Vision, Modeling, and Visualization (2025)
B. Egger and T. Günther (Eds.)

Alias-Free Shadows with Ray Cones for Alpha Tested Geometry

Felix Brüll Rene Kern Thorsten Grosch

TU Clausthal, Germany

(a) *Mip 0 Shadows (2.74 ms)* **(b)** *(Our) Ray Cones (3.43 ms)* **(c)** *(Our) Saturated Ray Cones (2.37 ms)*

Figure 1: *Pixel-perfect shadows of a tree in the Emerald Square scene [NB17]. Sampling ray-traced shadows from mipmap level 0 (a) results in pixelated shadows that flicker under camera motion. Using our ray cone method (b) produces stable shadows, but incurs additional performance cost. Our saturated shadows (c) are stable under motion and competitive in performance.*

Abstract

We present a method for computing alias-free, smooth shadows for alpha-tested geometry using a single ray per pixel. Without mipmap filtering, hard shadows from alpha-tested geometry appear very noisy under camera motion. Typically, many ray samples are required to soften the shadows and reduce noise. We propose using mipmaps instead, to achieve a fast and temporally stable solution. To determine the appropriate mipmap level, we introduce novel ray cone operations that account for directional and point light sources.

CCS Concepts

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

1. Introduction

The introduction of ray tracing hardware has enabled the use of pixel-perfect shadows in real-time applications. For analytical lights, only a single ray per pixel is required to obtain a perfect shadow. However, a small change in camera position can cause the shadow to flicker significantly, especially when the scene geometry contains many thin objects. This issue is particularly severe for alpha-tested geometry, where the shadow is determined by the alpha value of a high-resolution texture.

We describe how to prevent noisy shadows from alpha-tested geometry by extending the ray cone method [ACB*21] for shadow rays. In particular, we contribute:

- The **Ortho** operation, which removes the angular spread for directional lights (Sec. 3.1)
- The **Focus** operation, which focuses the ray cone on a point light source (Sec. 3.2)
- A saturated visibility accumulation method, which allows shadow rays to terminate early (Sec. 3.3)

DOI: 10.2312/vmv.20251226

2. Previous Work

2.1. Shadows

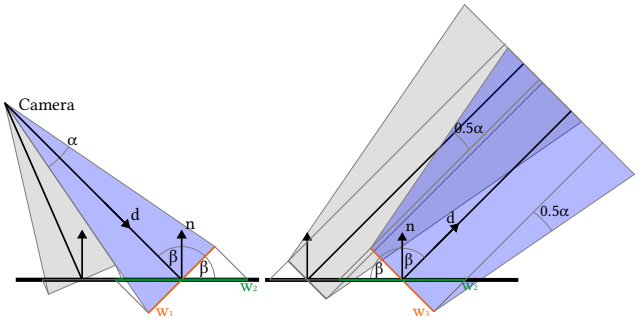
Shadows significantly enhance the perceived realism of rendered images. Shadow Mapping [Wil78] is a widely used technique for computing shadows. A shadow map is a depth texture that is generated from the perspective of the light source in a separate rendering pass. However, shadow maps are prone to aliasing and self-shadowing artifacts (commonly known as shadow acne), due to the limited resolution of the depth texture.

Aliasing artifacts can be mitigated by filtering the shadow. While the shadow map itself cannot be directly filtered, the result of the shadow test can be smoothed by repeating the depth comparison for nearby shadow map pixels using percentage-closer filtering (PCF) [RSC87]. This technique requires multiple texture lookups, which can be computationally expensive. Filterable shadow maps address this by allowing prefiltering and hardware-supported sampling through alternative, filterable depth representations. Several methods exist to support such filtering, including Fourier series expansions [AMB*07], exponential functions [AMS*08; Sal08], variance-based methods using Chebyshev's inequality [DL06; LM08], and moment-based techniques [PK15; PMWK16]. While these methods reduce aliasing and self-shadowing, they suffer from light leaking.

Another strategy to reduce aliasing is to adapt the light projection to better match the camera's view. For a comprehensive review of this class of techniques, we refer the reader to the Real-Time Shadows book [ESAW11].

