CHAPTER 26

# Deferred Hybrid Path Tracing

*Thomas Willberger, Clemens Musterle, and Stephan Bergmann*
*Enscape GmbH*

## ABSTRACT

We describe a hybrid rendering approach that leverages existing rasterization-based techniques and combines them with ray tracing in order to achieve real-time global illumination. We reduce the number of traced rays by trying to find an intersection in screen space and reuse information from previous frames via reprojection and filtering. Artificial lighting is stored in nodes of the spatial acceleration structure to ensure efficient memory access. Our techniques require no manual preprocessing and only a few seconds of precomputation. They were developed as a real-time rendering solution for architectural design but can be applied to other purposes as well.

## 26.1 OVERVIEW

Despite recent advances in GPU-accelerated ray tracing, it remains challenging to seamlessly scale ray tracing–based algorithms across a large variety of scene complexities while maintaining acceptable performance. This is especially true for scenarios (unlike games) where no artist can define to what extent and detail level ray tracing should be applied. We aim to provide global illumination on mostly static scene content without perceivable precomputation and with few assumptions about the scene.

To do so, we first render the scene to a G-buffer and then spawn rays from G-buffer pixels to evaluate the lighting. For each ray, we try to find intersections in screen space because, if successful, this is usually faster than tracing the ray in a spatial data structure. Additionally, we use fully lit pixels from the previous frame to get accumulated multiple-bounce lighting. If tracing in screen space is not successful, we continue tracing in a spatial data structure, in our case a bounding volume hierarchy (BVH), while keeping the visual quality degradation as low as possible. Figure 26-1 shows an actual image resulting from our implementation.


E. Haines, T. Akenine-Möller (eds.), *Ray Tracing Gems*, https://doi.org/10.1007/978-1-4842-4427-2_26

**Figure 26-1.** *Image rendered using the described approach. The majority of the required rays were traced in screen space while the reflection rays for the glass surfaces were traced in the BVH because the geometry reflected in the glass is offscreen. The scene contains a variety of materials that integrate plausibly with the scene's lighting, although most areas are only lit indirectly. Rays traced in the BVH result in one indirect bounce, while screen-space hits will benefit from recursive multiple bounces. (Image courtesy of Vilhelm Lauritzen Arkitekter for Novo Nordisk fonden, project "LIFE.")*

## 26.2 HYBRID APPROACH

We treat specular and diffuse lighting components separately to improve efficiency (e.g., reuse the view-independent diffuse component from the previous frame). The pipeline is shown in Figure 26-2. Both lighting components have the following overall concepts in common:

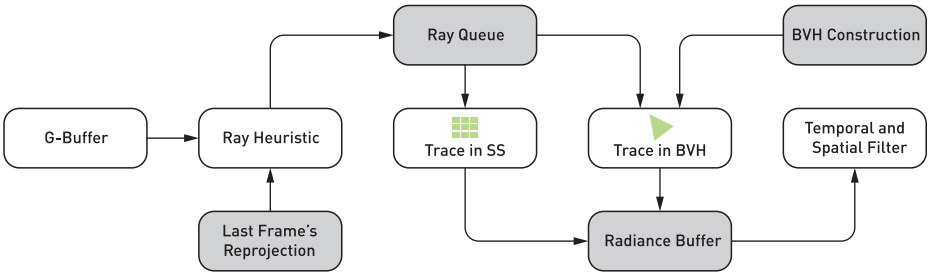


**Figure 26-2.** *Overview of the ray creation scheme for both diffuse and specular BRDFs.*

1. *Ray heuristic:* First, we decide whether or not we need a new ray. This is done by reprojecting the previous frame's unfiltered results with respect to the camera motion where possible, then comparing this for each pixel with a target ray count.

2. *Screen-space traversal:* We start the traversal in the last frame's depth buffer (Figure 26-3). Since we only use one Z-buffer layer, we assume a certain thickness $t$, proportional to the field of view and distance from the current march position to the camera, defined by

$$t = \frac{d \tan\left(\alpha_{\text{fov}} / 2\right)}{wh}, \tag{1}$$

where $\alpha_{\text{fov}}$ is the camera's field of view, $d$ is the fragment's distance to the camera, and $w$ is the width and $h$ the height of the screen in pixels. This thickness approximation is necessary to avoid rays penetrating closed surfaces at pixel edges where large depth gradients are present. For higher resolutions and a decreasing field of view, we need less thickness because the depth differentials become smaller. This solution does not guarantee watertightness, as it does not consider the geometric normal. However, for most scenarios this is sufficiently accurate. The relation to the screen resolution ensures watertight surfaces independent from the resolution or camera distance.

