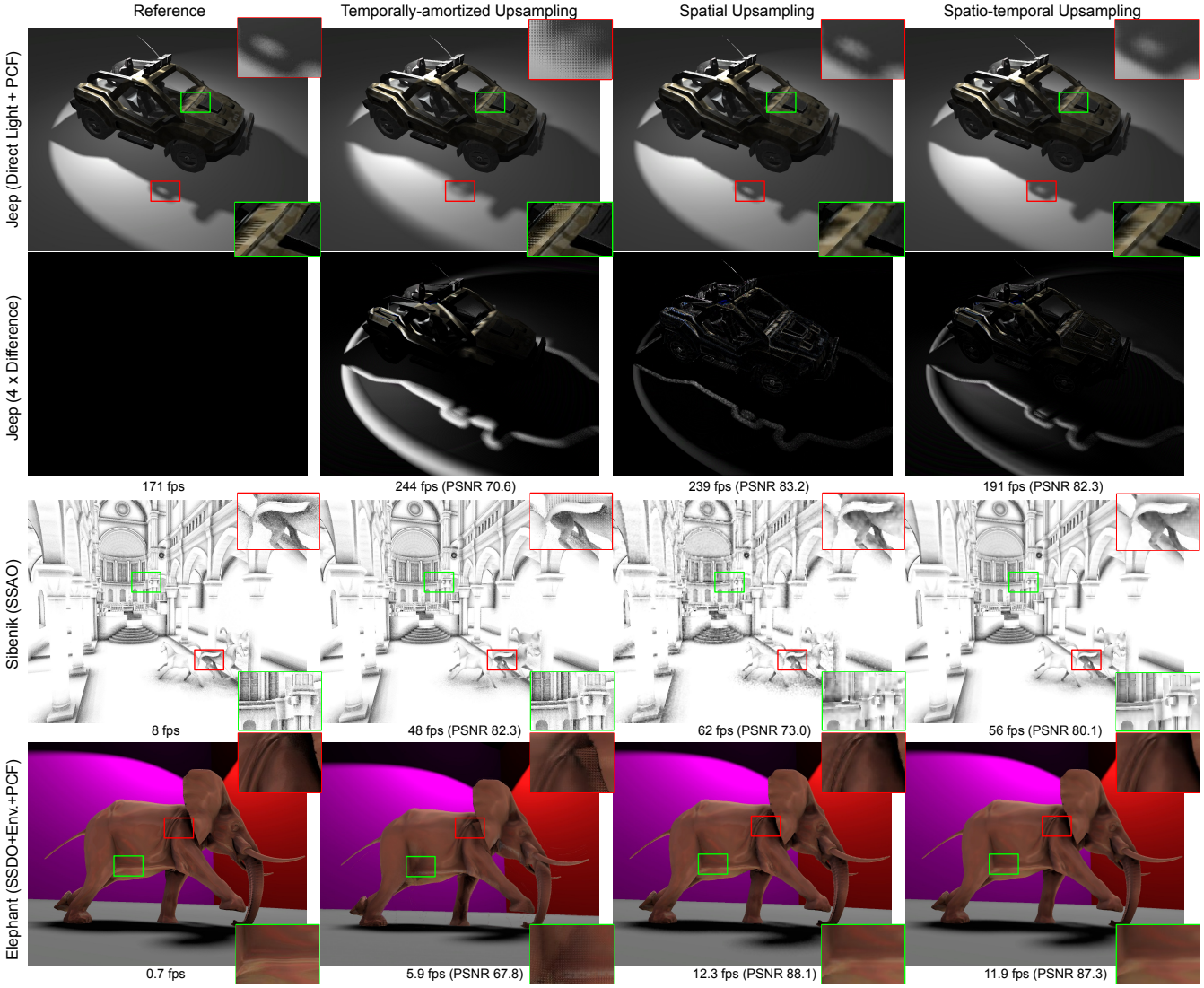

Figure 8: *Comparison of spatial, temporally-amortized, and spatio-temporal upsampling. The first two rows show screenshots and difference images (4 times scaled) of simple shading (static geometry and dynamic light). This is a difficult case for our method, but it performs better than amortized temporal upsampling. We obtain a smoother difference image than spatial upsampling. Geometric discontinuities are better handled leading to a more visually pleasing result. The third row shows results of a more expensive shader (SSAO) (dynamic scene and static camera). The background exhibits the high quality achieved by temporal convergence. The PSNR reflects the positive influence of our method. The bottom row shows a dynamic scene with high-frequency shading. (More results are shown in Figure 1.)*

