DOI: 10.1111/cgf.14425

COMPUTER GRAPHICS *forum*
Volume 41 (2022), number 1 pp. 106–121

Comparison of Modern Omnidirectional Precise Shadowing Techniques Versus Ray Tracing

Jozef Kobrtek, Tomas Milet, Michal Tóth and Adam Herout

Brno University of Technology
ikobrtek@fit.vut.cz

Abstract

This paper presents an in depth comparison of state-of-the-art precise shadowing techniques for an omnidirectional point light. We chose several types of modern shadowing algorithms, starting from stencil shadow volumes, methods using traversal of acceleration structures to hardware-accelerated ray-traced shadows. Some methods were further improved – robustness, increased performance; we also provide the first multi-platform implementations of some of the tested algorithms. All the methods are evaluated on several test scenes in different resolutions and on two hardware platforms – with and without dedicated hardware units for ray tracing. We conclude our findings based on speed and memory consumption. Ray-tracing is the fastest and one of the easiest methods to implement with small memory footprint. The Omnidirectional Frustum-Traced Shadows method has a predictable memory footprint and is the second fastest algorithm tested. Our stencil shadow volumes are faster than some newer algorithms. Per-Triangle Shadow Volumes and Clustered Per-Triangle Shadow Volumes are difficult to implement and require the most memory; the latter method scales well with the scene complexity and resolution. Deep Partitioned Shadow Volumes does not excel in any of the measured parameters and is suitable for smaller scenes. The source codes of the testing framework have been made publicly available.

Keywords: shadow algorithms, rendering, ray tracing, rendering systems

ACM CCS: • Computing methodologies → Rasterization; Ray tracing

1. Introduction

Shadows have been a subject of intense research for decades, with sustained effort to come up with either the fastest, most precise, or most realistic–looking shadows for various scenarios in computer graphics, as documented by several comprehensive publications [ESAW11, Woo12]. The problem of computing precise shadows for omni-directional lights is even more complex as it in addition to providing a precise solution, it also needs to cover the whole volume of the light source. Although the gaming industry is the primary consumer of shadowing algorithms, it is focused mostly on speed rather then perfection. Relatively recently, pixel-precise shadows were used in video games such as *Tom Clancy's The Division* and *Watch Dogs 2* using NVIDIA's Hybrid Frustum-Traced Shadows (HFTS), which combine triangle shadow frustum testing and shadow mapping [WHL15]. Areas such as CAD/CAM often require precise shadows from arbitrary triangle soup [MKZP14], thus popular and simple–to–implement shadow mapping methods are not a feasible option as they suffer from aliasing problems (e.g. mismatch between eye and camera-space sampling). The target audience is readers from CAD/CAM industry or implementers of various visualizations requiring fast precise shadows from omnidirectional light sources.

The contributions of this article are as follows. We provide a public and open-source shadow testing framework evaluating several shadowing algorithms for free. We use this framework to evaluate modern omnidirectional algorithms producing hard shadows on several types of popular scenes to cover different scenarios in terms of scene configuration and complexity in multiple screen resolutions. The algorithms are also compared on the basis of time complexity and their capabilities to support multiple light sources. To our knowledge, we are the first to compare these methods against state-of-the-art ray tracing using NVIDIA RTX on two platforms – as hardware and software-accelerated. We also present a stencil shadow volume implementation that is able to outperform even methods that claim to be superior in speed. Conclusions are drawn based on performance, memory consumption and implementation difficulty.

© 2021 Eurographics - The European Association for Computer Graphics and John Wiley & Sons Ltd

2. Related Work

A point in a 3D space lies in a shadow if there is occluding geometry between the point and the light source. As simple as the problem might seem, the research to accelerate shadow rendering has been going on for decades and there are numerous categories of methods based on the type of light source, its directionality, or the type of the shadow the algorithm produces. There are also dedicated techniques for solving shadows from many lights, as outlined in [OPB15].

Based on the shadow type, the shadowing algorithms produce either *hard* shadows, where the shadow information is only binary shadowed–lit, and *soft* shadows, where the amount of shadow can be expressed in the 0–1 range. The methods rendering hard shadows from omnidirectional light sources can be put into the following three categories: stencil shadow volumes, which rasterize shadow geometry into the stencil buffer; methods that use acceleration structures built from view samples or scene geometry; methods based on Irregular Z-Buffer; and ray-traced shadows.

Precise shadowing techniques are generally more complex and slower than shadow mapping. As these methods do not perform any resampling, the aliasing problems are less pronounced, but there is still some alias when sampling a binary signal on a regular grid. There are techniques that offer anti-aliased shadows with sub-pixel precision – for example Frustum-Traced Shadows [WHL15], casting multiple shadow rays per sample in ray-tracing or by rasterizing shadow volumes into a multi-sample framebuffer.

A typical problem of these methods is numerical stability and floating point precision [ESAW11, MKZP14]. As the methods operate on geometry and perform a lot of tests between various primitives (e.g. point-frustum containment test, point-plane distance, frustum-AABB test, triangle-point visibility test, etc.), these tests can be victims of floating point inaccuracy, causing a method to produce artefacts. These problems typically arise from, for triangles parallel to the light direction, degenerate triangles or inconsistent results between hardware (rasterization) and software sampling in a shader (a problem we encountered while implementing [SKOA14]).

2.1. Stencil shadow volumes

One of the oldest shadowing algorithms, Shadow Volumes [Cro77], constructs the shadow volume geometry by computing a set of silhouette edges from the 3D scene and then extruding them to infinity in the light direction. An example subset of silhouette edges is visualized in Figure 3. This method is later implemented in standard graphics hardware using the stencil buffer and has become known as *z-pass* [Hei91]. The problem with the camera in shadow is fixed in *z-fail* that essentially reverses the stencil test and requires the shadow volumes to be capped at both sides [Car00, EK02]. Z-fail was patented by Creative Labs [Cre99] but the patent expired in 2019, so the algorithm can now be used freely.

Stencil Shadow Volumes suffer from robustness problems when a triangle is almost parallel to the light direction. Due to floating point inaccuracy, such a triangle can cause an edge to be incorrectly tested as a silhouette and produce a shadow artefact, which exhibits itself as an infinitely-long quad. Adding bias to these computations only pushes the problem forward. This issue was addressed by Pečiva et al. [PSM*13] and further improved by Milet et al. [MKZP14] by sorting the triangle vertices.

A recent paper by Kobrtek et al. [KMH19] shows a technique to reduce the number of edges that need to be tested by building an octree from precomputed potentially silhouette edges using a voxelized scene space which further improves silhouette extraction of a 3D model as not all edges need to be tested. Fu et al. [FZW*20] focus on accelerating silhouette extraction using hash tables, targeting embedded hardware.

Several methods have tried to address the problem of z-pass when the camera is in shadow [HHLH05, ESAW11]. ++ZP [ESAW11] initializes a stencil buffer with a value obtained by rendering the scene into a 1×1 stencil buffer using orthographic projection from the light position towards the camera's near plane. The resulting stencil value, however, needs to be read back to initialize the stencil buffer. As the authors state, fitting the frustum introduces numerical errors and visual problems. AtomicZP [USB*19] extends the idea of ++ZP by casting a ray for every front-facing triangle to test if it lies between the light source and the camera, incrementing an atomic counter. This value is then used in a manual stencil test, discarding fragments having a different stencil value.

The downside of stencil-based methods is fillrate consumed by rendering infinitely large shadow volume sides. Robustness and numerical stability can also be a problem during silhouette computation in cases when the light direction is almost parallel to a triangle connected to the edge causing visual artefacts (e.g. from edges that are not supposed to cast a shadow). As we will demonstrate, these methods are still able to outperform more sophisticated solutions when implemented efficiently, contrary to results of previous evaluations, (e.g. Sintorn et al.) [SOA11] was able to outperform z-fail by a factor of 2 at 4096×4096; similarly Sintorn et al. [SKOA14] claim to be faster than z-pass but the tests were performed only at 1024×1024. Mora et al. [MGAG16] measured z-pass 50% – 300 % slower than their algorithm.

For more detailed information about the stencil Shadow Volumes, we direct the reader to the publications by Eisemann et al. [ESAW11] and Scherzer et al. [SWP11].

2.2. Methods using acceleration structures built from the view samples

The second category of methods builds acceleration structures from view samples in order to determine their visibility.

Aila and Laine [AL04] build a 2D BSP tree from view samples. They then test each triangle against the hierarchy for occlusion and traverse down if a node is at least partially occluded.

Per-Triangle Shadow Volumes (PTSV) [SOA11] builds a hierarchical depth buffer from the view samples. Every level of the hierarchy consists of tiles that represent 8×4 tiles of the level below, acting as a bounding box with a minimum and maximum depth. The algorithm then tests the shadow volume of every triangle in the clip space against the hierarchical depth buffer tiles, marks all nodes that lie inside the volume as shadowed and traverses those nodes that were intersected by any of the shadow volume planes, down to the

© 2021 Eurographics - The European Association for Computer Graphics and John Wiley & Sons Ltd

testing of individual view samples on the lowest level. Transparent shadow casters are also supported by the method.

This technique is further extended in Clustered PTSV (CPTSV) [SKOA14]. The view samples are first grouped into tiles of 8×8. Each tile is divided into several clusters and assigned a Morton code [Mor66] based on the depth and the screen coordinates. Then, a full 3D hierarchical tree is built from the clusters with a branching factor of 32. Each of the clusters is enclosed in an axis-aligned bounding box (AABB) for faster culling. The main idea behind this approach is to reduce the size of clusters for scenes with high depth complexity.

