Eurographics Symposium on Rendering (2024)
E. Garces and E. Haines (Editors)

Real-Time Pixel-Perfect Hard Shadows with Leak Tracing

René Kern Felix Brüll Thorsten Grosch

TU Clausthal, Germany

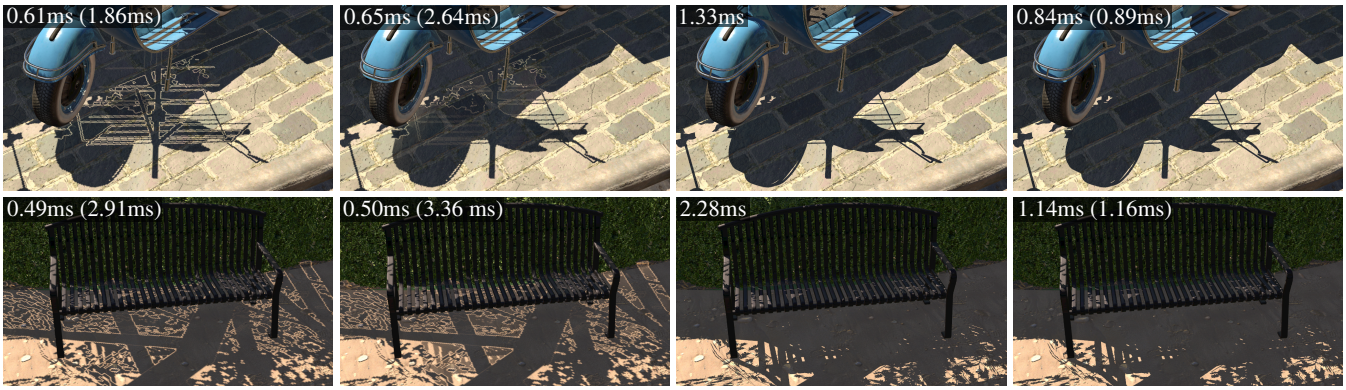

Figure 1: *Filterable shadow maps, such as VSM [DL06] and MSM [PK15], provide a fast approximation of shadows but often produce visible errors, namely light leaking. Ray tracing can calculate pixel-perfect shadows but is slower, especially for alpha-tested shadows. Our leak tracing algorithm traces rays in problematic areas to achieve pixel-perfect shadows similar to ray tracing but in a fraction of the time. While cascaded shadow maps require re-generation every frame, Leak Tracing can reuse shadow maps even with camera movement. Time format: Shadow evaluation time (+ shadow map generation). Scenes: Bistro [Lum17] (Top), Emerald Square [NHB17] (Bottom).*

Abstract

Accurate shadows greatly enhance the realism of a rendered image. Shadow mapping is the preferred solution for shadows in real-time applications. However, shadow maps suffer from discretization errors and self-shadowing artifacts, that need custom parameter tuning per scene. Filterable shadow maps such as variance or moment shadow maps solve both issues but introduce light leaking. With the advent of hardware ray tracing, it becomes more realistic to use shadow rays instead of a shadow map. However, distributing a shadow ray is often more expensive than evaluating a shadow map, especially if the ray hits alpha-tested geometry. We introduce leak tracing, where we use filterable shadow maps techniques on top of default shadow maps and eliminate the light leaks and aliased shadow edges with selective ray tracing. Our algorithm does not need any scene-dependent parameters. We achieve an average speedup ranging from 1.19 *to* 1.79*, with a top speedup of* 4.17*, depending on the scene and eliminate major performance drops caused by alpha-tested geometry during ray tracing. Our solution is temporally stable and reaches similar quality as pure ray tracing.*

CCS Concepts

• ***Computing methodologies*** → ***Ray tracing; Rasterization; Visibility;***

1. Introduction

Shadow Mapping [Wil78] is a popular method to render the shadow that a light source casts onto the scene by utilizing a depth map (shadow map) from the view of the light source. However, shadow mapping is plagued by aliasing artifacts or self-shadowing (shadow acne). Since its introduction, other works improved the issues of shadow maps by filtering [RSC87], modifying the perspective and relative resolution [SD02, WSP04, MT04, Eng06, OSK*14], or us-

DOI: 10.2312/sr.20241158

ing the shadow map as a filterable function [DL06, AMB*07, AMS*08, PK15].

With the advent of hardware ray tracing, ray traced shadows are a new option. They allow for unbiased shadows by tracing a ray to the light source. However, while offering a better quality, tracing a ray is still more expensive than the texture lookup used for shadow mapping.

We introduce leak tracing, a combination of filterable shadow maps and ray tracing, that is able to produce ray tracing quality shadows in a fraction of the time. For that, we only distribute rays on the error-prone areas (light leaks, shadow edge) of the filterable shadow maps. Our main contributions are:

- Our novel **Leak tracing** algorithm that combines filterable shadow maps with ray tracing. It resolves light leaking and discretization errors of filterable shadow maps by selectively tracing shadow rays on error-prone areas. The algorithm works on top of default shadow maps and is free of scene-dependent parameters.
- A 3-bit per light **temporal mask** that reduces the remaining errors of the leak tracing test by forcing rays to neighboring pixels on areas where the leak tracing test fails, while removing rays from unproblematic areas.

2. Related Work

Rendering shadows in real-time is a longstanding problem. For rasterization, Shadow Mapping [Wil78] and Shadow Volumes [Cro77] are two popular methods to determine shadows. Over the years, shadow mapping has turned out to be the more popular technique.

To use shadow mapping, first a depth texture from the view of the light source needs to be rendered. This depth texture, called 'shadow map', can then be used to determine if any point p of the scene is shadowed. This is done by transforming p to the perspective of the light source and comparing the depth values. However, due to the discretization, shadow maps are prone to aliasing and self-shadowing (shadow acne).

Undersampling is one of the main reasons for the aliasing artifacts. A better placement of the shadow maps can reduce that problem. Adapting the shadow map to the camera frustum [SD02, WSP04, MT04] or using 'virtual' shadow maps with adaptive resolution [OSK*14] can reduce the undersampling problem. For directional lights cascaded shadow maps [Eng06] are a popular solution, where multiple shadow maps are used for different parts of the camera's view frustum.

Another approach to reduce aliasing is filtering the shadow. Percentage closer filtering (**PCF**) [RSC87] can reduce the aliasing artifacts by repeating the shadow test and filtering its results with a filter kernel. However, PCF is expensive, as it requires multiple texture fetches.

With filterable shadow maps, the shadow maps can be prefiltered and sampled with hardware filtering, needing only one texture operation per pixel for an antialiased shadow. Variance shadow maps [DL06] (**VSM**) store z^2 in addition to z to approximate the shadow with Chebyshev's inequality. Convolution shadow maps [AMB*07] (**CSM**) approximate the shadow test by using a Fourier series expansion. Exponential shadow maps [AMS*08, Sal08] (**ESM**) approximate the shadow test with an exponential function. Exponential variance shadow maps [LM08] (**EVSM**) combine VSM and ESM by storing the first two moments of a positive and negative exponential function and evaluating Chebyshev's inequality for both. Moment shadow maps [PK15] (**MSM**) store the first four moments for the shadow map and solve the corresponding four-moment problem to approximate the shadow.

