Stable Sample Caching for Interactive Stereoscopic Ray Tracing

Henrik Philippi[1,2], Henrik Wann Jensen[2], and Jeppe Revall Frisvad[1]

[1]Technical University of Denmark
[2]KeyShot

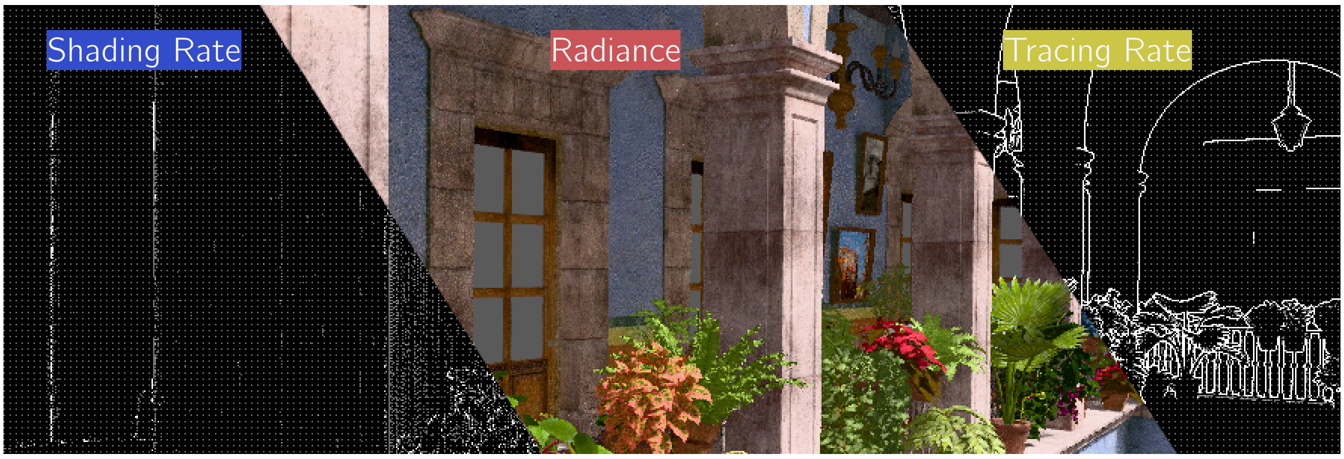


Figure 1: *Our method maintains a cache of stable samples suitable for adaptive blending of the images in a stereo pair. This leads to better temporal stability and better stereo consistency. The stable samples further enable us to reduce the performance hit by using adaptive shading and adaptive visibility tracing. Shading Rate and Tracing Rate indicate the values we use to dynamically control these adaptive techniques.*

Abstract

We present an algorithm for interactive stereoscopic ray tracing that decouples visibility from shading and enables caching of radiance results for temporally stable and stereoscopically consistent rendering. With an outset in interactive stable ray tracing, we build a screen space cache that carries surface samples from frame to frame via forward reprojection. Using a visibility heuristic, we adaptively trace the samples and achieve high performance with little temporal artefacts. Our method also serves as a shading cache, which enables temporal reuse and filtering of shading results in virtual reality (VR). We demonstrate good antialiasing and temporal coherence when filtering geometric edges. We compare our sample-based radiance caching that operates in screen space with temporal antialiasing (TAA) and a hash-based shading cache that operates in a voxel representation of world space. In addition, we show how to extend the shading cache into a radiance cache. Finally, we use the per-sample radiance values to improve stereo vision by employing stereo blending with improved estimates of the blending parameter between the two views.

CCS Concepts
• ***Computing methodologies*** $\rightarrow$ ***Rendering; Virtual reality;***

1. Introduction

With recent advances in graphics hardware, we start seeing ray-traced rendering of complex geometry and lighting effects within highly interactive real-time applications, such as video games. Meanwhile, stereoscopic applications like virtual reality (VR) rarely feature ray tracing as they require rendering a high-resolution image pair approximately every 10 milliseconds. This very limited frame time makes it a challenge to sample the image enough to avoid aliasing, flicker, and noise. In VR settings, these artifacts are very unpleasant as temporal incoherence and discrepancies between the two views can disturb motion parallax and depth perception and lead to user discomfort [LJ00, KT04].

Because stereo views have large overlaps and consecutive frames also contain similar content, reuse of rendering results between frames via filtering or caching has the potential to lower the rendering cost. Filtering generally comes at the cost of bias in the form of blur and/or ghosting while caching largely avoids these artifacts. Reuse of shading results from previous frames via caching [MNV*21] or filtering [SKW*17] can tremendously reduce computational load and avoid temporal artifacts. Similarly, temporal antialiasing (TAA) [YLS20] filters geometric edges by filtering temporally, thereby reducing flicker and aliasing at the cost of blur. Caching first-hit samples enables improved temporal stability without introducing blur but at a high cost of frame times 50% higher than native supersampling [DSK*17]. In this paper, we present a sample caching algorithm that is faster than previous work while retaining the benefit of temporal stability, and it can also serve as a shading cache. We improve efficiency by using adaptive shading and tracing, see Figure 1

2. Related Work

Our work is at the intersection between visibility caching, such as interactive stable ray tracing [DSK*17], and TAA [YNS*09, YLS20] as well as previous work on shading reuse, particularly in VR [MNV*21]. We improve stable ray tracing with a modified reprojection algorithm and data structure, and we add an adaptive tracing scheme that significantly improves performance and temporal stability. In addition, our shading cache allows reuse of shading work and filtering across stereo views.

Temporal screen space filtering. TAA [YNS*09, YLS20], recently improved with neural networks and extended with frame generation [NVI25], reuses primary visibility computations across frames. Instead of supersampling the frame to accurately resolve geometric edges and texture details, a history frame is projected from frame to frame via resampling and then temporally filtered with an exponential moving average. When radiance is demodulated to separate high-frequency components, the low-frequency indirect lighting can additionally be filtered spatiotemporally [CKS*17, SKW*17, TLP*22], allowing for heavier use of filtering without blurring geometric edges and texture details. Despite this, most approaches still use TAA as a post-process on the composited image to improve temporal stability and aliasing. This leads to the typical look of modern real-time rendering with blurry geometric edges and textures especially during movement. The loss of detail from resampling and ghosting is an inherent trade-off of temporal filtering [PFJ23]. The only technique we know of that avoids resampling blur while still reusing visibility computations is interactive stable ray tracing [DSK*17]. However, this method has a significant overhead even over native supersampling. Our objective is to achieve improved temporal stability without compromising image crispness and performance.

Shading caches. While visibility caching is constrained to screen space, shading caches [NSL*07] come with all kinds of data structures and methods. One way to cache shading results is to replace the 3D geometry with screen-aligned imposters [Sch96] which can be extended with voxel hierarchies [SS96] and ray-marched parallax corrections [MFL21]. More commonly, though, the 3D geometry is rendered as is and shading results are stored in a separate data structure. The shading atlas cache [MVD*18] stores shading results in a dynamically remapped texture space while other caches store the data in barycentric space [HSG*22]. More commonly, though, the (ir)radiance is stored in a world space cache such as a sparse voxel hierarchy [CP22, ZW23].

Using a hierarchical data structure to store radiance is constraining because one either has a fixed minimum voxel size and therefore limited precision or rendering requires constant building and updating a sparse data structure over the scene's geometry. Voxels are therefore often stored in a hashmap [DS07, PGSD13]. Storing average radiance in a voxel hash grid can simplify updates to sparse world-space caches [RWB*20, Gau21]. Storing the hash cells at dynamic LoD enables varying accuracy of the stored radiance depending on the view position. In addition, this enables fast and space-efficient reconstruction of low-frequency details such as diffuse global illumination in real-time [DDB*09, BMdD*23] and even glossy reflections to some degree [EMHB23]. The cache can also be shared between multiple views to achieve a coherent rendering for all viewing angles [WTS*23]. Some recent methods combine hash-based caching with screen space light probes, which are filtered in space and time [Wri21, BMdD*23].

Radiance can also be represented via a neural network. Recent work uses multi-layer perceptrons (MLPs) to query low-frequency secondary hit radiance and adjust training speed to scene changes, which leads to significant noise reduction [MRNK21]. Neural networks can also be used during the reconstruction step from a cache to retrieve high-quality first-hit radiance from imperfect barycentric irradiance caches [HSG*22]. Lastly, ReSTIR can also be considered a shading/radiance cache as it stores secondary ray samples at the primary hit point to reduce variance and acts as an unbiased filtering mechanism [BWP*20, OLK*21]. We consider ReSTIR to be mostly orthogonal to our work as we cache visibility rays instead of secondary rays and the provided variance reduction of ReSTIR may be combined with a shading cache.

We provide a screen space sample cache for caching both visibility and shading computations without requiring preprocessing or scene-specific optimisations. The cache is updated sparsely to reduce tracing of primary rays and shader invocations at superresolution. Our cache can be combined with existing methods for radiance caching, denoising, and filtering. In summary, we

- present an improved data structure and reprojection algorithm for sample caching,
- provide a method for adaptive primary ray tracing with the sample cache,
- extend the method with a shading cache with the ability to adaptively shade samples based on sample count or error, and
- demonstrate that the sample cache can improve a stereoscopic post-processing filter by providing a better variance estimate.