Further, our algorithm relies on the assumption that temporal changes in the shading are smooth and spatially coherent and can therefore be low-pass filtered. And all quickly varying signals are only due to aliasing or noise in the shading[1]. Otherwise, we could not reliably estimate our temporal weighting coefficient w_f, trading spatial with temporally-amortized upsampling, as our gradients would flicker. Hence, fine details in the shading that are also quickly changing in time are hard to detect. Fortunately, fast temporal changes are also hard to track by the human visual system and may appear blurry due to the integration in our eye (e.g. hold-type effect of modern LCD displays).

A second minor shortcoming is the influence of apparent motion blur in the temporally-amortized upsampling, which arises from the discretization in the bilinearly-filtered pixel flow accumulated over the history of reprojected frames. Even though our motion vectors are computed at high resolution, blur still appears when for example moving the camera because small deviations from the pixel center in the reprojection accumulate over several frames (see Fig. 9). A simple remedy also proposed in [Yang et al. 2009] is to increase the resolution of our exponential history buffer to sub-pixel accuracy. On the other hand, our spatio-temporal upsampling reduces the temporal influence of old pixels and shortens the lookup-chain leveraging the spatial coherence (even when shading is static pixels always have a small influence on the spatial neighborhood).

Our work relies on temporal gradients solely to adapt the temporal filter weights. Ideally, we would have wanted to extrapolate information based on these gradients, but this did not proof to be robust enough due to accumulated errors in those gradients.

Our algorithm produces better results with increasing frame-rates because incorrect pixels are quickly refreshed by newly computed

[1] Ideally, to avoid aliasing we would like to low-pass filter the input signal to the shading instead of the shading result itself. However, this way we would have to interfere with the shader algorithm, which we want to treat as a black-box decoupled from the upsampling

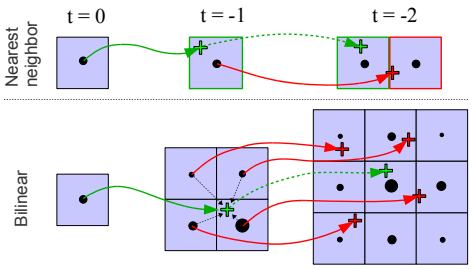

Figure 9: *Precision loss in temporal reprojection: (top) nearest neighbor reprojection results in discontinuities when the correct continuous motion flow (green arrows) and the approximate nearest-neighbor pixel-flow (red arrows) point to different pixels. (Bottom) bilinear texture filtering trades discontinuities with blur since an ever-growing neighborhood influences the final result.*

ones. A downside are the temporal patterns, that at least in screen shots, are clearly recognizable as the deterministic temporal sampling pattern. However, more sophisticated adaptive methods would easily annihilate our gained speedup. A dynamic change in sampling patterns could help to hide the problems behind noise, but we believe that more complex adaptations of the spatial-temporal weighting might be a better choice that we keep as future work.

References

ADELSON, S. J., AND HUGHES, L. F. 1995. Generating Exact Ray-Traced Animation Frames by Reprojection. *IEEE Computer Graphics & Applications 15*, 3, 43–53.

BALA, K., DORSEY, J., AND TELLER, S. 1999. Ray-Traced Interactive Scene Editing Using Ray Segment Trees. In *Proceedings of the 10th Eurographics Workshop on Rendering*.

BAVOIL, L., AND SAINZ, M. 2008. Image-space horizon-based ambient occlusion. In *SIGGRAPH 2008, Talk Program.*

BENNETT, E. P., AND MCMILLAN, L. 2005. Video enhancement using per-pixel virtual exposures. *ACM Transactions on Graphics 24*, 3, 845–852.

BISHOP, G., FUCHS, H., MCMILLAN, L., AND ZAGIER, E. J. S. 1994. Frameless rendering: double buffering considered harmful. In *Proceedings of SIGGRAPH '94*, 175–176.