While all the mentioned filterable shadow map techniques improve aliasing and self-shadowing, they are susceptible to light leaking, cracks of light inside the shadow (Figure 1 and 2). There are some solutions to reduce the light leaking by e.g. using summed-area tables [Lau07] or layered shadow maps [LM08]. However, they do not remove all light leaks.

Pixel-perfect shadows are another area of research, aiming to produce alias-free hard shadows similar to those generated by ray tracing. Adaptive Shadow Maps [FFBG01] iteratively refine the shadow map resolution at visually important regions (such as shadow edges) by estimating the footprint of a shadow map texel with mipmapping from the camera view, requiring a CPU readback of the camera view buffer. Lefohn et al. [LSK*05] demonstrated a full GPU implementation using a quadtree. Resolution Matched Shadow Maps [LSO07] improve on Adaptive Shadow Maps by directly approximating the required shadow map resolution for each node without iterative refinement. Shadow Silhouette maps [SCH03] use an additional silhouette texture from the light view to reconstruct the shadow silhouette using neighboring silhouette texels. Noticable artifacts can appear if two silhouette shadows project close to each other. Scherzer et al. [SJW07] use jittered shadow maps with temporal accumulation to produce pixel-perfect shadows after multiple frames. To converge to a pixel-perfect shadow, a confidence function is used, that weights samples in the shadow map texel center higher. However, temporally reprojected shadows struggle with moving objects and light animations.

Shadow alias artifacts result from a mismatch between the camera and shadow view query locations. Alia and Laine [AL04] and Johnson et al. [JLBM05] independently proposed similar solutions to this problem. First, a camera pass is executed to obtain the desired shadow map sample locations, which are then used in a shadow map generation pass. However, these sample locations lie on an irregular grid, making rasterization impossible. Arvo [Arv07], Sintorn et al. [SEA08] and Wyman et al. [WHL15] demonstrated graphics hardware implementations of an irregular z-buffer. All of these implementations require conservative rasterization, and the shadow pass needs to be evaluated every frame. Sintorn et al. [SKOA14] use per-triangle shadow volumes [SOA11] with a view based hierarchical clustering method to produce pixel-perfect shadows.

Advancements in hardware ray tracing allows for ray traced shadows as an artifact-free real-time alternative to shadow maps. However, ray traced shadows are still more expensive than a lookup in the shadow map. An approach that was developed before hardware ray tracing uses conservative rasterization to detect the areas where a ray is needed [Sto15]. AMD implemented a hybrid shadow map system in their FidelityFX framework [AMD21]. The hybrid shadow test is performed by repeating the shadow test mul-

tiple times, akin to PCF, and distributing a ray if the shadow test result differs for one or more samples. Choi [Cho21] presented a hybrid soft shadow algorithm, where ray tracing is used for opaque shadows and percentage closer soft shadows [Fer05] is used for alpha-tested geometry. The results require a denoiser.

2.1. Filterable Shadow Maps

Our leak tracing algorithm is built on top of filterable shadow maps that use an upper bound approximation to produce pixel-perfect hard shadows akin to ray traced shadows. In the following, we look at the properties and errors of the filterable shadow maps, which are used to selectively trace a shadow ray on the shadow edge and problematic areas.

The depth information from the view of the light is stored in a shadow map $\bar{z}(t)$ with the texture coordinate $t \in \mathbb{R}^2$. The depth $z \in [0,1]$ is stored as a vector $b(z) \in \mathbb{R}^n$ with n number of channels. A shadow test $s(\bar{z}(t),d)$ is performed with the depth vector from the shadow map $\bar{z}(t) := b$ and the light space depth of the current position d.

The simplest case is shadow mapping [Wil78], where the depth vector $b(z) := z$ is compared to d. The shadow test $s(\bar{z}(t),d)$ is a binary function which is 0 (shadowed) if $z < d$ and 1 (lit) otherwise. Filtering over the shadow map depth z does not make much sense, as only the depths would be averaged while the shadow test is still binary. However, filtering multiple shadow test results with PCF [RSC87] is possible. For that, a filter kernel K with the filter weights $k(t)$ is needed and the shadow function is defined as:

$$f_{PCF}(t,d) = \sum_{t_i \in K} k(t_i - t)\, s(\bar{z}(t_i),d) \tag{1}$$

which can also be modeled as a probability distribution function $P(z \geq d) \in [0,1]$, that expresses how many neighboring shadow map samples z are not occluded at d, essentially modeling the visibility of the light. However, this comes with a performance penalty because the shadow test, which samples the shadow map, needs to be repeated multiple times.

Filterable shadow maps aim to approximate the probability distribution $P(z \geq d)$ while allowing pre-filtering the shadow map by using different depth vector definitions and shadow tests. After creating a filterable shadow map, the values can be pre-filtered with:

$$\hat{z}(t) = \sum_{t_i \in K} k(t_i - t)\bar{z}(t_i) \tag{2}$$

where $\hat{z}(t)$ is the prefiltered shadow map. The different filterable shadow maps use different depth vectors. VSM [DL06] uses $b(z) := \{z, z^2\}$, ESM [AMS*08, Sal08] uses $b(z) := e^{c \cdot z}$, where c is a constant. EVSM [LM08] uses $b(z) := \{e^{c \cdot z}, e^{2c \cdot z}, -e^{-(c \cdot z)}, e^{-(2c \cdot z)}\}$ and MSM [PK15] uses $b(z) := \{z, z^2, z^3, z^4\}$.

All of these pre-filtered shadow maps approximate the probability of the shadow by computing the upper bounds to $P(z \geq d)$ with the shadow test $\hat{s}(\hat{z}(t),d) \leq P(z \geq d)$. Using the upper bounds will result in a shadow that is never too dark. VSM uses Chebychev's inequality. For EVSM, Chebychev's inequality is evaluated for the first and last two elements of the depth vector separately and the minimum between both is used as the approximated shadow probability. ESM uses $\hat{s}(\hat{z}(t),d) := \max(e^{-cd}\,\hat{z}(t), 1)$ as the shadow test. For MSM the Hamburger problem with four moments is solved (see MSM paper [PK15], Algorithm 2).

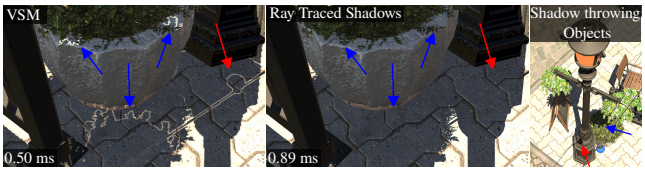

Figure 2: *Filterable shadow maps such as VSM produce light leaks on areas with increased variance, which usually happens if one occluder is in front of another.*

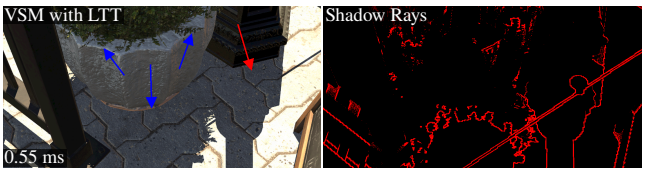

Figure 3: *VSM with LTT (Equation 3) at the same camera angle as Figure 2. The right image shows the regions where LTT is true since $\hat{s}$ is between ϵ and $(1-\epsilon)$ and a shadow ray is distributed. Note that all light leaks and discretization errors from Figure 2 are fixed while being 61% faster than pure ray tracing.*

Additionally, filterable shadow maps have the advantage over default shadow maps, in that no scene-dependent bias needs to be set by hand, as they are free from self-shadowing due to the upper bound approximation.