With the introduction of hardware-accelerated ray tracing, shadow rays have become feasible in real-time rendering. Ray-traced shadows offer a pixel-perfect alternative to shadow maps. A shadow is computed by casting a ray from the surface point toward the light source. If the ray intersects any geometry along its path, the point is considered shadowed. For alpha-tested materials, an additional texture lookup is required to determine whether the intersection point is opaque or transparent. We will use shadow rays instead of shadow maps in this paper.

Pixel-perfect shadows are also possible without ray tracing. Fernando et al. [FFBG01] proposed iterative refinement focusing on visually important areas such as shadow edges. Lefohn et al. [LSO07] improved this by directly estimating the required shadow map resolution per node, removing the need for iterative refinement. Shadow Silhouette Maps [SCH03] reconstruct shadow edges using an auxiliary silhouette map. CPU-based irregular grid shadow maps have been explored to achieve pixel-accurate results [AL04; JLBM05]. Wyman et al. [WHL15] demonstrated a GPU implementation of irregular grid shadow maps, while Sintorn et al. [SKOA14] applied a similar idea using shadow volumes [Cro77]. Scherzer et al. [SJW07] achieved pixel-perfect shadows by temporally accumulating shadow test results using jittered shadow maps. Kern et al. [KBG24] combined filterable shadow maps with selective hardware ray tracing to efficiently generate pixel-perfect shadows without light leaking.



(a) *Propagate + Hit* **(b)** *Reflect + Propagate*

Figure 2: ***(a)*** *The initial cone* $(0,\alpha)$ *is propagated to* (w_1,α)*. After the hit operation we get* (w_2,α)*.* ***(b)*** *The ray cone* (w_2,α) *is adjusted for the reflection to* (w_3,α)*. The figure shows the cone after a potential propagation step. The gray cone depicts the cone of a neighboring pixel.*

2.2. Ray Cones

For rasterization, the correct mipmap level for a texture is usually determined by the screen-space footprint of the triangle. This can be computed at runtime using the *ddx* and *ddy* intrinsic functions in the shader. However, *ddx* and *ddy* are not available for ray tracing, and a differential ray must be tracked manually. The two most popular approaches are ray differentials [Ige99] and ray cones [ACB*21]. Ray differentials are more accurate but require a significant amount of memory. Ray cones are more approximate but have a smaller memory footprint. Since we are interested in footprints for secondary shadow rays, where precise estimation is less critical, we use ray cones in this work.

A ray cone (w,α) is defined by its cone width w and spread angle α. For the initial ray, the ray cone width w_0 is initialized to zero, and the spread angle α_0 is computed as:

$$\alpha_0 = \arctan\left(\frac{2\tan(\psi/2)}{H}\right) \tag{1}$$

where ψ is the vertical field of view and H is the image height in pixels. This effectively computes the spread angle for a pixel in the center of the image, but it can also be used for all the other pixels for simplicity.

When a ray intersects a surface, the ray cone needs to be updated:

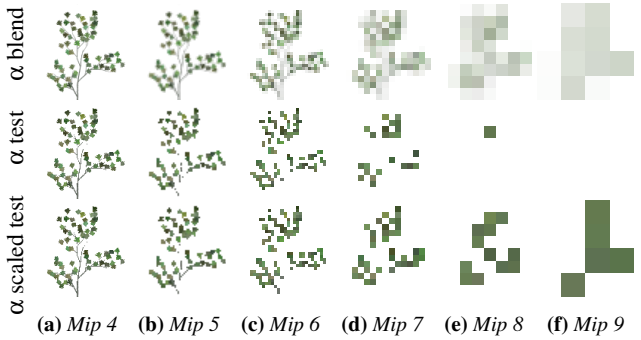
$$\text{Propagate}(w,\alpha,t) = (w+2t\tan(\alpha/2),\alpha) \approx (w+t\alpha,\alpha) \tag{2}$$

The `Propagate` operation updates the cone width based on the traveled distance t, as shown in Fig. 2. The approximation holds for small angles α, which is generally the case for camera rays. To transfer the ray cone to a non-perpendicular surface, the hit operation is used:

$$\text{Hit}(w,\alpha,n,d) = (w/|\text{dot}(n,d)|,\alpha) \tag{3}$$

Here, n is the surface normal and d is the ray direction. Finally, to reflect a ray cone, the reflect operation is used (Fig. 2b):

$$\text{Reflect}(w,\alpha,n,d) = (w\cdot|\text{dot}(n,d)|,\alpha) \tag{4}$$



(a) *Mip 4* **(b)** *Mip 5* **(c)** *Mip 6* **(d)** *Mip 7* **(e)** *Mip 8* **(f)** *Mip 9*

Figure 3: *Mipmaps of an oak leaf from the emerald square. The alpha blended reference is on top, followed by alpha testing on default mipmaps. The bottom row shows the results of alpha testing after performing proper alpha scaling on the mipmaps [NVI20].*

It is also possible to adjust the spread angle α based on the curvature of the surface, but this is not necessary for our use case. A method to handle refractions is described by Boksansky et al. [BCA21].

