International Conference on Artificial Reality and Telexistence
Eurographics Symposium on Virtual Environments (2015)
M. Imura, P. Figueroa, and B. Mohler (Editors)

Fast robust and precise shadow algorithm for WebGL 1.0 platform

T. Milet, M. Tóth, J. Pečiva, T. Starka, J. Kobrtek and P. Zemčík

Brno University of Technology, Czech Republic

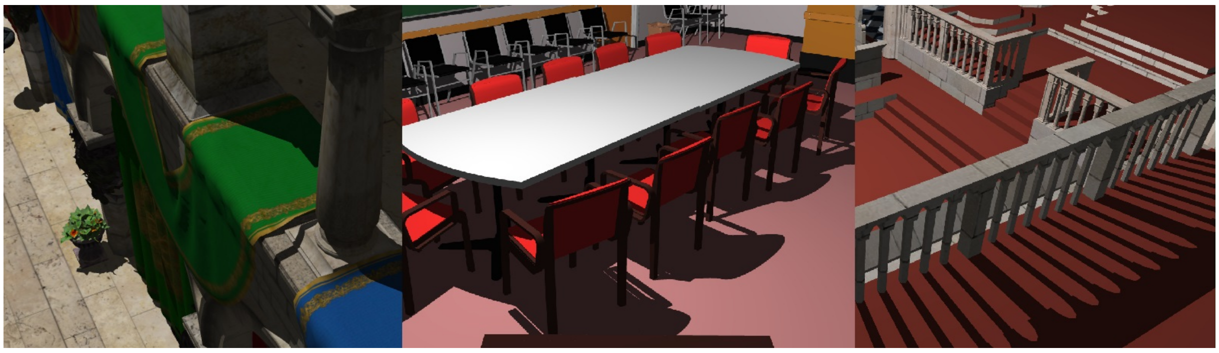


Figure 1: *Three testing scenes - Sponza, Conference Room and Sibenik*

Abstract
This paper presents fast and robust per-sample correct shadows for WebGL platform. The algorithm is based on silhouette shadow volumes and it rivals the standard shadow mapping performance. Our performance is usually superior when compared with high resolution shadow maps. Moreover, it does not suffer from a number of artefacts of shadow mapping and always provides per-pixel correct results.
WebGL 1.0 provides just vertex and fragment shaders. Thus, we put all our algorithms evaluating silhouette edges to vertex shaders. Specially precomputed data are fed to the vertex shaders that extrude shadow volume sides just for silhouette edges. Some optimizations are deployed for performance and data size reasons that are important especially on low performance configurations, such as cost-effective tablets and mobile phones. The paper evaluates our solution on number of models. Our solution performs on par with high resolution omni-directional shadow mapping.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture I.3.6 [Computer Graphics]: Methodology and Techniques—Graphics data structures and data types

1. Introduction

Shadows are an important visual cue for human perception of 3D space, providing human brain with additional information about the structure of the scene that is usually visualized as a 2D image on the computer screen.

Two popular methods for shadow visualizations are used nowadays - shadow volumes [Cro77] and shadow mapping [Wil78]. Shadow mapping got its hardware acceleration in 2001 with GeForce 3 and became popular shadow method in computer games and other areas. However, shadow mapping is prone to visual artefacts [LGMM07]. Tremendous amount of research was done to lessen these artifacts to some extent, for instance [SD02] [WSP04] [ZSXL06] [Ros12]. Another approach is to adjust the scene design in a way that artefacts will not become visible. Such approach is probably largely used in computer game industry but it is not acceptable in CAD or toher systems, where precise visualization of the model is required.

Shadow volumes got their hardware support in 1991 when hardware stencil buffer was introduced [Hei91]. Unfortunately, Heidmann's algorithm, that became known as z-pass, was not robust when the viewer himself was in shadow. Thus, [CK00] and [BS99] proposed z-fail algorithm. Finally, [EK02] presented robust z-fail algorithm that should provide correct visual results for any arbitrary scene.

As shadow volumes work partially in image space, they are capable to deliver per-pixel precise shadows. However, rasterizing large shadow volume extents made them fill-rate limited in most cases. The extrusion of shadow volumes only from silhouette edges, instead of every edge, became the main performance optimization. The algorithm for finding silhouette edges on manifold meshes was described by [Ber86]. [BS03] was the first to show the silhouette algorithm implemented entirely in graphics hardware. [McG04] described silhouette algorithm implemented in vertex shader only using specially precomputed mesh. [vW06] developed optimized silhouette construction using SSE2 instructions for computer game Doom 3. [SWK07] used then-new geometry shader for silhouette algorithm.

Many silhouette algorithms require 2-manifold triangle meshes to work correctly. More general algorithms were proposed to alleviate this restriction. [AW04] requires the mesh to be orientable only. [KKT08] showed the algorithm that works on arbitrary meshes however it exhibits visual artefacts from time to time due to limited numerical precision of both GPU and CPU computing units. [PSM*13] made the algorithm robust and verified its artefact-free property on a number of computing platforms. [MKZP14] showed the further performance gains when implemented using tessellation shaders.