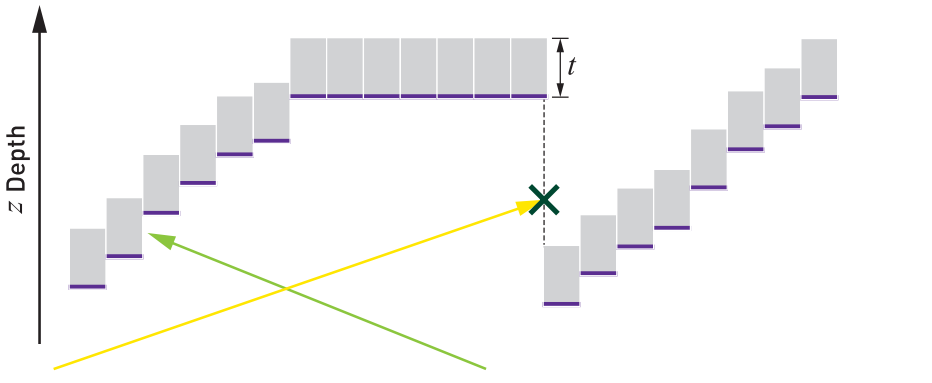


**Figure 26-3.** *Ray traversal in screen space. The front layer depth of the Z-buffer (purple line segments) is assumed to have a thickness t. This helps to prevent rays from penetrating closed surfaces (green ray). To minimize errors, we discard a screen-space ray when it enters areas behind the assumed thickness of the front layer (yellow ray) and continue tracing it in the BVH.*

We have to choose a rather small thickness to avoid false hits. During the screen-space ray march, it is possible that a sampling point lies behind (farther away from the camera than) the depth buffer, yet too

far away to be within the accepted thickness range. We lack reliable information about the occluded geometry. Therefore, the ray is not counted as being a hit, so we immediately stop the screen-space traversal. We classify such a ray as *no-hit* because we cannot be sure if there is more geometry that is not present in screen space.

Reading from the last frame's irradiance result, however, is inaccurate because some of the lighting information is view-dependent. To solve this, we store a buffer without the view-dependent components (specular) or alpha-blended geometry. This introduces an energy loss, which is currently compensated by a constant factor. The ray march result is written into a buffer of ray lengths. Then, we reconstruct the fetch position in the last frame's buffer for a subsequent pass to leverage the texture cache usage during traversal.

3. *BVH traversal:* We continue the ray traversal in our BVH, at the position where our screen space cast ends, and evaluate a radiance value. In the case of no hit, we write a skybox fetch into the accumulation buffer, which stores the radiance sum of all ray casts. This fetch can be slightly biased to reduce variance by reading from a filtered mip level depending on the estimated lobe size.

4. *Filtering:* Before compositing the traversal result, we employ a spatial filter followed by a temporal filter.

## 26.3 BVH TRAVERSAL

The range of complexity of our customers' scenes is rather large. To make sure that we can handle even large scenes with a reasonable impact on performance and memory requirements, our BVH does not contain all the scene geometry. This means that at any given time only a subset of the whole scene is included in the BVH. This subset is usually centered around the camera, which is achieved by continuously and asynchronously constructing the BVH, depending on the camera's position, and results in geometry being removed and added while the camera moves in the scene. Due to the temporal caching of various radiance buffers, the geometric change is mostly smooth. However, on some surfaces that lack a stable temporal accumulation, like alpha-blended geometry, it can be noticeable. The challenge is to include only the visually most relevant objects within our performance budget in the BVH.

### 26.3.1 GEOMETRY SELECTION

To select the relevant geometry, the total scene geometry has to be divided into meaningful parts that can be independently selected for inclusion in the BVH. This partition can be done at the hierarchy level of objects, but it was apparent that objects with a high triangle count needed to be subdivided further, so we included an automatic subdivision. We define a score function per object that describes the visual importance $j$,

$$j = \frac{a}{d^2} p, \tag{2}$$

where $a$ is the projected surface area and $d$ is the object's distance to the camera, making the first term comparable to the object's subtended solid angle as seen from the camera. The second term $p$ is an object-specific importance factor that is greater than one for emissive surfaces because their absence will have a larger visual impact than non-emissive surfaces, for which $p = 1$.

