High-Performance Graphics (2019)
T. Foley and M. Steinberger (Editors)

Short Paper

A Practical and Efficient Approach for Correct Z-Pass Stencil Shadow Volumes

B.Usta [†], L. Scandolo, M. Billeter, R. Marroquim and E. Eisemann

Delft University of Technology, The Netherlands

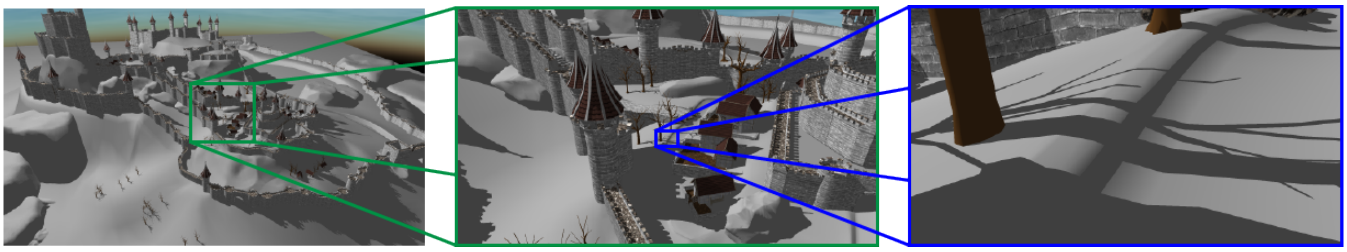


Figure 1: *Pixel-perfect hard shadows produced by our method in the Citadel scene from different viewpoints.*

Abstract

Shadow volumes are a popular technique to compute pixel-accurate hard shadows in 3D scenes. Many variants exist that trade off accuracy and efficiency. In this work, we present an artifact-free, efficient, and easy-to-implement stencil shadow volume method. We compare our method to established stencil shadow volume techniques and show that it outperforms the alternatives.

CCS Concepts

• ***Computing methodologies*** → ***Rendering;*** *Visibility;*

1. Introduction

Realistic shadows can provide crucial visual cues for understanding spatial relationships in a scene. Although many different methods for artifact-free shadows have been proposed, their pixel-accurate computation at real-time rates, even for point lights, remains a challenge. The majority of shadow-generation algorithms fall into one of four categories: shadow maps [Wil78], shadow volumes [Cro77], irregular Z-Buffer solutions [JLBM05, WHL15] and ray-tracing [App68]. Although shadow maps are very efficient, they suffer from difficult-to-avoid aliasing. The irregular Z-Buffer, while having led to efficient shadow computations [SEA08], still requires a significant overhead for larger screen resolutions. Real-time ray-tracing is still limited in availability and performance on current commodity hardware. Shadow volumes thus remain a portable and proven option for generating pixel-accurate hard shadows, yet many of its variants are prone to artifacts. For example, the original Z-Pass method [Hei91] fails when the observer is located in shadow. More recent solutions try to address this problem but introduce significant costs or are complex to implement. We present a faster artifact-free robust version of the ++ZP algorithm [ESAW11] that is easy to implement. After reviewing previous works (Sec. 2), we will describe this method (Sec. 3), then report results and comparisons (Sec. 4) before concluding (Sec. 5).