2. Algorithm

Our solution is based on the robust silhouette algorithm developed by [PSM*13] and enhanced algorithm by [MKZP14]. However, they present CPU, multi-core CPU, geometry shader, OpenCL and tessellation shader implementations in their papers, while the only GPU computing capabilities available in WebGL 1.0 are vertex and fragment shaders. For our solution, we have chosen vertex shader and took the idea of [McG04] to feed vertex shader by a specially constructed mesh data. These mesh data processed by vertex shaders will extrude shadow volume only on silhouette edges. [McG04] designed solution just for 2-manifold meshes so we had to design a new solution, merging McGuire's approach with the algorithm presented in [PSM*13]. Furthermore, we developed a number of data-related optimizations that will be described later.

2.1. Overview

The core idea of the algorithm is to find the edges that form the outline of the possible silhouette of the model for the given light position. Then, the edges of the silhouette are extruded to infinity and closed by caps at the model and at the infinity as required by z-fail algorithm, forming the shadow volume of the model. When we refer to silhouette of a model, we mean subset of all edges of the model that satisfies the following condition: edge is considered as silhouette edge when number of light-facing and light-back-facing triangles adjacent to this edge is not equal. We will call the extruded quads as shadow volume sides, cap at the model as front cap and cap at the infinity as back cap.

The computed shadow volume is used for shadow visualization using traditional stencil z-fail approach that is described in detail in [EASW09]. Two-sided stenciling optimization is used as it is supported by WebGL 1.0. Briefly, the algorithm works in three steps:

1. Render the regular scene, producing z-values to the z-buffer and producing scene with ambient light to the color buffer
2. Render the stencil mask using the shadow volume geometry and set stencil function to act whenever z-test fails; front faces are set to increment and back faces to decrement stencil value on z-test fail.
3. Render the regular scene with the light switched on while setting stencil test in such a way that the color buffer is updated only in places with zero stencil value, e.g. update only lit regions.

The core of our algorithm lies within the step 2.. We use two shaders in this step - a shader for sides-data and a shader for caps-data. These shaders computes Shadow Volume (SV) for particular scene transformation and light position.

2.2. Basic Input Data

First step of our algorithm is the construction of shadow-geometry. The shadow-geometry is composed of two parts: data for SV sides (sides-data) and data for SV caps (caps-data) (see Figure 5).

Data for sides are composed of all edges of input scene. Every edge is described with its vertices and a set of opposite vertices $\mathfrak{O} = \{\mathbf{O}_0, \mathbf{O}_1, \ldots, \mathbf{O}_{n-1}\}$. An edge and its opposite vertices form triangles that are attached to that edge. There are also additional data for every edge (see subsections 2.4 and 2.5).

Caps-data are composed of triangles of input scene with additional information.

We construct these two parts in preprocessing part of our algorithm. They are both stored in separate files along the scene files. Though that it's not necessary, we chose to trade the on load construction for download time. Neither one of the two approaches influences the measurements of performance.

2.3. Multiplicity computation

The multiplicity is computed as follows: We construct a plane using the light position and edge's two vertices. We call it a lightplane. A lightplane divides space into two subspaces. Then, we iterate through the opposite vertices of the edge and count their number in both light plane subspaces. Difference between these two numbers is the edge multiplicity, see Figure 2. The orientation of the lightplane determines which number we substract from which. It needs to be consistent. Then if it is zero, the edge is not silhouette edge and no edge extrusion happens. If it is non-zero, then absolute value of multiplicity gives the number how many times we need to extrude the side. The number of extrusion translates directly to the number of stencil buffer incrementations introduced by this side of the shadow volume. The sign of the edge multiplicity gives the extruded quad winding order, which will either increment or decrement stencil buffer value.

2.4. Vertex Shader for Sides

In order to compute multiplicity, VS has to receive an edge, a set of opposite vertices and a light source. A light source position can be set using uniform variable. We use vertex attributes for edges and sets of opposite vertices (see. Figure 3).

In this section we will closely describe shader for sides and format of sides-data. Every edge of the scene can extrude n sides of SV (see [KKT08]), where n is number of attached triangles to this edge, see Figure 2. We label n as maximal multiplicity - $maxMult$. We label computed current multiplicity as $curMult$ and it has the following property: $-maxMult \leq curMult \leq maxMult$. A computed multiplicity determines the number of sides extruded from the edge. We use the algorithm for computation of multiplicity proposed in [MKZP14].