All objects are ordered by their visual importance $j$. Depending on the desired quality level, we define a total cost budget that is allowed to ensure the desired frame rate. We include the objects with the highest importance score until that budget is reached. Beside the polygon count, the cost is also multiplied by an efficiency factor that tries to predict how many axis-aligned bounding box tests are necessary to successfully intersect a primitive or leave the model's bounding box. For this factor, we use a heuristic based on the number of shared vertices in the triangle meshes. This heuristic is motivated by our experience that if triangles rarely share vertices (as in the case of vegetation), traversal performance is usually less efficient.

We end up with BVH trees with less than 10 MB that are uploaded from CPU to GPU in a couple of milliseconds. This delay can usually be hidden by double buffering.

### 26.3.2 VERTEX PREPROCESSING

For each vertex in the BVH, we precompute a single irradiance value during BVH construction. This is done to avoid having to continue tracing the BVH after the first hit, which would be expensive because the rays will be increasingly incoherent and thus incur higher computation and memory access cost. For each vertex in the BVH, we compute an irradiance value using all the lights whose area of influence include the vertex. To test the visibility, we trace a shadow ray for each vertex-light combination. When these precomputed irradiance values are used in our

lighting computation when tracing, a path from a G-buffer pixel in the BVH has the following consequences/simplifications:

- We will include paths with two bounces in our lighting calculation when tracing within the BVH, although it is only a rather coarse approximation.
- We simplify the shading in the BVH traversal, to the extent that we only include diffuse shading components, and the irradiance at each point on the triangle's surface is assumed to be a barycentric interpolation of the irradiance of the triangle's vertices.

To avoid errors that are too perceptually noticeable due to the second simplification, we subdivide triangles where the difference between adjacent vertices' irradiance values exceeds a pre-configured threshold.

### 26.3.3 SHADING

To avoid an additional access to material data or UV coordinates, we store only a single albedo value per triangle in the BVH. The texture's albedo is therefore averaged when creating the BVH. For cutout masks, we calculate the number of visible pixels and approximate the ratio with a procedural cutout pattern, which can be cheaply evaluated in the intersection shader after the triangle/ray intersection test discards the hit. While these approximations work well for diffuse lighting, the missing material and texture information can become apparent for sharp specular reflections. Therefore, we sample the surface's albedo texture for glossy specular reflections. This mode is optional and can be disabled to ensure higher performance in scenarios like virtual reality.

The total shading then consists of the per-vertex artificial lighting amount, the shadow mapped sunlight, and an ambient amount to compensate for missing multiple bounces, which is only applied at the last ray intersection. The ambient amount is proportional to an atmosphere skybox read (convolved with a cosine distribution) and an ambient occlusion factor. We approximate ambient occlusion by multiplying an exponential function with $-d$ being the ray distance toward the surface and $k$ being a scaling factor that is chosen empirically. This is a hemisphere estimation with only one sample, but it helps to reduce the ambient factor in indoor scenarios to avoid light leaking leading to an ambient factor

$$a = m\left(e^{-dk} r_{\text{skybox}} + r_{\text{vertex}} + r_{\text{sun}}\right), \tag{3}$$

where $m$ is the albedo and $r$ represents various radiance sources.

## 26.4 DIFFUSE LIGHT TRANSPORT

In this section we describe the way we handle diffuse and near-diffuse indirect light. Descriptions within the following subsections explain the key blocks in Figure 26-2 and what happens in them. Figure 26-4 shows an example of user-generated content.

**Figure 26-4.** *Images showing various light transport scenarios in architecture. Left: the sunlight can be adjusted dynamically, with all other lights and scene contents usually updating in a fraction of a second. Right: much of the visual scene content is visible in the image, which allows for an accurate multiple-bounce approximation using screen-space rays. (Images courtesy of Sergio Fernando.)*

For every material, we divide the outgoing radiance into a diffuse and a specular component. The specular component is characterized by the amount of light that is reflected according to the Fresnel function, whereas the diffuse part may penetrate the surface and is independent of the view vector (at least in simpler models like Lambert). The diffuse lobe is generally larger, which results in a higher number of samples to reach convergence. Conversely, the diffuse component has less spatial variance, which allows for more aggressive filtering approaches that incorporate a larger spatial pixel neighborhood.