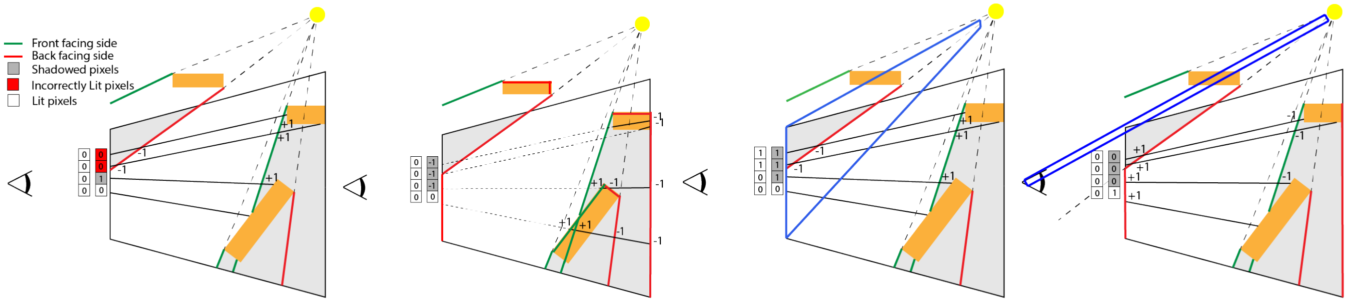
2. Related Work

Generating shadows in graphics applications is an important but difficult task, having a large body of research devoted to it. Eisemann et al. [ESAW11, EAS*13] present an in-depth overview of techniques for interactive and real-time applications. Here, we will focus on shadow-volume techniques [Cro77], which remain a popular option due to their pixel-accurate results (Fig. 1).

A shadow volume tightly encloses the space that lies in shadow with respect to a specific point light source. The boundary of a shadow volume can be found by extending the edges of the input geometry away from the light source. A common approach for determining shadows using shadow volumes relies on the stencil

† {b.usta,l.scandolo,m.j.billeter,r.marroquim,e.eisemann}@tudelft.nl

DOI: 10.2312/hpg.20191195



(a) *Z-Pass* **(b)** *Z-Fail* **(c)** *ZP+* **(d)** *++ZP*

Figure 2: *An illustration of the different existing shadow volume techniques. All methods count the amount of shadow volume boundaries traversed in a given path. Z-Pass counts from the near plane to the surface point. Z-Fail accounts for shadow boundaries from the surface point to a point infinitely far away. ZP+ computes traversed shadow boundaries from the light to the surface point via the near plane. ++ZP also counts boundaries from the light source to the surface point, but it does so via the camera position.*

buffer [Hei91]. This buffer is used to count the number of times a ray from the camera enters and leaves the volumes on its way to a surface point.

Methods relying on the stencil buffer [Hei91, Car00, EK02, HHLH05] draw (potentially) many shadow-volume boundaries, which can cause significant overdraw - one of the main drawbacks of shadow volumes. Limiting the number of planes by only constructing them for silhouette edges [AMA03, Yaz06, ZB11] is essential for these methods.

Overdraw can be further reduced by eliminating redundant shadow planes using an acceleration structure over the geometry [CBCJ99]. This approach was recently extended to avoid the stencil buffer completely [GMAG15, MGAG16]. The acceleration structure may alternatively be built over the pixel data [SOA11, SKOA14], where shadow planes are rasterized to an irregular buffer that combines writes to large regions to reduce the cost of overdraw. Per-triangle shadow volumes relax requirements such as avoiding the need for closed geometry. Nevertheless, these methods rely heavily on non-trivial compute operations on the GPU, including generation and updates of acceleration structures and software rasterization.

In this work, we revisit and build upon the ++ZP method proposed by Eisemann et al. [ESAW11]. Similar to other stencil-based approaches, our method utilizes the standard rendering pipeline together with just a custom geometry shader. Since shadow volumes are extruded from silhouette edges, the geometry needs to be closed (with few exceptions).

2.1. Z Pass

The Z-Pass algorithm [Hei91] is the first hardware-accelerated shadow-volume implementation [Cro77]. It works by using a screen-sized stencil buffer as a per-pixel counter of the amount of traversed shadow boundaries from the camera to the first visible surface. A first render pass of the scene geometry is performed to fill a depth buffer. The shadow volumes are then rendered into the stencil buffer, incrementing the stencil value by one when drawing a front-facing shadow boundary fragment and decrementing it by one for each back-facing one. Using the initialized depth buffer, only shadow-volume polygons in front of the first visible surface are counted. After drawing all shadow volumes, a pixel with a zero counter is considered lit, otherwise in shadow, as zero means that the ray from the camera to the visible surface point entered as many shadows as it exited.

This algorithm produces pixel-perfect hard shadows and it avoids counting non-visible shadow boundaries by using the depth test. Nevertheless, when the camera is inside a shadow volume, the results are invalid. The resulting misclassification of pixels can be seen in Fig. 2a, where red pixels are incorrectly considered lit.