2.2. Light Leaks

While filterable shadow maps are efficient, they suffer from light bleeding/leaks. That is the case because all approximate the shadow probability by calculating the upper bound to $P(z \geq d)$, which potentially results in a shadow that is too "bright". Light leaking in particular happens if the variance in the filter region is high, which usually happens if one occluder is in front of another occluder, as seen in the example in Figure 2. There, the silhouette from the wire leaks through the shadow of the street lamp.

3. Leak Tracing Shadow Test

As filterable shadow maps approximate the shadow probability distribution $P(z \geq d)$, the shadow test returns values in the range of $0 \leq \hat{s}(\hat{z}(t),d) \leq 1$. Fortunately, the distribution on fully shadowed (0) and fully lit (1) regions is reliable, so that only regions near the shadow edge or inside light leaks return values between 0 and 1. This property can be exploited, as we now know where the edges and problematic areas of the shadows are.

Therefore, we want to distribute a ray, whenever the filterable shadow map test results in a value between 0 and 1, which corresponds to the boolean test: $0 < \hat{s}(\hat{z}(t),d) < 1$. However, in practice the filterable shadow tests are numerically unstable, thus a small

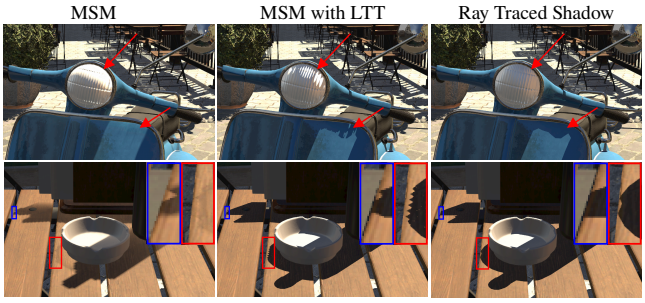

Figure 4: *Failure cases for our leak tracing shadow test caused by the discretization of the shadow map. Visible errors appear on objects where the geometry is nearly parallel to the light direction (top) and on shadow edges that are near gaps (bottom). The artifacts can be fully removed using our temporal mask (Section 3.1).*

Figure 5: *The roller and ashtray from Figure 4 with the different stages of the temporal LTT used on MSM. Combining LTT with the temporal tests forms our leak tracing (Lt) algorithm, which produces shadows similar to ray tracing (d).*

bias should be applied to this test (e.g. $\epsilon = 0.01$). With this, we define the leak tracing shadow test (**LTT**) as:

$$LTT := \epsilon < \hat{s}(\hat{z}(t), d) < (1 - \epsilon) \tag{3}$$

where a ray should be distributed whenever LTT evaluates true. With this simple test, the light leaks and most discretization errors on the shadow edge can be eliminated as seen in Figure 3.

However, while LTT works well for most cases, it has some failure cases caused by the discretization of the shadow map, which is shown in Figure 4. We opted to resolve these issues using temporal information, described in the next section.

3.1. Temporal Leak Tracing Mask

To resolve the errors shown in Figure 4 with temporal information, we first need to classify the possible outcomes of LTT and then identify the problematic cases. In addition to the LTT result, we need information about the shadow test result (**ST**) and the shadow ray result (**RS**). ST and RS are booleans and are defined as follows:

$$\begin{aligned} ST &:= \textit{shadow test value} < 0.5 \text{ (shadow)} \\ RS &:= \text{ray hits any geometry (shadow)} \end{aligned} \tag{4}$$

All possible combinations are shown in Table 1, with a visual representation shown in Figure 5a. In the following, we will reference the corresponding color when discussing a specific case.

First, the remaining discretization error of LTT are eliminated by expanding shadow rays to regions that are error-prone (red arrows/box in Figure 5a) in Section 3.1.1. Then runtime is improved in Section 3.1.2 by removing rays on regions where the shadow ray is identical to the shadow map result (blue arrow in Figure 5a). A complete overview of the leak tracing algorithm is then shown in Section 3.1.3.

In the following, we assume that the light is mostly static, where only small light movements are allowed. Camera movement and dynamic geometry are fully supported by the temporal mask using re-projection with motion vectors. If re-projection fails, we force a shadow ray.

Color	LTT	ST	RS	Effect
Black	0	0	-	Conf. lit or backfacing light
Green	0	1	-	Confidently shadowed
White	1	1	1	Shadowed ST; Ray shadowed
Purple	1	0	1	Lit ST; Ray shadowed
Red	1	0	0	Lit ST; Ray lit
Yellow	1	1	0	Shadowed ST; Ray lit

Table 1: *Boolean classifications for a pixel in the temporal leak tracing mask.*

3.1.1. Ray Expansion

As the filterable shadow maps approximate the upper bound of the shadow, the only theoretical remaining discretization errors are in regions where the shadow test wrongfully evaluated lit and the LTT test evaluated false (*black* case). However, while all failure cases are in the *black* case, not all pixels in the *black* case are failure cases. When analyzing the neighboring pixels of a failure case (red arrow/box in Figure 5a), it shows that a failure case is always adjacent to the *purple* case, where LTT corrected a lit shadow test. Ideally, we would want to completely ray trace along the silhouette of the shadow. This means for our LTT, that a correction of a lit shadow test (*purple*) should always be adjacent to a confirmation of a lit shadow test (*red*). Therefore, we want to set the current LTT result to true which forces a shadow ray if this property was not fulfilled in the last frame. To be fully consistent we also need to handle the same case for wrongly shadowed regions (*yellow*), even though it is theoretically impossible due to the upper-bound approximation, it can exist in practice. Therefore, a corrected shadow result (*yellow*) should always be adjacent to a shadow confirmation (*white*).

To summarize, when LTT evaluates false for the current pixel, we check if the *purple* or *yellow* case is present in any of the adjacent neighboring pixels. If that is the case, the property is not fulfilled and we need to expand the ray and set the current LTT result to true. The expand rays check (**ER**) that is checked for each adjacent neighboring pixel in the temporal mask M, can be expressed

as the following:

$$ER_{purple} := M_{LTT} \wedge (M_{RS} \wedge \neg ST)$$
$$ER_{yellow} := M_{LTT} \wedge (\neg M_{RS} \wedge ST) \quad (5)$$

with ST being the current shadow test result, M_{LTT} and M_{RS} being the LTT and RS results from last frame. Both boolean expressions can also be combined into a singular expression. Additionally, we also want to distribute a ray if the shadow map result changes compared to the last frame, to cover slight light movements and disocclusion. Combining this with both equations from Equation 5 results in the boolean expression:

$$ER_{LTT} := (M_{LTT} \wedge (M_{RS} \oplus ST)) \vee (M_{ST} \oplus ST) \quad (6)$$

that is tested for all adjacent neighboring mask pixels and the LTT result is set to true if Equation 6 is true for any neighboring pixel. As temporal information is used to fix the remaining errors of LTT, a couple of frames are needed to completely fix the failure cases. However, as the failure cases often occupy a small area, $4-8$ frames are usually enough to fully fix those errors, resulting in a mask that can be seen in Figure 5b, which from this point on is fully temporally stable.