2.3. Methods based on Irregular Z-Buffer

The methods described in this section share a similar acceleration structure, the Irregular Z-Buffer [JMB04, JLBM05]. The technique works similarly to shadow maps, but instead of depth the 2D spatial data structure stores a list of all view samples that have been reprojected to a particular IZB cell.

Story [Sto15] has proposed the Deep Primitive Map (DPM) technique that works similarly to the IZB, but instead of storing view samples, it makes lists of all triangle IDs that cover a particular IZB cell. The data structure seen in Figure 2 is sampled as a shadow map to obtain the list of triangle IDs that are ray-tested from the view sample position towards the light. For the method to work correctly, conservative rasterization needs to be used as all triangles touching a particular IZB cell must be stored in its list. As the method stores triangle IDs, the maximum length of the lists has to be experimentally determined for every scene.

Frustum Traced Shadows [WHL15] conservatively renders the scene geometry against the IZB. A shadow volume is created from a triangle and the light source and every IZB cell touched or covered by the triangle then tests its view-space samples against the triangle's frustum (shadow volume).

A common problem of all the IZB-based methods is long lists, which cause low GPU occupancy and slow IZB traversal. In order to match eye and light space sampling, Wyman et al. [WHL15] propose that ideal parametrization for these methods are cascades, opting for the technique described by Lauritzen et al. [LSL11]. Story [Sto16] uses dynamic reprojection of the light space area where the list lengths exceed a selected threshold. A second projection matrix is computed that projects the selected area into the second IZB list head texture.

2.4. Methods building acceleration structures from the scene geometry

The methods described in this section use BSP-trees to build a hierarchy from the scene's geometry. This approach was first described in Shadow Volumes BSP (SVBSP) [CF89]. The BSP tree is built incrementally from a front-to-back sorted set of polygons with respect to the light source, thus a triangle being processed is tested only against the planes that are already in the BSP tree. Triangles that intersect a plane need to be split, which complicates the build process. Sample visibility is determined by traversing the tree and finding out if the sample's coordinates lie in an occluded part of the tree or not. The resulting data structure needs to be rebuilt every time the light changes.

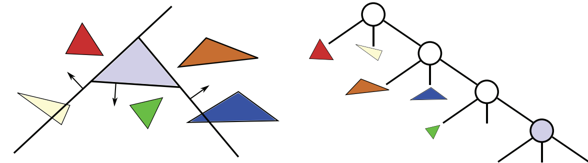

Figure 1: *Construction of the* TOP tree. *The purple triangle creates a volume from the three side shadow planes and itself. Each plane acts as a node in the TOP tree, its children being triangles outside the plane, intersecting the plane or on their inner side. The right side depicts a possible scenario when the purple triangle is added first and all other triangles afterwards. Used both in [GMAG15] and [MGAG16].*

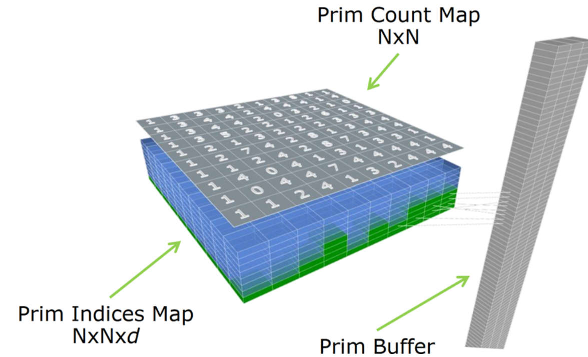

Figure 2: *Data structures of Deep Primitive Map, consisting of the buffer storing indices to triangles covering or intersecting the particular texel of the primitive map. Primitive count map stores the number of triangles in each texel of the primitive map. Primitive buffer is the list of processed triangles [Sto15].*

Gerhards et al. [GMAG15] have proposed a tree that is ternary rather than binary, calling it *TOP tree*, built and traversed on the GPU. Instead of splitting the triangles during the build process as SVBSP, this algorithm assigns triangles that intersect a split plane into a separate intersection node, as seen in Figure 1. This addresses the robustness issues resulting from triangle splitting. Triangles are also inserted in random order, which produces a TOP tree of different quality every frame.

Mora et al. [MGAG16] improve on the TOP tree method by introducing stackless and hybrid traversals, as well as depth optimization. Stackless and hybrid traversal try to trade register allocation of stack-based traversal for memory bandwidth, a hybrid method providing an optimal solution in the majority of the described test cases.

Deves et al. [DMAG18] have improved the scalability of the SVBSP. The geometry in the scene is clustered into and encapsulated with either a bounding sphere or a capsule and stored in a metric tree [Uhl91]. The traversal is similar to methods mentioned above except that the view sample is first tested against the

© 2021 Eurographics - The European Association for Computer Graphics and John Wiley & Sons Ltd

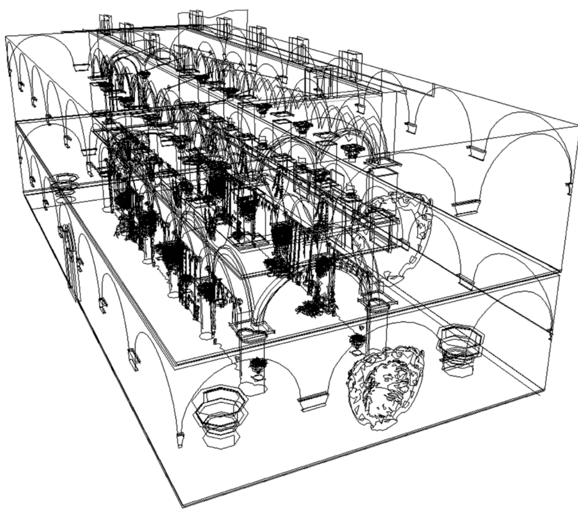

Figure 3: *Visualization of Sponza silhouette edges from a given light position. The set of silhouette edges is only a small subset of all the edges.*

bounding volumes. According to the authors' results the method is more suitable for scenes with a large amount of geometry (millions of triangles); other methods are faster for smaller scenes.

These group of methods share one common feature – their acceleration structure is built every frame in a random order of nodes; thus, its quality varies from frame to frame, which will be seen in the measurements.

2.5. Ray–Traced shadows

One of the first computer graphics techniques for rendering shadows is ray tracing, used for off-line rendering via a scan-line approach [App68]. Recent development of graphics hardware brings ray-tracing from interactive to real-time graphics also for the consumer market. The current generation of graphics hardware has dedicated units for ray tracing acceleration, including the latest game consoles. This makes ray tracing an attractive method for shadow rendering. Shadow computation using ray-tracing can be easily extended from hard to soft shadows by casting multiple shadow rays towards an area light source. These techniques sample multiple random locations on the area light source and accumulate the visibility contribution from each shadow ray cast from a single origin. As the sampled locations cannot cover the whole area of the light source, the shadows suffer from noise, thus a de-noising filter needs to be applied. Such a technique has been proposed by Boksanský et al. [BWB19]. The algorithm utilizes the recently introduced Vulkan ray tracing extension, coupled with the rasterization pipeline. The method uses penumbra detection and adaptive sampling to lower the number of shadow rays based on the sample visibility in the four previous frames. When tracing hard shadows, the method was able to outperform shadow mapping on scenes with multiple light sources. The method, however, produces temporal shadow artefacts and ghosting when used in combination with a moving light source or shadow caster due to reprojection from previous frames.

2.6. Other methods

In the context of this paper we note several key algorithms for Shadow Mapping [Wil78] parametrization that aim to minimize alias and produce hard shadows. Cascade-based approaches [ZSXL06, LSL11] cover the camera frustum by a series of shadow maps, each covering a differently sized area. Warping the shadow map in order to redistribute texels based on over or undersampling of the shadow map regions has been tried by several researchers, notably Martin et al. [MT04] (trapezoidal warping), Wimmer et al. [WSP04] (light frustum warping for directional lights), Rosen [Ros12] (rectilinear warping) and Milet et al. [MNZ15] (non-orthogonal warping). Scherzer et al. [SJW07] use temporal reprojection and jittering to increase the sampling rate of a shadow map. The hybrid approach combining shadow maps and shadow volumes by Chan et al. [CD04] uses shadow volumes only on the shadow borders.

2.7. Sub–Pixel precise methods

Several algorithms were designed to provide sub–pixel anti–aliased hard shadows. Ray tracing can achieve sub-pixel precision naively by simply casting more secondary rays, or by beam [ORM07] or packet tracing [BEL*07]. Sub-pixel shadow maps [LMSG14] try to address both perspective and projection alias of shadow maps by storing fixed-sized partial representation of the scene geometry. Conservative rasterization is then used to generate fragments covering the entire triangle. Sub-pixel precision is achieved by testing multiple subsamples of fragments on the shadow boundary. A similar approach is used in [WHL15] that is capable of up to 32 samples per pixel precision by creating a μquad from screen pixel and projecting it onto the fragments' tangent base. Its subsamples are then marked when testing a triangle shadow frustum sides against the μquad. Du et al. [DFY14] use a triangle-based G-buffer generated from the original coverage triangle of each screen pixel and propose a faithful geometric filter to alleviate aliasing. Several hard-shadow anti-aliasing techniques are discussed by Li et al. [LCF17].

3. Selected Methods and Implementation Details

We tested and evaluated several methods representing each category discussed in the previous section. Stencil shadows are represented by an implementation in the compute shader (CSSV) based on the algorithm proposed by Milet [MKZP14]. We also evaluated implementations in tessellation and geometry shaders, but we were able to optimize the compute shader versions of the algorithm to be the fastest and most consistent of all hardware platforms.