2.2. Z-Fail

Z-Fail [Car00, EK02] is a similar but robust alternative of the Z-Pass algorithm. Instead of counting the amount of shadow boundaries from the camera position to the first visible surface, it counts from the first visible surface to a point infinitely far away along the same line of sight (see Fig. 2b). The principle is essentially equivalent to the Z-Pass algorithm, but assumes instead that the camera is at this infinitely far away point. The volumes are capped at the far plane to ensure that the far point is outside of any shadow volume. Thanks to this, it sidesteps the main failure case of the original algorithm.

The main drawback of Z-Fail is that it generates a higher overdraw than the Z-Pass algorithm, since it requires rendering all shadow volumes up to an infinitely far-away point, as well as the shadow volumes' front and back caps.

2.3. ZP+

The ZP+ algorithm [HHLH05] aims to address the failure case of the Z-Pass algorithm by initializing the stencil buffer with a value that ensures the correct result. It performs an initial render pass from the point of view of the light, matching its far plane to the camera near plane (blue frustum in Fig. 2c), and using the same stencil buffer which is used for the Z-Pass algorithm. During this pass, only the original light-facing geometry is counted. Combined with the Z-Pass, this effectively amounts to counting the intersections of a ray from the light to the near plane and then to the first visible surface point. As the light source itself cannot be in its own shadow, this counting process will be correct. While this algorithm retains most of the advantages of Z-Pass, it requires an extra render pass and a careful match of the pixel locations during the two different stencil render passes, which makes the implementation quite complex.

2.4. ++ZP

The ++ZP algorithm [ESAW11] takes a similar approach to ZP+ by creating a two-segment path from the light to the visible fragment. In the case of ++ZP, the path goes via the camera position instead of the near plane. A single 1x1 stencil buffer orthographic render pass is used to count the amount of shadow blockers from the light position to the camera position (depicted in blue in Fig. 2d). This resulting value is used as the reference during the shading pass to determine whether a pixel is shadowed. The full-screen stencil pass is performed next, which is similar to Z-Pass, with the exception that shadow volume fragments between the camera and the near plane are clamped to the near plane, which can be efficiently achieved with the corresponding OpenGL extension. The implementation is straightforward and handles the limitations of Z-Pass. Our method is inspired by ++ZP and addresses its main shortcoming: the need for an extra geometry pass.

3. AtomicZP

The main drawback of ++ZP is the additional render pass to find the number of shadow blockers between light and camera. Although this pass computes a single value, the scene geometry must be read and processed. We propose to skip this render step, and instead compute this information during the geometry stage of the screen-size forward stencil pass.

As part of the forward stencil pass, we utilize a geometry shader to identify and extrude silhouette edges for light-facing triangles using adjacency information. During this step, for each light-facing triangle, we additionally perform a fast ray-triangle intersection [MT97] test to determine if it lies between light source and camera. If so, we increment a global atomic counter (Fig. 3), which implicitly represents the one-pixel buffer of the ++ZP method. This eliminates the need for an extra geometry pass.

The resulting atomic counter can be retrieved before the final shading pass and used to initialize the stencil buffer, similar to the ZP+ method, or simply to set the reference value for the stencil test, similar to our implementation of the ++ZP method. Depending on the capabilities of the graphics library and specific hardware, this step may require reading back the value from the GPU to main memory (the read-back may be avoided using ARB_shader_stencil_export, however our setup did not support this OpenGL extension). However, other work can be scheduled while waiting for this read-back result, such that no stall occurs. Alternatively, it is even possible to skip this step altogether and emulate the stencil test during final shading by reading the fragment's stencil value, comparing it against the value of the atomic counter, and discarding the fragment if the values do not match.

Timings in Section 4 use the last approach, which proved most efficient. In our implementation, reading back the atomic counter

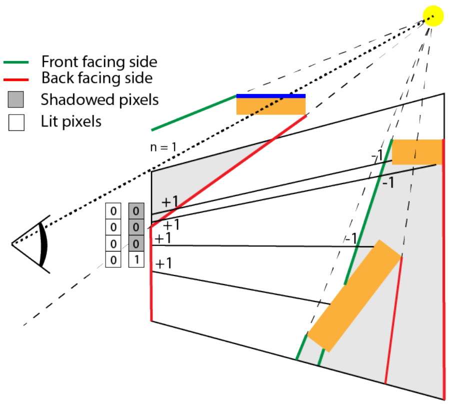


Figure 3: *AtomicZP: First, a Z-Pass-like operation updates the stencil values depending on the facing of the generated shadow volume quads. Similar to ++ZP, the pass uses depth clamping. When extruding the shadow volume quads, a check determines if a light facing triangle lies between the light source and the camera's origin (dotted line). Each such triangle (shown in blue) increments a counter (n). Fragments are lit (white pixel) if their stencil value equals the value of the counter (i.e., $n = 1$ in this example), and shadowed otherwise (gray pixel).*

