Eurographics Symposium on Rendering (2009), pp. 1–10
Hendrik Lensch and Peter-Pike Sloan (Guest Editors)

Soft Textured Shadow Volume

Vincent Forest[1], Loïc Barthe[1], Gaël Guennebaud[2] and Mathias Paulin[1]

[1]University of Toulouse IRIT-CNRS, France
[2]INRIA Bordeaux, France

(a) (b) (c)

Figure 1: *Illustration of shadows cast by perforated triangles (fences) using our soft textured shadow volume algorithm.*

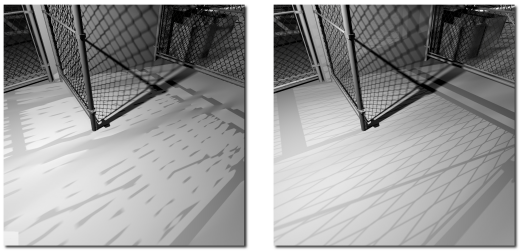
Abstract

Efficiently computing robust soft shadows is a challenging and time consuming task. On the one hand, the quality of image-based shadows is inherently limited by the discrete property of their framework. On the other hand, object-based algorithms do not exhibit such discretization issues but they can only efficiently deal with triangles having a constant transmittance factor. This paper addresses this limitation. We propose a general algorithm for the computation of robust and accurate soft shadows for triangles with a spatially varying transmittance. We then show how this technique can be efficiently included into object-based soft shadow algorithms. This results in unified object-based frameworks for computing robust direct shadows for both standard and perforated triangles in fully animated scenes.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Color, shading, shadowing, and texture

1. Introduction

Shadows enhance the quality of virtual 3D scenes and give information about the relationship between objects. Generating accurate shadows requires the computation of the light-surface visibility interaction. Such information is particularly difficult to compute in real-time since it requires a global knowledge of the scene organization. Hard shadows are the simplest to generate. They require only the visibility information between a point **p** in the scene and the light center. The image-based shadow map approach [Wil78] is the easiest way to compute real-time hard shadows. It holds for all rasterizable primitives and it is not explicitly influenced by the geometric complexity of the scene. Unfortunately, its computational complexity and memory consumption increase when it has to deal with omni-directional light sources. In addition, its accuracy is limited by the discrete representation of the scene used for the visibility queries. This results in minification and magnification artifacts that can be limited by increasing the resolution [TQJN99, LTYM06, ZSXL06], re-parameterizing [SD02, WSP04, MT04], or filtering [RSC87, DL06, AMB*07] the



(a) SSM 1024 (b) DCS+STSV

Figure 2: *Comparison between image-based and object-based soft shadows. (a) Soft Shadow Mapping based on the author's implementation [GBP07] (shadow map resolution: 1024^2). (b) Depth Complexity Sampling [FBP08] combined with our Soft Textured Shadow Volume. The STSV computes the shadows casted by the perforated triangles (fences).*

shadow map. On the other hand, the object-based shadow volume algorithm [Cro77] does not exhibit the previous drawbacks. Despite its geometric constraints and fill-rate bottleneck, it allows a *robust per pixel exact* evaluation of real-time hard shadows [BS99, Car00, EK02].