The second category will be represented by the Per-Triangle Shadow Volumes (PTSV [SOA11]) and Clustered PTSV (CPTSV [SKOA14]).

We chose Deep Partitioned Shadow Volumes from the category of methods building acceleration structures from scene geometry as its shader sources are publicly available and the scale of the evaluated models should suit the method.

Although IZB-based methods are not explicitly omnidirectional, we have implemented the Frustum-Traced Shadows [WHL15]

© 2021 Eurographics - The European Association for Computer Graphics and John Wiley & Sons Ltd

using omnidirectional parameterization, similar to omnidirectional shadow mapping (Omnidirectional Frustum-Traced Shadows, OFTS).

As we used a single point light source in our test scenes, we implemented a hardware-accelerated ray tracer casting one shadow ray per fragment using NVIDIA RTX. The source codes for both testing programs are freely available on GitHub as a reference for other researchers and as a public benchmark. (https://github.com/dormon/Shadows, https://github.com/neithy/NeiGinPublic.

All methods except for the ray tracing were implemented in a multi-platform framework using OpenGL 4.5 core. Ray tracing was implemented in Vulkan using `VK_NV_ray_tracing` extension. Both test programs use the same mechanics – we utilized a deferred pipeline to first render the G-buffer (position, depth, normal, color, triangle ID) which serves as the input for the shadowing method. As all these methods use different ways of applying shadows to the scene (stencil mask, acceleration structure traversal, shadow map), we decided to unify the tested algorithms in terms of output. Each method is supposed to fill a *shadow mask* texture that is then used in the final rendering pass to apply shadows to the view samples.

The following subsections describe our implementations for all these methods in more detail.

3.1. Stencil shadows

We based our stencil shadow volumes on z-fail using techniques from [MKZP14] and [KKT08], as it addresses the robustness issues of shadow volumes. The connectivity information is extracted on the CPU and without loss of generality, we set the maximum possible multiplicity to 2 in our tests – edges having more than two adjacent triangles were split into several instances. The compute shader then tests every edge for silhouetteness. In order to minimize the amount of data written to the global memory, the compute shader only outputs one encoded integer per silhouette edge consisting of edge ID and its multiplicity. Then, the geometry shader receives this encoded information as a vertex attribute and casts the edge side as many times as its multiplicity with correct triangle winding based on the multiplicity sign. To speed up the computation process even more, we replace storing opposite vertices with the edges with precomputed triangle planes that would otherwise have to be calculated every time in the silhouette testing shader from the edge vertices and from the opposite vertex. The shadow volume caps are rendered similarly using the geometry shader, we compute multiplicity from the triangle and the light source, the sign of the multiplicity determining the winding of the triangle.

3.2. PTSV and CPTSV

Both PTSV and CPTSV aim to reduce the shadow volume rasterization times as it is the most demanding step. They do not use traditional rasterization of shadow volumes as the stencil methods, instead they hierarchically rasterize the shadow volumes into an acceleration structure. PTSV uses a hierarchical depth buffer and a shadow mask buffer. CPTSV uses a 3D tree of view sample clusters, as seen in Figure 4. All hierarchical structures have a branching factor equal to 32 (SIMD size) in the original design. Threads in each warp cooperate in rasterization of one shadow frustum.

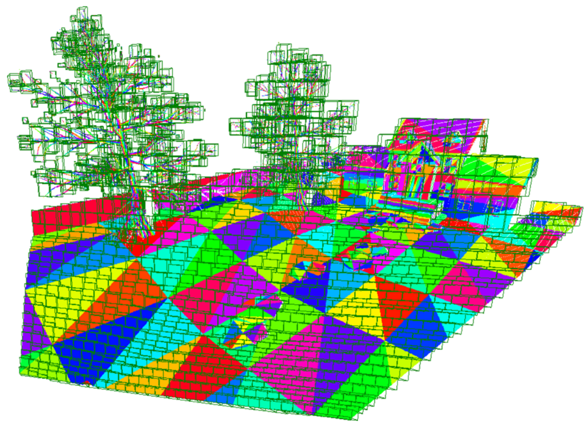

Figure 4: *The image shows view-samples (coloured triangles) and clusters (green boxes) on one level of the CPTSV's hierarchy*

We re-implemented both methods in OpenGL according to the original papers and available source codes. The authors provided us with the implementation of PTSV in CUDA. We modified the implementation to run on different hardware (AMD) and to support different resolutions and fixed some visual artefacts. We tested different memory storage types (textures and buffers) for intermediate results. In the end, we chose textures as they performed slightly better.

CPTSV was implemented from scratch with all the extensions mentioned in the original paper, except for load balancing. We implemented the traversal shader using a small stack since template recursion is not possible in OpenGL.

We further optimized the construction of upper parts of the 3D cluster tree by storing the IDs of active nodes of the previous level and launching only the appropriate number of threads. We also tried to minimize the number of operations required to compute Morton codes for x, y, and z components with different bit lengths. Furthermore, we optimized the traversal step of the algorithm by reducing the number of registers required for the stack using local memory.

According to our measurements, we are convinced that our implementation is on a par with, or faster, than the original implementation.

3.3. Deep partitioned shadow volumes

The shaders for DPSV [MGAG16] have been made publicly available by the authors; we provided all the necessary inputs and outputs for the method. We implemented a deterministic shadow plane construction based on [MKZP14], which helped with blinking artefacts we encountered with this method. After testing all three variants of the method's TOP tree traversal (stack, stackless, hybrid), we opted for the hybrid variant, combining both approaches, as it was the fastest of all three versions.

© 2021 Eurographics - The European Association for Computer Graphics and John Wiley & Sons Ltd

3.4. Omnidirectional frustum-traced shadows

This method was implemented using all optimizations mentioned in the original paper [WHL15] (tight-fitting projection, discarding back-facing view samples, removal of already shadowed view samples, depth buffer initialization) including the reprojection of areas with very long lists using off-centre projection [Sto16]. Our implementation works similarly to omnidirectional shadow mapping, utilizing six light frusta to cover all directions. We cull those frusta in two stages – first on the CPU by computing collisions of light frusta with camera frustum and producing a bit mask, which speeds up the heat map construction. The second frustum culling occurs when computing new projection matrices, as some of the visible light frusta may not contain any view samples. The whole method runs in a single pass using compute shaders to compute the heat map (list lengths per light-space texel), new projection matrices and IZB; the depth buffer optimization and the IZB traversal utilize geometry shader and layered rendering to draw to several textures simultaneously. The IZB traversal pass requires conservative rasterization; we use NVIDIA's OpenGL extension `GL_CONSERVATIVE_RASTERIZATION_NV`.

3.5. Ray tracing

Shadow rendering by ray tracing was implemented separately in a Vulkan test program using NVIDIA's `VK_NV_ray_tracing` extension. Similarly to other implemented methods, it used geometry information provided by the G-buffer to create a per-fragment shadow mask. A simple ray generation shader casts a ray from the fragment position towards the light and the corresponding miss shader marks the fragment as lit. As a precise triangle intersection is not required, we enabled an optimization in the form of `TerminateOnFirstHit` flag, that is terminating BVH traversal upon hitting any triangle towards the light source.

To use the ray tracing extension, a two-level bounding volume hierarchy is required. The bottom level consists of elements of the geometry, and the top level is built from bounding volumes of object instances. In the case of static geometry, the acceleration structure is created once and never updated. For a scene with moving objects, only the top level needs to be updated with transformations. The bottom level structure needs to be updated only when the geometry changes (elastic simulation, skinning, etc.). In order to simulate several possible scenarios, we included measurements without BVH updates as the best-case scenario and a full per-frame BVH rebuild as the worst case. Building is done on the GPU; therefore, all memory needs to be allocated in advance. The API provides an upper estimate of the required memory amount based on the provided geometry, but the final size of the BVH is usually around 50 % of the estimate.

4. Measurements

The measurements were carried out on a set of popular scenes; details can be seen in Table 1. Although 'Villa' is a small scene compared to modern standards, it was used in [SKOA14], and we will demonstrate that this method was designed specifically for scenes of this type – high depth complexity and falls behind on other scene types. Even "Hairball", with its 2.8 million triangles, poses a challenge for the tested algorithms. All scenes were tested with a single point light source. The 'Buddha' and 'Hairball' scenes were slightly modified – we positioned both models on a plane, acting as a shadow receiver. Each scene was tested using a camera flythrough that took 1000 frames. Every frame was rendered five times and the average time of the shadow mask creation was written to a `.csv` log file. The tests are focused on resolutions of 1920×1080 and 3840×2160, but we have also tested on other resolutions, from $1K \times 1K$ to $4K \times 4K$, when evaluating the resolution dependency. The memory consumption of all methods is analysed as well.

Table 1: *Test scenes used for method evaluation.*

Scene	Triangle count	Edge count
Villa	88 870	136 663
Conference	124 619	195 019
Sponza	279 163	431 246
Closed Citadel	613 567	921 555
Buddha	1 087 476	1 630 522
Hairball	2 880 002	4 290 005

Table 2: *Memory consumption of all tested algorithms on all scenes.*

Scene	RTX	CSSV	DPSV	PTSV	CPTSV
Villa	5.63	7.82	10.85	8.13	9.49
Conference	7.88	11.16	15.21	17.85	20.83
Sponza	17.50	24.68	34.08	39.48	46.06
Citadel	38.56	52.73	74.90	84.37	98.43
Buddha	68.40	93.30	132.75	149.27	174.16
Hairball	181.02	245.48	351.56	392.76	458.22

Notes: The sizes for the ray tracer are only for the BVH structure. As we only cast one secondary ray per fragment, they should fit into the GPU registers. The memory footprint of Sintorn's methods (PTSV, CPTSV) depends on the resolution and other factors, so the table only shows the size of shadow frusta buffers. All sizes are in MB.