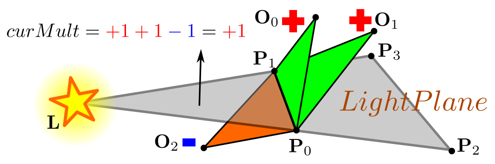


Figure 2: *Edge $\mathbf{P}_0 \rightarrow \mathbf{P}_1$ has $n = 3$ opposite vertices: $\mathfrak{O} = \{\mathbf{O}_0, \mathbf{O}_1, \mathbf{O}_2\}$. Its maximum multiplicity is $n = 3$. Current multiplicity of this edge, given by light source and transformation, is $+1$. One quad $\mathbf{P}_0, \mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3$ is extruded using edge and light source. This quad has the same orientation as light plane (black arrow).*

We speed up computation of multiplicity using parallelization by vertex shaders (VS). In order to prevent writes and reads from memory we compute and draw shadow-data in one step. To make this streaming concept possible, VS has to receive all data for all potential SV sides. Let us denote an index of potential side as s.

One side of the volume is composed of 2 triangles and 6 vertices. Let us denote an index of vertex as v. The vertex shader has to receive $maxMult \cdot 6$ vertices for every edge. As example, that is 18 vertices for $maxMult = 3$. For every vertex, VS computes current multiplicity - $curMult$. $curMult$ determines whether a vertex is useful or not.

Vertex that lies on such side for which $s > |curMult|$ is useless. All useless vertices are transformed to $(0,0,0)$. This transformation ensures that useless sides of SV will be degenerated and will not be rasterized.

Useful vertices have to be moved to one of the four possible positions: $\mathbf{P}_{id}$ (see Figure 2):

$$toInf(\mathbf{P},\mathbf{L}) = (p_x l_w - l_x, p_y l_w - l_y, p_z l_w - l_z, 0) \quad (1)$$

$$\begin{aligned} \mathbf{P}_0 &= (a_x, a_y, a_z, 1) \\ \mathbf{P}_1 &= (b_x, b_y, b_z, 1) \\ \mathbf{P}_2 &= toInf(\mathbf{P}_0, \mathbf{L}) \\ \mathbf{P}_3 &= toInf(\mathbf{P}_1, \mathbf{L}) \end{aligned} \quad (2)$$

In equation 2 $\mathbf{P}_0 \rightarrow \mathbf{P}_1$ symbolizes an edge and $\mathbf{L} = (l_x, l_y, l_z, l_w)$ symbolizes light position. id depends on index of vertex v and computed multiplicity $curMult$ according to following equations:

$$id = \begin{cases} idCCW & \text{if } curMult > 0 \\ idCW & \text{otherwise} \end{cases} \quad (3)$$

$$idCW = \begin{cases} 6 - v & \text{if } v > 2 \\ v & \text{otherwise} \end{cases} \quad (4)$$

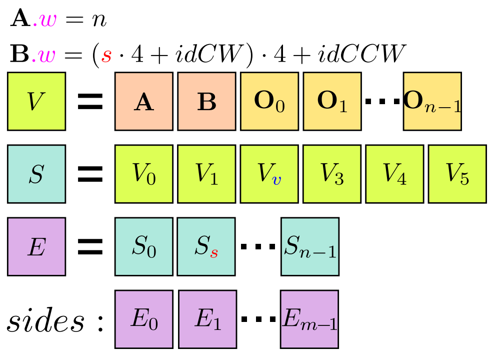
$$idCCW = \begin{cases} v - 2 & \text{if } v > 2 \\ 2 - v & \text{otherwise} \end{cases} \quad (5)$$

Pseudo code implemented in VS for sides-data (Figure 3) is in Algorithm 1.

2.5. Vertex Shader for Caps

In order to compute multiplicity, VS has to receive all vertices of triangles and light source position. Similarly to sides, all vertices of a triangle are sent to VS using vertex attributes (see Figure 4).

A shader for SV caps works with scene triangles. Six vertices are needed in order to create a couple of caps (front and back cap). The vertex shader is therefore executed $6 \cdot m$ times, where m is the number of scene triangles.



$\mathbf{A}.w = n$
$\mathbf{B}.w = (s \cdot 4 + idCW) \cdot 4 + idCCW$
$V = \mathbf{A}\ \mathbf{B}\ \mathbf{O}_0\ \mathbf{O}_1 \cdots \mathbf{O}_{n-1}$
$S = V_0\ V_1\ V_v\ V_3\ V_4\ V_5$
$E = S_0\ S_s \cdots S_{n-1}$
$sides : E_0\ E_1 \cdots E_{m-1}$

Figure 3: *The image shows data structure for sides of SV. Every vertex (V) contains $n+2$ vectors of size 4.* **A** *and* **B** *are vertices of an edge and* $\mathbf{O}_j$ *are opposite vertices. Forth components of* **A** *and* **B** *are used for special purposes - storing number of opposite vertices, index of side and indices idCW, idCCW. Side (S) contains 6 vertices (two triangles). If maxMult is n, there has to be n sides for every edge (E). Sides data is composed of m edges of input model.*