Checking all adjacent neighbors requires a 3×3 kernel. After experimenting with multiple sample patterns, a 2×2 gather with a subpixel jitter, so that a 3×3 region is covered over 4 frames, was the most performant option while delivering indistinguishable results from a full 3×3 filter kernel.

Expanding the rays fixes the remaining discretization errors in LTT. However, the mask decreases performance slightly as it includes read and write operations for the mask. To compensate for this, we also want to use the temporal mask to remove rays that are identical to the shadow map result, which is either caused by poor shadow map approximation or light leaks.

3.1.2. Ray Removal

In order to improve performance, we want to avoid distributing rays in regions where the shadow ray only confirms the shadow map result ($ST == M_{RS}$). Additionally, we want to verify that we are still in the same shadow case as in the last frame ($ST == M_{ST}$). Looking at our classification (Table 1), this corresponds to the *white* case for shadows and the *red* case for lit regions. Therefore, we can assume that a ray is not needed if all adjacent neighbors only contain confirmations (*white, red*) or are already confident in the result (*black, green*). With this, we define the following remove ray (**RR**) expressions:

$$RR_{white,red} := M_{LTT} \wedge (ST == M_{ST}) \wedge (ST == M_{RS})$$
$$RR_{black,green} := \neg M_{LTT} \wedge (ST == M_{ST}) \quad (7)$$

We can remove a ray if either $RR_{black,green}$ or $RR_{white,red}$ evaluates true for all adjacent neighboring pixels in the mask M. To not overwrite the ER expressions, we only use RR if the original LTT test evaluated true. Equation 7 can be combined and simplified to:

$$RR_{LTT} := (ST == M_{ST}) \wedge (\neg M_{LTT} \vee (ST == M_{RS})) \quad (8)$$

which is the (optional) remove ray test for LTT. Additionally, when a ray is removed, the shadow value from the shadow test needs to be set to either 0 or 1 to stay consistent with the ray tracer. The result after a couple of frames of using ER and then RR can be seen in Figure 5c.

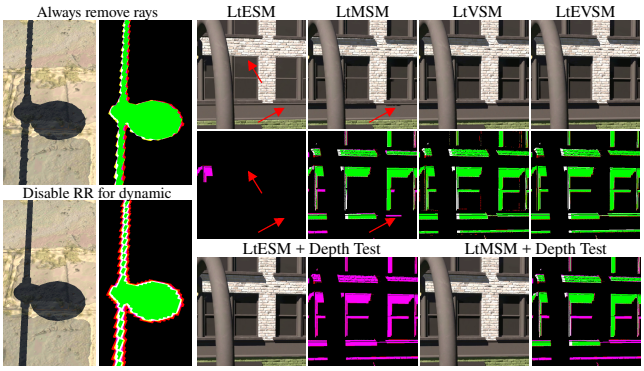

Figure 6: ***(a)*** *Discretization error that happens when remove ray is used with dynamic geometry, as the ray traced shadow edge is reduced. The screenshot was captured in motion (wire moves from right to left) with LtEVSM* ***(b)*** *A house wall of the Emerald Square scene with a right-to-left camera motion. Visible disocclusion can occur in ESM and MSM when remove rays is used, as the shadow values from the shadow map are near to 1. Using an additional temporal depth test can fix this problem.*

However, removing rays is potentially dangerous in two cases. One is dynamic geometry, where movements can cause visible discretization errors (Figure 6a). To avoid these errors, we disable RR_{LTT} for dynamic shadows. Section 3.4 goes into more detail on how we handle dynamic geometry. The second case happens due to dissocclusion artifacts. When the whole shadow is only covered by the *purple* case and a confidently lit object (*black* case) occludes it, most of the shadow will be removed, as the LTT result is overwritten before a shadow ray is distributed (see Figure 6b). Our experiments show that this only happens with LtEVM and LtMSM, as LtVSM and LtEVSM approximate the inner part of the shadow more confidently. For LtEVM and LtMSM the error can be resolved using an additional temporal depth test, by only removing the ray if the camera depth from the last frame is similar to the current depth.

3.1.3. Combined Algorithm

Combining LTT with the temporal mask test forms our leak tracing (**Lt**) algorithm. An overview of the algorithm looks as follows:

1. Evaluate the filterable shadow map ($\hat{s}(\hat{z}(t), d)$).
2. Use LTT (Equation 3) to initially check if a ray is needed.
3. If temporal re-projection fails, force a shadow ray.
4. *Expand Rays:* If LTT evaluates **false**, check Equation 6 on all adjacent neighbors and set the LTT result to true if any neighbor evaluated true.
5. *Remove Rays (optional):* If LTT evaluates **true** in step 2, check Equation 8 on all adjacent neighbors and set the LTT result to false if all neighbors evaluated true. Note the special cases mentioned in Section 3.1.2 where RR should not be used.

Following these steps, our leak tracing algorithm gets rid of the

artifacts of filterable shadow maps and recreates a shadow similar to ray tracing, as seen in Figure 5d. Step 5 is optional and not necessary for artifact-free shadows, as it is only a performance optimization.

3.2. Memory Optimizations

Our experiments show that a shadow map filter kernel of the size 2×2 is sufficient to identify where a ray is needed. Therefore, a default shadow map with linear depth($b(z) := z$) can be used to replicate a filterable shadow map. When evaluating the shadow map, a gather texture operation (i.e. 2×2 filter kernel) can be used to fetch the depths. These depths are converted to the filterable shadow map format (e.g. $z \rightarrow \{z, z^2\}$ for VSM) and then interpolated bilinearly. This decreases the memory bandwidth and increases performance for texture fetches (see Figure 8a), without visual differences to the pre-filtered counterpart.

3.3. Cascaded Shadow Mapping

Rendering the depth map multiple times per frame can be expensive. With rasterization, reusing the shadow map for slight movements can lead to noticeable jumps at the shadow edge. However, as our leak tracing calculates the correct edge, reusing the (static) cascaded shadow map is possible without any noticeable changes on the shadow edge.

When creating a shadow map for a cascaded level, we increase the cascaded frustum by a percentage (e.g. 15%), which allows reusing the shadow map on slight movements. For every frame, we check for each level if all edges for the cascaded frustum from the current frame are within the (increased) cascaded frustum from the last frame. If it is outside the bounds we increase the bounds of the current cascaded frustum by the percentage and render the cascaded shadow map level. If it is inside, we can reuse the shadow map from the last frame cascaded level.

This drastically reduces the performance impact cascaded shadow maps would have on our leak tracing algorithm, as the shadow map can be reused most of the time and only needs to be re-rendered on strong movements. Possible jumps in the shadow, when the cascaded shadow map level is replaced, are compensated by the second part of our expand ray expression (Equation 6).