3. Method

Our method consists of four parts, visualised in Figure 2: Reprojection, Primary Ray Tracing, Shading/Secondary Ray Tracing, and Reconstruction. In the reprojection phase, all cached samples are forward reprojected into the new frame according to animations

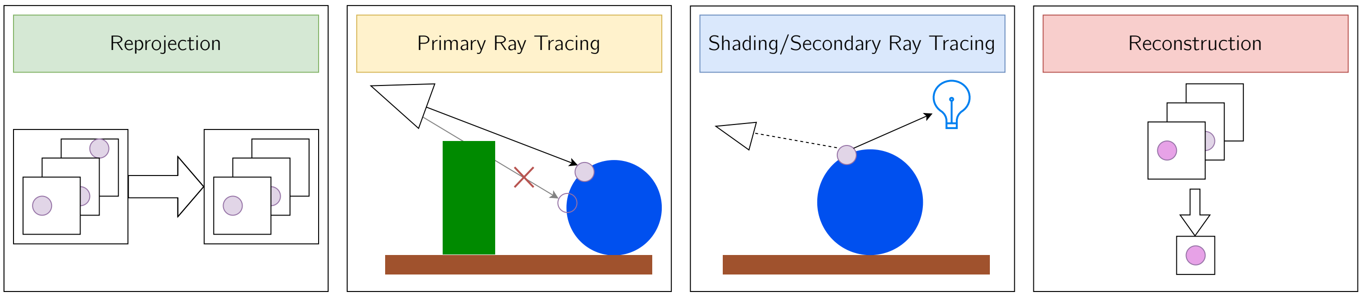


Figure 2: *Overview of our rendering algorithm. The sample cache is reprojected each frame and visibility is redetermined using primary rays. A subset of visibility samples is then reshaded and shading results of all samples within a pixel are used to produce a final colour in a reconstruction step.*

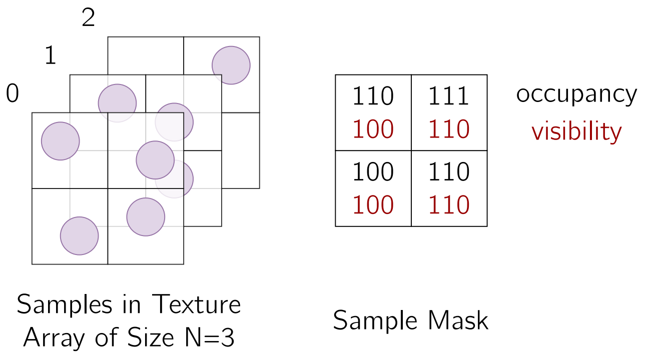


Figure 3: *Visualising the content of the radiance cache. Each pixel holds a dynamic number of samples with at most N (in this case $N = 3$) samples. The data for the samples is held in an array of textures, a bitmask holds occupancy and visibility information.*

and changes of the camera view. We use the resulting distribution of samples to identify holes in the sample cache and samples that were likely occluded by the scene changes. Where necessary, new samples are created or pruned, and a subset of persistent samples is retraced to update their visibility. Furthermore, the shading pass can use the sample information to shoot secondary rays into the scene and accumulate radiance information over time. Finally, the reconstruction pass produces a final colour for the display based on the visible samples in the cache and their radiance values and material parameters.

The core of our algorithm is a screen space data structure that holds an array of samples for each pixel. Each sample contains a surface point hit by a primary ray. The cache has N slots for each pixel, so that it can hold zero to N samples per pixel. We identify a sample within a pixel by its index within the cache $s \in \{0,\dots,N-1\}$. The cache data is held in an array of textures with each texture holding all samples of index s as illustrated in Figure 3. In addition, a sample mask carries occupancy o_s and visibility bits v_s of all slots within a pixel. The information is encoded into a texture with one integer per pixel. Since each sample slot takes two bits, o_s and v_s, a 32-bit mask can support up to 16 samples, 32 for a 64-bit mask. Each sample may also hold additional fields to carry shading information like radiance L_s from secondary rays with an associated accumulated sample count c_S. To avoid confusion, we refer to the samples stored in the cache as *visibility samples* and the accumulated secondary rays as *radiance samples*.

Listing 1: *Pseudocode to encode a first hit into the 128 bits of* `fh`.

```
input: float2 bary, uint instance, uint prim, uint mat
output: float4 fh
uint2 b = uint(round(clamp(bary,0,1) * float(0xFFFFFF)))
fh.x = asfloat((b.x & 0xFFFFFF) | ((mat & 0xFF) << 24))
fh.y = asfloat((b.y & 0xFFFFFF) | ((mat & 0xFF00) << 16))
fh.z = asfloat(prim)
fh.w = asfloat(instance)
```

Each slot contains enough information to reconstruct the local geometry and shading parameters of a primary hit sample. In our case, we pack an instance index, a primitive index, a material index, and barycentric coordinates into a float4 vector as shown in Listing 1. Using the scene's vertex buffer and material textures, we reconstruct the position, normal, and material parameters of the sample in order to perform reprojection, shading, and retracing. The reconstructed world space position is used to reproject the samples within the cache from frame to frame and retrace samples to keep visibility information up-to-date. The accumulated radiance L_s and its sample count c_S are stored in a single float4 (128-bits) value.

3.1. Forward Reprojection

At the beginning of each frame, we reproject the cache of the previous frame into a new cache for the current frame. We obtain the destination of each sample by computing the sample position $\boldsymbol{p}_s$ and projecting that into the view of the current frame. When the target pixel has been computed, a slot has to be reserved in the list of pixels in which the sample can be stored. This is done via atomic operations on the sample mask's occupancy and visibility bits. Unlike the reprojection algorithm in interactive stable ray tracing [DSK*17], where the subpixel position of a sample determines the samples's cache location, a sample in our method may be stored in any of the available slots of a pixel. Therefore, a failed reservation on a pixel's sample slot may be retried on another slot until success or no more slots are available.

When a sample s is to be reprojected into the cache of a target pixel with sample slots $i = 0 \dots N-1$, we first read back the target sample mask to determine the indices of the slots that are unoccupied. If a slot is unoccupied, the sample tries to reserve it via an

Listing 2: *Pseudocode for cache slot allocation, where* `pixel` *is the target pixel for the projection,* `i` *is the index of the sample we would like to reserve,* `s` *is the sample we are reprojecting, and* `m` *is a 32-bit integer containing the visibility and occupancy bits of a pixel. We obtain* `m` *from the texture* `mask_tex`.

```
bool try_reserve(pixel, i, s, inout m):
    int o_bit = 1 << i
    int v_bit = 1 << (i + N)
    int s_val = o_bit | (is_visible(s) ? v_bit : 0)
    m = InterlockedOr(mask_tex[pixel], s_val)

    // check previous mask value to confirm reservation
    bool success = (m & o_bit) == 0
    if (is_visible(s) && !success)
        success |= (m & v_bit) == 0

    m |= s_val
    return success

void write_sample(s, pixel):
    int m = mask_tex[pixel]
    while (slots_available(s, m)):
        i = choose_target_slot(s, m)
        if (try_reserve(pixel, i, s, m)):
            write_sample_data(s, pixel, i)
            break
```

atomic OR operation on the sample mask. The visibility bit, which is encoded into the same integer is updated at the same time. If unsuccessful due to a lost race with another thread, the operation is retried as long as unoccupied slots remain. If all slots are occupied, we overwrite invisible samples by racing to set the visibility bit to 1. Listing 2 provides pseudo-code.

This approach contains two possible race conditions. One may happen between reserving a slot and writing it and the other one while writing the sample data. The first one happens when a thread projecting a visible sample overtakes a thread projecting an invisible sample into the same pixel and slot. The thread with the invisible sample will read success and overwrite the sample data of the visible sample. The second race condition can happen if the sample data requires multiple write operations for all data. Common GPUs typically support atomic writes for 128-bit writes, but our cache can exceed that if additional radiance data is stored within the samples. In both cases, the sample data might become inconsistent, meaning that either the radiance information or the visibility bit does not belong to the written sample information.

Both of these can be fixed by splitting the reprojection into two steps. Instead of performing the sample write directly, the sample address is written to a target texture. A second step then simply copies all samples including their visibility and occupancy bits. In practice, we found that this step is not required as we did not observe any problems as a result of these race conditions. A similar observation was made before [DSK*17].

3.2. Primary Ray Tracing

The reprojected sample cache needs an update to reflect changes in the scene and the view. Some pixels may not contain any samples while others may contain samples that were visible last frame but occluded in the current frame. To make sure all holes are filled and visibility is updated, we device a three-step procedure:

1. (only for adaptive sampling) analyse samples and mark a subset to be retraced,
2. fill holes by creating new samples, remove samples from regions with too many samples,
3. trace a subset of samples to update their visibility.

We test three different strategies for choosing the samples to be retraced: single-, multi-, and adaptive sampling. In single-sampling, only a single sample for each pixel is chosen (round robin) for retracing. This may lead to artefacts from lagging visibility, so for reference, the multi-sampling approach retraces all samples every frame. Adaptive sampling analyses the samples before deciding which subset is to be retraced. The adaptive sampling traces fewer rays than multi-sampling. More details on this in Section 3.4.

In the second step, the density of the samples is controlled to be roughly n_{target} with an allowed tolerance δ. We measure the average density of visibility samples within a 3×3 neighborhood around each pixel. If the density is out of bounds, we then either add or purge visibility samples (randomly) while ensuring that at least one sample remains per pixel. To delete a sample, we simply mark it as unoccupied and invisible in the sample mask. New samples are either created in an unoccupied slot or, when none are available, an invisible sample is overwritten.

Following target selection and density adjustment, the chosen subset of samples in each pixel is retraced with rays from the eye point to the stored hit positions. This is in principle like shooting shadow rays, which can be done with minimal loss of precision by choosing the ray direction directly as the vector between the camera and the hit position [vA23]. The visibility bit is set to 1 if the closest hit is within ϵ_0 (we use 0.001) of the stored sample and set to 0 if a closer hit is encountered when tracing the ray. New samples are traced by generating a random direction within the pixel they reside in and storing the first hit information within the cache upon a hit. To ensure that at least one visible sample remains for each pixel, the last visible sample is overwritten if it fails the verification test.

3.3. Shading and Reconstruction

Expensive shading operations such as tracing of secondary rays for global illumination can be cached in the radiance value of the sample cache. Each frame, a subset of samples are then chosen for reshading, by default we select one sample for each pixel in a round-robin fashion. Shading samples within the cache can happen completely independently of the primary rays shot in the frame because the sample cache contains all information necessary to reconstruct the local geometry and shading parameters by consulting the vertex buffer and scene textures. In our case, we shoot secondary rays into the scene to compute ray traced global illumination for one of the samples in each pixel.

To compute the final colour of each pixel, we average the observed radiance stored in all visibility samples with $v_s = 1$ within each pixel. The radiance can be stored demodulated. In this case, the pass also recombines albedo and demodulated radiance to compute the final colour. Additionally, the environment is evaluated where the stored samples represent the rendered background.

3.4. Adaptive Ray Tracing

The visibility needs to be up-to-date for all samples to fully avoid image artifacts. In practice, this means retracing all samples for every frame, which is prohibitively expensive. However, we can approximate the ideal behaviour with a sparse tracing pattern and an adaptive heuristic based on analysing the samples within each pixel. We trace a single sample within a 4×4 block of pixels and trace all samples where the geometry of a pixel is inconsistent with a flat smooth surface. To this end, we compute the distance between the camera and the surface for each sample. Based on the closest and furthest distances, $t_{\min}$ and $t_{\max}$, of the visible samples within a pixel, we can identify image regions that likely contain occluded samples and shoot additional primary rays.