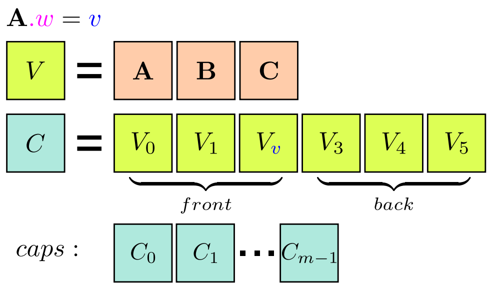
Algorithm 1: Pseudo code for one vertex of a volume side extrusion in Vertex Shader.

```
Data: Edge vertices $\mathbf{A}, \mathbf{B}$, set $\mathfrak{O}$, light position $\mathbf{L}$,
      side id $s$, indices $idCCW$, $idCW$,
      model-view projection matrix $\mathbf{M}$
Result: $gl\_Position$
compute vertices $\mathbf{P}_i$ according to equation 2;
$curMult$ = computeMultiplicity($\mathbf{A}, \mathbf{B}, \mathfrak{O}, \mathbf{L}$);
if $s < |curMult|$ then
    compute $id$ according to equation 3;
    $gl\_Position = \mathbf{M} \cdot \mathbf{P}_{id}$;
else
    $gl\_Position = (0,0,0,0)$;
```

In order to prevent errors, both caps have to properly oriented. The orientation has to be the same as orientation of sides and has to be performed deterministically. We use deterministic computation of multiplicity destribed in [MKZP14] to do so. Compared to sides, there is only one opposite vertex.

First, VS finds reference edge using method described in [MKZP14]. Then it computes multiplicity using third vertex and reference edge. The multiplicity determines orientation of particular triangle.

Furthermore, we propose a method that shifts front cap to infinity. After transforming the vertex into the clip space, we simply set its z component to its w. The shifting can be seen on Figure 5. This ensures that the front cap always fails the depth test. We avoid the far plane clipping by using the modified projection matrix, which effectively sets the far plane into infinity [MHE*03]. Let $\mathbf{A} = (a_x, a_y, a_z, a_w)$ be vertex of



$\mathbf{A}.w = v$
$V = \mathbf{A}\ \mathbf{B}\ \mathbf{C}$
$C = \underbrace{V_0\ V_1\ V_v}_{front}\ \underbrace{V_3\ V_4\ V_5}_{back}$
$caps : C_0\ C_1 \cdots C_{m-1}$

Figure 4: *The image shows data structure for caps of SV. Every vertex (V) contains 3 vectors of size 4.* **A**, **B**, **C** *are vertices of scene's triangles. Forth component of* **A** *is used for special purposes - storing index of vertex v. Six vertices forms couple of front and back cap. Caps data is composed of m couples for m triangles of input model.*

front cap in clip space. The shifted vertex **B** toward infinity from camera can be computed according to equation 6:

$$\mathbf{B} = (b_x, b_y, b_z, b_w) = (a_x, a_y, a_{\mathbf{w}}, a_{\mathbf{w}}) \tag{6}$$

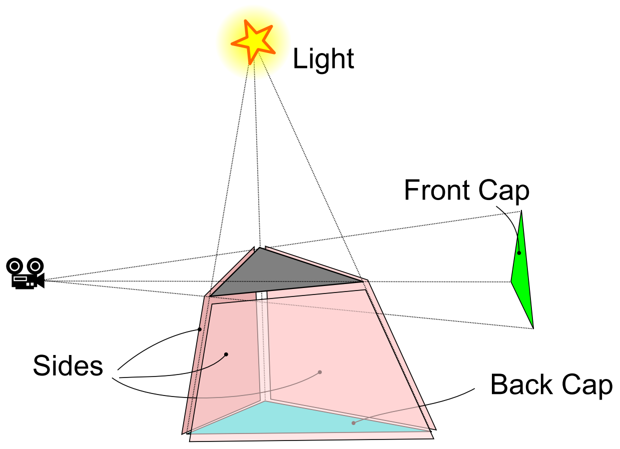


Figure 5: *The image shows shadow volume that is constructed from a triangle. The front cap is shifted to infinity using homogeneous coordinates in order to prevent self-shadowing artifacts.*

Pseudo code implemented in VS for caps-data (Figure 4) can be seen in Algorithm 2.

2.6. Data size optimizations

The size of shadow-geometry can be large in comparison to size of input scene. For example, for input scene having $e = 1000$ edges and $maxMult = 3$, the size of sides-data can be evaluated using equation 7.

$$e \cdot maxMult \cdot 6 \cdot (2 + maxMult) \cdot 4 \cdot 4 = 1440000[bytes] \tag{7}$$

Algorithm 2: Pseudo code for caps in Vertex Shader.

```
Data: Triangle vertices $\mathbf{P}_0, \mathbf{P}_1, \mathbf{P}_2$, light position
      $\mathbf{L}$, vertex id $v$, model-view projection
      matrix $\mathbf{M}$
Result: $gl\_Position$
1  $curMult$ = computeMultiplicity($\mathbf{P}_0, \mathbf{P}_1, \mathbf{P}_2, \mathbf{L}$);
2  if $curMult = 0$ then
3      $gl\_Position = (0,0,0,0)$;
4      return;
5  if $curMult < 0$ then
6      swap($\mathbf{P}_0, \mathbf{P}_1$);
7  if $v < 3$ then
8      $\mathbf{V} = \mathbf{M} \cdot \mathbf{P}_v$;
9      $gl\_Position = (V_x, V_y, V_w, V_w)$;
10 else
11     $gl\_Position = \mathbf{M} \cdot toInf(\mathbf{P}_{5-v}, \mathbf{L})$;
```

Every edge can produce up to 3 sides. Every side contains 6 vertices. Every vertex contains 5 vertex attributes with size of 4 floats.