All tests were carried out on an AMD ThreadRipper 1920X system with 32 GB of RAM and a GeForce RTX 2080 Ti graphics card with hardware ray tracing support. Some of the test were run on a GeForce GTX 1080Ti that supports RTX API in fallback mode using compute shaders. The system runs on Windows 10, and both test applications were compiled using Visual Studio 2019.

4.1. Memory consumption

We measured the amount of memory each method requires for its acceleration structures; the results can be seen in Table 2. The memory footprint of OFTS is not affected by the scene geometry, as it depends only on the method parameters and the screen resolution. We used two sets of parameters based on profiling – lower resolutions (up to 1920×1080) used 1024 by 1024 for one slice of the IZB's head texture, and higher resolutions used 2048×2048. The reason for this was to cope with the performance drops caused by insufficient reprojection of the densest areas of the heat map. The heat map had a resolution of 512×512 for all screen resolutions.

© 2021 Eurographics - The European Association for Computer Graphics and John Wiley & Sons Ltd

This resulted in memory consumption of up to 98 MB for lower resolutions (up to 1920 × 1080), and up to 403 MB at the resolution of 4000 × 4000. The reason for such a high amount of memory at the higher resolutions is that the IZB's head texture has 12 layers (six sides with reprojection), the same for the depth texture for the z-buffer optimization pass. At lower resolutions, the benefit of being independent on the scene geometry prevails on larger models; however, at higher resolutions (3840 × 2160 and more), the method is the second most demanding, even at the 'Hairball' scene.

From the rest of the tested methods, of which the memory footprint is dependent on the amount of scene geometry, ray tracing reports the lowest amount of video RAM (based on the upper estimate reported by the RTX API), followed by CSSV. The stencil method only needs one buffer to store the edge information and another to write the resulting encoded multiplicity and the edge ID. DPSV stores the TOP tree nodes in a single linear buffer and its memory consumption is around 40% higher than CSSV.

The memory requirements of PTSV and CPTSV depend on two factors: the number of triangles and the resolution. Both algorithms need to allocate a shadow frusta buffer, the size of which can be seen in Table 2.

Apart from the shadow frusta buffer, PTSV uses two acceleration structures – one hierarchically stores the depth ranges per tile, the other contains the actual shadowed/lit information. The depth range is represented by two float values; the number of tiles on every level can be expressed by Equation (1), where T is the number of tiles, R the resolution, N the number of levels, B the branching factor (workgroup size) and i the index of the level. The size of these hierarchical structures is 5 MB at 4K resolution.

$$T = \frac{R}{b^{N-i}} \quad (1)$$

The memory footprint of CPTSV's hierarchical structures is much larger than PTSV. The actual size of the structures depends greatly on resolution, branching factor, and z' bits in the cluster key; the amount of memory can go up to 6 GB at $4K \times 4K$ (buffer containing the AABBs of the clusters). We have also implemented the memory reduction scheme as mentioned in the original paper, reducing the amount of memory approximately 6.5-times at a cost of about 5% of the performance.

4.2. Evaluation at 1920 × 1080

The results of all methods and scenes at 1920 × 1080 can be seen in Figure 5 and the average shadow mask creation times in Table 3. PTSV was removed from the 'Buddha' graph (Figure 5e) for clarity, as its average performance was 352 ms.

Although RTX without BVH rebuild is the fastest method of all, it is also an ideal condition that would probably not be achieved in a real scenario. 'RTX AVG' as an average is probably closer to a practical case, but even then it was the fastest in the most cases. Ray tracing was also the most stable method tested.

CSSV's average time was spoiled by the "Hairball" scene, where the silhouette is not simple – the model is comprised chiefly of thin geometry, producing a complex silhouette that generates a lot of shadow volumes requiring rasterization. Otherwise, this method would be second to ray tracing in this scenario, even surpassing the "RTX AVG" time twice. Compared to OFTS, stencil shadow volumes perform better on enclosed scenes ('Conference', 'Sponza') and scenes with a relatively simple silhouette ('Buddha', as well as 'Conference', has a relatively simple silhouette). Tree branches on 'Villa' cast shadows that cause a high fill rate, which leads to lower performance of the stencil method compared to other techniques.

Table 3: *Average shadow mask creation times across all the test scenes at* 1920 × 1080.

	Villa	Conf.	Sponza	Citad.	Buddha	Hairb.	AVG
CPTSV	**1.30**	8.51	3.08	4.93	7.04	33.55	9.74
CSSV	2.23	**0.68**	**1.57**	4.15	**2.24**	29.90	6.80
DPSV	1.87	3.11	2.91	5.16	8.47	120.88	23.74
OFTS	2.05	2.05	3.70	**3.30**	5.22	**14.32**	5.08
PTSV	12.73	12.73	10.25	10.66	352.64	97.90	81.42
RTX rebuild	1.18	1.47	2.33	4.33	7.23	16.50	5.50
RTX	0.24	0.28	0.37	0.26	0.08	0.54	0.29
RTX AVG	0.71	0.88	1.35	2.30	3.66	8.52	2.90

Notes: The bold values represent the fastest algorithm (except ray tracing). 'RTX_AVG' represents an average of 'RTX' and 'RTX rebuild'. The 'AVG' column contains the average shadow mask creation time across all frames on all scenes. All values are in milliseconds.

PTSV is the slowest method on most of the scenes. The 'Buddha' scene seems to pose a non-trivial problem for PTSV as the model contains a high number of small triangles positioned mostly in the middle of the viewport. This triangle distribution causes an imbalanced GPU load when traversing the acceleration structure of the view samples, as only a handful of the view samples are affected by the vast majority of the scene geometry, causing the method to perform very poorly in this test case.

CPTSV excels on the 'Villa', as the scene was specifically designed for this method.Both PTSV and CPTSV perform unusually on the 'Conference' scene, where the average time is slower than on 'Buddha', despite having just 11% of its geometry. This is probably caused by missing load-balancing optimization, as there are larger triangles in the scene (table, floor, etc.). A single triangle is processed by one warp, which causes improper load balancing when the triangle covers a large portion of the screen, as a lot of nodes need to be processed. Apart from these two cases, the method can be considered an average one – not the fastest, not the slowest.

DPSV randomly builds its TOP tree every frame, meaning that the quality of the acceleration structure differs from frame to frame. It can be observed as fluctuations mostly on the 'Hairball' scene. The complex and concentrated geometry in this scene poses a challenge for the build phase of the algorithm, resulting in huge variation of the frame times. The method was faster than CPTSV up to 'Sponza', but does not scale as well as other methods with the increasing amount of geometry.

OFTS had to be tuned for smaller (up to 1920 × 1080) and larger resolutions separately, as the parameters greatly affected sudden performance drops caused by the reprojection area being too large and the most exposed lists did not get properly redistributed. This

© 2021 Eurographics - The European Association for Computer Graphics and John Wiley & Sons Ltd