The texture **LOD (level of detail)** for a triangle is computed as:

$$\mathrm{LOD} = \log_2\left(w\sqrt{t_a/p_a}\right) \tag{5}$$

$$t_a = T_w T_h \left|(t_{1x}-t_{0x})(t_{2y}-t_{0y}) - (t_{2x}-t_{0x})(t_{1y}-t_{0y})\right| \tag{6}$$

$$p_a = ||(P_1-P_0)\times(P_2-P_0)|| \tag{7}$$

Here, w is the ray cone width after the `Hit` operation. t_a and p_a are twice the area of the triangle in texture space and world space, respectively. T_w and T_h are the texture's width and height in pixels. t_{ix} and t_{iy} are the texture coordinates of the i-th triangle vertex, and P_i is the world-space position of the i-th vertex.

2.3. Texture LOD Generation

Calculating proper mipmaps for alpha-tested textures differs slightly from the standard approach. When the alpha channel is simply averaged during mipmap computation, the percentage of pixels that pass the alpha test ($\alpha > 0.5$) is generally not preserved, as shown in Fig. 3 (middle row). This is a well-known problem and is described in detail by Izdebski [Izd20].

The issue can be circumvented by scaling the alpha channel after mipmap generation, such that the percentage of pixels passing the alpha test is preserved (see Fig. 3, bottom row). This technique is implemented in the NVIDIA Texture Tools [NVI20] as an iterative algorithm that scales all alpha values by a factor f until the percentage of pixels passing the alpha test in the current mipmap closely matches the percentage in Mip 0. After each iteration, if the percentage of passing pixels is too high, f is slightly decreased; if it is too low, f is slightly increased.

2.4. Antialiasing

Rendering images with a single sample per pixel results in jagged, pixelated edges. Rendering with multiple samples per pixel and averaging the results produces a smoother image but is computationally expensive. Multi-sample anti-aliasing (MSAA) can be used in rasterization to maintain a shading cost of about one sample per pixel, but it is not applicable to deferred shading or ray tracing.

Fast Approximate Anti-Aliasing (FXAA) [Lot11] and Subpixel Morphological Anti-Aliasing (SMAA) [JESG12] are post-processing techniques used to smooth the edges of a rendered image afterwards. They detect edges in the image and then apply a blur, which reduces aliasing. However, since they operate on single images, they do not account for temporal coherence, resulting in temporal aliasing artifacts such as flickering between frames.

Temporal Anti-Aliasing (TAA) [YLS20] addresses this by using the previous frame to smooth the current one, reducing aliasing both within and across frames. While effective to some extent, TAA leads to a blurrier image and may introduce ghosting artifacts, particularly during rapid camera movement. DLSS [NVI25] is a more recent technique that leverages a neural network to improve upon TAA, but it still suffers from similar issues to some degree.

Some applications cannot afford the extra cost of post-processing anti-aliasing. In other cases, a sharper image may be preferred by the art director or user. For these reasons, we want to focus on alias-free shadows without relying on post-processing techniques.

3. Our Method

Our goal is to produce alias-free shadows that eliminate temporal noise under camera motion. To achieve this, we use ray cones to estimate an approximate footprint for ray traced shadow rays.

Ray cones for shadow rays behave differently than those for camera rays. Directional lights remove the angular spread, while point lights focus the ray cone on the light source. We describe how to compute the appropriate LOD for both cases.

- Sec. 3.1 describes the implementation for directional lights.
- Sec. 3.2 describes the implementation for point lights.
- Sec. 3.3 explains how to obtain the shadow visibility.
- Sec. 3.4 covers additional implementation details.
- Sec. 4 presents the results of our method.