For optimal performance, the light direction or position should only be updated sparingly to reuse the cascaded level as much as possible. Additionally, stronger changes in light position or direction are currently not supported, as the temporal mask only compensates camera and geometry movements.

3.4. Dynamic Geometry

Reusing a shadow map is only possible for static geometry. Therefore, we use a separate shadow map for static and dynamic geometry. The static shadow map is reused if possible and the dynamic shadow map is rendered for every frame. For the filterable shadow test, both shadow maps are sampled. However, because the dynamic geometry is rendered on top of an empty shadow map, pre-filtering is not possible, as it would blend the dynamic shadow edges with the clear value. To still get the filter effect, we sample the static and dynamic shadow map using a gather operation. We take the minimum of both the static and dynamic shadow maps and manually interpolate the values. Bigger kernels would also be possible, using the same principle. Additionally, we store if a static gather sample was replaced by a dynamic one to disable the remove ray expression in this case.

4. Results

We implemented our leak tracing algorithm in the Falcor framework [KCK*22]. The tests were performed on an NVIDIA RTX 2080 SUPER with a screen resolution of 1920x1080 on a range of different scenes. The used scenes are:

- **Bistro** [Lum17]: The Amazon Lumberyard Bistro with $2.8m$ tris. Objects with foliage are present all over the scene and it has a handful of animated wires with light bulbs.
- **EmeraldSquare** [NHB17]: A small city scene with a park that has many trees with alpha-tested leaves. The scene has $10m$ tris.
- **MedievalCity**: A low poly medieval city with around $1.2k$ objects, of which 208 are low poly trees with a total of $723k$ tris.
- **Forest**: A small forest path with a high poly animated bike ($1.9m$ tris) and an animated character walking through the forest. It contains many instanced low poly trees and grasses by Nicholas-3D [ND23]. The scene has a total of $4m$ tris.

The settings used for each test scene are the same, except for the cascaded split range, which is adjusted for each scene. We always used 4 cascaded levels, where the shadow of levels 0 to 2 is determined using our hybrid algorithm with a 2048^2 shadow map while the last level is fully ray traced. The shadow map format is always 32-bit float per channel. We use a fixed min variance of $1e^{-10}$ for VSM and EVSM and a moment bias of $3e^{-6}$ for MSM, which worked well for all tested scenes. We use $c = 80$ for ESM and $c_{pos} = 20$ and $c_{neg} = 5$ for EVSM. Additionally, we used an 8-sample camera jitter (DirectX standard pattern) and DLSS 3.1 for antialiasing. The BRDF is also evaluated in the same pass as the shadow. We added the abbreviation *Lt* before the shadow map technique to signal that our algorithm is used.

To test if our parameters are independent of the scene extent, we additionally tested other scenes with very different scales. Furthermore, we scaled all our test scenes by 10^{-3} and 10^3. Our algorithm performed well and always reproduced the ray traced shadow, regardless of the scale, as long as the cascaded level splits were reasonably chosen.

First, we compare how well the different types of filterable shadow maps work with our leak trace algorithm in Section 4.1. Then we test against full ray traced shadows in Section 4.2 and show the limitations of our technique in Section 4.3.

4.1. Filterable Shadow Map types

We examined our algorithm with the upper-bound filterable shadow maps introduced in Section 2.1. Quality-wise, ESM and MSM often result in a poor approximation on steep shadow edges with a shadow value near 1. Therefore, ESM and MSM need an additional temporal depth test to compensate for this error.

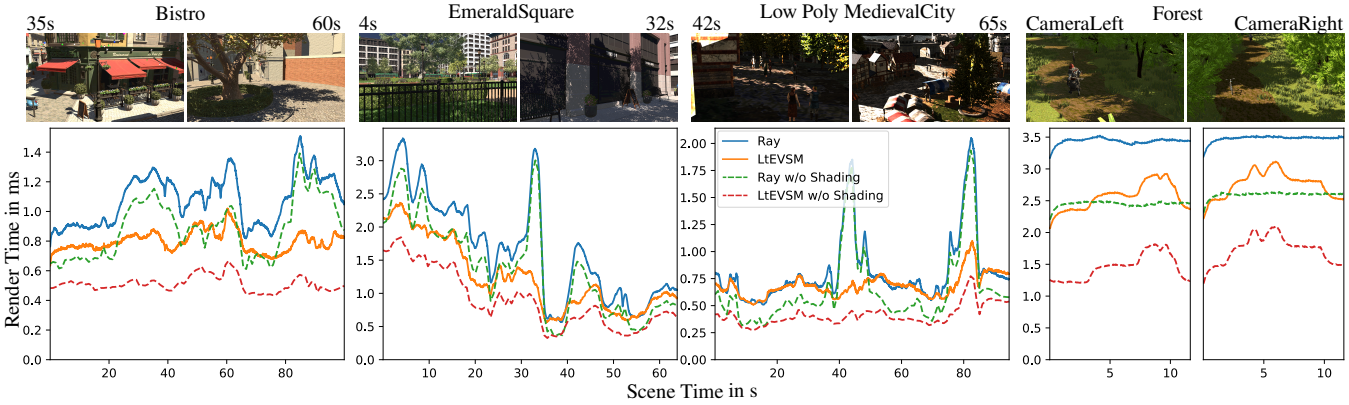

Figure 7: *Performance comparison between ray traced shadows and our LtEVSM. The times on the y-axis respond to the scene time for the camera paths. The corresponding videos are provided in the Supplemental Materials.*

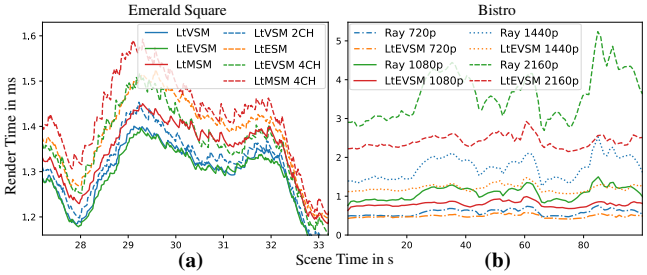

Figure 8: ***(a)*** *Performance comparison of the different filterable shadow map versions of the camera path in the Emerald Square scene. The filterable shadow maps without channels use the memory optimizations (Section 3.2).* ***(b)*** *Comparison of LtEVSM and ray shadows with different screen resolutions in the Bistro scene. After each method, we added the screen pixel height with a* 16 : 9 *screen pixel ratio.*

	Bistro		EmSq		MedCity		Forest	
Shading		w/o		w/o		w/o		w/o
Avg.	1.36	1.74	1.34	1.55	1.19	1.57	1.30	1.61
Min.	1.12	1.21	0.91	0.90	0.92	1.01	1.12	1.24
Max.	1.76	2.62	2.88	3.90	2.71	4.17	1.52	2.04

Table 2: *Speedups for scenes from Figure 7. The average, minimum and maximum with and without shading are shown.*

Figure 8a shows a performance comparison of our algorithm with the different upper-bound filterable shadow maps. The memory optimization from Section 3.2 increases the performance by a constant factor, as less memory bandwidth is used. The cost for the 4 channel shadow maps is reduced by around ~ 0.1ms on average and around ~ 0.05ms on average for LtVSM. Our algorithm with VSM and EVSM has the best performance, with EVSM being slightly faster. Due to the more confident approximation of the inner shadow part, fewer rays need to be distributed. LtESM often needs rays when the shadow and the sender are close to each other, resulting in many unnecessary rays that cannot be safely removed. LtMSM has an unsure LTT test around smaller occluders (e.g. see Figure 6b), resulting in a less confident approximation and more rays as LtVSM or LtEVSM.