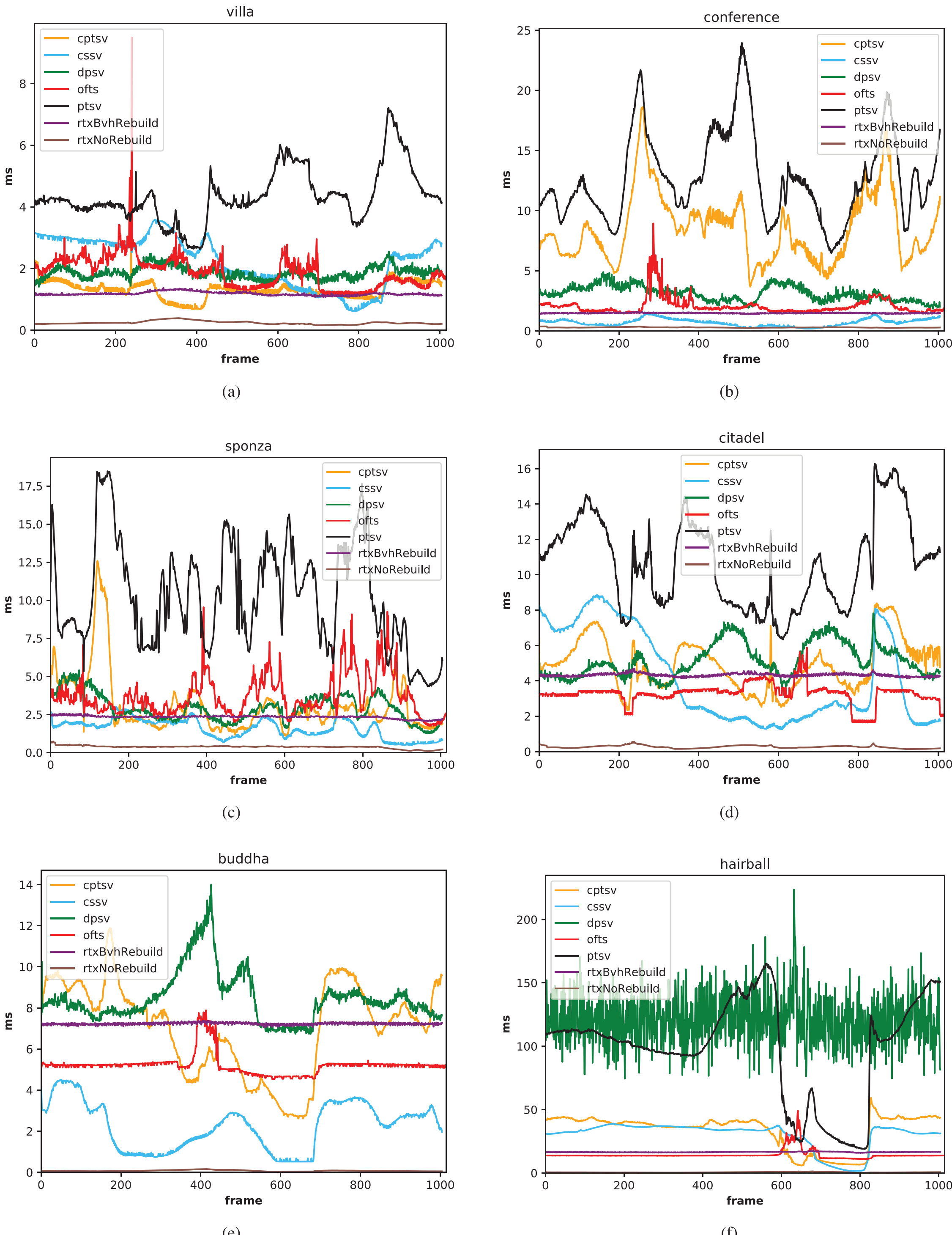

Figure 5: *Comparison of all the methods using flythroughs on various test scenes, resolution* 1920 × 1080*, sorted by the amount of triangles. Each graph represents one tested scene with results from methods tested. Ray tracing is measured twice – without any modification to the BVH (as 'rtxNoRebuild') being the best possible scenario and with full BVH rebuilt every frame ('rtxBvhRebuild') for the worst case. The 'Buddha' scene does not include the PTSV method, as its average performance was* 352 ms *and was removed for clarity.*

© 2021 Eurographics - The European Association for Computer Graphics and John Wiley & Sons Ltd

Table 4: *Average shadow mask creation times across all the test scenes at* 3840 × 2160.

	Villa	Conf.	Sponza	Citad.	Buddha	Hairb.	AVG
CPTSV	**2.16**	18.51	7.52	12.61	5.96	29.95	12.79
CSSV	5.42	**1.74**	**4.27**	9.68	**3.97**	64.29	14.90
DPSV	4.95	8.52	7.01	10.62	10.61	129.97	28.61
OFTS	4.25	4.78	8.18	**4.43**	6.21	**17.52**	7.57
PTSV	5.40	41.62	23.52	13.19	316.87	91.77	82.06
RTX rebuild	1.85	2.23	3.36	5.0	7.31	17.66	6.24
RTX	0.90	1.06	1.40	0.99	0.25	1.93	1.09
RTX AVG	1.38	1.65	2.38	3.0	3.78	9.80	3.67

Notes: The table description is identical to Table 3.

Table 5: *Average shadow mask creation times across all the test scenes at* 1920 × 1080 *running on GeForce GTX 1080Ti.*

	Villa	Conf.	Sponza	Citad.	Buddha	Hairb.	AVG
CPTSV	2.14	10.30	4.36	7.05	11.22	49.48	14.09
CSSV	2.31	**0.74**	**1.79**	4.51	**2.90**	32.01	7.38
DPSV	2.93	4.63	4.19	6.87	9.86	137.08	27.59
OFTS	**1.87**	2.22	3.97	**3.66**	5.94	**16.39**	5.67
RTX rebuild	3.62	3.22	5.12	7.26	8.53	26.28	9.01
RTX	2.39	1.65	2.74	2.45	0.42	7.66	2.89
RTX AVG	3.01	2.44	3.93	4.86	4.47	16.97	5.95

Notes: The table description is identical to Table 3.

happens most often when the camera is very close to the geometry or when the reprojection itself cannot benefit from the shape of the reprojection area (e.g. when the longest lists are in opposite corners of the heat map). Such spikes can be seen on the ‘Villa’ scene. Its performance also depends on the number of active light frusta, which can be seen for example on the ‘Sponza’ scene where the light is positioned in a way so that all 6 frusta are facing some of the scene’s geometry, thus there are always multiple frusta active during the flythrough; compared to, for example ‘Buddha’, where most of the scene geometry will be concentrated in a single frustum. OFTS can handle complex geometry, like ‘Hairball’, better than other conventional methods. This method suffers from the same problem as omnidirectional shadow mapping – seams between the cubemap faces, exhibiting as an occasional line of lit fragments.

4.3. Evaluation at 3840 × 2160

The results of the $4K$ flythroughs are presented in Figure 6 and Table 4; the results of the ‘Buddha’ scene in Figure 6e are again missing the PTSV graph, as the method performed very slowly – 316 ms on average. We were surprised by the results of the stencil method on the $4K$ resolution, as we estimated that the rasterization of the shadow volumes geometry would cause CSSV to perform as one of the slowest, but the results seem to follow a similar trend as at 1920 × 1080. In terms of the average across all scenes, OFTS was again second-fastest to ray tracing, followed by CSSV (mainly because of poor performance on the ‘Hairball’ scene) and CPTSV. Interestingly, the PTSV was able to outperform CPTSV on the ‘Hairball’ scene compared to the full-HD test; the reasons will be disclosed below. OFTS had to be tuned for higher resolution, as we often experienced spikes in frame times; for example, there were 270 ms spikes on both ‘Sponza’ and ‘Hairball”. We had to adjust the resolution and reprojection threshold to cope with them, but they are still visible. Ray tracing is again the fastest solution.

4.4. Frame time decomposition

The timings of the particular components of each method can be seen in Figure 7, measured on the ‘Sponza’ scene at 1920 × 1080 resolution. Sponza was chosen because the omni directional light source can demonstrated be very well in its enclosed atrium.

CSSV can be broken down into two parts – silhouette computation and shadow volume rasterization. We used the *z-fail*, thus we rendered both the front and back caps for each shadow volume. The silhouette extraction takes only around 0.05 ms (‘compute’). The extruded sides take the longest to render, as they consume a lot of fillrate. Drawing caps takes on average 9.5% of the total shadow computation time (0.15 ms in average).

The time to build the hierarchical tree from the view samples in CPTSV is significantly faster than in PTSV, 0.44 versus 3.17 ms on average. Wedge optimizations for faster tile culling sped up the rasterization of the shadow volumes against the hierarchical tree structure.

All stages of OFTS except traversal take about 1 ms combined. The traversal itself takes 2.72 ms on average on this scene, about 73% of the shadow compute time. ‘Sponza’ as an enclosed scene provides a good example, as the number of active frusta frequently changes in the course of the test as well as projected area in each frustum. Sudden spikes are caused by IZB lists being long.

The ray tracer spent most of the frame time building the BVH structure; the tracing itself is only around 20% of the total time. We have also found out that the initial BVH build takes up 10 ms more than all subsequent rebuilds, probably due to memory allocation. The traversal part is very fast, also because all the rays are coherent and converge to a single point.

4.5. Evaluation on GeForce GTX 1080Ti

We repeated the measurements using GeForce GTX 1080Ti, which supports the RTX API in fallback mode, representing a well-optimized software ray tracing solution. The results can be seen in Figure 8 and Table 5. As PTSV was the slowest method of all, we excluded it from this measurement.

Ray tracing, on average, performs 2.1-times slower than on the RTX 2080Ti; rebuild 1.6-times and the traversal-only scenario 10-times. The acceleration structure traversal benefits mostly from the hardware acceleration. Compared to other methods on this platform, ray tracing without rebuild is not the fastest method until the Citadel scene. Although a combined average time of ray tracing was second to pure traversal on the 2080Ti, it was surpassed by OFTS on the 1080Ti. CSSV is also faster than RTX with rebuild on the legacy

© 2021 Eurographics - The European Association for Computer Graphics and John Wiley & Sons Ltd