Size of sides-data can be reduced $maxMult \cdot 6$ times. In the example shown in equation 7, the reduced size would be $80000[bytes]$. Reduction can be achieved by using combination of instancing and drawing a shadow-geometry using multiple draw calls. The instancing is present in every current mainstream browser as an extension.

Modification is simple. First step is to remove additional data $idCW$, $idCCW$ and s from vertex attribute $B.w$, see Figure 3. After this step, every vertex V in edge E in the Figure 3 has the same data. Therefore, it is necessary to compute $idCW$ and $idCCW$ inside VS using $gl\_VertexID$ and equations 3,4 and 5. Unfortunately, WebGL does no support $gl\_VertexID$, so we have to simulate its behaviour using additional vertex attribute. Also, the variable s has to be sent to VS using another vertex attribute. The variable s can be also computed using $gl\_InstanceID$, but it is not present in current WebGL. Alternatively, we could use uniform variable and multiple drawcalls, which is what we do. The final equation for computation of size of sides-data is as follows:

$$16 \cdot (2 + maxMult) \cdot e[bytes] \tag{8}$$

e is number of edges in input scene.

Size of caps-data can be also reduced using similar approach. The final equation for computation of size of caps-data is as follows:

$$48 \cdot t[bytes] \tag{9}$$

where t is number of triangles in input scene.

3. Measurements and evaluation

Our evaluation focused on two aspects:

- performance comparison with shadow mapping methods
- performance on various devices using popular scenes

For comparison with the shadow mapping, we considered that shadow volumes provide omnidirectional shadows while shadow mapping only directional. For omnidirectional shadows, cube shadow mapping need to be utilized.

We used standard shadow mapping instead of one of its optimized versions, such as LiSPSM [WSP04] or CSM [Dim07]. These optimized versions focus on shadow visual quality and do not improve the performance.

For the second point, we evaluated our algorithm on several test scenes and hardware configurations, including handheld devices. Desktop measurements were carried out on high-end graphics hardware. Handheld platform was represented by NVIDIA SHIELD tablet with Tegra K1 SOC running Android 5.0.1, desktop platform by Radeon R9 290X and GeForce GTX 980 running in system with Intel Core i7 4790, 16GB of memory and Windows 7. We used Firefox browser version 36.0 on both - desktop and NVIDIA SHIELD tablet. Firefox browser was chosen as it provided the best means for measurements. WebGL 1.0 lacks support for OpenGL queries, so we had to opt for timing draw commands ourselves, which was not possible in some browsers like Google Chrome and Opera because glFinish is implemented asynchronously in these browsers. We measured our algorithm both using ANGLE and with disabled ANGLE.

We have chosen some popular well-known scenes: Crytek Sponza, Conference Room and Sibenik Cathedral using scenes flythrough.

Every test scene contains single point-light source. The canvas resolution for flythroughs was 1920x1080, unless specified otherwise. On NVIDIA SHIELD Tablet, we were able to create cube map having only 2k by 2k texels per side due to hardware limitations.

3.1. Popular scenes

For our measurements on popular scenes, we have chosen following well-known models Crytek Sponza (262k triangles), Conference Room (331k triangles) and Sibenik Cathedral (75k triangles). Performance on these scenes has been measured by camera flythrough. Averaged frame rendering time for each scene is shown in the tables 1, 2.

As can be seen in the Table 1, our shadow volumes are faster than 8k omnidirectional shadow mapping (e.g. 8192x8192x6) on both - high-end NVIDIA and AMD cards. Even on shadow mapping optimized pipeline, we are slightly faster. This is no longer true for 4k and 2k shadow mapping, however 2k and 4k shadow mapping does not provide sufficient shadow resolution, see Figure 6. On the other side,

scene	Sponza		Conf. room		Sibenik	
method	R290	G980	R290	G980	R290	G980
no shadow	3.3	3.9	1.0	0.9	0.7	1.1
SM1	13.5	13.8	4.0	3.5	1.7	1.6
SM2	13.4	14.1	5.2	3.8	3.0	2.4
SM4	16.1	15.3	9.8	6.4	8.5	5.7
SM8	37.0	21.8	28.2	15.5	28.9	18.7
SV	14.5	19.2	7.4	11.0	7.1	10.1

Table 1: *Average frame time (ms) for flythroughs using ANGLE.*

scene	Sponza			Conf. room			Sibenik		
method	R290	G980	Shield	R290	G980	Shield	R290	G980	Shield
no shadow	2.5	2.5	31.8	1.0	1.3	21.5	0.8	1.1	12.9
SM1	6.8	5.6	77.8	3.8	2.9	43.9	1.9	1.8	37.1
SM2	8.0	5.6	75.9	5.3	3.5	57.5	3.6	2.8	55.0
SM4	15.0	8.4	0.0	11.2	6.0	0.0	10.3	6.0	131.2
SM8	41.7	22.9	0.0	34.2	16.0	0.0	36.0	18.8	0.0
SV	14.7	22.2	207.5	8.0	15.3	165.4	7.6	13.1	73.0

Table 2: *Average frame time (ms) for flythroughs without using ANGLE.*

our shadow volumes always provide per-pixel precise solution or per-fragment, if antialiasing is used, because shadow volumes perform its computation in screen space.