A high difference $t_{\max} - t_{\min}$ in the sample distances helps us identify edges, where most visibility changes usually occur. However, simply thresholding the difference would lead to many false positives since surfaces hit at oblique angles also exhibit high differences in the same pixel. To distinguish oblique angled surfaces from geometric edges, we compute a slope $\Delta_{\vec{n}}$ based on the surface normal $\vec{n}$ and the distance t_s of sample s from the camera (see App. A):

$$\Delta_{\vec{n}} = t_s \left| 1 - \frac{\cos\theta}{\cos(\theta - \phi)} \right| , \tag{1}$$

where θ is the angle between the direction toward the camera $\vec{\omega}_o$ and the surface normal ($\cos\theta = \vec{n} \cdot \vec{\omega}_o$) and ϕ is the angle subtended by the diagonal of a single pixel. This slope serves as an expected distance within a pixel frustum. Using this, we can threshold the difference using

$$t_{\max} < (1+\epsilon_1)(t_{\min} + 2\Delta_{\vec{n}}) \tag{2}$$

with some suitable $\epsilon_1 > 0$ to account for numerical precision. We use $\epsilon_1 = 0.005$.

Because the sample cache may have holes and scene changes may change visibility outside of geometric edges, we analyse a 3×3 neighbourhood of each pixel so that visibility changes propagate through the image within a few frames. To avoid reading and reconstructing surface points for all samples within the 3×3 block, we adjust the reprojection step to write out minimum and maximum depth textures. Whenever a sample is successfully reprojected into a pixel, that pixel's min and max depth values are updated via atomic operations using bit reinterpretation of the depth as integers. The normal slope $\Delta_{\vec{n}}$ is carried in the lower 32-bit within a 64-bit integer together with the closest depth in the higher 32-bit. The primary tracing pass can then cheaply access the closest and furthest distances by reading these textures. Figure 4 visualises the computation of the trace mask based on the inequality (2), determining the samples to be traced.

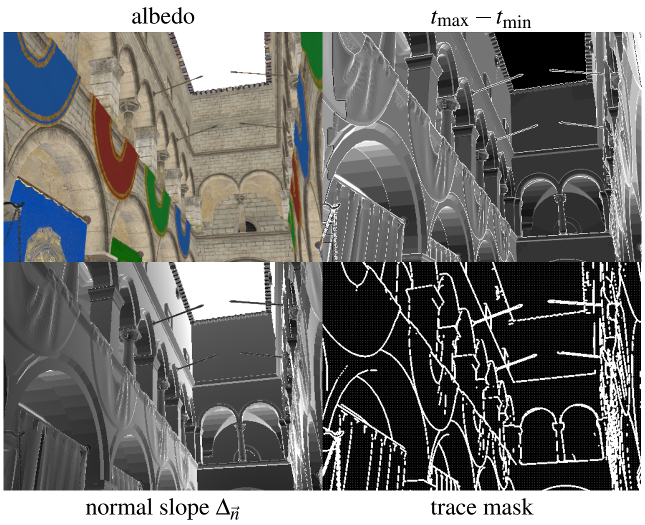


Figure 4: *Analysing the distance between samples within a pixel and its neighbourhood can detect edges in the image, where more primary rays should be used to ensure up-to-date visibility values in the sample cache. To avoid false positives, the difference between maximum and minimum distances is adjusted by the slope of the surface before being used in the decision to trace all samples within a pixel or not. A sparse subset of the samples is always traced as can be seen by the faint grid in the trace mask.*

3.5. Adaptive Shading

Our stable set of samples enables us to recompute the shading results for the samples adaptively. Since the sample information is self-contained, a sample may even be updated when it is not verified in the same frame. This is especially interesting when the sample cache carries Monte Carlo estimates. We set up a simple adaptive shading scheme that focuses computation on disoccluded regions and aims to equalise the accumulated radiance samples across the entire frame. As in the case of adaptive primary ray tracing, we shade one sample in every 4×4 block of pixels as a base shading rate and dispatch the rest dynamically.

For the dynamic dispatch decision, each sample is assigned an importance that indicates how often it should be updated. This importance value may be a rate of change in the shading result as is done in [MNV*21]. To use the shading cache as a form of radiance cache, one may take the number of accumulated radiance samples c_s to set the importance to $w_{\text{shade}} = (1+c_s)^{-1}$ which gives higher priority to samples with lower sample counts. During reprojection, we compute a histogram from these importance values by counting the samples of each pixel into $M (M = 13)$ logarithmically sized buckets via atomic instructions. This histogram is illustrated in Figure 5. The minimum and maximum importance for the full frame, w_{min} and w_{max}, are also computed via atomics and used in the next frame to compute the bounds of the M bins.

The dispatch decision is done in a separate kernel that writes out sample indices for the shading pass to compact the work. The shading rate is user-defined as a percentage where 100% means that the number of shading invocations is equal to the number of pixels. With the shading rate, we can compute an importance threshold using the prefix sum of the histogram bins. Buckets that fall below are not shaded, buckets above are dispatched for shading. For samples in the bucket that cross the shading rate threshold, samples are dispatched randomly based on a weighted coin toss for each sample so that, in expectation, the correct amount of samples is dispatched.

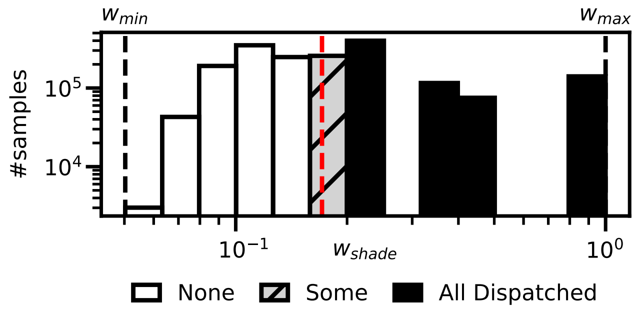


Figure 5: *Adaptive shading works by dispatching the samples with highest importance. A histogram is used to make this decision per sample. A splitting line is defined by the shading rate. Samples that fall in bins above are all dispatched.*

3.6. Stereo Blending

Rendering in stereo warrants special care with respect to how the images are perceived by the user. Monte Carlo noise and Aliasing can induce a stereo discrepancy error that is generally considered uncomfortable for the user [KT04]. We use the sample cache to reduce this stereo discrepancy. Since we treat both eyes separately, each having its own cache, we use a post-process to combine the views after reconstruction via a stereoscopic filter.

The optimal blending factor might depend on material and geometry being rendered in each pixel. Suppose X is a sampled pixel value in one image that we project to a pixel in another image with a sampled value Y. If we want to use linear interpolation to improve the pixel value in Y through replacement with

$$\mathrm{lerp}(X,Y,\beta) = (1-\beta)X + \beta Y\,, \tag{3}$$

the theoretically optimal blending parameter β in terms of mean squared error is [PFJ23]

$$\beta' = 1 - \frac{\sigma_Y^2}{\mathrm{Bias}^2 + \sigma_X^2 + \sigma_Y^2}\,, \tag{4}$$

where σ_X^2 and σ_Y^2 are variances of the initial sampled pixel values and $\mathrm{Bias}^2 = (\mathbb{E}[X] - \mathbb{E}[Y])^2$ with $\mathbb{E}$ denoting the expected value. To find a good stereo blending parameter, we need good estimates of the variances and expected values in this equation. This is where we utilise the sample cache.

For a pixel with $n \in [0, N-1]$ visibility samples $s_0, \dots, s_{n-1}$, each containing a radiance estimate $L_0, \dots, L_{n-1}$, we derive an estimate of the optimal blending parameter using the first and second raw moments of the underlying distribution. These moments are $\mu_1(X) = \mathbb{E}[X]$ and $\mu_2(X) = \mathbb{E}[X^2]$, and we estimate them from our shading cache using

$$\mu_1(X) \approx \frac{1}{c_X}\left(\sum_{s=0}^{n_X-1} c_s L_s\right), \quad \mu_2(X) \approx \frac{1}{c_X}\left(\sum_{s=0}^{n_X-1} c_s L_s^2\right), \tag{5}$$

where c_s is the effective number of samples [YNS*09] of a visibility sample in the pixel with sampled value X and

$$c_X = \sum_{s=0}^{n_X-1} c_s \tag{6}$$

is the total effective sample count for this pixel. The variance of a random variable X is

$$\sigma_X^2 = \mathbb{E}[X^2] - (\mathbb{E}[X])^2 = \mu_2(X) - (\mu_1(X))^2\,, \tag{7}$$

which we use to rewrite Eq. 4 in terms of the first and second raw moments. We then estimate β' using

$$\beta' = 1 - \frac{\mu_2(Y) - (\mu_1(Y))^2}{\mu_2(X) + \mu_2(Y) - 2\mu_1(X)\mu_1(Y)}\,. \tag{8}$$

As compared with the original technique for finding per pixel stereo blending parameters [PFJ23], we utilise the multiple samples per pixel of our sample cache which avoids the blur resulting from estimates based on neighborhood sampling.

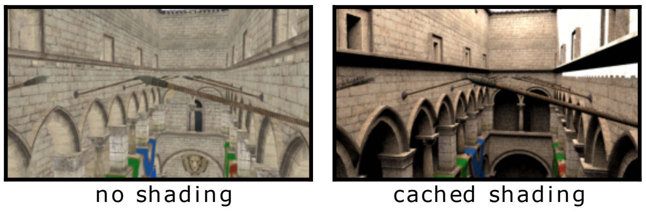
In practice, we store μ_2 for all pixels within the reconstruction pass and then implement stereo blending with a texture fetch. We also reproject using the closest visibility sample's position which gives a more stable depth value compared to taking either the targeted sample's position or a jittered first hit common in other techniques. The final colour is then given by inserting Eq. 8 in Eq. 3.

4. Implementation

We implemented our method in Vulkan (SDK Version 1.3.296) using hardware ray tracing [KHBW20] and refer to it as `SampleCache`. Unless mentioned otherwise, we used $N = 12$ layers, a target density of $n_{\text{target}} = 4$, and a sample tolerance of $\delta = 1$. Each layer of the cache requires 4 bytes per pixel for the mask and 16 bytes for each sample with an additional 16 bytes for each sample in the shading cache. That's approximately 31Mb for each layer at a resolution of 1080p.