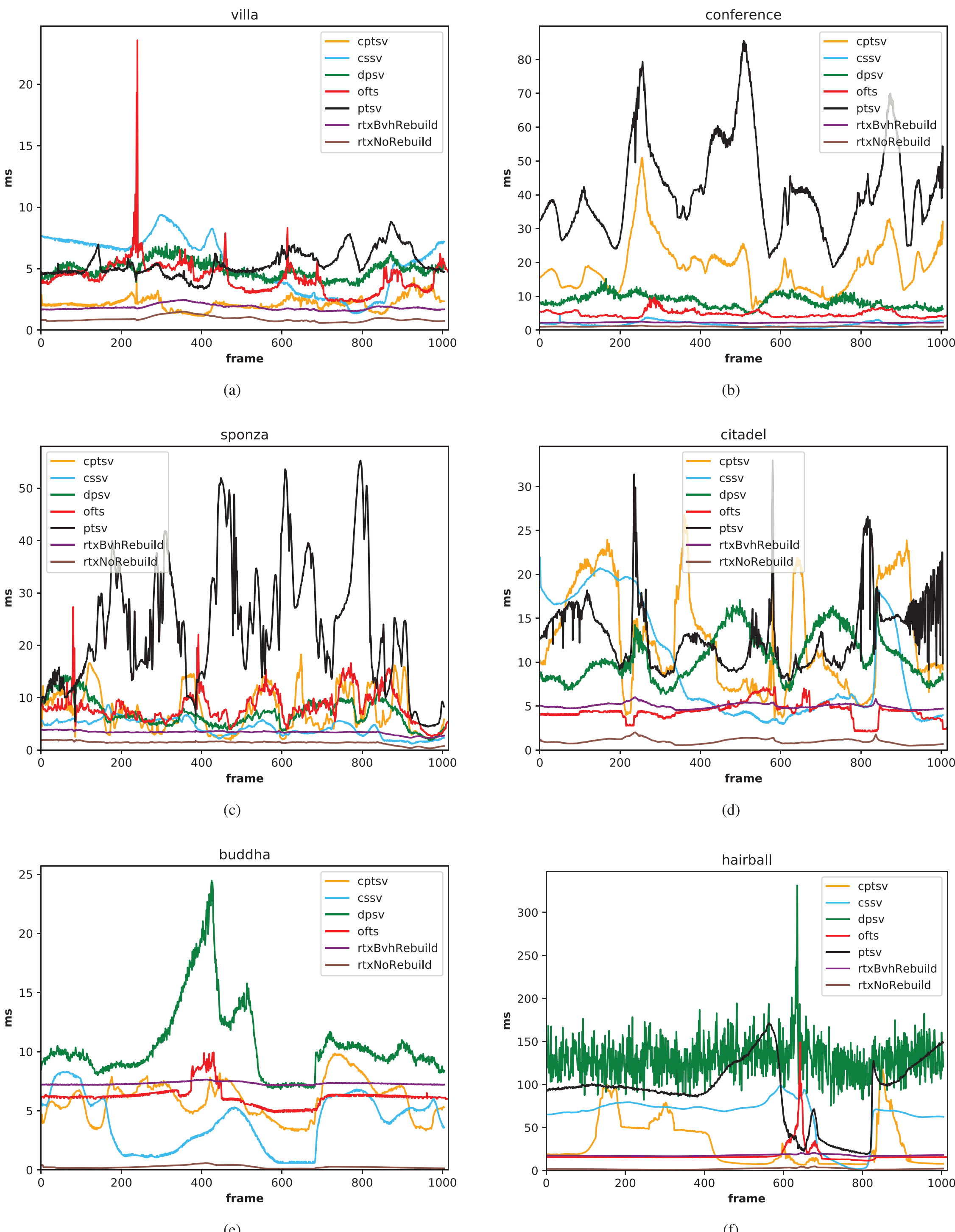

Figure 6: *Comparison of all the methods using flythroughs on various test scenes, resolution* 3840 × 2160*. The methods are labelled the same way as in Figure 5. The test on the 'Buddha' scene again does not include the PTSV method, as its average performance was* 316 ms.

© 2021 Eurographics - The European Association for Computer Graphics and John Wiley & Sons Ltd

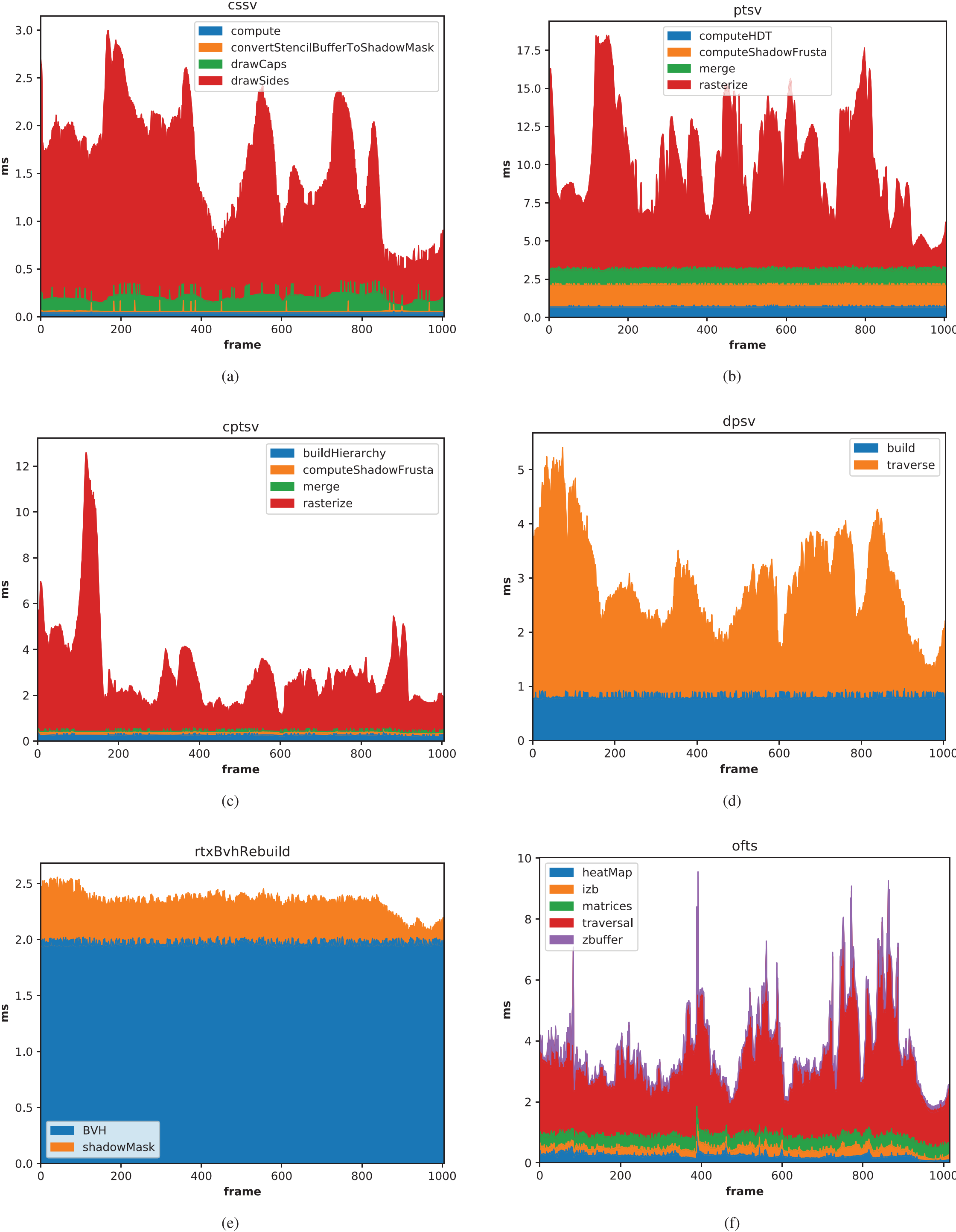

Figure 7: *Frame time decomposition of all methods on the Sponza scene at* 1920 × 1080*. Ray tracing is represented by its worst-case scenario (full BVH rebuild every frame).*

© 2021 Eurographics - The European Association for Computer Graphics and John Wiley & Sons Ltd

Figure 8: *Evaluation at* 1920 × 1080 *using GeForce GTX 1080Ti, having only software support for ray-tracing. The methods are labelled the same way as in Figure 5. PTSV was excluded from the measurements for clarity.*

© 2021 Eurographics - The European Association for Computer Graphics and John Wiley & Sons Ltd

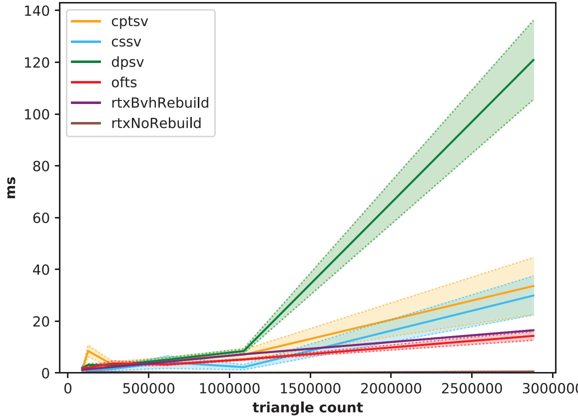

Figure 9: *Dependency on the triangle count across all tested scenes at 1920 × 1080. PTSV was excluded for clarity.*

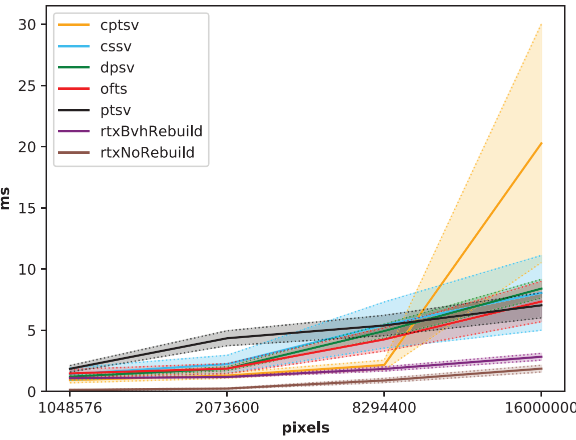

Figure 10: *Computational times as they depend on the screen resolution on the "Villa" scene.*

platform. Conventional methods were, on average, 15.5 % (8%–30 %) slower than on the 2080Ti.