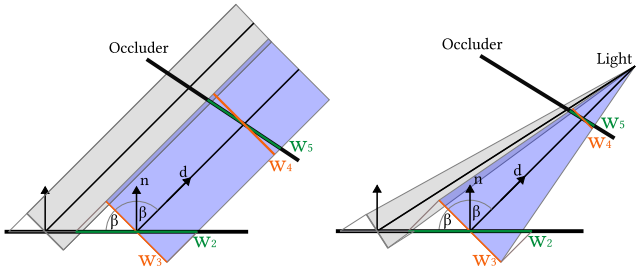
3.1. Directional Lights

The rays from a directional light source are assumed to be parallel. Therefore, when calculating the LOD for an occluder between the light source and the shaded point, the angular spread of the ray cone should be zero:

$$\mathrm{Ortho}(w,\alpha) = (w,0) \tag{8}$$

This is illustrated in Fig. 4a. The steps to calculate the LOD are as follows:

1. Initialize the ray cone $(0,\alpha)$.
2. **Propagate** the ray cone to (w_1,α).
3. **Hit** the shading surface and update the ray cone to (w_2,α).
4. **Reflect** the ray cone towards the light direction (w_3,α).
5. **Orthogonalize** the ray cone to $(w_3,0)$.
6. **Propagate** the ray cone to $(w_4,0)$.
7. **Hit** the occluder and update the ray cone to $(w_5,0)$.
8. Calculate the texture LOD using $w := w_5$ with Eq. 5.



(a) *Reflect+Ortho+(Propagate)+Hit* **(b)** *Reflect+Focus+Propagate+Hit*

Figure 4: ***(a)*** *The ray cone* (w_2, α) *is adjusted for the orthographic reflection to* $(w_3, 0)$. ***(b)*** *The ray cone* (w_2, α) *is adjusted to focus on a point light. In both cases,* w_4 *is the ray cone width after propagation and* w_5 *for the final hit. Furthermore,* w_5 *will be used for the LOD calculation (Eq. 5).*

3.2. Point Lights

Calculating the LOD from the perspective of a point light is similar to calculating it from the camera perspective, since all rays share a common origin. However, in our case, we start from the point to be shaded and move toward the light source. Therefore, the angular spread must be adjusted so that the ray cone width w converges to zero at the light source:

$$\text{Focus}(w, \alpha, t) = (w, 2\arctan\left(\frac{-w}{2t}\right)) \approx (w, \frac{-w}{t}) \tag{9}$$

Here, t is the distance to the light source. A subsequent call of $\text{Propagate}(w, \alpha, t)$ will then result in a ray cone width of zero. This is illustrated in Fig. 4b. The steps to calculate the LOD are as follows:

1. Initialize the ray cone $(0, \alpha)$.
2. **Propagate** the ray cone to (w_1, α).
3. **Hit** the shading surface and update the ray cone to (w_2, α).
4. **Reflect** the ray cone toward the light direction (w_3, α).
5. **Focus** the ray cone to (w_3, α').
6. **Propagate** the ray cone to (w_4, α').
7. **Hit** the occluder and update the ray cone to (w_5, α').
8. Calculate the texture LOD using $w := w_5$ with Eq. 5.

3.3. Texture Sampling

After calculating the corresponding texture LOD using either Sec. 3.1 or Sec. 3.2, the opacity texture of the occluder can be sampled. The extracted opacity value α is then used to determine the shadow contribution. Depending on how the alpha value is interpreted, the shadow ray may need to be continued or can be terminated early. We present results for the following three cases, each with different benefits and drawbacks:

Raw: The raw α value is used without modification. The remaining visibility of the shadow ray is then $(1-\alpha)$, and the ray continues until it is either terminated or the accumulated visibility falls below a small threshold (e.g., 0.01). The main drawback of this method is that it produces overly soft shadows. This occurs because the alpha test used in geometry shading treats texels with $\alpha > 0.5$ as fully opaque. In contrast, using the raw alpha value causes the same texel to appear partially transparent (e.g., 50% transparent at $\alpha = 0.5$), leading to significantly softer shadows, especially when sampling from higher mipmap levels.

Alpha Test: The extracted α value is converted to a binary visibility by testing it against a threshold of 0.5. If the texel is classified as opaque, the visibility is set to 0, and the shadow ray can be terminated immediately. The problem with this approach is, that shadows will be hard shadows, and only slightly less noisy than the texels from mipmap 0.