We compare our results to previous works in visibility and shading caches. We compare our sample cache with and without shading to TAA [YLS20] and DLSS [NVI25]. As for caching shading results, we omit comparisons to straight shading caches and instead use our shading cache as a radiance cache in which secondary ray results for diffuse global illumination are accumulated. The sample cache can then be compared to existing hash-based radiance caching with TAA for Antialiasing. We implement a state-of-the-art hashing-based radiance cache, `HashCache`, and compare with this in terms of quality and performance. We choose the Level-of-Detail (LoD) of the voxels such that they are smaller than a pixel. This ensures a fair comparison with our sample cache. We also combine the two caches into a `Hybrid` in which our sample cache without shading storage is used for adaptive first hit tracing and to provide stable lookups into the hashing-based radiance cache. The radiance caches for both our sample cache and the hash-based cache reconstruct radiance as is with only non-resampled temporal filtering applied to make sure sources of blur can be accurately discriminated and assessed. In a full rendering pipeline, the output indirect lighting may be denoised using a spatiotemporal denoiser.

Our implementation of `HashCache` roughly follows the existing literature on hash-based radiance caching [RWB*20, BMdD*23, WTS*23, MKK24]. A hash-based radiance cache relies on double hashing of voxel coordinates where the first hash is used to index the cache and the second is used to resolve cache collisions. Entries are found by linear probing (open addressing), and each entry carries the information of the last access to the entry. When a value has



Method	Reserve (ms)	Copy (ms)	Total (ms)
single-pass-no-shade	1.48 ± 0.02	/	1.48 ± 0.02
single-pass	2.10 ± 0.03	/	2.10 ± 0.03
two-pass-no-shade	1.12 ± 0.03	1.68 ± 0.03	2.79 ± 0.06
two-pass	1.01 ± 0.03	2.49 ± 0.08	3.50 ± 0.11
simple reprojection	1.00 ± 0.01	/	1.00 ± 0.01

Figure 6: *Comparison of 1-pass and 2-pass reprojections at 1080p on an RTX 3060. Without shading, the cache carries only 128 bits per sample. This grows to 256 bits when caching shading results for each sample. While 2-pass is technically more correct, artefacts are rare and performance is much better with 1-pass.*

not been accessed for a while, it is free to be overwritten by subsequent insertions into the cache. The cache is stored in two textures, one containing a 64-bit integer with a last access value and the collision hash, and a second texture containing the radiance information including sample count for each voxel in a 4-component 32-bit float vector. This results in a total of 24 bytes per voxel entry. For more coalesced memory accesses, the voxels are stored in spatially coherent 2D tiles which represent a volume projected onto the major axis of the hit normal, an optimisation introduced by [BMdD*23].

For HashCache and Hybrid, the rendering and access are similar to the algorithm for SampleCache. We trace jittered first hits at 1 spp and each primary hit position is used to perform a cache lookup. Upon reserving the cache cell, a secondary ray is traced and the radiance information is updated. The updated radiance information is used to compute the final colour. This is done in only two passes, one for primary tracing and one for cache lookup, update, and final colour computation. Just like in SampleCache, we decouple albedo from lighting for the HashCache. To smooth out voxel artefacts and keep nearby cache cells alive, we apply voxel jittering [BFK18] before lookup and apply TAA [YLS20] to the final image using temporal weights as derived by Philippi et al. [PFJ23]. The TAA removes the jitter artefacts resulting from first hit and voxel jitter. For Hybrid, we compute the final colour from the hit voxel and its parent one level-of-detail above. The update randomly updates one of the two levels. To keep our implementation within a real-time frame budget, we only cache first hits and apply no spatial denoising, making the methods comparable to our SampleCache.

5. Results

We first test the different algorithmic options in our method and then move on to comparisons with other methods as well as documenting the improved temporal stability and stereo consistency.

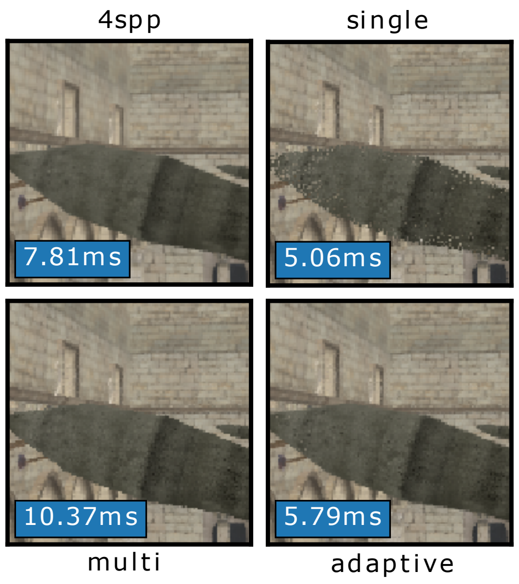
Reprojection performance. In Section 3.1, we discussed single-pass (1-pass) and dual-pass (2-pass) versions of the forward reprojection, where 2-pass corrects the race conditions of the reprojection in a deferred copy pass. Figure 6 lists performance numbers for both algorithms with and without a shading payload. The latter means that the cache has an additional 128 bits of radiance data in each cache cell for a total of 256 bits per sample cache entry. The 2-pass reprojection has almost twice the overhead of 1-pass. We observed no artefacts with 1-pass (neither on other graphics hardware, see App. B), so we use this version in all the following results.

Adaptive sampling. Figure 7 demonstrates the difference between the three retracing strategies described in Sections 3.2 and 3.4 (single, multi, and adaptive). The sample cache reprojection and the reconstruction have significant overhead as compared with simple ray tracing. While single-sampling improves temporal stability and filters textures with good performance, geometric edges exhibit temporal lag (ghosting) during movement. Multi-sampling, tracing all samples, completely removes such artefacts at the cost of lower performance, a strategy used in prior work [DSK*17]. On the other hand, our adaptive strategy has a performance that is close to single-sampling despite the overhead of the shading cache (larger payload). It also exhibits very few artefacts except for edge cases where extreme visibility changes happen from one frame to the next. Even then, the heuristic's neighbourhood scan makes sure these changes propagate within few frames. In the following, results involving the SampleCache use adaptive sampling unless mentioned otherwise.

Temporal stability without excessive blur. To evaluate the antialiasing and temporal stability of our sample cache, we compare our method with 4 spp supersampling (as a baseline) and 4 spp plus DLAA (DLSS version 3.5.10 at native resolution without upscaling) as a temporal antialiasing method. This is done in Figure 8 and a supplementary video using a hairball lit by a single directional light for a non-stochastic shader evaluation. Our method uses adaptive sampling with $n_{target} = 4$ and has the shading cache enabled. The reference is rendered at 16 spp. Our method shows slightly worse antialiasing than 4 spp rendering but less flicker. The temporal integration of DLAA makes its result the most stable at the cost of excessive blurring and ghosting, which is typical for temporal antialiasing methods. The performance impact of DLAA is slightly higher than that of our method despite the scene being a challenging edge case for our adaptive sampling scheme.

Temporally stable radiance lookups. A still view is the optimal situation for both the HashCache and our SampleCache. Figure 9 and a supplementary video shows the antialiasing and temporal stability of these methods with and without TAA applied. As seen, our method is significantly less aliased and temporally stable without relying on TAA and thereby introducing blur. When using the HashCache method, many voxels can fall into the same pixel at oblique viewing angles, making lookups temporally unstable as a different voxel is queried for each frame. Applying an LoD adjustment based on the normal incidence avoids this problem at the cost of increased blur along the surface, mimicking the look of trilinear texture filtering at high angles of incidence.

Comparison with interactive stable ray tracing. Figure 10 compares quality and performance of our method with a version of



Method	Reprojection (ms)	Tracing (ms)	Total (ms)
single	1.43 ± 0.01	1.95 ± 0.03	5.06 ± 0.06
multi	1.49 ± 0.02	7.15 ± 0.16	10.37 ± 0.26
adaptive	2.47 ± 0.04	1.46 ± 0.03	5.79 ± 0.10

Figure 7: *Comparing the different tracing modes of our* `SampleCache` *against simple supersampling at 1080p on an RTX 3060, see also supplementary video. Tracing a single sample each frame exhibits artifacts during movements, while multi-sampling increases the tracing cost significantly. The total frame time also includes reconstruction time, shading is skipped in this test.*

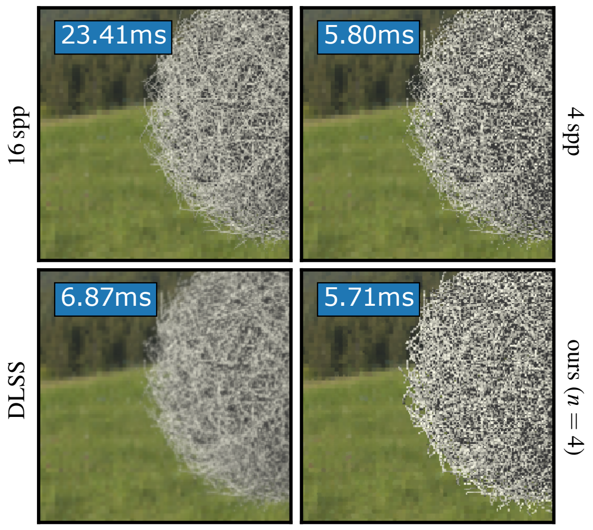


Figure 8: *Antialiasing comparison between native rendering, DLAA (DLSS without upscaling) and our method at 1080p on an RTX 3060. Our method reaches an image quality close to 4 spp with much better temporal stability and without the blur of temporal methods like DLSS.*

`SampleCache` following the layout and reprojection logic of interactive stable ray tracing (`isrt`) [DSK*17]. These results demonstrate that our method improves the reprojection algorithm and the sample cache data structure. For better comparison, we omit adaptive sampling here and only trace one sample per pixel which leads to edge artifacts during movements. Our method shows better performance, quality, and temporal stability when carrying radiance information in each visibility sample. Due to more visibility samples staying alive after each reprojection, our sample cache can accumulate a longer history for each radiance in a sample which improves temporal stability and quality. The superresolution cache of `isrt` is disadvantageous for memory accesses since samples are scattered sparsely within the grid of each pixel, whereas our data structure keeps the samples in the first layers of each pixel, leading to coalesced memory accesses into the sample cache during reprojection and reconstruction.