4.6. Dependency on triangle count

Figure 9 shows performance dependency on triangle count across all of the tested scenes and methods at 1920 × 1080. It was calculated as the average and mean absolute deviation from all the frame times of the flythrough on a particular scene. Due to PTSV's behaviour on the 'Buddha' scene (described above), the method was excluded from the graph. It also has the highest dependency on the triangle configuration of all the tested methods; its mean absolute deviation at 4000 × 4000 was 30 ms. It can be seen that ray tracing without rebuild does not put a lot of stress on the RT cores of the GPU; we are tracing 2 megarays at 1920 × 1080 shadow rays per frame, where the hardware is, in theory, capable of 8 gigarays per second. OFTS shares similar triangle dependency with ray tracing with full rebuild, having the lowest average and mean absolute deviation of the conventional methods. CSSV is something of a surprise, as the method needs to rasterize a lot of infinite shadow volume geometry. Although having lower triangle dependency than CPTSV in the tested scenarios, the tide would change for larger scenes as the average curves of both methods converge. With the increasing number of triangles, DPSV was gradually outperformed by other methods, and its curve steep final segment is the result of the 'Hairball' scene.

4.7. Dependency on screen resolution

We produced a graph in a similar fashion for dependency on the resolution as well; see 10.

CPTSV's average shadow computation time increased 7.5-times at 4096 × 4096 compared to $4K$ resolution despite the fact the acceleration structures have the same size in both cases. This occurred on every test scene. We found out that it is caused by incorrect view sample depth clustering. The majority of the view samples in all screen tiles fell into the same cluster even if their depth was different. The reason is the exponential division of the view frustum's depth. The amount of bits allocated for encoding depth using the Morton code depends, apart from other factors, on the vertical screen resolution and on the distance of the near clipping plane. To encode the depth at 4096 × 4096, the bit count would exceed the allocated 10 bits, causing the z-part of the Morton key to overflow and thus storing all view samples into the same furthest cluster. This resulted in a very long AABB of the cluster (along the z axis) which causes very slow traversal, as most of the clusters have to be visited by the majority of the shadow frusta. One of the possible fixes would be to allocate more bits (13 or more) for the depth, but in that case, a 32-bit integer would be insufficient and an arbitrary bit array would slow down the method. Using 64-bit keys would double the already high memory consumption. Another possibility would be to divide the frustum using a different scheme.

Provided CPTSV would not be affected by the problem mentioned above, we extrapolated a hypothetical average frame time between 3.8 and- 5.1 ms (using quadratic regression and power curve) for $4K \times 4K$ resolution. In such a case, the method would have had one of the lowest resolution dependencies of all the tested methods.

Although PTSV and CPTSV have a lower resolution dependency thanks to the hierarchy they build, they have larger initial overhead and don't scale well with the increasing amount of triangles. As expected, CSSV is more sensitive to resolution than other methods; its curve is similar to OFTS but absolute deviation is the widest (except for $4k \times 4k$). DPSV follows CSSV in this scenario, but ends up slightly above. Hardware-accelerated ray tracing has the lowest sensitivity on all the test scenes.

5. Time Complexity

Although the authors of the original papers do not state the complexity of their algorithms, we tried to estimate the average time complexity based on increasing scene complexity, screen resolution and multiple light sources for every major part of the tested algorithms. These findings can be seen in Table 6. In terms of resolution, most of the algorithms scale linearly with the increasing amount of pixels except for CPTSV. When multiple lights are used, ray tracing seems

© 2021 Eurographics - The European Association for Computer Graphics and John Wiley & Sons Ltd

Table 6: *Time and memory complexities of the tested algorithms, broken down into stages.*

Method	Resolution	Geometry	Multiple lights
CSSV (adjacency)	O(1)	O(N log(N))	O(1)
CSSV (silhouette)	O(1)	O(N)	O(N)
CSSV (rasterize)	O(N)	O(N)	O(N)
PTSV (shadow frusta)	O(1)	O(N)	O(N)
PTSV (depth stencil)	O(N)	O(1)	O(N)
PTSV (traversal)	O(N)	O(N)	O(N)
CPTSV (shadow frusta)	O(1)	O(N)	O(N)
CPTSV (view cluster hierarchy)	O(N)	O(1)	O(1)
CPTSV (traversal)	O(log(N))	O(N)	O(N)
OFTS (shadow frusta)	O(1)	O(N)	O(N)
OFTS (IZB build)	O(N)	O(1)	O(N)
OFTS (traversal)	O(N)	O(N)	O(N)
RTX (build)	O(1)	O(N log(N))	O(1)
RTX (trace)	O(N)	O(log(N))	O(N)
DPSV (build)	O(1)	O(N log(N))	O(N)
DPSV (traverse)	O(N)	O(log(N))	O(N)

Notes: An empty cell means the stage of a particular algorithm is not dependent on the number of triangles or screen pixels.

again to be the best possible method as it does not require a BVH rebuild, unlike all the other evaluated methods.

6. Discussion and Conclusion

We were able to compare several modern omni directional shadowing methods with precise hard shadows. These methods were tested on several popular test scenes under multiple resolutions.

Hardware-accelerated ray tracing is the clear winner in terms of speed, implementation difficulty and even memory consumption in the most cases. It is clear that having dedicated hardware units can reduce the BVH traversal time by a factor of 10. Ray tracing (traversal only) on legacy hardware was able to outperform all the methods on scenes having more than 600,000 triangles. Ray tracing also has the lowest triangle and resolution dependency in our tests. Unlike all other tested algorithms, support for multiple light sources does not require a BVH rebuild, as the structure can be reused, since it is independent on the light position. Transparent casters and sub-pixel precision are also supported.

Although the authors of Frustum-Traced Shadows did not design the method primarily for omnidirectional parametrization, the method works very well with this configuration. It has one of the lowest triangle dependencies, a predictable memory footprint and was able to handle the ‘Hairball’ scene second best to ray tracing. Its implementation is straightforward and does not require any preprocessing. The downsides are higher memory consumption on higher resolutions, and performance drops due to long lists, which need to be addressed using parameters which are scene-dependent. We noticed a light-leaking artefact on the seams between the frusta, probably caused by different projections. Although CSSV was faster on the enclosed test scenes, OFTS has better geometry dependency and, in most cases, better resolution dependency. It was the only method to outperform the average of combined RTX time on the 1080Ti.

The CSSV algorithm was a surprise, as almost all previous papers presented stencil-based shadow volumes as being too slow for larger resolutions. Our implementation was able to compute an object’s silhouette in 0.03 to 0.1 ms across the test scenes, thus the majority of the method’s time was spent on the rasterization of shadow volumes. Although based on z-fail, CSSV was the fastest conventional method on ‘Buddha’, ‘Conference’ and ‘Sponza’ in both 1920×1080 and 3840×2560. These scenes have relatively a simple silhouette, mostly observable on ‘Buddha’, where the method is on average 48% faster than OFTS behind it. As no bias was used during any of its stages, the method has the most accurate shadows of the tested algorithms. We think the advance in the graphics hardware and increased rasterization performance can also be credited for the performance of the stencil shadow volumes. The method’s implementation is among the easier ones; its memory consumption is second to ray tracing. The downsides are edge extraction as a preprocess step and unpredictable performance due to shadow volume rasterization. If not for the “Hairball” scene, which was the worst-case for this method, we would declare CSSV to be the second-fastest algorithm.

DPSV was improved by deterministic shadow plane calculations, which helped with robustness of the method but we still experienced blinking triangles in the ‘Buddha’ scene, as the model consists of very small triangles. The randomness of the TOP tree build quality between consecutive frames caused the method to perform less stably locally, most notably on the ‘Hairball’ scene. Our measurements have shown that this method is more suitable for scenes with up to 1 million triangles. The method is easy to implement, as source codes for both its shaders are already available and requires no geometry preprocessing. The memory consumption was average compared to other methods. Overall, the method ends up 4th in our comparison, as it did not excel in any of the observed parameters.

We succeeded in porting PTSV and CPTSV algorithms from CUDA to OpenGL, making them available to other hardware platforms. These methods, particularly CPTSV, are difficult to implement and their memory consumption is the highest of all tested methods, as it depends not only on the geometry, but also on the screen resolution. CPTSV’s acceleration structure, containing AABBs for view sample clusters, can take up to 6 GB at 4096×4096 if the memory optimization is not used, but can still take around 1 GB when optimized. The method also suffers from an incorrect acceleration structure build when the screen resolution is very high, which negatively affects its performance. If not for this issue, this method would have had one of the lowest sensitivities to screen resolution. PTSV was the slowest algorithm in the test. In general, we don’t recommend either of these methods for practical use, as there are faster and easier-to-implement methods that also consume less memory.

Except for ray tracing, there is no universal method that would be suitable for all scenarios; all methods have their best and worst cases. If ray tracing is not available, OFTS is suitable for more opened scenes, whereas CSSV handles closed or scenes with simple silhouette. Our tests also conclude that more complex code does not necessarily yield faster frame times.

In the future, the ray tracing implementation could be optimized for other hardware platforms, including the current generation of

© 2021 Eurographics - The European Association for Computer Graphics and John Wiley & Sons Ltd

game consoles. It would be interesting to see even bigger scenes, although ‘Hairball’ was already challenging for the most of the algorithms as the fastest time behind ray tracing was 14ms at 1920 × 1080. Multiple light sources is also an issue that is not frequently evaluated on these methods.

References

[AL04] Aila T., Laine S.: Alias-free shadow maps. In *Proceedings of the Fifteenth Eurographics Conference on Rendering Techniques* (Aire-la-Ville, Switzerland, Switzerland, 2004), EGSR’04, Eurographics Association, pp. 161–166.

[App68] Appel A.: Some techniques for shading machine renderings of solids. In *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference* (New York, NY, USA, 1968), AFIPS ’68 (Spring), ACM, pp. 37–45.