Saturate: The extracted α value is saturated to $\alpha' = \min(2\alpha, 1)$. This approach is more compatible with alpha-tested geometry, as texels with $\alpha > 0.5$ are classified as fully opaque. As a result, the visibility decreases more rapidly, allowing the ray to terminate earlier than in the *Raw* case.

3.4. Implementation Details

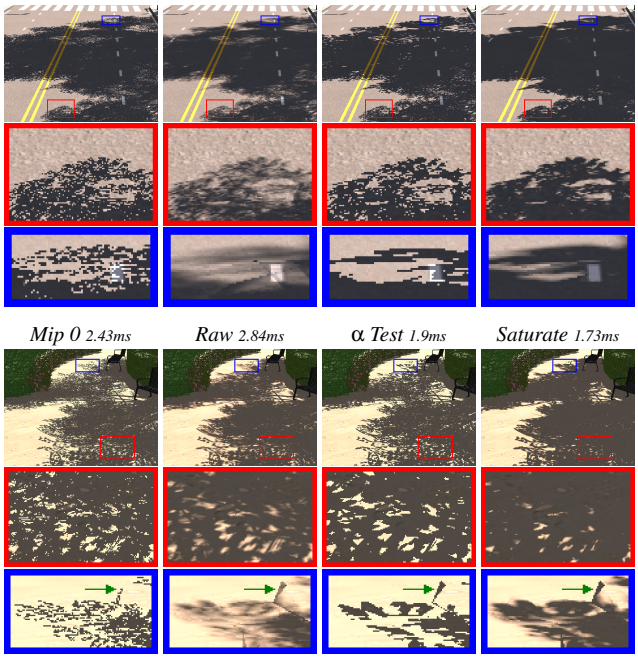
We implemented the algorithm in Falcor [KCK*22]. The implementation stores the position and normal of the first opaque hit in a G-buffer. A separate full-screen pass then computes the required shadow ray for a given light source and uses inline ray tracing (ray queries) to detect occluders. If an occluder is alpha-tested, we calculate the LOD as described above and sample the texture at the corresponding level of detail. The resulting texture α value is then converted to visibility as described in Sec. 3.3. We accumulate the product $(1-\alpha)$ from all occluders to obtain the final ray visibility value. If the accumulated visibility falls below 0.01, we terminate the shadow ray early and assume the pixel is fully shadowed. The full source code can be found on the project page: `https://github.com/TU-Clausthal-Rendering/AliasfreeShadowRayCones`

4. Results

We evaluated our direct lighting implementation in the Emerald Square scene [NB17]. For the point light case, we created a Cornell box containing a point light and a tree with alpha-tested leaves [Spe17]. Renderings were recorded at a resolution of 1920×1080 on an NVIDIA GeForce RTX 2080 Ti using one sample per pixel.

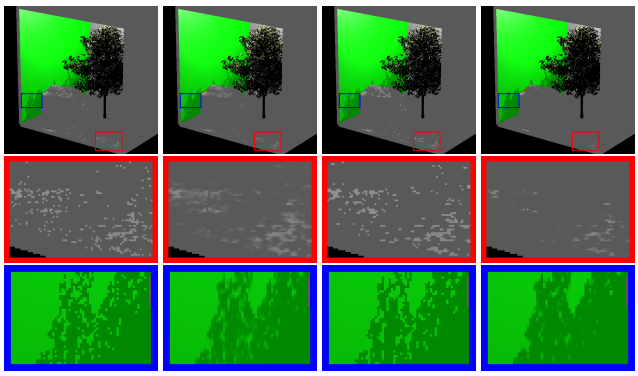
Fig. 5 shows the results for a directional light in the Emerald Square scene. Sampling the alpha test from mipmap level 0 (a) is very slow and produces noisy shadows under camera motion. Using the raw alpha value from the corresponding texture LOD (b) is even slower, but the shadows are free from noise. Applying the alpha test (c) to texture LODs is slightly faster, but shadows remain noisy with camera motion due to the discrete nature of the alpha test. The saturate operation (d) is the fastest option, and the shadows are not noisy. However, this method results in noticeably thicker shadows. The performance improvement is due to two factors: First, sampling from higher texture LODs is generally faster, as it leverages the texture cache more efficiently. Second, the saturation operation allows the shadow ray to terminate earlier.