5.1. Moving Camera and Performance

In supplementary videos, we provide video results with a moving camera for each method using the three scenes in Figure 11. The scenes were rendered with three-bounce global illumination and Russian roulette starting from the second bounce, a common choice in real-time rendering. The three scenes are `sponza`, `bistro`, and `san-miguel`. A Preetham sun and sky model illuminates the `bistro` and `san-miguel` scenes, while `sponza` is lit by a constant background and a directional light source. All three show significant camera movement with `san-miguel` playing back captured position data recorded from a VR headset during use. Detailed frame times and RSE are plotted in Figure 12. We apply TAA on top of all methods to level the playing field for the error comparison. However, our method is decently temporally stable without TAA. These results demonstrate that we achieve the same error levels as state-of-the-art hash-based radiance caching (`HashCache`). The additional performance hit of our method is a trade-off to improve temporal stability without excessive blurring and to enable better stereo consistency (the latter is evaluated in Section 5.2).

Both `HashCache`, `SampleCache`, and `Hybrid` perform well in terms of measured error. However, RSE does not capture temporal stability problems. For this reason, we provide a closeup of the animation in the `bistro` scene in a supplementary video showing how temporal flickering is reduced with our method. All caching methods have trouble during fast movements where many cache misses occur and rendering has to reconverge. The `san-miguel` scene shows this well as the frame sequence plays back a captured VR session with a sudden head turn around frame 130 that invalidates most of the cache and therefore increases error toward the level of a naïve 1 spp rendering.

In all three methods, performance is mostly unaffected by the number of cache evictions. This matches our objective of a method suitable for preview applications in VR. Figure 13 has a breakdown of frame times. Overall, performance was consistently lower for `SampleCache` and `Hybrid` due to reprojection overhead. Cache lookups and updates run with similar execution times for both `SampleCache` and `HashCache`. `Hybrid` requires a total of three lookups per pixel (one for shading, two for reconstruction) into both its sample cache and its hash map, adding additional over-

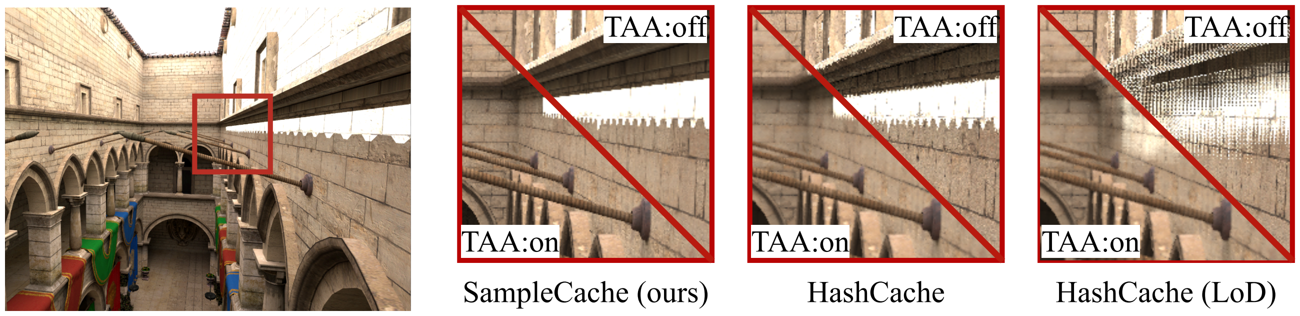
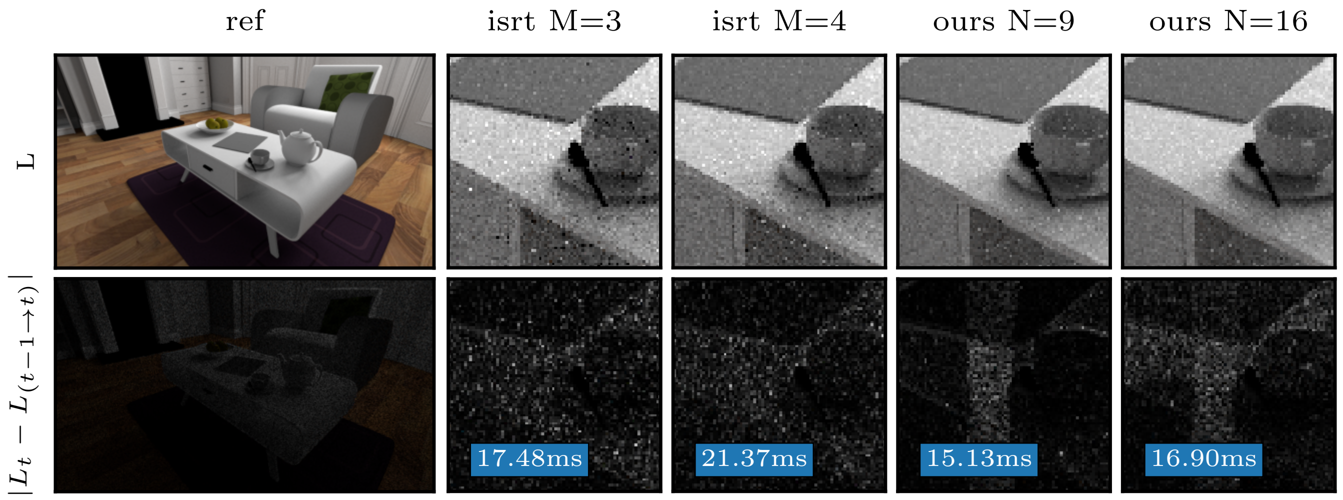


Figure 9: *From left to right:* SampleCache, HashCache *using only distance to choose a Level-of-Detail (LoD) and next to it the same* HashCache *method using distance and normal to determine LoD. Each method is shown with and without TAA. Our method shows better antialiasing and less blur for illumination edges like hard shadows. Temporal stability of the* HashCashe *can be significantly improved by choosing the LoD based on the angle of indicence at the first hit surface albeit at the cost of blurring the shadow edges further. The same scene is attached as a video in the supplementary materials.*



	isrt ($M = 3$)	isrt ($M = 4$)	ours ($N = 9$)	ours ($N = 16$)
Reprojection Time (ms)	4.66 ± 0.26	6.69 ± 1.52	3.15 ± 0.41	3.33 ± 0.56
Resolve Time (ms)	1.14 ± 0.01	1.67 ± 0.12	1.16 ± 0.02	1.20 ± 0.03

Figure 10: *Comparison of quality and temporal stability between* isrt *and* ours *at 1080p on an RTX 3060 without adaptive sampling. The columns show results with superresolution factor M for* isrt *(M=3 means $3 \times 3 = 9$ visibility samples per pixel) and number of layers N for* ours. *The target density is $n_{\text{target}} = 4$ and $n_{\text{target}} = 6$ for $N = 9$ and $N = 16$, respectively. The top row shows a closeup of rendered radiance and the bottom row shows a frame-to-frame reprojected difference, which serves as a proxy for temporal stability for the shading cache. Our method is faster and more stable when holding onto samples, which is seen in the lower temporal differences. The total frame time includes around 1 ms for first-hit tracing and 10 ms (12 ms for* isrt*) for material evaluation and secondary hit tracing.*

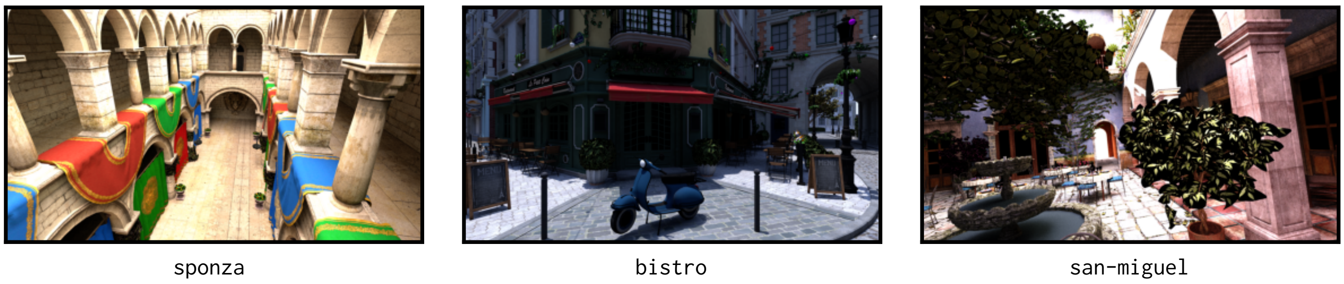


Figure 11: *Three test scenes. Rendered here using our screen space shading cache. Despite continuous movement, radiance caching methods can significantly reduce noise and quickly converge to a high-quality result while providing real-time feedback by showing noisy, but temporally coherent frames during convergence.*

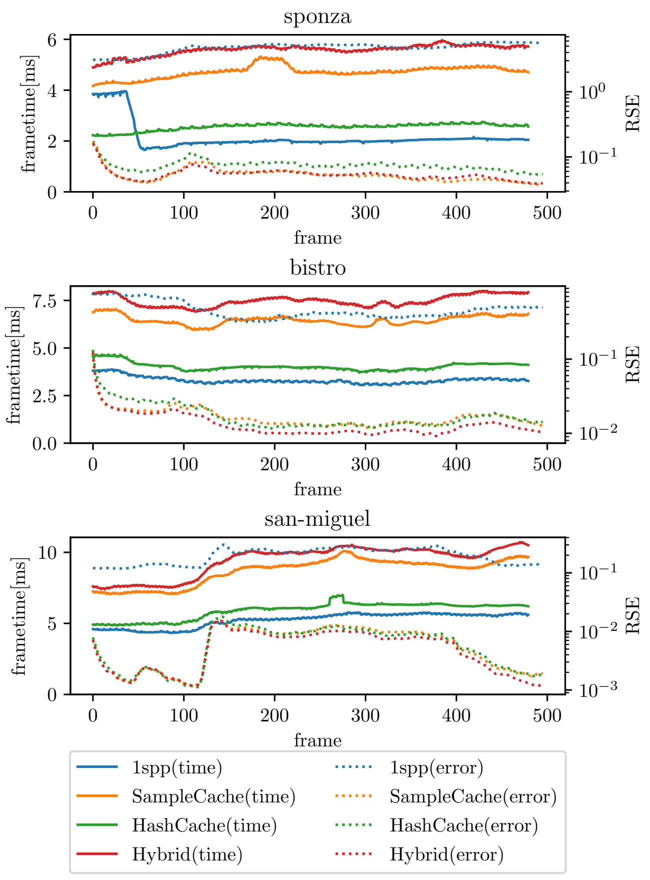


Figure 12: *Mean relative squared error (RSE) and frame times for sequences of 500 frames on the three scenes at 1080p on an RTX 4090. A sharp rise in error indicate a large camera movement that invalidated large parts of the cache. Performance is mostly unaffected and dependent on ray tracing time. Lines are smoothed via a moving average for visualisation purposes.*