[BEL*07] Boulos S., Edwards D., Lacewell J. D., Kniss J., Kautz J., Shirley P., Wald I.: Packet-based whitted and distribution ray tracing. In *Proceedings of Graphics Interface 2007* (New York, NY, USA, 2007), GI ’07, Association for Computing Machinery, pp. 177–184.

[BWB19] Boksanský J., Wimmer M., Bittner J.: Ray traced shadows: Maintaining real-time frame rates. In *Ray Tracing Gems: High-Quality and Real-Time Rendering with DXR and Other APIs*, Haines E., Akenine-Möller T., (Eds.). Apress, Berkeley, CA, 2019, pp. 159–182.

[Car00] Carmack J.: Email communication with Mark Kilgard. online, 26 May 2000. https://fabiensanglard.net/doom3_documentation/CarmackOnShadowVolumes.txt.

[CD04] Chan E., Durand F.: An efficient hybrid shadow rendering algorithm. In *Proceedings of the Fifteenth Eurographics Conference on Rendering Techniques* (Goslar, DEU, 2004), EGSR’04, Eurographics Association, pp. 185–195.

[CF89] Chin N., Feiner S.: Near real-time shadow generation using bsp trees. In *Proceedings of the 16th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1989), SIGGRAPH ’89, ACM, pp. 99–106.

[Cre99] Creative Technology Ltd: Method for rendering shadows using a shadow volume and a stencil buffer. US Patent US6384822B1, 1999. https://patents.google.com/patent/US6384822B1/en.

[Cro77] Crow F. C.: Shadow algorithms for computer graphics. *SIGGRAPH Computer Graphics 11*, 2 (July 1977), 242–248.

[DFY14] Du W., Feng J., Yang B.: Sub-pixel anti-aliasing via triangle-based geometry reconstruction. *Computer Graphics Forum 33*, 7 (Oct. 2014), 81–90.

[DMAG18] Deves F., Mora F., Aveneau L., Ghazanfarpour D.: Scalable real-time shadows using clustering and metric trees. In *Proceedings of the Eurographics Symposium on Rendering: Experimental Ideas & Implementations* (Goslar Germany, Germany, 2018), SR ’18, Eurographics Association, pp. 83–93.

[EK02] Everitt C., Kilgard M. J.: *Practical and Robust Stenciled Shadow Volumes for Hardware-Accelerated Rendering*. Tech. rep., nVidia Corporation, 2002.

[ESAW11] Eisemann E., Schwarz M., Assarsson U., Wimmer M.: *Real-Time Shadows*, 1st ed. A. K. Peters, Ltd., Natick, MA, USA, 2011.

[FZW*20] Fu Z., Zhang H., Wang R., Li Z., Yang P., Sheng B., Mao L.: Dynamic shadow rendering with shadow volume optimization. In *Advances in Computer Graphics* (Cham, 2020), Magnenat-Thalmann N., Stephanidis C., Wu E., Thalmann D., Sheng B., Kim J., Papagiannakis G., Gavrilova M., (Eds.), Springer International Publishing, pp. 96–106.

[GMAG15] Gerhards J., Mora F., Aveneau L., Ghazanfarpour D.: Partitioned shadow volumes. *Computer Graphics Forum 34*, 2 (May 2015), 549–559.

[Hei91] Heidmann T.: Real shadows real time. *IRIS Universe 18* (November 1991), 28–31.

[HHLH05] Hornus S., Hoberock J., Lefebvre S., Hart J.: Zp+: Correct z-pass stencil shadows. In *Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2005), I3D ’05, Association for Computing Machinery, pp. 195–202.

[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*, 4 (Oct. 2005), 1462–1482.

[JMB04] Johnson G. S., Mark W. R., Burns C. A.: *The Irregular Z-Buffer and its Application to Shadow Mapping*. Tech. Rep. TR-04-09, The University of Texas at Austin, 2004.

[KKT08] Kim B., Kim K., Turk G.: A shadow-volume algorithm for opaque and transparent nonmanifold casters. *Journal of Graphics, GPU and Game Tools 13* (2008), 1–14.

[KMH19] Kobrtek J., Milet T., Herout A.: Silhouette extraction for shadow volumes using potentially visible sets. In *International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG)* (2019), Union Agency, pp. 9–16.

[LCF17] Li H., Cao Y., Feng X.: Analysis of hard shadow antialiasing. In *Advances in Computer Science and Ubiquitous Computing* (Singapore, 2017), Park J. J. J. H., Pan Y., Yi G., Loia V., (Eds.), Springer Singapore, pp. 421–429.

[LMSG14] Lecocq P., Marvie J.-E., Sourimant G., Gautron P.: Sub-pixel shadow mapping. In *Proceedings of the 18th Meeting of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2014), I3D ’14, Association for Computing Machinery, pp. 103–110.

© 2021 Eurographics - The European Association for Computer Graphics and John Wiley & Sons Ltd

[LSL11] LAURITZEN A., SALVI M., LEFOHN A.: Sample distribution shadow maps. In *Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2011), I3D '11, Association for Computing Machinery, pp. 97–102.

[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 Rendering: Experimental Ideas & Implementations* (Goslar, Germany, 2016), EGSR '16, Eurographics Association, pp. 73–83.

[MKZP14] MILET T., KOBRTEK J., ZEMČÍK P., PEČIVA J.: Fast and robust tessellation-based silhouette shadows. In *WSCG 2014 - Poster papers proceedings* (2014), University of West Bohemia in Pilsen, pp. 33–38.

[MNZ15] MILET T., NAVRÁTIL J., ZEMČÍK P.: An improved non-orthogonal texture warping for better shadow rendering. In *WSCG 2015 - Full Papers Proceedings* (2015), Union Agency, pp. 99–107.

[Mor66] MORTON G. H.: A Computer Oriented Geodetic Data Base and a New Technique in File Sequencing. Tech. rep., IBM Ltd, Ottawa, Canada, 1966.

[MT04] MARTIN T., TAN T.-S.: Anti-aliasing and continuity with trapezoidal shadow maps. In *Eurographics Workshop on Rendering*, Keller A., Jensen H. W., (Eds.). The Eurographics Association, 2004, pp. 153–160.

[OPB15] OLSSON O., PERSSON E., BILLETER M.: Real-time many-light management and shadows with clustered shading. In *ACM SIGGRAPH 2015 Courses* (New York, NY, USA, 2015), SIGGRAPH '15, Association for Computing Machinery.

[ORM07] OVERBECK R., RAMAMOORTHI R., MARK W. R.: A real-time beam tracer with application to exact soft shadows. In *Proceedings of the 18th Eurographics Conference on Rendering Techniques* (Goslar, DEU, 2007), EGSR'07, Eurographics Association, pp. 85–98.

[PSM*13] PEČIVA J., STARKA T., MILET T., KOBRTEK J., ZEMČÍK P.: Robust silhouette shadow volumes on contemporary hardware. In *Conference Proceedings of GraphiCon'2013* (2013), GraphiCon Scientific Society, pp. 56–59.

[Ros12] ROSEN P.: Rectilinear texture warping for fast adaptive shadow mapping. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2012), I3D '12, Association for Computing Machinery, pp. 151–158.

[SJW07] SCHERZER D., JESCHKE S., WIMMER M.: Pixel-correct shadow maps with temporal reprojection and shadow test confidence. In *Proceedings of the 18th Eurographics Conference on Rendering Techniques* (Goslar, DEU, 2007), EGSR'07, Eurographics Association, pp. 45–50.

[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* (New York, NY, USA, 2014), I3D '14, ACM, pp. 111–118.

[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* (New York, NY, USA, 2011), SA '11, ACM, pp. 153:1–153:10.

[Sto15] STORY J.: Hybrid ray-traced shadows. *Game Developers Conference* (March 2015).

[Sto16] STORY J.: Advanced geometrically correct shadows for modern game engines. *Game Developers Conference* (March 2016).

[SWP11] SCHERZER D., WIMMER M., PURGATHOFER W.: A survey of real-time hard shadow mapping methods. *Computer Graphics Forum 30* (03 2011), 169–186.

[Uhl91] UHLMANN J. K.: Satisfying general proximity/similarity queries with metric trees. *Information Processing Letters 40*, 4 (1991), 175–179.

[USB*19] USTA B., SCANDOLO L., BILLETER M., MARROQUIM R., EISEMANN E.: A Practical and Efficient Approach for Correct Z-Pass Stencil Shadow Volumes. In *High-Performance Graphics - Short Papers* (2019), Steinberger M., Foley T., (Eds.), The Eurographics Association.

[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* (New York, NY, USA, 2015), i3D '15, ACM, pp. 15–23.

[Wil78] WILLIAMS L.: Casting curved shadows on curved surfaces. *SIGGRAPH Computer Graphics 12*, 3 (Aug. 1978), 270–274.

[Woo12] WOO A.: *Shadow Algorithms Data Miner*. CRC Press, Hoboken, NJ, 2012.

[WSP04] WIMMER M., SCHERZER D., PURGATHOFER W.: Light space perspective shadow maps. In *Proceedings of the Fifteenth Eurographics Conference on Rendering Techniques* (Goslar, DEU, 2004), EGSR'04, Eurographics Association, pp. 143–151.

[ZSXL06] ZHANG F., SUN H., XU L., LUN L. K.: Parallel-split shadow maps for large-scale virtual environments. In *Proceedings of the 2006 ACM International Conference on Virtual Reality Continuum and Its Applications* (New York, NY, USA, 2006), VRCIA '06, Association for Computing Machinery, pp. 311–318.

© 2021 Eurographics - The European Association for Computer Graphics and John Wiley & Sons Ltd