### 26.4.1 RAY HEURISTIC

The challenge in a sampling strategy is that we want to get a pseudo-random sample distribution that contains as much information as possible within the radius of the filter that is applied later. Current offline renderers use sampling strategies that maximize the spatial and temporal sample variety to increase the convergence rate, like correlated multi-jitter sampling [9]. We chose a simpler logic because of different circumstances:

- Usually, samples for an image pixel are not dependent on past samples that have been accumulated on other image areas or previous frames via screen-space intersections. In our case, samples are accumulated along multiple screen- and view-space positions (due to the reprojection), distributed across several frames.

- The ray traversals itself are comparatively cheap, which makes complex sampling logic unattractive.
- The bias, introduced by sample reuse and lighting approximations, is larger than the potential convergence gain of a more advanced Quasi-Monte Carlo approach.

We start by sampling a cosine distribution, which is given by a $64^2$ pixel tiled blue noise texture with a Cranley Patterson rotation [2] of a Halton 2, 3 sequence [4] that alternates each frame. The desired sample-per-pixel count depends on the quality settings and the amount of direct light. If our history reprojection (see 26.4.2) of the diffuse radiance buffer contains more than that many samples, no new ray is cast. We account for view-dependent diffuse models by multiplying a function that depends on the dot product of the normal vector **n** and the view vector **v** and a roughness factor, similar to the precomputed specular DFG (distribution, Fresnel, geometry) term [6]. This decoupling from the view vector is necessary to allow reuse of the samples from different view angles.

Once we decide to query a new ray, the request is appended to a list (according to the ray queue in Figure 26-2). This request is then used by the screen-space traversal. If we find a valid hit in the previous frame's depth buffer, the result is written into the radiance accumulation texture (radiance buffer). If not, a ray traversal in the global BVH is initiated.

### 26.4.2 LAST FRAME'S REPROJECTION

The purpose of reprojection is to reuse shading information from previous frames. However, between two frames the camera generally moves, so the color and shading information contained in a certain pixel is possibly no longer valid for this pixel but needs to be reprojected to a new pixel location. The reprojection happens in screen space only and is agnostic of the origin of the stored radiance (BVH or screen-space ray traversal). This can be done for diffuse shading only because it is view-independent.

For a successful reprojection, we need to determine whether the shading point in question (i.e., the currently processed pixel) was visible in the previous frame; otherwise we cannot reproject. To determine whether we have a reliable source for color information, we consider the motion vector and check if the previous frame's depth buffer content for the processed shading point is consistent with the motion vector. If it is not, we probably have a disocclusion at the current position and need to request a new ray.

Another reason to request a new ray is the change of the geometric configuration: As the camera moves through the scene, some surfaces change their distance and angle to the camera. This causes a geometric distortion of the image content in screen space. When reprojecting the diffuse radiance buffer, geometric distortions have to be considered. We want to achieve a constant density of rays per screen pixel, and the described geometric distortions can change the local sample density. We store the radiance premultiplied by the number of samples that we were able to accumulate and use the alpha channel to store the sample count. The reprojection pass has to weigh the history pixels with a bilinear filter and apply a distortion factor $b$, according to Equation 4, for each of the four unfiltered fetches. Note that this factor can be $\geq 1$, for example when moving away from a wall. The distortion factor is expressed as

$$b = \frac{\mathbf{n} \cdot \mathbf{v}_{\text{current}}}{\mathbf{n} \cdot \mathbf{v}_{\text{previous}}} \frac{d^2_{\text{current}}}{d^2_{\text{previous}}}, \tag{4}$$

where $d$ is the pixel's distance to the camera and $\mathbf{v}$ is the view vector.

Figure 26-5 illustrates scene areas where reprojection caused the ray heuristic to request a new ray, either due to disocclusions or due to insufficient sample density.