A specific artifact of texture LOD sampling can be seen in the bottom row of the Emerald Square scene (Fig. 5, green arrow). A triangle-shaped shadow appears in the top right of our results. This occurs because the occluder in that region is nearly perpendicular



(a) *Mip 0 3.75ms* **(b)** *Raw 4.08ms* **(c)** α *Test 3.14ms* **(d)** *Saturate 2.81ms*

Figure 5: *Shadows from directional light in the emerald square, evaluated without mipmaps (a) and with our method (b-d).*



(a) *Mip 0 0.36ms* **(b)** *Raw 0.53ms* **(c)** α *Test 0.42ms* **(d)** *Saturate 0.35ms*

Figure 6: *Shadows from a point light in a cornell box, evaluated without mipmaps (a) and with our method (b-d).*

to the light direction, leading to a large estimated footprint due to the *Hit* operation. As a result, the highest LOD is sampled, which contains a single opaque texel.

Fig. 6 shows the result for a point light in the Cornell box scene. The visual observations are similar to those in the directional light case. However, using texture LODs does not provide a significant performance benefit in this scene, as it only has a single texture. Nevertheless, shadows remain stable during camera motion when using our raw or saturated method.

To demonstrate the temporal stability of our method, we recorded a video in the Emerald Square scene, which is included in the supplementary material. The video shows that naive mipmap level 0 sampling and our alpha test method result in highly noisy shadows, while the raw and saturated methods produce shadows that are stable under motion. Our raw method occasionally exhibits minor flickering within alpha-tested shadows, but this is not caused by texture LOD sampling. Instead, it stems from small opaque branches modeled with actual geometry rather than textures, and thus falls outside the scope of what our method can address. This issue is less noticeable in the saturated method, as the thicker shadows tend to mask most of the small branches.

We also include a second supplementary video that evaluates how existing anti-aliasing methods perform with naive mipmap level 0 shadows. As discussed in Sec. 2.4, SMAA fails to produce a temporally stable image, even when using a variant that incorporates one temporal frame (T2X). TAA offers better temporal stability but results in a noticeably blurrier image. Additionally, the TAA implementation in Falcor is unable to fully eliminate temporal flickering due to the high contrast between lit and shadowed regions. Finally, DLSS performs best among the tested methods, effectively removing most temporal flickering.

Combining our method with TAA or DLSS can completely eliminate temporal flickering in alpha-tested shadows, while also accelerating shadow sampling through the saturated approach. However, note that DLSS in itself is generally sufficient to remove aliasing artifacts in the final image, and thus our method generally does not provide any visual benefit in this case.

5. Conclusion

We presented a method for computing alias-free, temporally stable ray-traced shadows for alpha-tested geometry using a single shadow ray per pixel. By extending ray cones with two novel operators — **Ortho** and **Focus** — we account for the behavior of shadow rays towards directional and point light sources. Our method estimates an appropriate texture LOD for each occluder hit, enabling the use of mipmaps to smooth aliasing which prevents flickering under camera motion. We further demonstrated that saturating alpha values during visibility accumulation leads to stable and more performant shadows.

Limitations. While our approach produces alias-free and stable shadows, it is limited to alpha-tested geometry and does not address geometric aliasing. The use of ray cones can overestimate the footprint at grazing angles, occasionally resulting in artifacts such as blurry shadows or visible triangles (Fig. 5, green arrow).

Future Work. Our method could be extended to support anisotropic ray cones, which would more accurately model stretched footprints. Although we evaluated the technique only for alpha-tested occluders, estimating footprints for alpha-blended occluders should be equally feasible, wherein the *raw* method should be used instead of *saturate*. Finally, ray cones could be extended to support area lights or other complex light types, as our current implementation only handles point and directional lights.

References

[ACB*21] AKENINE-MÖLLER, TOMAS, CRASSIN, CYRIL, BOKSANSKY, JAKUB, et al. "Improved Shader and Texture Level of Detail Using Ray Cones". *Journal of Computer Graphics Techniques (JCGT)* 10.1 (Jan. 2021), 1–24. ISSN: 2331-7418. URL: http://jcgt.org/published/0010/01/01/ 1, 2.

[AL04] AILA, TIMO and LAINE, SAMULI. "Alias-Free Shadow Maps". *Eurographics Symposium on Rendering*. EGSR'04. The Eurographics Association, 2004. ISBN: 3-905673-12-6. DOI: 10.2312/EGWR/EGSR04/161-166 2.