CALLE, D., AND MONTANVERT, A. 1998. Super-resolution inducing of an image. *Image Processing, International Conference on 3*, 232.

DACHSBACHER, C., AND STAMMINGER, M. 2005. Reflective shadow maps. In *I3D '05: Proceedings of the symposium on Interactive 3D graphics and games*, 203–231.

DAMEZ, C., DMITRIEV, K., AND MYSZKOWSKI, K. 2003. State of the art in global illumination for interactive applications and high-quality animations. *Computer Graphics Forum 22*, 1 (Mar.), 55–77.

DAYAL, A., WOOLLEY, C., WATSON, B., AND LUEBKE, D. P. 2005. Adaptive frameless rendering. In *Rendering Techniques*, 265–275.

EISEMANN, E., AND DURAND, F. 2004. Flash photography enhancement via intrinsic relighting. In *ACM Transactions on Graphics (Proceedings of Siggraph Conference)*, ACM Press, vol. 23.

HASSELGREN, J., AND AKENINE-MÖLLER, T. 2007. PCU: the programmable culling unit. In *Proc. of SIGGRAPH '07*, ACM.

KOLTUN, V., CHRYSANTHOU, Y., AND COHEN-OR, D. 2000. Virtual occluders: An efficient intermediate pvs representation. In *11th Eurographics Workshop on Rendering*, 59–70.

KOPF, J., COHEN, M. F., LISCHINSKI, D., AND UYTTENDAELE, M. 2007. Joint bilateral upsampling. *ACM Transactions on Graphics 26*, 3 (July).

MARK, W., MCMILLAN, L., AND BISHOP, G. 1997. Post-rendering 3D warping. In *1997 Symposium on Interactive 3D Graphics*, ACM SIGGRAPH, 7–16.

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*.

PETSCHNIGG, G., SZELISKI, R., AGRAWALA, M., COHEN, M., HOPPE, H., AND TOYAMA, K. 2004. Digital photography with flash and no-flash image pairs. *ACM Transactions on Graphics (Proceedings of Siggraph Conference) 23*, 3, 664–672.

REEVES, W. T., SALESIN, D. H., AND COOK, R. L. 1987. Rendering antialiased shadows with depth maps. In *Proc. of Siggraph'87*.

RITSCHEL, T., GROSCH, T., AND SEIDEL, H.-P. 2009. Approximating Dynamic Global Illumination in Screen Space. In *Proc. of ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*.

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

SEGOVIA, B., IEHL, J. C., MITANCHEY, R., AND PROCHE, B. 2006. Bidirectional Instant Radiosity. In *Rendering Techniques*, 389–397.

SHINYA, M. 1993. Spatial Anti-aliasing for Animation Sequences with Spatio-temporal Filtering. In *Proceedings of SIGGRAPH '93*, 289–296.

SITTHI-AMORN, P., LAWRENCE, J., YANG, L., SANDER, P. V., AND NEHAB, D. 2008. An improved shading cache for modern gpus. In *Graphics Hardware*.

SITTHI-AMORN, P., LAWRENCE, J., YANG, L., SANDER, P. V., NEHAB, D., AND XI, J. 2008. Automated reprojection-based pixel shader optimization. In *SIGGRAPH Asia '08*, ACM, 1–11.

TEKALP, A. M. 1995. *Digital video Processing*. Prentice Hall.

WALD, I., KOLLIG, T., BENTHIN, C., KELLER, A., AND SLUSALLEK, P. 2002. Interactive global illumination using fast ray tracing. In *Rendering Techniques*, 15–24.

WALTER, B., DRETTAKIS, G., AND PARKER, S. 1999. Interactive Rendering using the Render Cache. In *Proceedings of the 10th Eurographics Workshop on Rendering*, 235–246.

YANG, L., SANDER, P. V., AND LAWRENCE, J. 2008. Geometry-aware framebuffer level of detail. In *Rendering Techniques*.

YANG, L., NEHAB, D., SANDER, P. V., SITTHI-AMORN, P., LAWRENCE, J., AND HOPPE, H. 2009. Amortized supersampling. *Proc. of SIGGRAPH Asia '09*.