**Figure 26-5.** *This image highlights the areas where a new diffuse ray is requested (green). Left: the camera is stationary. Due to the temporal antialiasing camera subpixel offset, the reprojection fails at geometry edges. Right: the camera is moving to the right. New rays are requested at geometric occlusions and areas that were not present in the previous view. Note the partially green walls due to a decrease in the density of accumulated samples.*

### 26.4.3 TEMPORAL AND SPATIAL FILTERING VIA OPTIMIZED MULTI-PASS

Image filters quickly become bandwidth bound due to the large number of memory reads. One way to alleviate this is to apply a sparse filter, recursively, in multiple iterations $n$ with $s$ samples. The effective amount of samples contributing to the filter result is $s^n$. If this sparsity is randomly distributed with a seed that varies within a 3 × 3 pixel window, the result can be filtered with a *neighborhood clamp* temporal filter [7]. The neighborhood clamp filter creates a rolling exponential

average of the pixel value and leverages the assumption that the old pixel is a blend of the new pixel's neighborhood to reject pixel history, therefore avoiding ghosting. The clamp window can be spatially extended to reduce flickering [11], which in turn increases ghosting.

The fetch positions are chosen to minimize the amount of redundant fetches within this window, while staying in the desired radius. To do so, we calculate a list of source pixel fetch positions that are effectively included after $n$ iterations of the filter. The loss function of our genetic evolution-based numerical optimization algorithm is the number of duplicate source pixel fetches within a final 3 × 3 pixel window. This ensures a maximum sample diversity within the temporal filter. The radius $r$ should be chosen according to the size $s_{\text{seed}}$ of the diffuse ray direction seed texture in order to hide tiling artifacts according to $r = s_{\text{seed}}/n$. This radius is unrelated to the 3 × 3 window for temporal accumulation, as the temporal accumulation happens after the filtering. Usually, we would weigh in a Gaussian distribution to model the relevance of nearby samples by the distance to sample. In a multi-pass approach, this is not necessary because iteratively sampling a circle-shaped kernel yields a suitable nonlinear falloff, without penalizing outer memory reads (see the article by Kawase [8]). None of the reads use hardware texture filtering to ensure discrete depth and normal weights from our aliased source buffers. Those G-buffer normals and view-space depths are then used to scale the bilateral weight, similar to a technique by Dammertz et al. [3]. Figure 26-6 compares our multi-pass approach with that of Schied et al. [12]

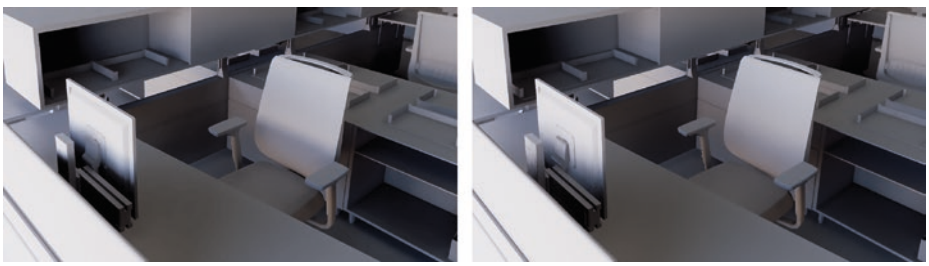

**Figure 26-6.** *Left: spatiotemporal variance-guided filtering (SVGF) [12] (2.6 ms). Right: our multi-pass filter (0.5 ms). The total screen resolution is 1920 × 1080. Both filters cover the same maximum radius, with our filter being sparse and lacking the variance-based edge-stopping function. SVGF is more accurate at preserving indirect lighting details, at a higher cost. The sparse filtering allows the final temporal antialiasing filter to darken fireflies, whereas in SVGF that filter tends to locally illuminate the temporal antialiasing clamp window.*

## 26.5 SPECULAR LIGHT TRANSPORT

Unlike diffuse filtering, specular filtering is prone to visually overblur details in reflections. We have to carefully pick and weigh the samples that we merge to create an estimate of the specular lobe. Inspired by Stachowiak [13], we trace our specular rays in half resolution and resolve to full resolution afterward using a ratio estimator. The introduced bias is acceptable, and the estimator is able to preserve normal map details and roughness variations. The major challenge remains in reducing noise in high-variance scenarios, such as rough metallic surfaces, while adding as little bias as possible. During sampling, we only importance-sample our microfacet's distribution term. The Fresnel and geometry terms are approximated by a lookup table [6].