[AMB*07] ANNEN, THOMAS, MERTENS, TOM, BEKAERT, PHILIPPE, et al. "Convolution shadow maps". *Eurographics Symposium on Rendering*. EGSR'07. Grenoble, France: Eurographics Association, 2007, 51–60. ISBN: 9783905673524. DOI: 10.2312/EGWR/EGSR07/051-060 2.

[AMS*08] ANNEN, THOMAS, MERTENS, TOM, SEIDEL, HANS-PETER, et al. "Exponential shadow maps". *Proceedings of Graphics Interface*. GI '08. Windsor, Ontario, Canada: CIPS, 2008, 155–161. ISBN: 9781568814230 2.

[BCA21] BOKSANSKY, JAKUB, CRASSIN, CYRIL, and AKENINE-MÖLLER, TOMAS. "Refraction Ray Cones for Texture Level of Detail". *Ray Tracing Gems II: Next Generation Real-Time Rendering with DXR, Vulkan, and OptiX* (2021), 115–125 3.

[Cro77] CROW, FRANKLIN C. "Shadow algorithms for computer graphics". *Proceedings of the 4th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '77. San Jose, California: ACM, 1977, 242–248. ISBN: 9781450373555. DOI: 10.1145/563858.563901 2.

[DL06] DONNELLY, WILLIAM and LAURITZEN, ANDREW. "Variance shadow maps". *I3D 2006*. Redwood City, California: ACM, 2006, 161–165. ISBN: 159593295X. DOI: 10.1145/1111411.1111440 2.

[ESAW11] EISEMANN, ELMAR, SCHWARZ, MICHAEL, ASSARSSON, ULF, and WIMMER, MICHAEL. *Real-Time Shadows*. 1st. USA: A. K. Peters, Ltd., 2011. ISBN: 1568814380 2.

[FFBG01] FERNANDO, RANDIMA, FERNANDEZ, SEBASTIAN, BALA, KAVITA, and GREENBERG, DONALD P. "Adaptive shadow maps". *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '01. ACM, 2001, 387–390. ISBN: 158113374X. DOI: 10.1145/383259.383302 2.

[Ige99] IGEHY, HOMAN. "Tracing ray differentials". *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*. 1999, 179–186 2.

[Izd20] IZDEBSKI, LUKASZ. *Improving the quality of the alpha test (cutout) materials*. https://www.asawicki.info/articles/alpha_test.php5. Accessed on: 10 Jan. 2024. 2020 3.

[JESG12] JIMENEZ, JORGE, ECHEVARRIA, JOSE I., SOUSA, TIAGO, and GUTIERREZ, DIEGO. "SMAA: Enhanced Subpixel Morphological Antialiasing". *CGF* 31.2pt1 (May 2012), 355–364. ISSN: 0167-7055. DOI: 10.1111/j.1467-8659.2012.03014.x 3.

[JLBM05] JOHNSON, GREGORY S., LEE, JUHYUN, BURNS, CHRISTOPHER A., and MARK, WILLIAM R. "The irregular Z-buffer: Hardware acceleration for irregular data structures". *ACM Trans. Graph.* 24.4 (2005), 1462–1482. ISSN: 0730-0301. DOI: 10.1145/1095878.1095889 2.

[KBG24] KERN, RENÉ, BRÜLL, FELIX, and GROSCH, THORSTEN. "Real-Time Pixel-Perfect Hard Shadows with Leak Tracing". *Eurographics Symposium on Rendering*. The Eurographics Association, 2024. ISBN: 978-3-03868-262-2. DOI: 10.2312/sr.20241158 2.

[KCK*22] KALLWEIT, SIMON, CLARBERG, PETRIK, KOLB, CRAIG, et al. *The Falcor Rendering Framework*. Jan. 2022. URL: https://github.com/NVIDIAGameWorks/Falcor 4.

[LM08] LAURITZEN, ANDREW and MCCOOL, MICHAEL. "Layered variance shadow maps". *Proceedings of Graphics Interface*. GI '08. Windsor, Ontario, Canada: CIPS, 2008, 139–146. ISBN: 9781568814230 2.

[Lot11] LOTTES, TIMOTHY. *FXAA*. https://developer.download.nvidia.com/assets/gamedev/files/sdk/11/FXAA_WhitePaper.pdf. Whitepaper. 2011 3.