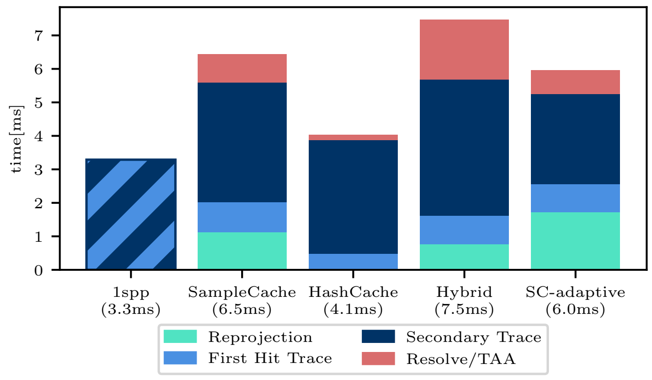


Figure 13: *Execution time breakdown for the* `bistro` *scene with* `1spp`, `SampleCache`, `HashCache`, `Hybrid`, *and* `SC-adaptive` *at 1080p on an RTX 4090. The latter is* `SampleCache` *with adaptive shading enabled. Secondary hit tracing includes the cache lookup and update and takes most time in all the methods. In* `1spp`, *first and secondary hit tracing are integrated into a single stage. The red bar sums the reconstruction (here: "resolve") and TAA passes.*

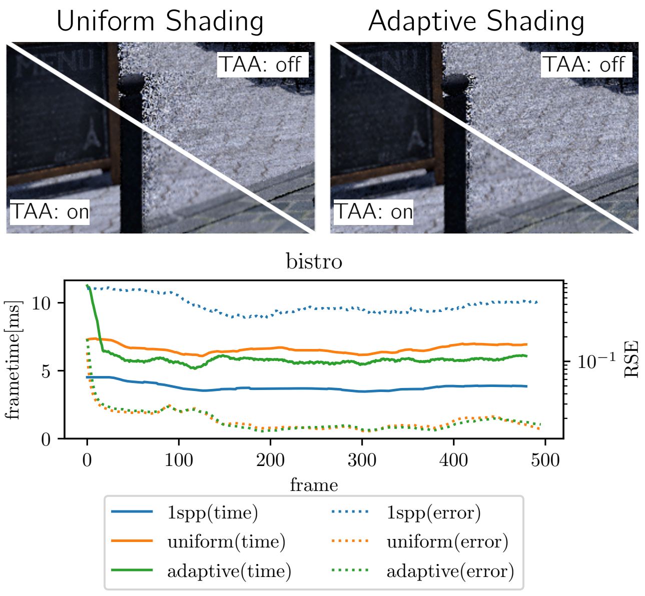


Figure 14: *Adaptive shading results for the Bistro scene at 1080p on an RTX 4090. Adaptive shading focuses secondary rays where radiance sample count is low thereby reducing noisy trails. Despite adaptive shading using only 70% of the shading rate used for uniform shading, the error is equally low.*

head in shading and resolve. `HashCache` has a lower shading load because nearby pixels may hit the same voxel in which case only one is traced. The overhead of the caching methods on top of 1 spp tracing is significant but fairly modest when considering the achieved error reduction. Figure 13 also includes `SC-adaptive`, which is our SampleCache with adaptive shading enabled. Figure 14 demonstrates that adaptive shading improves the performance of `SampleCache` without compromising the overall error.

5.2. Stereo Discrepancy Error

To assess how well stereo signals are improved by our stereo blending algorithm (outlined in Section 3.6), we use the `bistro` scene with view-dependent glossy materials switched on. To let the highlights change with view changes, we added a temporal filter in the sample update procedure using a fixed blending parameter $\alpha = 0.1$. The resulting image is used as input for stereo blending. We compare with related work by Philippi et al. [PFJ23], where the stereo blending parameter β is estimated using a 3×3 kernels (while we use Eq. 4). Because kernel estimation tends to blur the result, which may or may not be desirable, we introduce a sharpness parameter $R \in [0,1]$ in our technique that controls how much the 1st and 2nd raw moments are blurred before being used in an estimate. Specifically, we blur the estimates within a 3×3 neighbourhood where the weight of each pixel is $w = 1 - R$ except for the center pixel for which the weight is always $w = 1$. The raw moments from the

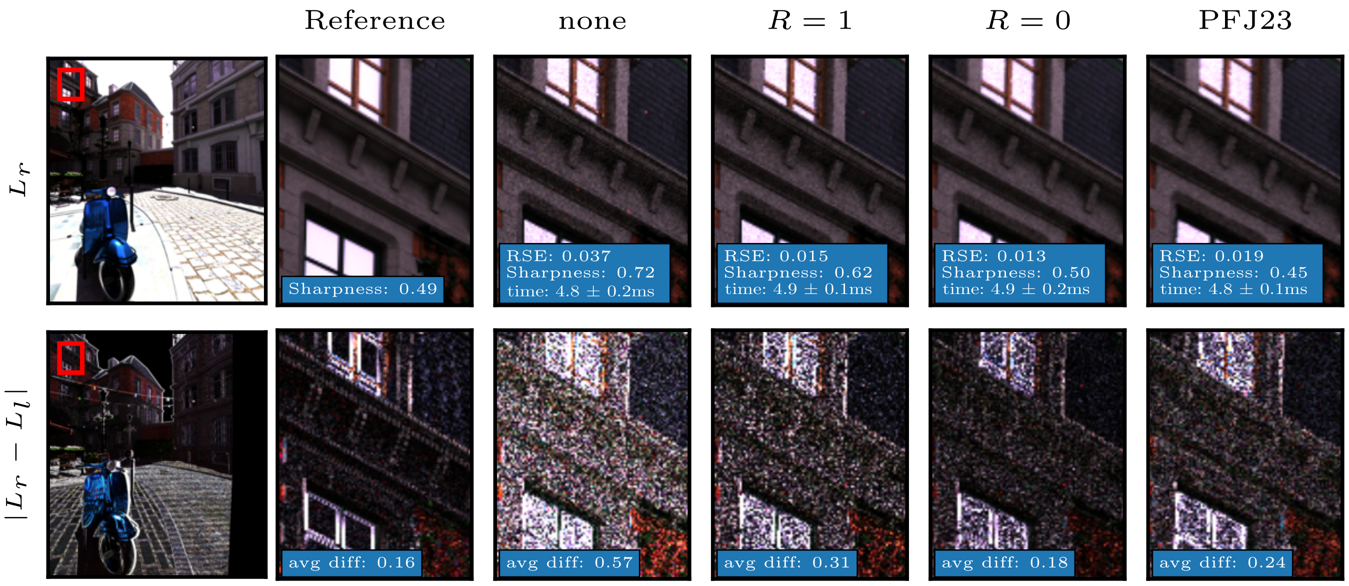


Figure 15: *Comparing stereo blending techniques. From left to right: reference,* `SampleCache` *without any stereo blending, with our stereo blending and $R = 1$, then $R = 0$, and finally stereo blending using a neighbourhood estimated blending parameter as in related work [PFJ23]. Using variance estimated from samples, we can tune the sharpness parameter R and get a desired trade-off between blur and error. The bottom row shows the stereo difference of the images, in which the left eye was reprojected onto the right view to show stereo discrepancy error. At lowest R, our method gives lower RSE, sharpness (CPBD) similar to the reference, and lower stereo discrepancy error than previous work with no significant performance overhead. The sharpness metric was used in a similar way in work on stable ray tracing [DSK*17]. Frame times were measured using an RTX 4090 at $2\times960\times1080$ resolution.*

projected eye are smoothed using Dodgson's quadratic resampling kernel [Dod97] with R as the spline parameter.

Figure 15 shows the quality and performance of our method in comparison with related work. We provide results for $R = 1$ as well as $R = 0$. We note that while $R = 1$ does not perform as well in terms of error and stereo discrepancy, it provides a conservative option for cases where any kind of blur is undesirable. It is slightly cheaper too, as it needs fewer texture fetches. Best quality is achieved with an $R = 0$, which improves RSE, sharpness, and stereo discrepancy as compared with the related work. We measure sharpness using cumulative probability of blur detection (CPBD) [NK11]. Ideally, we would like the same level of sharpness in our images as in the reference image, which is achieved very nicely with $R = 0$.

To better visualise the improvement in stereo discrepancy, we provide a supplementary video that switches between the two views, comparing our sample-based stereo blending to previous work. We use $R = 0.8$, a common value for the spline parameter in Dodgson's kernel [Dod97]. The scene uses a more aggressive temporal filter $\alpha = 0.25$ and a more complex environment texture to exaggerate the effect. In stereoscopic vision, such as VR, even low discrepancies between the eyes can lead to discomfort [KT04], which might not be immediately visible in a single view video. The video shows that our method achieves a better compromise between blur and improved stereo vision.

6. Discussion and Conclusion

To test our method for more view-dependent materials with varying degrees of glossiness, we use a simple Cornell box scene with a metal and a glass sphere. We find that our method still significantly improves temporal stability over multisampling, but the temporal blending needs to be controlled for view-dependent materials leading to a tradeoff between bias and variance, see App. C. A test of the method for animated objects is in App. D.

Our video results demonstrate qualitatively that sample caching can significantly increase temporal stability at a minor cost in terms of performance. Unlike interactive stable ray tracing [DSK*17], our method does not require validating all samples every frame and provides a better compromise between performance and quality for VR. Screen space methods that utilise backwards reprojection can show good results for many scenes, but are limited as resampling from frame to frame is lossy. We obtain real-time frame rates and practical convergence times in our results that show converged global illumination without sophisticated sampling techniques, reservoir resampling, or path guiding. Such techniques could be used to improve the radiance values we cache.

Foveated rendering or focusing shading on noisier regions are extensions worth considering, especially since we can compute good variance estimates for each frame given the visibility samples (as we did for stereo blending in Section 3.6). Another avenue of future work would be to estimate optimal temporal blending parameters using subpixel estimates. Computing an adaptive temporal parameter based on measured gradients has been shown to improve the responsiveness of view-dependent effects [SPD18].