### 26.5.1 TEMPORAL ACCUMULATION

Similar to the diffuse pass, we try to find the pixel's history by reprojecting its position into our previous specular buffer. This is done by using the virtual ray length correction techniques of Stachowiak [13] and Aizenshtein [1]. To avoid artifacts due to hardware bilinear filtering, we have to weigh the four bilinear samples individually and keep track of the total weight. In a case where the reprojection fails completely, like a disocclusion, we can only use the newly upsampled result. We use a 3 × 3 Gaussian blurred version of the upsampled buffer with a nonlinearity, like the perceptual quantizer electro-optical transfer function (used as a gamma curve in high dynamic range video signal processing), to hide fireflies.

The variance-based neighborhood clamp of temporal filtering allows us to discard incorrect reprojections. However, if the targeted radiance is occasionally not part of the local YCoCg bounding box, then flickering occurs. This can be countered by biasing the specular lobe [13], applying a variance-based post filter after temporal accumulation [14], enlarging the spatial size of the neighborhood, or simply darkening the bright pixels that introduce the bias. We observe that the flickering is mostly caused by a temporally unstable maximum luminance component. Therefore, we chose to temporally smooth the maximum luminance of the resulting color clamp. This only requires storing one additional value and causes few side effects.

### 26.5.2 REUSE OF DIFFUSE LOBE

The specular pass is performed after the diffuse pass. We reuse the filtered diffuse result in our specular pass for two reasons:

- Low-variance fallback for high-roughness, dielectric specular lobes: Using the diffuse lobe as an approximation for the specular lobe is inaccurate. However, it is visually plausible since the lobe energy resides in a similar range. This saves performance on rough surfaces with moderate visual impact. For metals, we cannot rely on this simplification because the specular component is too visible.
- Ambient lighting amount for the geometry in the reflection: In cases where we do not want to trace further and gather the incoming lighting at the hit point, we need to assume an ambient lighting factor. The reflective surface's diffuse lighting proved to be a good approximation with little cost.

### 26.5.3 PATH TRACED INDIRECT LIGHTING

Adding path traced indirect lighting is required for mirror-like surfaces. It is costly due to its incoherent memory reads and suffers from high variance. Liu [10] proposes filtering the indirect diffuse component along with the indirect diffuse component of the mirrored surfaces. To properly decouple the filtered lighting from albedo and direct light, we would need to store and fetch multiple additional buffers for our reflections. Instead, we chose to filter across the dimensions of the random seed texture (5 × 5 in our case) during the resolve pass, combined with a tone-mapped average to reduce fireflies. The filter is bilateral and takes the reflection ray length and G-buffer normal into account to preserve geometry silhouettes in reflections and normal map details. Both the special filtering and the indirect diffuse filtering is only applied for low-roughness metal surfaces, which makes the extra work affordable. A faster variant, without tracing additional rays, consists of the ambient factor combined with a screen-space ambient occlusion factor based on the ray lengths, which can be interpreted as virtual screen depth.

### 26.5.4 LOBE FOOTPRINT ESTIMATION

Similar to Liu's work [10], we scale the number of filtering fetches according to the screen-space size of the projected reflection lobe footprint. This can be done by calculating the dimensions of a two-dimensional scale matrix.

Since most surfaces are not planar, we also need to estimate the local curvature and distort the footprint accordingly. This is done by computing the local derivatives of the G-buffer normal. The neighbors are chosen according to the eigenvectors of

the two-dimensional lobe distortion matrix, which describes the lobe elongation and shrinking in the tangent space, projected to screen-space units. The smallest derivative of both neighbors is used to avoid artifacts at geometry edges. Finally, the number of samples is proportional to the matrix's determinant. If the filter size is smaller than $\sqrt{2}$ times the tracing resolution, we switch to a fixed 3 × 3 pixel kernel instead. This ensures that we consider all neighbors, which increases the reconstruction quality when dealing with curved (or normal-mapped) glossy surfaces at half resolution tracing. This is summarized in the following code.

```
mat2 footPrint;
// "Bounce-off" direction
footPrint[0] = normalize(ssNormal.xy);
// Lateral direction
footPrint[1] = vec2(footPrint[0].y, -footPrint[0].x);

vec2 footprintScale = vec2(roughness*rayLength / (rayLength + sceneZ));

// On a convex surface, the estimated footprint is smaller.
vec3 plane0 = cross(ssV, ssNormal);
vec3 plane1 = cross(plane0, ssNormal);
// estimateCurvature(...) calculates the depth gradient from the
// G-buffer's depth along the directions stored in footPrint.
vec2 curvature = estimateCurvature(footPrint, plane0, plane1);
curvature = 1.0 / (1.0 + CURVATURE_SCALE*square(ssNormal.z)*curvature);
footPrint[0] *= curvature.x;
footPrint[1] *= curvature.y;

// Ensure constant scale across different camera lenses.
footPrint *= KERNEL_FILTER / tan(cameraFov * 0.5);

// Scale according to NoV proportional lobe distortions. NoV contains
// the saturated dot product of the view vector and surface normal
footPrint[0] /= (1.0 - ELONGATION) + ELONGATION * NoV;
footPrint[1] *= (1.0 - SHRINKING) + SHRINKING * NoV;

for (i : each sample)
{
    vec2 samplingPosition = fragmentCenter + footPrint * sample[i];
    // ...
}
```