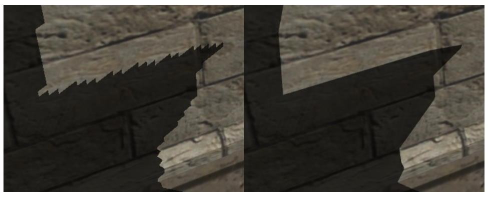


Figure 6: *The image shows diference in quality between shadow mapping and shadow volumes.*

We performed more detailed analysis of our flythrough animation. The frame times are shown in the Figure 7, Figure 8 and Figure 9. As can be seen, shadow volume performance is view dependent. Anyway, the performance is higher than 8k shadow mapping for about two thirds of our flythrough in Crytek Sponza scene.

On Radeon R9 290X, the performance of shadow volumes is more stable as shown in the graphs 8. Shadow volumes performance roughly equals to the performance of 4k shadow mapping, sometimes even attacking 2k shadow maps.

Finally, Figure 9 shows the performance on NVIDIA SHIELD tablet. The performance is not on interactive level, but we consider 4 FPS in FullHD on tablet for such large scene as acceptable. The performance is more stable than on GeForce GTX 980 (only about 20% fluctuation compared to 40%) and roughly only ten times slower than high-end NVIDIA GPU. This seems to go in hand with number of shader processors available on the devices, e.g. 192 vs. 2048. Unfortunately, we have not been able to create larger cube shadow maps on NVIDIA SHIELD device and measure them. We only succeeded with 2k cube map.

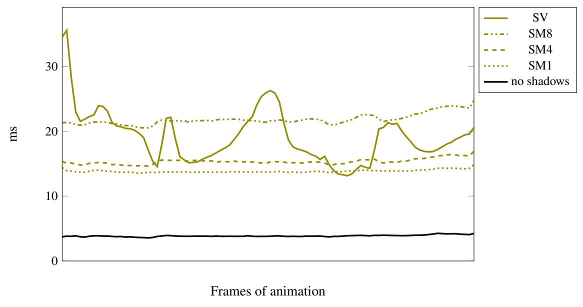


Figure 7: *The figure shows flythrough Crytek Sponza on GeForce GTX 980. We observe that SM is very well optimized, even for large textures. But SV also manage to be real-time for most view points, although not as stable.*

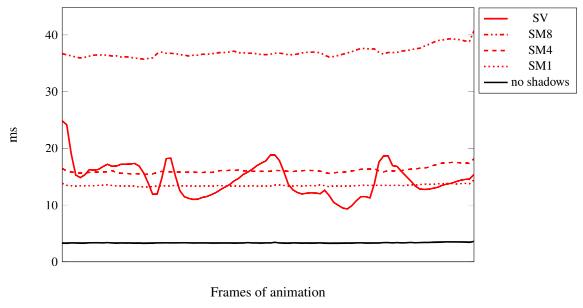


Figure 8: *The figure shows flythrough Crytek Sponza On AMD platform. The SV seem to provide an interesting aleternative to SM. While it behaves more stable than on NV platform, it also manages to be faster than 4k SM in average and completely outperforming 8K SM.*

3.2. In-depth performance analysis

We measured the time of all rendering stages of our application. The stages are:

1. rendering of ambient scene
2. silhouette edge set evaluation in vertex shaders
3. rasterization of shadow volumes
4. blending of lit parts of the scene

The results depend on output video resolution. Thus, we made the tests on four output resolutions: 800x600, 1920x1080, 2560x1600 and 3840x2160, that corresponds roughly to 0.5, 2, 4 and 8 megapixels video output. The test was carried out on Sponza flythrough.

First, we show the results of the flythrough in figures, but only for 1920x1080 resolutions due to limited paper space.