After evaluating the different filterable shadow maps, we came to the conclusion that LtVSM and LtEVSM are the best filterable shadow map candidates for our algorithms. LtESM is slower and shows errors, which is why we would advise against using ESM as the filterable shadow map for our algorithm. MSM is a possible candidate, but the performance is worse than LtVSM and LtEVSM due to the worse shadow approximation and the additional temporal depth test ($\sim 0.04ms$) that is needed.

4.2. Performance

In Figure 7 we compare LtEVSM, our best-performing variant, to ray traced shadows. The graphs show the render time for the shading of the scene with one directional light. We also included the render time without evaluating the BRDF. Additionally, the speedups for each scene are shown in Table 2.

Overall, our algorithm has a 1.19 to 1.36 speedup on average with shading and an average speedup from 1.55 to 1.74 without evaluating the BRDF. This is probably due to the latency-hiding capabilities of the GPU. While the ray is in flight, the GPU can perform some independent calculations, which effectively hides the cost of dispatching a ray to some degree. The graph also shows, that our leak tracing is able to drastically reduce the peaks caused by alpha-tested rays. This can lead to huge speedups in those areas ranging from 1.76 to 4.17.

However, our algorithm can sometimes be slower than ray tracing, which happens in the EmeraldSquare and MedievalCity scene, where the lowest speedup was 0.9. Our algorithm can get slower when the static shadow map needs to be constantly updated, due to camera rotations, and the visible shadows are cast by simple objects. In these cases, the cost of updating the shadow maps outweighs the benefit gained from not tracing the shadow rays in the core of the shadow. As this happens only on stronger camera rotations, these cases are more the exception than the rule. Additionally,

	SM	VSM	MSM	Ray	Lt	Lt w/o RR	Lt w/o Reuse	LTT only	HySM [AMD21]
Avg.	2.54	2.79	3.39	1.67	1.31	1.38	2.85	1.03	1.31
Min.	1.29	1.57	2.18	0.56	0.58	0.59	1.34	0.49	0.58
Max.	3.58	3.81	4.40	3.33	2.41	2.60	4.52	1.91	2.39

Table 3: *Render times in ms for the camera path in Emerald Square with different shadow techniques and settings for our Leak Tracing.*

we do not use simplified geometry for the latter cascaded levels in our research prototype.

Our leak tracing can even perform well for scenes with an extensive amount of dynamic triangles if there are enough alpha tests, as in our Forest scene. A bike with 2 million triangles is moving through the scene, drastically increasing the render time of the dynamic shadow map. This is also visible in the graph (Figure 7), as the render time for our LtEVSM peaks when the bike needs to be rasterized for all cascaded levels.

In Table 3, we compare the runtime of various shadow techniques and leak tracing with some optimizations disabled. Default shadow maps and filterable shadow maps have worse render times than ray tracing if all cascaded levels are re-generated every frame. The shadow evaluation time was around 0.5 ms for all three shadow map techniques. Similarly, without our reuse optimization (Lt w/o Reuse), our leak tracing is significantly slower than ray tracing. With reuse optimization, the average shadow map generation time is 0.01 ms, and without it, the time is 1.53 ms, demonstrating the importance of reuse optimization. The ray removal optimization provides a modest performance gain (avg. 1.05 speedup). Disabling the temporal mask entirely (LTT only) results in an average speedup of 1.27. This is mainly because LTT does not correctly detect all shadows within trees, whereas the temporal mask forces expensive alpha-rays in these regions. Lastly, we replaced the LTT test with AMD's FidelityFX hybrid shadows [AMD21] with 2x2 PCF while using our reuse optimization and temporal mask. The render times are identical to leak tracing; however, a shadow bias must be carefully set to avoid dispatching many unnecessary rays. Without the temporal mask, the artifacts produced by hybrid shadows are slightly worse than using LTT alone.

4.2.1. Performance for Different Resolutions

Figure 8b we compared the performance of our LtEVSM with ray traced shadows with different screen resolutions. Both methods use shading for this comparison. The graph shows, that with increasing screen size, our algorithm scales better, as relatively fewer rays need to be distributed. For a resolution of $1280x720$, the average speedup is 1.21, with a minimum speedup of 1.0. With a resolution of $2560x1440$, we reach an average speedup of 1.44 with a min of 1.13 and a max of 1.97. For the resolution $3840x2160$ we achieve an average speedup of 1.5, with a max of 2.06 and a min of 1.17. Therefore, we can conclude that our algorithm scales better than ray tracing shadows with increasing screen size.

We also experimented with varying shadow map resolutions. In the Bistro scene, we achieved an average speedup of 1.39 with a shadow map resolution of 512^2, 1.37 with 1024^2 and 1.24 with 4096^2. However, the performance difference between the different shadow map resolutions is mostly because of shadow map rendering. In some scenes, lower resolutions can negatively impact performance, as seen in Figure 9. Also, lower resolution shadow maps can lead to some problems (see Section 4.3), which were present in the Bistro scene with 512^2 and 1024^2 shadow maps.

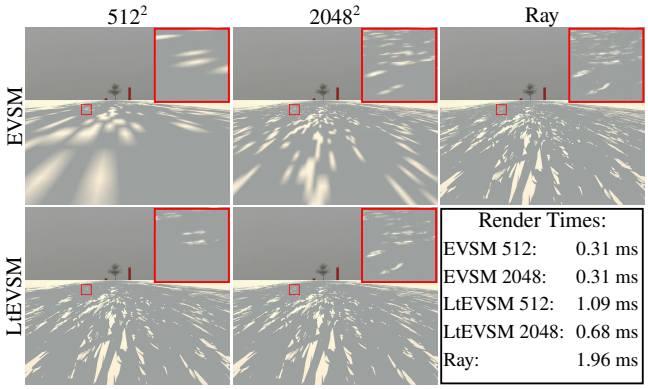

Figure 9: *A faraway tree that is lit by a directional light with a grazing angle while the camera is moving. Even with poor shadow map resolution, our leak tracing is able to approximate the ray traced shadow. However, some details are lost due to the limited resolution and the temporal depth test needs to be enabled to prevent disocclusion artifacts.*

4.3. Limitations

Our leak tracing algorithm is able to remove all the artifacts from filterable shadow maps. However, in some situations, it can fail to correctly replicate the ray traced shadow.

It can happen that very small lit regions (∼1-2 pixel in screen space) are either not found or closed by ray removal. This happens when the lit region does not appear in the shadow map due to its resolution or when our temporal mask closes this lit pixel. With insufficient shadow map resolution, these lit holes can pop into existence when the cascaded shadow map level is refreshed. An example would be the LtEVSM with a 512^2 shadow map from Figure 9. Similarly, very small shadow casters can be missed when they are not rasterized in the shadow map. In this case, leak tracing will propagate all shadow it recognizes, but can fail to replicate the ray traced shadow if the shadow map information is missing and there is no connected ray traced shadow pixel, as shown in Figure 10. However, as long as the shadow map resolution is sufficient for the scene, these errors are not noticeable and can only be found if directly comparing the ray shadow to our hybrid shadow.