## 26.6 TRANSPARENCY

Alpha-blended surfaces' reflections are more complex, since we do not want to store the pixel's history for each alpha layer. This is possible but would increase the implementation's memory requirements. Instead, we use the main temporal

antialiasing filter to take care of stochastic noise. This is acceptable because we assume that most alpha-blended surfaces (like glass) have a low roughness and therefore do not suffer from much variance during importance sampling of the specular distribution. Our order-independent transparency approach sorts the alpha pixels into layers before shading them, which allows us to employ different quality settings for each layer. We trace all layers in half resolution, just as for our specular component on opaque geometry. In contrast to the specular pass, we lack a G-buffer, which is why we cannot use the identical upscale algorithm. Instead, we implement a spatiotemporal shuffle by using blue noise–based offsets per pixel in the full resolution pass. This can be seen as a blur filter with only one fetch. Combined with the temporal antialiasing filter, this can be used to trade undersampling artifacts with noise.

## 26.7 PERFORMANCE

The performance results were measured using NVIDIA Titan V hardware at a resolution of 1920 × 1080. The current implementation still uses custom shaders for traversal, instead of DirectX Raytracing, for example. The scene contains 15 million polygons and represents an average architectural scene, as shown in Figure 26-7. The total frame time during these measurements was continuously below 9 ms.

**Figure 26-7.** *Test scene for benchmarking. The scene was created in Autodesk Revit and includes various interior objects, trees, water, and a variety of materials.*

Table 26-1 illustrates the timings of relevant sections in a real-time walkthrough scenario with our default high-quality configuration. Many system parameters can be adjusted to increase the quality and approach ground truth much more closely, e.g., for still images and videos, or to gain more performance for virtual reality (VR) rendering where low frame times are essential for the experience. Besides common parameters, like the number of samples and light bounces, the filter kernel sizes, and the number of BVH polygons, we also found adjusting the maximum ray lengths and the threshold for the specular-to-diffuse fallback (see 26.5.2) to be effective tools to strike a balance between quality and frame time for the desired use case.

**Table 26-1.** *Pass times of specular and diffuse light transport. Timings of diffuse passes are given for one indirect bounce. The number of new rays depends on the success of the last frame's reprojection. Therefore, camera movement causes higher workload. The diffuse filtering only depends on the percentage of geometry pixels visible on the screen. The specular tracing is performed in half resolution. Unlike the diffuse pass, the reprojection for specular light transport happens in the temporal filter. The spatial filter runtime increases with rough materials due to their larger footprint.*

| | | Path Tracing | | |
|---|---|---|---|---|
| Pass Times (in ms) | Reprojection | Screen Space | BVH | Filtering |
| Diffuse Resting | 0.18 | 0.05 | 0.25 | 0.47 |
| Diffuse Moving | 0.18 | 0.21 | 1.10 | 0.47 |
| Specular | - | 0.20 | 0.49 | 1.07 |

### 26.7.1 STEREO RENDERING FOR VIRTUAL REALITY

For VR, we chose one eye to be dominant and alternate our choice each frame. For the dominant eye, we update the diffuse lighting. For the other eye, the past frame's information is reprojected in the same way as we reproject our diffuse and specular buffers in a regular scene rendering cycle. However, this approach creates artifacts. Geometric occlusion causes holes during camera movement. Due to the stochastic nature of our sampling, differences in the integration results become apparent when viewed with a stereoscopic headset. The differences can be the result of different sampling seeds at the same world-space location. To address both issues, we reuse the newly updated information of the dominant eye by reprojecting it. It is then merged with a constant blend factor $\gamma$ onto the other eye. If the past information of the identical eye from the last frame could not be used, but we have a successful reprojection, $\gamma = 1$.