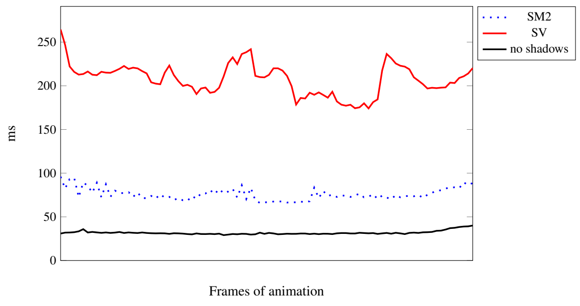


Figure 9: *The figure shows flythrough Crytek Sponza on NVIDIA SHIELD. Both methods perform only in interactive time. Although SM performs better here, it must be taken into account that it is only 2k cubemap, compared to previous results where SV are on-par with 4K SM and difference between 2K SM and SV were much bigger. Thus quality-wise, shadow volumes are better alternative to SM also on this platform.*

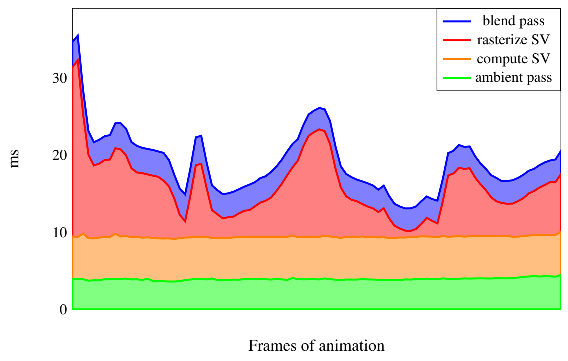


Figure 10: *Timings for stages for Sponza scene on NVIDIA GeForce GTX 980 - 1920x1080*

Then, we present all the results for all output resolution in the tables using just average values.

As can be seen in Figure 10, the shadow volume rasterization is the most varying part. As the scene view changes during the animation, the shadow volumes changes occupied space on the screen, sometimes producing smaller, sometimes higher rasterization work.

Table 3 and Table 4 show average times of all four rendering stages on Sponza flythrough for four output resolutions. As can be seen from the results, the first and last stage (ambient scene rendering and blending of lit scene) grows rather slowly with increasing output resolution. For resolution increase from 0.5 to 8 megapixels, e.g. 16 times, ambient scene rendering time increases only about 50% on GeForce GTX 980 and by 25% on Radeon R9 290X. For blending stage, GeForce GTX 980 time increases only about 8% and Radeon R9 290X time increases by 25%. The second stage (silhouette edge set evaluation) takes constant time. Shadow volume rasterization grows rapidly. About 36 times on GeForce GTX 980 and about 16 times on Radeon R9 290X .

stage	800x600	1920x1080	2560x1600	3840x2160
4	3.1	3.1	3.4	3.8
3	0.8	6.7	14.3	28.8
2	5.3	5.5	5.2	5.8
1	3.6	3.9	4.6	5.3

Table 3: *GeForce GTX 980 average times in milliseconds for Sponza Flythroughs for all four stages and resolutions.*

stage	800x600	1920x1080	2560x1600	3840x2160
4	1.8	1.9	2.0	2.3
3	1.5	6.1	12.2	24.0
2	3.1	3.2	3.1	3.2
1	3.5	3.5	3.9	4.4

Table 4: *Radeon R9 290X average times in milliseconds for Sponza Flythroughs for all four stages and resolutions.*

4. Conclusion and future work

This paper presented fast and precise method for rendering shadows in web applications using WebGL API. Despite limitations of WebGL 1.0 (only vertex and fragment shader, no queries, etc.), our algorithm is able to compute silhouette shadow volumes on GPU. The algorithm is robust and works with any arbitrary triangle meshes, producing precise per-sample correct shadows.

We tested the algorithm on number of well-known scenes and it proved to be faster than omnidirectional shadow mapping with resolution 8k on GeForce GTX 980 and Radeon R9 290X while not suffering from any artefacts seen on shadow mapping. On Radeon R9 290X, it performs even better and outperforms even 4k omnidirectional shadow mapping. Among the algorithms for precise shadows, our algorithm is the fastest one for WebGL up to our knowledge.

Contribution of the paper is the silhouette edge evaluation implemented in vertex shader, shadow data space optimizations to limit large amounts of shadow-data transmitted over network and robust rendering of shadow volume caps by moving them to infinity.

For the future work, we consider to lower bandwidth requirements by not transmitting the topology data for shadow rendering but generating them on the client side from the scene data. We are considering to use WebGL for this task as we are not sure about Javascript performance for such computationally expensive task. We also consider a kind of hierarchical edge evaluation that would drop large number of edges if they do not contribute to the shadows on the screen. A tempting idea is to use compute shaders for the algorithm once it is available as WebGL extension or included in future WebGL releases.

Acknowledgements

This work was supported by the Ministry of Education, Youth and Sport of the Czech Republic under the research program TE01020415 (V3C - Visual Computing Competence Center) and The IT4Innovations Centre of Excellence (IT4I, ED1.1.00/02.0070). Additional data and algorithms were provided by Cadwork. We gratefully acknowledge the support of NVIDIA Corporation with the donation of the SHIELD tablets for this research.