Shadow realism is enhanced when adding penumbra (soft shadows). Their computation requires the evaluation of the "amount of light" visible from a scene point **p** that can be derived from the visibility between **p** and an infinite number of samples onto the light source. This is particularly time consuming and as a result, most real-time applications use an approximation of soft shadows by decorrelating shadow generation from lighting computation. Thus, for each visible point of the scene, a *visibility coefficient* (v_coef) is computed, corresponding to an approximation of the visibility integral for the directions toward the light source. The v_coef is used to modulate the direct lighting contribution computed from the light center. On the one hand, the v_coef can be approximated using heuristics [AHT04, ED06] or an adaptive filtering of the hard shadow borders [Fer05, DAM*08]. This generates visually plausible soft shadows only correct in very specific cases. On the other hand, the v_coef can be derived from the evaluation of the light area visible from the points **p** seen in the scene. Despite convincing physically plausible results, these last algorithms are often based on the discrete and surjective scene representation of the shadow map framework that limits their accuracy [AHL*06, GBP07, SS07] (magnification aliasing, light leaking, shadow popping, *etc.*). Object-based soft shadow algorithms are not influenced by the previous drawbacks (Figure 2). They can either approximate a physically plausible v_coef [AAM03] analytically, or allow the numerical integration of the direct lighting [FBP08] independently of the light type. In fact, they propose a robust solution to generate *accurate* hard [EK02] or soft [FBP08] shadows in real time (we define accurate shadows as the un-biased shadow generation according to the underlying geometry discretization). These object-based techniques rasterize *penumbra wedges* in screen space. Sintorn *et al.* [SEA08] experimented an algorithm rasterizing per triangle soft shadow volumes in an alias-free shadow map. Unfortunately, the per-triangle treatment and the alias-free data structure construction significantly limit its performance when dealing with many light sources (*e.g.* a single omni-directional light requires *six* alias-free shadow maps). Based on the same idea, the soft irregular shadow mapping [JHB*09] avoids the construction of the specific regular alias-free data structure by using an irregular software rasterizer.

Following these observations, penumbra-wedge-based approaches are particularly attractive. However, they exhibit an important drawback: they can only deal efficiently with triangle-based geometries having a constant transmittance factor. Thus, unlike image-based shadows, object-based algorithms cannot handle perforated triangles. Such geometry is widely used in real-time and offline applications in order to represent highly perforated objects with few triangles and alpha textures encoding their binary opacity (*e.g.* a wire fence). As a first step, Hasselgren and Akenine-Möller [HAM07] propose an extension of the shadow volumes computing *pixel-exact* hard shadows cast by triangles with a spatially varying transmittance (including perforated triangles).

Contribution: We propose a general algorithm computing robust and accurate *soft shadows* for triangles with a spatially varying transmittance denoted as S-triangles (Figure 1). We also show how this technique can be included into object-based soft shadow frameworks [AAM03, FBP08], and we present an efficient and practical GPU implementation with binary transmittance textures (a texel is fully transparent or full occluder). In general, the use of perforated triangles allows a significant reduction of the geometric overhead. Our approach takes benefit of this property since our soft shadow computations is accelerated by a factor varying between 25 and 35 when compared with object-based soft shadows generated from an equivalent geometry represented by meshes.

The rest of the paper is organized as follows. Section 2 details our algorithm. In order to provide an unified and robust shadow framework, we show in Section 3 how our approach can be naturally and easily integrated into existing object-based soft shadow algorithms. Section 4 describes our GPU implementation. Finally, we present our results in Section 5 and we conclude with a discussion and directions for future work.

2. Soft Textured Shadow Volume

In this section we present our algorithm that allows the generation of *accurate soft shadows* cast by S-triangles (Section 2.1). In practice, this boils down to the computation of the influence of S-triangles on the visibility between the receivers and the area light source.

In fact, our approach merges the properties of both soft [AAM03, FBP08] and textured [HAM07] shadow volumes: it extrudes conservative volumes from S-triangles in order to define their soft shadow influence. We thus call our technique *Soft Textured Shadow Volumes* (STSV).

2.1. Overview

Targeting objects with spatially varying transmittance, we divide the occluding geometries into opaque and S-triangles. The shadows from opaque triangles are evaluated in a conventional way (using any soft shadow algorithm) while S-triangles are treated separately using Algorithm 1.