[LSO07] LEFOHN, AARON E., SENGUPTA, SHUBHABRATA, and OWENS, JOHN D. "Resolution-matched shadow maps". *ACM Trans. Graph.* 26.4 (2007), 20–es. ISSN: 0730-0301. DOI: 10.1145/1289603.1289611 2.

[NB17] NICHOLAS HULL, KATE ANDERSON and BENTY, NIR. *NVIDIA Emerald Square, Open Research Content Archive (ORCA)*. July 2017. URL: http://developer.nvidia.com/orca/nvidia-emerald-square 1, 4.

[NVI20] NVIDIA. *NVIDIA Texture Tools: FloatImage::scaleAlphaToCoverage*. https://github.com/castano/nvidia-texture-tools. Accessed on: 10 Jan. 2024. 2020 3.

[NVI25] NVIDIA. *Deep Learning Super Sampling*. 2025. URL: https://developer.nvidia.com/rtx/dlss 3.

[PK15] PETERS, CHRISTOPH and KLEIN, REINHARD. "Moment Shadow Mapping". *I3D 2015*. San Francisco, California: ACM, 2015, 7–14. ISBN: 978-1-4503-3392-4. DOI: 10.1145/2699276.2699277 2.

[PMWK16] PETERS, CHRISTOPH, MÜNSTERMANN, CEDRICK, WETZSTEIN, NICO, and KLEIN, REINHARD. "Beyond Hard Shadows: Moment Shadow Maps for Single Scattering, Soft Shadows and Translucent Occluders". *I3D*. ACM, 2016, 159–170. ISBN: 978-1-4503-4043-4/16/03. DOI: 10.1145/2856400.2856402 2.

[RSC87] REEVES, WILLIAM T., SALESIN, DAVID H., and COOK, ROBERT L. "Rendering antialiased shadows with depth maps". *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '87. ACM, 1987, 283–291. ISBN: 0897912276. DOI: 10.1145/37401.37435 2.

[Sal08] SALVI, MARKO. "Rendering filtered shadows with exponential shadow maps". *ShaderX6: Advanced Rendering Techniques*. Charles River Media, 2008, 257–274. ISBN: 978-1584505440 2.

[SCH03] SEN, PRADEEP, CAMMARANO, MIKE, and HANRAHAN, PAT. "Shadow silhouette maps". *SIGGRAPH*. San Diego, California: ACM, 2003, 521–526. ISBN: 1581137095. DOI: 10.1145/1201775.882301 2.

[SJW07] SCHERZER, DANIEL, JESCHKE, STEFAN, and WIMMER, MICHAEL. "Pixel-Correct Shadow Maps with Temporal Reprojection and Shadow Test Confidence". *Eurographics Symposium on Rendering*. EGSR'07. The Eurographics Association, 2007. ISBN: 978-3-905673-52-4. DOI: 10.2312/EGWR/EGSR07/045-050 2.

[SKOA14] SINTORN, ERIK, KÄMPE, VIKTOR, OLSSON, OLA, and ASSARSSON, ULF. "Per-triangle shadow volumes using a view-sample cluster hierarchy". *I3D 2014*. San Francisco, California: ACM, 2014, 111–118. ISBN: 9781450327176. DOI: 10.1145/2556700.2556716 2.

[Spe17] SPEEDTREE. *SpeedTree, Open Research Content Archive (ORCA)*. July 2017. URL: http://developer.nvidia.com/orca/speedtree 4.

[WHL15] WYMAN, CHRIS, HOETZLEIN, RAMA, and LEFOHN, AARON. "Frustum-traced raster shadows: revisiting irregular z-buffers". *I3D 2015*. San Francisco, California: ACM, 2015, 15–23. ISBN: 9781450333924. DOI: 10.1145/2699276.2699280 2.

[Wil78] WILLIAMS, LANCE. "Casting curved shadows on curved surfaces". *SIGGRAPH Comput. Graph.* 12.3 (1978), 270–274. ISSN: 0097-8930. DOI: 10.1145/965139.807402 2.

[YLS20] YANG, LEI, LIU, SHIQIU, and SALVI, MARCO. "A Survey of Temporal Antialiasing Techniques". *Computer Graphics Forum* 39.2 (July 2020), 607–621. DOI: 10.1111/cgf.14018 3.