For the diffuse ray heuristic, we increase the desired sample density at the center of the image. On outside regions, we also tolerate sample densities below one. These can occur after a reprojection, but are still acceptable in most scenarios. We use this foveation approach to concentrate our computational resources where they are most effective.

### 26.7.2 DISCUSSION

Our described global illumination algorithm is able to scale across different performance requirements. It can output high-quality images with multiple bounces, and with a different parameter set, it is able to reach the low frame times required for VR—with almost the same code path. Some of the state-of-the-art image G-buffer–based techniques, like post-processed depth of field or motion blur, work sufficiently well while being highly efficient. Others, like shadow

mapping, can be improved by ray tracing. Replacing a high number of shadow-mapped lights by ray tracing remains a performance challenge, yet it already promises high-quality results [5].

We also see room for improvement in the scalability of ray traced reflections on multiple alpha-blended layers. This is related to the calculation of subsurface scattering phenomena that are currently approximated by lighting in a volume texture in our case. For diffuse and specular integration, we would like to make specular ray tracing benefit from a ray heuristic, instead of equally sampling all screen regions each frame.

## ACKNOWLEDGMENTS

We thank Tomasz Stachowiak and the editors for their valuable input, corrections, and suggestions that greatly improved this chapter.

## REFERENCES

[1] Aizenshtein, M., and McMullen, M. New Techniques for Accurate Real-Time Reflections. Advanced Graphics Techniques Tutorial, SIGGRAPH Courses, 2018.

[2] Cranley, R., and Patterson, T. N. L. Randomization of Number Theoretic Methods for Multiple Integration. *SIAM Journal on Numerical Analysis 13*, 6 (1976), 904–914.

[3] Dammertz, H., Sewtz, D., Hanika, J., and Lensch, H. P. A. Edge-Avoiding À-Trous Wavelet Transform for Fast Global Illumination Filtering. In *Proceedings of High-Performance Graphics* (2010), pp. 67–75.

[4] Halton, J. H. Algorithm 247: Radical-Inverse Quasi-Random Point Sequence. *Communications of the ACM 7*, 12 (1964), 701–702.

[5] Heitz, E., Hill, S., and McGuire, M. Combining Analytic Direct Illumination and Stochastic Shadows. In *Symposium on Interactive 3D Graphics and Games* (2018), pp. 2:1–2:11.

[6] Karis, B. Real Shading in Unreal Engine 4. Physically Based Shading in Theory and Practice, SIGGRAPH Courses, August 2013.

[7] Karis, B. High-Quality Temporal Supersampling. Advances in Real-Time Rendering in Games, SIGGRAPH Courses, 2014.

[8] Kawase, M. Frame Buffer Postprocessing Effects in DOUBLE-S.T.E.A.L (Wreckless). Game Developers Conference, 2003.

[9] Kensler, A. Correlated Multi-Jittered Sampling. Pixar Technical Memo 13-01, 2013.

[10] Liu, E. Real-Time Ray Tracing: Low Sample Count Ray Tracing with NVIDIA's Ray Tracing Denoisers. Real-Time Ray Tracing, SIGGRAPH NVIDIA Exhibitor Session, 2018.

[11] Salvi, M. High Quality Temporal Supersampling. Real-Time Rendering Advances from NVIDIA Research, Game Developers Conference, 2016.

[12] Schied, C., Kaplanyan, A., Wyman, C., Patney, A., Chaitanya, C. R. A., Burgess, J., Liu, S., Dachsbacher, C., Lefohn, A. E., and Salvi, M. Spatiotemporal Variance-Guided Filtering: Real-Time Reconstruction for Path-Traced Global Illumination. In *Proceedings of High-Performance Graphics* (2017), pp. 2:1–2:12.

[13] Stachowiak, T. Stochastic Screen-Space Reflections. Advances in Real-Time Rendering in Games, SIGGRAPH Courses, 2015.

[14] Stachowiak, T. Towards Effortless Photorealism through Real-Time Raytracing. Computer Entertainment Developers Conferences, 2018.