By combining our SampleCache with HashCache, we show that stable surface points can improve existing radiance caching methods. While we did use TAA for some of our results, the Hybrid method delivers a good picture without it. As TAA always reduces variance at the cost of bias (blur and ghosting), removing the need for temporal filtering in any part of a rendering algorithm can improve image clarity.

Temporal stability and stereo discrepancy error play a big role in stereoscopic ray tracing, especially when using Monte Carlo techniques. We show that antialiasing and shading reuse can be achieved with a sample cache that lends well to stereoscopic ray tracing.

Acknowledgement

The test scenes are from McGuire's Computer Graphics Archive [McG17]. The bistro scene is from the Open Research Content Archive (ORCA) [Ama17].

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 956585 (PRIME).

References

[Ama17] Amazon Lumberyard: Amazon lumberyard bistro, open research content archive (ORCA), July 2017. URL: http://developer.nvidia.com/orca/amazon-lumberyard-bistro. 12

[AMNA*19] Akenine-Möller T., Nilsson J., Andersson M., Barré-Brisebois C., Toth R., Karras T.: Texture level of detail strategies for real-time ray tracing. In *Ray Tracing Gems: High-Quality and Real-Time Rendering with DXR and Other APIs*, Haines E., Akenine-Möller T., (Eds.). Apress, 2019, pp. 321–345. doi:10.1007/978-1-4842-4427-2_20. 13

[BFK18] Binder N., Fricke S., Keller A.: Fast path space filtering by jittered spatial hashing. In *SIGGRAPH 2018 Talks* (2018), ACM, pp. 71:1–71:2. doi:10.1145/3214745.3214806. 7

[BMdD*23] Boissé G., Meunier S., de Dinechin H., Bartels P., Veselov A., Eto K., Harada T.: GI-1.0: A fast and scalable two-level radiance caching scheme for real-time global illumination. arXiv:2310.19855 [cs.GR], 2023. doi:10.48550/arXiv.2310.19855. 2, 6, 7

[BWP*20] Bitterli B., Wyman C., Pharr M., Shirley P., Lefohn A., Jarosz W.: Spatiotemporal reservoir resampling for real-time ray tracing with dynamic direct lighting. *ACM Transactions on Graphics 39*, 4 (2020), 148:1–148:17. doi:10.1145/3386569.3392481. 2

[CKS*17] Chaitanya C. R. A., Kaplanyan A. S., Schied C., Salvi M., Lefohn A., Nowrouzezahrai D., Aila T.: Interactive reconstruction of Monte Carlo image sequences using a recurrent denoising autoencoder. *ACM Transactions on Graphics 36*, 4 (July 2017). doi:10.1145/3072959.3073601. 2

[CP22] Cosin Ayerbe A., Patow G.: Clustered voxel real-time global illumination. *Computers & Graphics 103* (2022), 75–89. doi:10.1016/j.cag.2022.01.005. 2

[DDB*09] Debattista K., Dubla P., Banterle F., Santos L., Chalmers A.: Instant caching for interactive global illumination. *Computer Graphics Forum 28*, 8 (2009), 2216–2228. doi:10.1111/j.1467-8659.2009.01435.x. 2

[Dod97] Dodgson N.: Quadratic interpolation for image resampling. *IEEE Transactions on Image Processing 6*, 9 (1997), 1322–1326. doi:10.1109/83.623195. 11

[DS07] Dietrich A., Slusallek P.: Adaptive spatial sample caching. In *Symposium on Interactive Ray Tracing* (2007), IEEE, pp. 141–147. doi:10.1109/RT.2007.4342602. 2

[DSK*17] Dal Corso A., Salvi M., Kolb C., Frisvad J. R., Lefohn A., Luebke D.: Interactive stable ray tracing. In *Proceedings of High Performance Graphics (HPG '17)* (2017), ACM, pp. 1:1–1:10. doi:10.1145/3105762.3105769. 2, 3, 4, 7, 8, 11

[EMHB23] Eto K., Meunier S., Harada T., Boissé G.: Real-time rendering of glossy reflections using ray tracing and two-level radiance caching. In *SIGGRAPH Asia 2023 Technical Communications* (2023), ACM, pp. 4:1–4:4. doi:10.1145/3610543.3626167. 2

[Gau21] Gautron P.: Practical spatial hash map updates. In *Ray Tracing Gems II*. Springer, 2021, ch. 41, pp. 659–671. doi:10.1007/978-1-4842-7185-8_41. 2

[HSG*22] Huang T., Song Y., Guo J., Tao C., Zong Z., Fu X., Li H., Guo Y.: Real-time deep radiance reconstruction from imperfect caches. *Computer Graphics Forum 41*, 7 (2022), 267–278. doi:10.1111/cgf.14675. 2

[KHBW20] Koch D., Hector T., Barczak J., Werness E.: Ray tracing in Vulkan. Khronos Blog, March 2020. URL: https://www.khronos.org/blog/ray-tracing-in-vulkan. 6

[KT04] Kooi F. L., Toet A.: Visual comfort of binocular and 3D displays. *Displays 25*, 2 (2004), 99–108. doi:10.1016/j.displa.2004.07.004. 1, 6, 11

[LJ00] LaViola Jr J. J.: A discussion of cybersickness in virtual environments. *ACM SIGCHI Bulletin 32*, 1 (2000), 47–56. doi:10.1145/333329.333344. 1

[McG17] McGuire M.: Computer Graphics Archive, July 2017. URL: https://casual-effects.com/data. 12

[MFL21] Misiak M., Fuhrmann A., Latoschik M. E.: Impostor-based rendering acceleration for virtual, augmented, and mixed reality. In *Symposium on Virtual Reality Software and Technology (VRST)* (2021), ACM, pp. 3:1–3:10. doi:10.1145/3489849.3489865. 2

[MKK24] Murphy M., Knapik J., Kozlowski P.: RT: Overdrive in Cyberpunk 2077 Ultimate Edition - pushing path tracing one step further. Game Developers Conference (GDC), 2024. URL: https://www.nvidia.com/en-us/on-demand/session/gdc24-gdc1002. 6

[MNV*21] Mueller J. H., Neff T., Voglreiter P., Steinberger M., Schmalstieg D.: Temporally adaptive shading reuse for real-time rendering and virtual reality. *ACM Transactions on Graphics 40*, 2 (April 2021), 11:1–11:14. doi:10.1145/3446790. 2, 5, 14

[MRNK21] Müller T., Rousselle F., Novák J., Keller A.: Real-time neural radiance caching for path tracing. *ACM Transactions on Graphics 40*, 4 (2021), 36:1–36:16. doi:10.1145/3450626.3459812. 2

[MVD*18] Mueller J. H., Voglreiter P., Dokter M., Neff T., Makar M., Steinberger M., Schmalstieg D.: Shading atlas streaming. *ACM Transactions on Graphics 37*, 6 (2018), 199:1–199:16. doi:10.1145/3272127.3275087. 2

[NK11] Narvekar N. D., Karam L. J.: A no-reference image blur metric based on the cumulative probability of blur detection (CPBD). *IEEE Transactions on Image Processing 20*, 9 (2011), 2678–2683. doi:10.1109/TIP.2011.2131660. 11

[NSL*07] Nehab D., Sander P. V., Lawrence J., Tatarchuk N., Isidoro J. R.: Accelerating real-time shading with reverse reprojection caching. In *Graphics hardware* (2007), vol. 41, pp. 61–62. 2

[NVI25] NVIDIA Corporation: NVIDIA DLSS, 2025. Accessed on 4 April 2025. URL: https://developer.nvidia.com/rtx/. 2, 6

[OLK*21] Ouyang Y., Liu S., Kettunen M., Pharr M., Pantaleoni J.: ReSTIR GI: Path resampling for real-time path tracing. *Computer Graphics Forum 40*, 8 (2021), 17–29. doi:10.1111/cgf.14378. 2

[PFJ23] Philippi H., Frisvad J. R., Jensen H. W.: Practical temporal and stereoscopic filtering for real-time ray tracing. In *Eurographics Symposium on Rendering (EGSR)* (2023), Eurographics Association, pp. 111–118. doi:10.2312/sr.20231129. 2, 6, 7, 10, 11

[PGSD13] Popov S., Georgiev I., Slusallek P., Dachsbacher C.: Adaptive quantization visibility caching. *Computer Graphics Forum 32*, 2 (2013), 399–408. doi:10.1111/cgf.12060. 2

[RWB*20] Roth T., Weier M., Bauszat P., Hinkenjann A., Li Y.: Hash-based hierarchical caching and layered filtering for interactive previews in global illumination rendering. *Computers 9*, 1 (2020), Article 17. doi:10.3390/computers9010017. 2, 6

[Sch96] Schaufler G.: Exploiting frame-to-frame coherence in a virtual reality system. In *Proceedings of Virtual Reality Annual International Symposium* (1996), IEEE, pp. 95–102. doi:10.1109/VRAIS.1996.490516. 2

[SKW*17] Schied C., Kaplanyan A., Wyman C., Patney A., Chaitanya C. R. A., Burgess J., Liu S., Dachsbacher C., Lefohn A., Salvi M.: Spatiotemporal variance-guided filtering: real-time reconstruction for path-traced global illumination. In *Proceedings of High Performance Graphics (HPG '17)* (2017), ACM, pp. 2:1–2:12. doi:10.1145/3105762.3105770. 2

[SPD18] Schied C., Peters C., Dachsbacher C.: Gradient estimation for real-time adaptive temporal filtering. *Proceedings of the ACM on Computer Graphics and Interactive Techniques 1*, 2 (August 2018), 24:1–24:16. doi:10.1145/3233301. 12, 14

[SS96] Schaufler G., Stürzlinger W.: A three dimensional image cache for virtual reality. *Computer Graphics Forum 15*, 3 (1996), 227–235. doi:10.1111/1467-8659.1530227. 2

[TLP*22] Thomas M. M., Liktor G., Peters C., Kim S., Vaidyanathan K., Forbes A. G.: Temporally stable real-time joint neural denoising and supersampling. *Proc. ACM Comput. Graph. Interact. Tech. 5*, 3 (2022), 21:1–21:22. doi:10.1145/3543870. 2

[vA23] van Antwerpen D. G.: Solving self-intersection artifacts in DirectX raytracing. NVIDIA Developer Blog, August 2023. Accessed on 30 September 2023. URL: https://developer.nvidia.com/blog/solving-self-intersection-artifacts-in-directx-raytracing/. 4

[Wri21] Wright D.: Radiance caching for real-time global illumination. In *Advances in Real-Time Rendering in Games: Part II* (2021), SIGGRAPH 2021 Courses. URL: https://advances.realtimerendering.com/s2021/. 2