References

[AW04] Aldridge G., Woods E.: Robust, geometry-independent shadow volumes. In *Proceedings of the 2nd international conference on Computer graphics and interactive techniques in Australasia and South East Asia* (New York, NY, USA, 2004), GRAPHITE '04, ACM, pp. 250–253. URL: http://doi.acm.org/10.1145/988834.988877, doi:10.1145/988834.988877. 2

[Ber86] Bergeron P.: A general version of crow's shadow volumes. *IEEE Computer Graphics and Applications 6* (1986), 17–28. 2

[BS99] Bilodeau B., Songy M.: Real time shadows. *Creativity 1999* (1999). 2

[BS03] Brabec S., Seidel H.-P.: Shadow volumes on programmable graphics hardware. *Computer Graphics Forum (Eurographics) 2003* (2003), 433–440. 2

[CK00] Carmack J., Kilgard M.: John Carmack on Shadow Volumes. *NVIDIA Developer Zone* (2000). URL: http://web.archive.org/web/20090127020935/http://developer.nvidia.com/attach/6832. 2

[Cro77] Crow F. C.: Shadow algorithms for computer graphics. In *Proceedings of the 4th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1977), SIGGRAPH '77, ACM, pp. 242–248. URL: http://doi.acm.org/10.1145/563858.563901, doi:10.1145/563858.563901. 1

[Dim07] Dimitrov R.: *Cascaded Shadow Maps*. Tech. rep., NVIDIA Corporation, August 2007. URL: http://developer.download.nvidia.com. 5

[EASW09] Eisemann E., Assarsson U., Schwarz M., Wimmer M.: Casting shadows in real time. In *ACM SIGGRAPH ASIA 2009 Courses* (New York, NY, USA, 2009), SIGGRAPH ASIA '09, ACM. URL: http://doi.acm.org/10.1145/1665817.1722963, doi:10.1145/1665817.1722963. 2

[EK02] Everitt C., Kilgard M. J.: *Practical and Robust Stenciled Shadow Volumes for Hardware-Accelerated Rendering*. Tech. rep., NVIDIA Corporation, 2002. URL: http://developer.nvidia.com. 2

[Hei91] Heidmann T.: Real shadow real time. IRIS Universe, pp. 28–31. 2

[KKT08] Kim B., Kim K., Turk G.: A shadow-volume algorithm for opaque and transparent nonmanifold casters. *J. Graphics Tools 13*, 3 (2008), 1–14. URL: http://dblp.uni-trier.de/db/journals/jgtools/jgtools13.html#KimKT08. 2, 3

[LGMM07] Lloyd D. B., Govindaraju N. K., Molnar S. E., Manocha D.: Practical logarithmic rasterization for low-error shadow maps. In *Proceedings of the 22Nd ACM SIGGRAPH/EUROGRAPHICS Symposium on Graphics Hardware* (Aire-la-Ville, Switzerland, Switzerland, 2007), GH '07, Eurographics Association, pp. 17–24. URL: http://dl.acm.org/citation.cfm?id=1280094.1280097. 1

[McG04] McGuire M.: Efficient shadow volume rendering. In *GPU Gems*, Fernando R., (Ed.). Addison-Wesley, 2004, pp. 137–166. 2

[MHE*03] McGuire M., Hughes J. F., Egan K., Kilgard M., Everitt C.: *Fast, Practical and Robust Shadows*. Tech. rep., NVIDIA Corporation, Austin, TX, Nov 2003. URL: http://developer.nvidia.com. 4

[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. URL: http://www.fit.vutbr.cz/research/view_pub.php?id=10587. 2, 3, 4

[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. URL: http://www.fit.vutbr.cz/research/view_pub.php?id=10408. 2

[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, ACM, pp. 151–158. URL: http://doi.acm.org/10.1145/2159616.2159641, doi:10.1145/2159616.2159641. 1

[SD02] Stamminger M., Drettakis G.: Perspective shadow maps. *ACM Trans. Graph. 21*, 3 (July 2002), 557–562. URL: http://doi.acm.org/10.1145/566654.566616, doi:10.1145/566654.566616. 1

[SWK07] Stich M., Wächter C., Keller A.: Efficient and robust shadow volumes using hierarchical occlusion culling and geometry shaders. In *GPU Gems 3* (2007), Nguyen H., (Ed.), Addison Wesley Professional, pp. 239–256. 2

[vW06] van Waveren J.: Shadow Volume Construction. *Intel Software Network* (2006). URL: http://fabiensanglard.net/doom3_documentation/37730-293752.pdf. 2

[Wil78] Williams L.: Casting curved shadows on curved surfaces. *SIGGRAPH Comput. Graph. 12*, 3 (Aug. 1978), 270–274. URL: http://doi.acm.org/10.1145/965139.807402, doi:10.1145/965139.807402. 1

[WSP04] Wimmer M., Scherzer D., Purgathofer W.: Light space perspective shadow maps. In *Rendering Techniques 2004 (Proceedings Eurographics Symposium on Rendering)* (June 2004), Keller A., Jensen H. W., (Eds.), Eurographics, Eurographics Association, pp. 143–151. URL: http://www.cg.tuwien.ac.at/research/publications/2004/Wimmer-2004-LSPM/. 1, 5

[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, ACM, pp. 311–318. URL: http://doi.acm.org/10.1145/1128923.1128975, doi:10.1145/1128923.1128975. 1