The temporal mask in our leak tracing algorithm does not accommodate moving light sources, as it only compensates for camera and geometry movement using camera motion vectors. However, leak tracing still functions with small changes in the position or direction of the light (e.g. sun), as the base approximation from LTT is effective in most cases, and the temporal mask remains valid for most pixels. Fast-moving lights that require per-frame shadow map updates are likely to be slower than ray-traced shadows, except

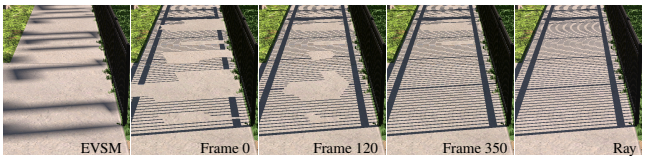

Figure 10: *A worst-case scenario, where temporal maks convergence is shown without any previous history. A fence in Emerald Square with a 64^2 EVSM (left) is shown. While most of the fence is slowly closed within 350 frames, a hole in the top remains that could not be closed as the information in the shadow map is missing and the ray shadow pixels are too far apart.*

for very alpha-heavy scenes. To reduce the frequency of shadow map generation for slower-moving lights, our algorithm could be combined with Parallax-Corrected Cached Shadow Maps [Tur19] or other similar techniques that allow the reuse of shadow maps with slightly different light directions.

5. Discussion

We presented a simple algorithm to generate pixel-perfect shadows that closely resemble ray-traced shadows by distributing rays in error-prone areas of filterable shadow maps. We achieved an average speedup of 1.19 to 1.74 across various test scenes, with speedups of up to 4.17 for alpha-tested shadows, where ray tracing notoriously struggles.

The algorithm can be used on top of default shadow maps that often are already used in games and does not need any scene dependent parameters. The leak tracing shadow test (Equation 3) provides a good approximation of the shadow with minimal error cases occurring with near-parallel incoming light. Our proposed temporal mask addresses the remaining issues, requiring only 3 bits per light, making it very memory-efficient and allowing for support of a larger number of lights. The static part of the shadow map can be reused over multiple frames and only needs sparse updates, which significantly reduces shadow map generation time. Leak tracing works with dynamic geometry and slow light movements (e.g. sun).

Comparison to existing work: In addition to ray-traced shadows, our work shares similarities with previous work on pixel-perfect shadows. View-dependent shadow map techniques such as Adaptive Shadow Maps [FFBG01, LSK*05], Resolution Matched Shadow Maps [LSO07], temporal reprojected shadow maps [SJW07], methods based on the irregular z-buffer [AL04, JLBM05, Arv07, SEA08, WHL15] and per-triangle shadow volumes [SOA11, SKOA14] all require generating the shadow evaluation structure every frame, which can be expensive for scenes with higher triangle counts, especially with more complex shadow structures. As shown in Section 4.2 and Table 3, even default cascaded shadow maps are slower than ray tracing if the shadow map needs to be generated every frame. Additionally, some techniques rely on conservative rasterization, which scales very poorly with larger scenes. Our Leak Tracing can reuse the shadow map over multiple frames and uses a default shadow map as a shadow evaluation structure.

Work based on shadow maps, such as Adaptive Shadow Maps, Resolution Matched Shadow Maps, Shadow Silhouette Maps [SCH03], temporal reprojected shadow maps and hybrid PCF shadow maps [AMD21] all require setting a scene-dependent depth bias to avoid self-shadowing. Since our Leak Tracing is based on filterable shadow maps, we do not need to set any scene-dependent parameters. Additionally, Shadow Silhouette Maps can exhibit noticeable zig-zag artifacts.

Adaptive Shadow Maps and temporal reprojected shadow maps do not handle dynamic scenes well, while our algorithm can handle dynamic objects and slow-moving lights.

6. Conclusion and Future Work

We introduced Leak Tracing, a simple algorithm that approximates a ray traced shadow using a shadow map. Our algorithm is easy to implement into existing systems and significantly speeds up ray traced shadows on alpha-test heavy scenes.

For future work, it would be interesting to examine the scaling of our algorithm with more than one light source, we imagine that our algorithm will scale very well with an increased amount of lights. Up to 42 lights would currently be possible with our temporal mask as one RGBA32 texture. Due to the extent of a real light source, perfect hard shadows do not exist. Therefore, it would be interesting to adjust our algorithm to support soft shadows. The inner part of the shadow could still be produced by a shadow map, while the penumbra region could be determined by ray tracing.

Supplemental Materials

Full source code for our implementation, an intractable demo and videos are available at: https://github.com/TU-Clausthal-Rendering/LeakTracing

Acknowledgements

This work was supported by the German Research Foundation (DFG) grant GR 3833/4-1, Project Nr. 524961573: Optimal Combination of Ray Tracing and Rasterization (R^2)

References

[AL04] AILA T., LAINE S.: Alias-Free Shadow Maps. In *Eurographics Symposium on Rendering* (2004), EGSR'04, The Eurographics Association. doi:10.2312/EGWR/EGSR04/161-166. 2, 9

[AMB*07] ANNEN T., MERTENS T., BEKAERT P., SEIDEL H.-P., KAUTZ J.: Convolution shadow maps. In *Eurographics Symposium on Rendering* (2007), EGSR'07, Eurographics Association, p. 51–60. doi:10.2312/EGWR/EGSR07/051-060. 2

[AMD21] AMD: FidelityFX Hybrid Shadows, 2021. Part of the FidelityFX framework. URL: https://gpuopen.com/fidelityfx-hybrid-shadows/. 2, 8, 9

[AMS*08] ANNEN T., MERTENS T., SEIDEL H.-P., FLERACKERS E., KAUTZ J.: Exponential shadow maps. In *Proceedings of Graphics Interface* (2008), GI '08, CIPS, p. 155–161. 2, 3

[Arv07] ARVO J.: Alias-free shadow maps using graphics hardware. *Journal of Graphics Tools 12*, 1 (2007), 47–59. doi:10.1080/2151237X.2007.10129231. 2, 9

[Cho21] CHOI J.: Hybrid shadows, 2021. Presentation, in *NVIDIA GPU Technology Conference* (GTC) 2021. URL: https://www.nvidia.com/en-us/on-demand/session/gtcspring21-e32638/. 3

[Cro77] CROW F. C.: Shadow algorithms for computer graphics. In *Proceedings of the 4th Annual Conference on Computer Graphics and Interactive Techniques* (1977), SIGGRAPH '77, ACM, p. 242–248. doi:10.1145/563858.563901. 2

[DL06] DONNELLY W., LAURITZEN A.: Variance shadow maps. In *Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games* (2006), I3D '06, ACM, p. 161–165. doi:10.1145/1111411.1111440. 1, 2, 3

[Eng06] ENGEL W.: Cascaded shadow maps. In *Shader X5: Advanced Rendering Techniques* (2006), Charles River Media, Inc., pp. 197–206. 1, 2