value introduces a stall in rendering, since no other operations are scheduled, increasing total frame time slightly (about 500μs). Time for the final shading pass (excluding the read-back) is however the same as for the rest of the previously described methods. As already mentioned, if the stall can be avoided by scheduling other operations between the read-back and the final shading step, using this approach may be advantageous as it can utilize early-out stencil tests if such are supported.

4. Results and Discussion

We performed our experiments on a NVIDIA GTX 1080TI GPU, running under Microsoft Windows 10 on an Intel i7-8700 CPU with 16GB of RAM. All methods have two main steps: stencil buffer update by rendering the shadow volumes, and final scene render using said stencil buffer. In addition, for ZP+ and ++ZP there is an extra stencil buffer render pass. We have selected three scenes for our experiments:

- Buddha: simple scene with one model and two planes (543k vertices, 1M triangles)
- Sponza: indoor scene with exterior light source where the camera is often in shadow and facing the light source(153k vertices, 262k triangles)
- Citadel: outdoor scene with high depth complexity (310k vertices, 613k triangles)

For each scene, we have predetermined a camera path, and executed each algorithm 10 times. We store the median execution time for each frame in order to eliminate most variation due to other concurrent processes. Some variation is, nevertheless, still noticeable in the results, especially for slower methods that might be more prone to interruption.

In all cases the final shading step times are nearly identical, since all methods perform practically the same work after the shadow volumes have been rendered. Furthermore, we use a simple Phong shading model for the experiments. In a more realistic application, with more complex shading models, any differences would become even more negligible as the shading computations would dominate the total time. Therefore, we have focused our analysis on the more relevant step for this work, that is, evaluating the stencil buffer.

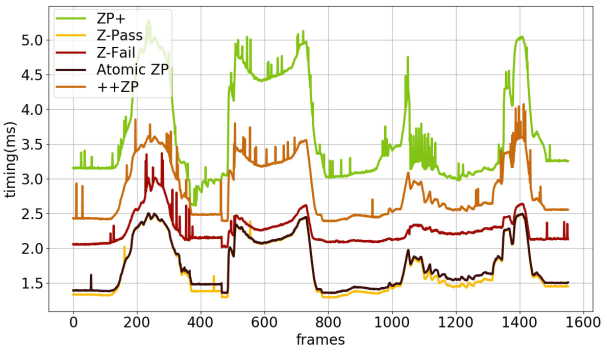
Fig. 4 shows timings for the shadow-volume creation step for the three test scenes. For ZP+ and ++ZP these timings also include the stencil-buffer initialization. The two fastest algorithms have practically the same performance, namely, Z-Pass and AtomicZP. Yet, our method produces the correct result for all cases, while Z-Pass is not artifact free, as discussed in Section 2.1.

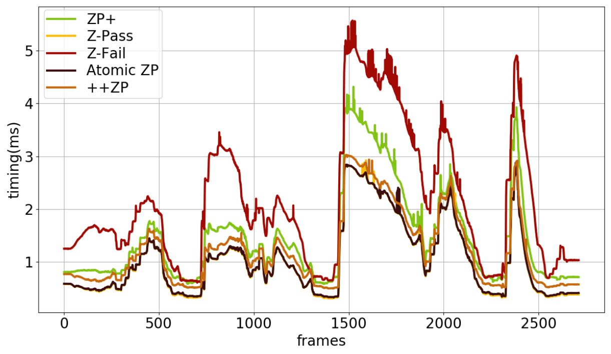
Our method outperforms other artifact-free methods consistently. The difference to the next best method, ++ZP, is as expected primarily the time required to count the number of blockers between light and camera. This time is quite stable across the runs in our tests (around 190μs for the Sponza scene when measured in isolation), as the complexity depends only on the amount of geometry, given that the render target has resolution 1×1. Nevertheless, in geometry-bound applications, this difference may grow larger. ZP+ occasionally shows a larger performance difference, which is likely due to increased fill-rate requirements in some views.