Algorithm 1 STSV(Triangles T, Light l)

```
Require: samples distributed onto the light source
 1: clear_light_sample_buffer(l.LS_buffer);
 2: for all t ∈ T do
 3:     stsv_t ← build_soft_textured_shadow_volume(t, l);
 4:     for all visible points p ∈ stsv_t do
 5:         for all light samples s as seen by p do
 6:             tmp ← texture_access( t, t.tex_transmittance, s, p);
 7:             update_light_sample(l.LS_buffer[p][s], tmp);
 8:         end for
 9:     end for
10: end for
```

For light l, a light sample buffer (LS_buffer) stores the information necessary to derive the visibility interaction between a visible point **p** and a set of light samples. We point out that the visible points **p** are already defined (for instance using a first rendering pass initializing the Z-buffer). For each S-triangle **t** (line 2), we extrude a *Soft Textured Shadow Volume* (STSV) (line 3 and Section 2.2). The STSV conservatively includes the region in which light visibility is influenced by **t**. For all **p** lying in the STSV (line 4 and Section 2.3), we finally update the LS_buffer according to the influence of **t** on the visibility between **p** (Section 2.4) and a set of light samples (line 6, 7, and Section 4.1).

Note that we do not explicitly define the information stored into the LS_buffer yet. Indeed, the LS_buffer can encode any data allowing the computation of the visibility interactions. For instance, the LS_buffer could store the percentage of light intensity attenuated by the S-triangles lying between any visible point **p** and a set of light samples **s**.

2.2. Soft Textured Shadow Volume Extrusion

We extrude a volume that conservatively includes both umbra and penumbra regions of S-triangle **t** (Figure 3). We use the robust z-fail strategy [Car00, EK02] to rasterize the resulting primitive and thus we have to close the Soft Textured Shadow Volume with front and back caps.

For a robust construction [AAM03], we define which vertex of **t** is the closest to the light center. Then, we move the others towards the light center until the distance is the same for all vertexes. The resulting vertexes describe a new triangle $\mathbf{t}'$ that defines the front cap of the STSV. For each edge of $\mathbf{t}'$ we extrude the outer-quad of its corresponding penumbra wedge primitive (Figure 3 left) (we refer to [AAM03] for additional details). At this step we connect these quads in order to build a closed volume defining an upper-bound of the shadow region of $\mathbf{t}'$ (Figure 3 right). This is done by first evaluating the cross product of the normals of *two* adjacent outer-quads to define the direction of their intersection. Then we extrude the vertex belonging to $\mathbf{t}'$ and shared by the quads along this direction. Finally, we define the back cap triangle of the STSV with the set of extruded vertexes.

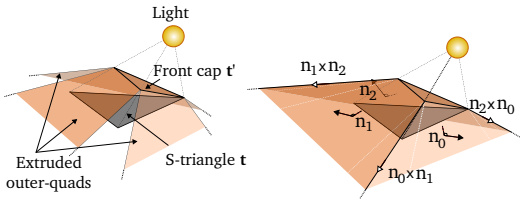


Figure 3: *Construction of the Soft Textured Shadow Volume. (left) We first extrude the penumbra wedge outer-quads from the front cap triangle edges. (right) Then we close the volume by connecting the adjacent outer-quads.*

2.3. Points into Soft Textured Shadow Volume

A given point **p** lies inside a STSV if it is on the same side of the *four* STSV planes. This test is done efficiently using the interpolation algorithm of [HAM07]. Once efficiently implemented, this evaluation only requires *two* multiplications and *one* addition per visible point and per STSV plane.

2.4. Accessing the Transmittance Texture

The access to the transmittance texture of the occluding triangle **t** can be performed by shooting a ray from **p** to a light sample. The barycentric coordinates of the ray-triangle intersection [Bad90, MT97] are then used to interpolate the texture coordinates of **t** and access the transmittance value.