[Fer05] FERNANDO R.: Percentage-closer soft shadows. In *ACM SIGGRAPH 2005 Sketches* (2005), SIGGRAPH '05, ACM, p. 35–es. doi:10.1145/1187112.1187153. 3

[FFBG01] FERNANDO R., FERNANDEZ S., BALA K., GREENBERG D. P.: Adaptive shadow maps. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques* (2001), SIGGRAPH '01, ACM, p. 387–390. doi:10.1145/383259.383302. 2, 9

[JLBM05] JOHNSON G. S., LEE J., BURNS C. A., MARK W. R.: The irregular z-buffer: Hardware acceleration for irregular data structures. *ACM Trans. Graph. 24*, 4 (2005), 1462–1482. doi:10.1145/1095878.1095889. 2, 9

[KCK*22] KALLWEIT S., CLARBERG P., KOLB C., DAVIDOVIČ T., YAO K.-H., FOLEY T., HE Y., WU L., CHEN L., AKENINE-MÖLLER T., WYMAN C., CRASSIN C., BENTY N.: The Falcor rendering framework, 2022. URL: https://github.com/NVIDIAGameWorks/Falcor. 6

[Lau07] LAURITZEN A.: Summed-area variance shadow maps. In *GPU Gems 3: Chapter 8* (2007), Addison-Wesley Professional. URL: https://developer.nvidia.com/gpugems/gpugems3/part-ii-light-and-shadows/chapter-8-summed-area-variance-shadow-maps. 2

[LM08] LAURITZEN A., MCCOOL M.: Layered variance shadow maps. In *Proceedings of Graphics Interface* (2008), GI '08, CIPS, p. 139–146. 2, 3

[LSK*05] LEFOHN A., SENGUPTA S., KNISS J., STRZODKA R., OWENS J. D.: Dynamic adaptive shadow maps on graphics hardware. In *ACM SIGGRAPH 2005 Sketches* (2005), SIGGRAPH '05, ACM, p. 13–es. doi:10.1145/1187112.1187126. 2, 9

[LSO07] LEFOHN A. E., SENGUPTA S., OWENS J. D.: Resolution-matched shadow maps. *ACM Trans. Graph. 26*, 4 (2007), 20–es. doi:10.1145/1289603.1289611. 2, 9

[Lum17] LUMBERYARD A.: Amazon lumberyard bistro, open research content archive (ORCA), 2017. URL: http://developer.nvidia.com/orca/amazon-lumberyard-bistro. 1, 6

[MT04] MARTIN T., TAN T.-S.: Anti-aliasing and continuity with trapezoidal shadow maps. In *Eurographics Symposium on Rendering* (2004), EGSR'04, Eurographics Association, p. 153–160. 1, 2

[ND23] NICHOLAS-3D: Low poly tree scene free, 2023. URL: https://sketchfab.com/Nicholas01. 6

[NHB17] NICHOLAS HULL K. A., BENTY N.: Nvidia emerald square, open research content archive (ORCA), 2017. URL: http://developer.nvidia.com/orca/nvidia-emerald-square. 1, 6

[OSK*14] OLSSON O., SINTORN E., KÄMPE V., BILLETER M., ASSARSSON U.: Efficient virtual shadow maps for many lights. In *Proceedings of the 18th Meeting of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (2014), I3D '14, ACM, p. 87–96. doi:10.1145/2556700.2556701. 1, 2

[PK15] PETERS C., KLEIN R.: Moment shadow mapping. In *Proceedings of the 19th ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (2015), i3D '15, ACM, pp. 7–14. doi:10.1145/2699276.2699277. 1, 2, 3

[RSC87] REEVES W. T., SALESIN D. H., COOK R. L.: Rendering antialiased shadows with depth maps. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques* (1987), SIGGRAPH '87, ACM, p. 283–291. doi:10.1145/37401.37435. 1, 2, 3

[Sal08] SALVI M.: Rendering filtered shadows with exponential shadow maps. In *ShaderX6: Advanced Rendering Techniques* (2008), Charles River Media, p. 257–274. 2, 3

[SCH03] SEN P., CAMMARANO M., HANRAHAN P.: Shadow silhouette maps. In *ACM SIGGRAPH 2003 Papers* (2003), SIGGRAPH '03, ACM, p. 521–526. doi:10.1145/1201775.882301. 2, 9

[SD02] STAMMINGER M., DRETTAKIS G.: Perspective shadow maps. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques* (2002), SIGGRAPH '02, ACM, p. 557–562. doi:10.1145/566570.566616. 1, 2

[SEA08] SINTORN E., EISEMANN E., ASSARSSON U.: Sample Based Visibility for Soft Shadows using Alias-free Shadow Maps. *Computer Graphics Forum* (2008). doi:10.1111/j.1467-8659.2008.01267.x. 2, 9

[SJW07] SCHERZER D., JESCHKE S., WIMMER M.: Pixel-Correct Shadow Maps with Temporal Reprojection and Shadow Test Confidence. In *Eurographics Symposium on Rendering* (2007), EGSR'07, The Eurographics Association. doi:10.2312/EGWR/EGSR07/045-050. 2, 9

[SKOA14] SINTORN E., KÄMPE V., OLSSON O., ASSARSSON U.: Per-triangle shadow volumes using a view-sample cluster hierarchy. In *Proceedings of the 18th Meeting of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (2014), I3D '14, ACM, p. 111–118. doi:10.1145/2556700.2556716. 2, 9

[SOA11] SINTORN E., OLSSON O., ASSARSSON U.: An efficient alias-free shadow algorithm for opaque and transparent objects using per-triangle shadow volumes. In *Proceedings of the 2011 SIGGRAPH Asia Conference* (2011), SA '11, ACM. doi:10.1145/2024156.2024187. 2, 9

[Sto15] STORY J.: Hybrid ray-traced shadows, 2015. Presentation, in *Game Developers Conference* (GDC) 2015. URL: https://developer.download.nvidia.com/assets/events/GDC15/hybrid_ray_traced_GDC_2015.pdf. 2

[Tur19] TURCHYN P.: Parallax-corrected cached shadow maps. In *GPU Zen 2: Advanced Rendering Techniques*, Engel W., (Ed.). Black Cat Publishing, 2019, pp. 143–152. 9

[WHL15] WYMAN C., HOETZLEIN R., LEFOHN A.: Frustum-traced raster shadows: revisiting irregular z-buffers. In *Proceedings of the 19th Symposium on Interactive 3D Graphics and Games* (2015), i3D '15, ACM, p. 15–23. doi:10.1145/2699276.2699280. 2, 9

[Wil78] WILLIAMS L.: Casting curved shadows on curved surfaces. *SIGGRAPH Comput. Graph. 12*, 3 (1978), 270–274. doi:10.1145/965139.807402. 1, 2, 3

[WSP04] WIMMER M., SCHERZER D., PURGATHOFER W.: Light space perspective shadow maps. In *Proceedings of the Fifteenth Eurographics Conference on Rendering Techniques* (2004), EGSR'04, Eurographics Association, p. 143–151. doi:10.2312/EGWR/EGSR04/143-151. 1, 2