Apart from its superior performance, our method has the additional advantage of being easier to implement, since we do not have to handle special cases and avoid an extra geometry pass, unlike ZP+ and ++ZP.

A minor limitation of stencil-based shadow volume methods relates to the stencil buffer's 8-bit limit. This is a common problem for stencil-buffer applications, as the buffer can overflow. Using the extension to wrap the buffer values solves many issues but when passing from 255 to 0, the resulting zero will indicate that the region is lit, while the ray towards it actually traversed 256 shadow boundaries. This is an extremely rare case, and in practice, we noticed no artifacts due to buffer overflow. In cases, where correctness is imperative, the stencil buffer can be replaced by a 16-bit texture and the stencil test can be performed manually at the fragment level.

During our analysis, we found that shadow-volume quads for which both extruded edges extend to infinity behind the camera frustum were carrying a significant performance penalty. We avoid this case by manually clipping these quads against the plane perpendicular to the view direction and passing through the camera position. We perform this in all relevant cases (our method, Z-Pass, Z-Fail, and ++ZP). Furthermore, in our experiments enabling depth clamping for Z-Pass slightly increased performance, while only affecting failure cases, so it was enabled during our tests. Both of these observations might be specific to the OpenGL implementation of our test system. In this context, an improved interface that allows setting the stencil reference value directly from a value computed during rendering could simplify our solution and ++ZP further. In OpenGL, this would require an API similar to glMultiDrawElementsIndirectCountARB where the count is a value stored in an OpenGL parameter buffer.





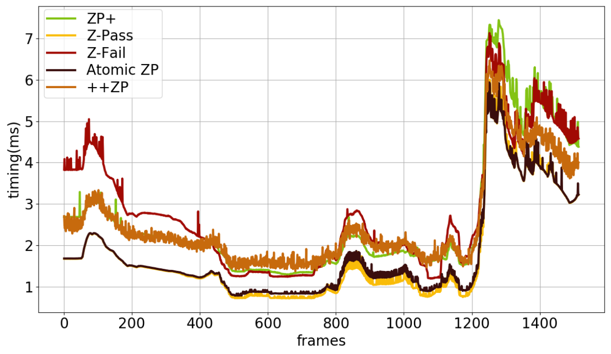


Figure 4: *Top to bottom: Shadow volume creation times for Buddha, Sponza, Citadel. Note that our method's performance is very close to Z-Pass, causing their graphs to overlap.*

Since we are dealing with watertight objects, one could adopt a watertight ray-triangle intersection test [WBW13] to increase robustness over other ray-triangle intersection tests. Standard ray-triangle intersection methods may produce gaps even in watertight meshes. However, we did not encounter this problem in our tests.

5. Conclusion

We introduce AtomicZP, a stencil shadow volume algorithm with minimal overhead over Z-Pass that is, nevertheless, correct regardless of the view and the scene configuration. We show that our approach performs better than competing methods such as Z-Fail, ZP+ and ++ZP while remaining easy to implement. Like the original Z-Pass, our method requires only a single stencil render pass. This advantage becomes especially evident in applications that work with models that have high geometric complexity.

Acknowledgments

The Buddha model is courtesy of the Stanford 3D scanning repository. The Citadel scene was provided by Erik Sintorn. The Sponza scene is based on the original model by Marko Dabrovic, updated by Crytek GmbH.

This work was supported by DyViTo that is funded by the European Union's Horizon 2020 programme under grant agreement No 765121 and partially funnded by Swiss National Science Foundation Advanced Postdoc Mobility Project 174321.