Instead, we reduce the per-light sample computations using projections. Given S-triangle **t**, its normal n, its world space vertex coordinates v_0, v_1, v_2, their associated texture coordinates t_0, t_1, t_2, and the orthogonal distance k from the point light to **t**, we project each light sample onto the triangle plane as seen by **p**. Let e denote the orthogonal distance from **p** to **t**. We retrieve the texture transmittance value affecting the visibility of **p** and **s** by computing the homogeneous texture transmittance coordinates $\mathbf{s}'_h$ as follows:

$$\mathbf{s}'_h = S \cdot \overbrace{W \cdot M'}^{J} \cdot \mathbf{s} \qquad (1)$$

In this equation, M' is the transformation from the world

to an homogeneous triangle space (Equation 2), W is the matrix setting $\mathbf{p}$ as the projective center and performing the transformation into the homogeneous barycentric space of the S-triangle (Equation 3), and S is the transformation from the homogeneous barycentric triangle space to the homogeneous texture space (Equation 4).

$$M' = \begin{pmatrix} \overrightarrow{v_0v_1}.x & \overrightarrow{v_0v_2}.x & -n.x \\ \overrightarrow{v_0v_1}.y & \overrightarrow{v_0v_2}.y & -n.y \\ \overrightarrow{v_0v_1}.z & \overrightarrow{v_0v_2}.z & -n.z \end{pmatrix}^{-1} \quad (2)$$

$$W = \begin{pmatrix} 1 & 0 & -(M' \cdot (v_0 - \mathbf{p})).x/e \\ 0 & 1 & -(M' \cdot (v_0 - \mathbf{p})).y/e \\ 0 & 0 & 1 \end{pmatrix} \quad (3)$$

$$S = \begin{pmatrix} \overrightarrow{t_0t_1}.x & \overrightarrow{t_0t_2}.x & t_0.x \\ \overrightarrow{t_0t_1}.y & \overrightarrow{t_0t_2}.y & t_0.y \\ 0 & 0 & 1 \end{pmatrix} \quad (4)$$

In a first step, we evaluate the S and M' matrices per S-triangle since they are constant for a triangle. Then, for each visible point $\mathbf{p}$ lying into the STSV, we compute its corresponding transformation W, and define the transformation $J = W \cdot M'$ from world space to homogeneous barycentric coordinates (Equation 1). Due to the conservative nature of the STSV, some light samples as seen by $\mathbf{p}$ may not be occluded by $\mathbf{t}$. Let $\mathbf{s}_h^* = J \cdot \mathbf{s}$. A light sample $\mathbf{s}$ is occluded if $(\mathbf{s}_h^*.x \geqslant 0)$, $(\mathbf{s}_h^*.y \geqslant 0)$, and $(\mathbf{s}_h^*.z \geqslant \mathbf{s}_h^*.x + \mathbf{s}_h^*.y)$. We transform the homogeneous barycentric coordinates $\mathbf{s}_h^*$ of the occluded light samples into homogeneous texture coordinates $\mathbf{s}'_h$ by applying matrix S. We finally retrieve the texture transmittance position $\mathbf{s}'$ by simply performing the projection:

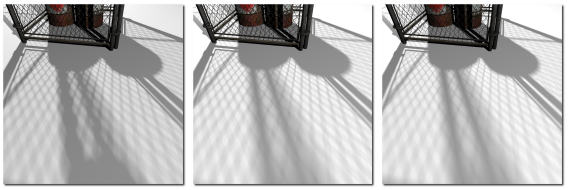
$$\mathbf{s}' = \frac{1}{\mathbf{s}'_h.z} \begin{pmatrix} \mathbf{s}'_h.x \\ \mathbf{s}'_h.y \end{pmatrix} \quad (5)$$

Even though several computations are performed for each visible point $\mathbf{p}$, the use of Equation 1 is still more efficient than a ray-traced approach to access the transmittance texture (Appendix A). Indeed, the matrix J is computed once for a given $\mathbf{p}$ (whatever the number of light samples). Thus, the computations of $\mathbf{s}_h^*$ only require 15 scalar instructions and for the occluded light samples, the evaluation of the texture transmittance coordinates is obtained with 12 scalar instructions.