[WTS*23] Weinrauch A., Tatzgern W., Stadlbauer P., Crickx A., Hladky J., Coomans A., Winter M., Mueller J. H., Steinberger M.: Effect-based multi-viewer caching for cloud-native rendering. *ACM Transactions on Graphics 42*, 4 (2023), 87:1–87:16. doi:10.1145/3592431. 2, 6

[YLS20] Yang L., Liu S., Salvi M.: A survey of temporal antialiasing techniques. *Computer Graphics Forum 39*, 2 (2020), 607–621. doi:10.1111/cgf.14018. 2, 6, 7

[YNS*09] Yang L., Nehab D., Sander P. V., Sitthi-amorn P., Lawrence J., Hoppe H.: Amortized supersampling. *ACM Transactions on Graphics 28*, 5 (2009), 1–12. doi:10.1145/1618452.1618481. 2, 6

[ZLY*21] Zeng Z., Liu S., Yang J., Wang L., Yan L.-Q.: Temporally reliable motion vectors for real-time ray tracing. In *Computer Graphics Forum* (2021), vol. 40, Wiley Online Library, pp. 79–90. 14

[ZW23] Zhang H., Wang B.: World-space spatiotemporal path resampling for path tracing. *Computer Graphics Forum 42*, 7 (2023), Article e14974. doi:10.1111/cgf.14974. 2

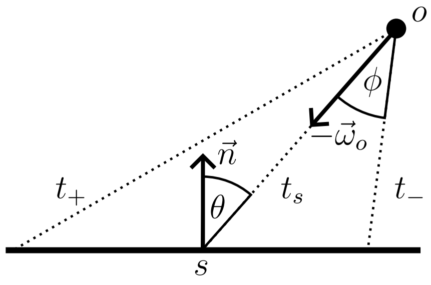


Figure 16: *Figure for deriving Eq. (1).*

Appendix A: Derivation of Equation (1)

Testing whether two first hits belong to the same surface is done using ray cones. Inspired by previous work on level of detail (LoD) selection using ray cones [AMNA*19], we derive an expected distance extent $\Delta_{\vec{n}}$ based on the diagonal field of view of a single pixel ϕ and the angle of incidence θ between the camera ray $\vec{\omega}_o$ and the surface normal $\vec{n}$.

Considering the setup in Figure 16, we are looking for the difference $\Delta_{\vec{n}} = |t_s - t_\pm|$, where the sign in the subscript indicates whether a sample within the pixel at the farthest possible distance from the intersection point along the locally flat surface is closer to or further away from the ray origin o than the intersection point s (that is, $t_- < t_s < t_+$). Using the law of sines in a triangle and trigonometric identities, we have

$$t_\pm = t_s \frac{\sin(\frac{\pi}{2} \pm \theta)}{\sin(\pi - (\frac{\pi}{2} \pm \theta) - \phi)} = t_s \frac{\cos\theta}{\cos(\theta \pm \phi)} ,$$

Thus,

$$\Delta_{\vec{n}} = |t_s - t_\pm| = t_s \left| 1 - \frac{\cos\theta}{\cos(\theta \pm \phi)} \right| .$$

Because we take the absolute value and ϕ is very small, the sign only makes a difference at almost tangential observation of the surface. Using $\theta - \phi$ avoids the singularity at $\theta \pm \phi = \frac{\pi}{2}$ since $\theta \in [0, \frac{\pi}{2}]$. Selecting $\theta - \phi$ as a good approximation (since the surface geometry does not extend to infinity in any case), we have Eq. (1), which is bounded.

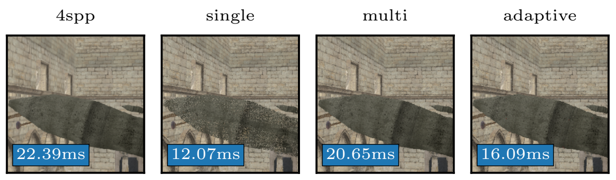
Considering samples within a pixel's 3×3 neighbourhood, the angle between a ray in the centre pixel and a ray in another pixel in the neighbourhood is at most 2ϕ. If the sample closest to the observer in a pixel is at the distance $t_{\min}$, we obtain the maximum distance to a sample within the 3×3 neighbourhood by adding $\Delta_{\vec{n}}$ computed using 2ϕ instead of ϕ. Alternatively, $2\Delta_{\vec{n}}$ with $\theta - \phi$ as the argument of the cosine function in the denominator is between $\Delta_{\vec{n}}$ with $\theta - 2\phi$ and $\Delta_{\vec{n}}$ with $\theta + 2\phi$ while still avoiding the singularity. This is a good compromise, so we use this and have that the distance $t_{\max}$ to the farthest sample is less than $t_{\min} + 2\Delta_{\vec{n}}$. Multiplying this by 1 with an added epsilon value, we have the inequality (2).

Appendix B: Single-Pass Reprojection on Other Hardware

To verify that the single-pass reprojection works on graphics hardware from other vendors, we reran the experiment in Figures 6 and

Table 1: *To verify the single-pass reprojection on non-NVIDIA hardware, we tested on an AMD Radeon 760M Graphics iGPU. Being a mobile chip, performance is very low but gives similar relative frametimes for the single-pass and two-pass method.*

Method	Reserve (ms)	Copy (ms)	Total (ms)
single-pass-no-shade	5.13 ± 0.04	/	5.13 ± 0.04
single-pass	9.06 ± 0.31	/	9.06 ± 0.31
two-pass-no-shade	3.62 ± 0.02	4.41 ± 0.02	8.02 ± 0.04
two-pass	3.53 ± 0.02	7.03 ± 0.08	10.56 ± 0.09
simple reprojection	0.97 ± 0.00	/	0.97 ± 0.00



Method	Reprojection (ms)	Tracing (ms)	Total (ms)
single	4.84 ± 0.05	3.20 ± 0.52	12.07 ± 0.63
multi	4.72 ± 0.09	11.99 ± 0.25	20.65 ± 1.11
adaptive	6.22 ± 0.23	5.81 ± 0.09	16.09 ± 0.66

Figure 17: *A rerun of the experiment in Figure 7 on an AMD Radeon 760M Graphics iGPU shows similar relative performance and equivalent image quality.*

7 at 720p on an AMD Radeon 760M Graphics iGPU. The results are in Table 1 and Figure 17. The results demonstrate equivalent relative performance and image quality.

Appendix C: Caching Specular Materials

While previous work has analysed to what degree view-dependent results can be reused over time [MNV*21], modern rendering techniques often rely on a temporal filter to combine shading results of many frames into one. We therefore include an analysis of the degree to which this is possible without a shading cache. For this purpose, we run our `SampleCache` method with adaptive tracing and uniform shading on a Cornell box with two spheres with a metal and a glass material of varying roughness ($\alpha_{GGX} \in \{0, 0.2, 0.4\}$). Because our method relies on the reprojection of first hits, we cannot use advanced techniques to compensate the motion of view-dependent effects [ZLY*21] and thus caching view-dependent effects is limited. To allow the shading cache to accommodate some view-dependent effects, we clamp the temporal blending factor (α) to a minimum value (α_{min}) depending on the roughness

$$\alpha_{min} = (1-\alpha_{GGX}) \cdot 0.5 + \alpha_{GGX} \cdot 0.1\,.$$

For a more generalised technique to get an adaptive temporal blending parameter, we refer to previous work [SPD18]. Results are shown in Figure 18 as well as a video in the supplementary files. Caching materials with high roughness is possible without introducing large errors in practice. The optimal temporal filter depends on the noise of the rendered effect, as well as the bias introduced by ghosting.

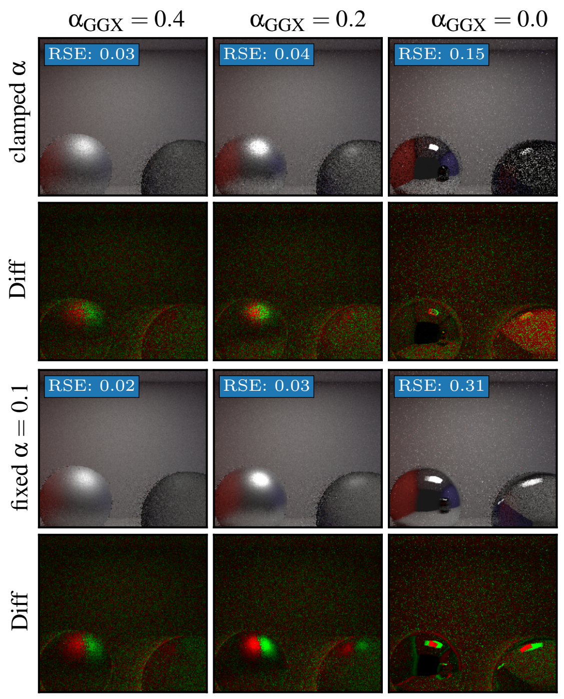


Figure 18: *Caching and temporally filtering view-dependent effects such as specular highlights and refractive transmissions is challenging since results cannot be accumulated indefinitely when the camera moves. The second and fourth rows show the difference to the reference (scaled by 3) with positive/negative differences seen as green/red. For the specular/glossy objects, the first two rows clamp the temporal blending factor α to a minimum α_{min}. The last two rows use a fixed $\alpha = 0.1$ for these objects. As seen in the difference images, the blending factor α can be used in this way to control the tradeoff between variance and bias.*

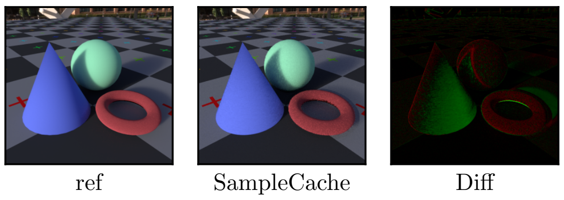


Figure 19: *Object animations can lead to temporal lag in the shading. In this example, the rotating cone stays too bright where its surface turns away from the sun.*

Appendix D: Moving Objects

As with view-dependent effects, moving objects may change the lighting of the scene in a way that requires reshading. We test our shading cache with a fixed temporal filter on a scene with animated objects. No additional changes are necessary to accomodate animations, since the first-hit information contains all necessary information, such as an instance ID, to fetch the transformation from the scene data. The result is in Figure 19 and the accompanying video. As with view dependence, the shading exhibits a temporal lag.