References

[AMA03] Akenine-Möller T., Assarsson U.: On the degree of vertices in a shadow volume silhouette. *Journal of Graphics Tools 8*, 4 (2003). doi:10.1080/10867651.2003.10487591. 2

[App68] Appel A.: Some techniques for shading machine renderings of solids. In *AFIPS Spring Joint Computer Conference* (1968), vol. 32. doi:10.1145/1468075.1468082. 1

[Car00] Carmack J.:. Email to mail list., 2000. (link accessed 2019-03-27). URL: http://fabiensanglard.net/doom3_documentation/CarmackOnShadowVolumes.txt. 2

[CBCJ99] Costa Batagelo H., Costa Junior I.: Real-time shadow generation using BSP trees and stencil buffers. In *XII Brazilian Symposium on Computer Graphics and Image Processing* (1999). doi:10.1109/SIBGRA.1999.805714. 2

[Cro77] Crow F. C.: Shadow algorithms for computer graphics. *SIGGRAPH Computer Graphics 11*, 2 (1977). doi:10.1145/965141.563901. 1, 2

[EAS*13] Eisemann E., Assarsson U., Schwarz M., Valient M., Wimmer M.: Efficient real-time shadows. In *ACM SIGGRAPH Courses* (2013). doi:10.1145/2504435.2504453. 1

[EK02] Everitt C., Kligard M.: Practical and robust stenciled shadow volumes for hardware-accelerated rendering. Online at developer.nvidia.com, 2002. (link accessed 2019-03-27). URL: https://fabiensanglard.net/doom3_documentation/0301002.pdf. 2

[ESAW11] Eisemann E., Schwarz M., Assarsson U., Wimmer M.: *Real-Time Shadows*. Taylor & Francis, 2011. URL: https://www.realtimeshadows.com/. 1, 2, 3

[GMAG15] Gerhards J., Mora F., Aveneau L., Ghazanfarpour D.: Partitioned shadow volumes. *Computer Graphics Forum 34* (2015). doi:10.1111/cgf.12583. 2

[Hei91] Heidmann T.: Real shadows, real time. *Iris Universe* (1991). 1, 2

[HHLH05] Hornus S., Hoberock J., Lefebvre S., Hart J.: ZP+: Correct Z-pass stencil shadows. In *Proceedings of the Symposium on Interactive 3D Graphics and Games* (2005). doi:10.1145/1053427.1053459. 2

[JLBM05] Johnson G. S., Lee J., Burns C. A., Mark W. R.: The irregular Z-buffer: Hardware acceleration for irregular data structures. *ACM Transactions on Graphics 24* (2005). doi:10.1145/1095878.1095889. 1

[MGAG16] Mora F., Gerhards J., Aveneau L., Ghazanfarpour D.: Deep partitioned shadow volumes using stackless and hybrid traversals. In *Proceedings of the Eurographics Symposium on Renderings* (2016). doi:10.2312/sre.20161212. 2

[MT97] Möller T., Trumbore B.: Fast, minimum storage ray-triangle intersection. *Journal of Graphic Tools 2* (1997). doi:10.1080/10867651.1997.10487468. 3

[SEA08] Sintorn E., Eisemann E., Assarsson U.: Sample-based visibility for soft shadows using alias-free shadow maps. *Computer Graphics Forum (Proc. of EGSR) 27* (2008). doi:10.1111/j.1467-8659.2008.01267.x. 1

[SKOA14] Sintorn E., Kämpe V., Olsson O., Assarsson U.: Per-triangle shadow volumes using a view-sample cluster hierarchy. In *Proceedings of the Symposium on Interactive 3D Graphics* (03 2014). doi:10.1145/2556700.2556716. 2

[SOA11] Sintorn E., Olsson O., Assarsson U.: An efficient alias-free shadow algorithm for opaque and transparent objects using per-triangle shadow volumes. *ACM Transactions on Graphics 30* (2011). doi:10.1145/2070781.2024187. 2

[WBW13] Woop S., Benthin C., Wald I.: Watertight ray/triangle intersection. *Journal of Computer Graphics Techniques (JCGT) 2*, 1 (June 2013). URL: http://jcgt.org/published/0002/01/05/. 4

[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), ACM. doi:10.1145/2699276.2699280. 1

[Wil78] Williams L.: Casting curved shadows on curved surfaces. *ACM SIGGRAPH Computer Graphics 12* (1978). doi:10.1145/965139.807402. 1

[Yaz06] Yazheng S.: Performance comparison of CPU and GPU silhouette extraction in a shadow volume algorithm. 2

[ZB11] Zerari A. E. M., Babahenini M.: Shadow volume in real-time rendering. In *International Conference on Communications, Computing and Control Applications (CCCA)* (2011). doi:10.1109/CCCA.2011.6031479. 2