3. Unified Object-Based Soft Shadow Framework

Theoretically our STSV approach can be used for the generation of soft shadows from both standard meshes and S-triangles. However, it requires unnecessary computations for triangles without a transmittance property and conventional object-based approaches are far more efficient. Thus, we show how our STSV algorithm can be naturally integrated in common object-based soft shadow frameworks.

Most S-triangles with spatially varying transmittance are used to represent perforated triangles, *i.e.* the transmittance texture encodes its spatially varying binary opacity (fence, branch, grass, *etc.*). Indeed, such representation is naturally and efficiently integrated in real-time renderers. Even though the STSV algorithm works with any transmittance value, our implementation thus focuses on perforated triangles.



(a) PWedge+STSV (b) DCS+STSV (c) RT

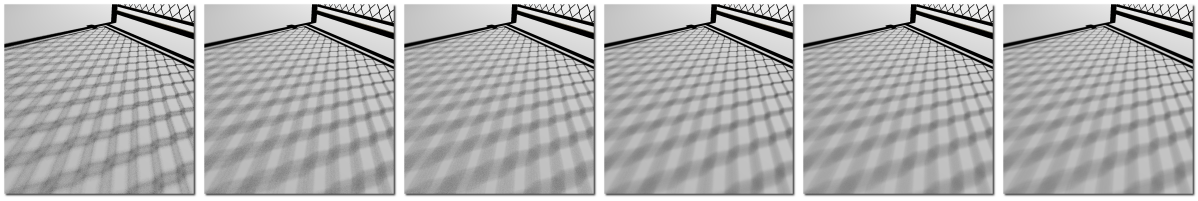
Figure 4: *Comparison between the shadows computed by (a) the Penumbra Wedge+Soft Textured Shadow Volume algorithm, (b) the Depth Complexity Sampling+Soft Textured Shadow Volume technique, and (c) a Ray Traced reference [mi] computed onto the fences represented by meshes.*

3.1. Penumbra Wedge

The penumbra-wedge algorithm [AMA02] generates soft shadows by modulating the direct lighting from the light center using a physically plausible v_coef. This approach is based on the assumption that the silhouette loops detected from the light center do not overlap. When *two* silhouette edges overlap, the overlapping occluded area is counted twice, resulting in an under-estimated v_coef and thus in an over-shadowed region.

For common meshes, a v_coef is computed per visible point $\mathbf{p}$, using the penumbra-wedge algorithm. The shadows cast by perforated triangles are then evaluated with the STSV approach (Algorithm 1). For each visible point $\mathbf{p}$, the light sample buffer (LS_buffer) stores a bit mask where each bit encodes visibility between $\mathbf{p}$ and a light sample. The LS_buffer is computed as follows. For a given perforated triangle $\mathbf{t}$, we have to compute a bit mask for each point $\mathbf{p}$ lying into its extruded STSV. For each of these $\mathbf{p}$, the value of each bit is derived from the transmittance value affecting its light sample visibility (Equation 1). Then, we perform a logical OR of the resulting bit mask with the corresponding mask stored in the LS_buffer. When all perforated triangles are treated, we evaluate the per $\mathbf{p}$ STSV v_coef with the bit mask of the LS_buffer. We add this v_coef with the one computed during the penumbra-wedge pass in order to evaluate the final v_coef used to modulate the direct illumination.

In the case of overlapping S-triangles, the STSV v_coef does not exhibit under-estimation artifacts. However, the inherent shadow overlapping artifact of the penumbra wedges still occurs when perforated triangles overlap common meshes (Figure 4(a)).



(a) 16 uniform (b) 16 stratified (c) 16 poisson disk (d) 64 low discrep. (e) 64 poisson disk (f) 64i poisson disk

Figure 5: *Impact of the sampling strategy on the shadow quality. Shadows are generated with either* 16 *(a, b, c) or* 64 *(d, e, f) decorrelated light samples using a (a) uniform, (b) stratified, (d) low-discrepancy, or (c, e) Poisson disk sampling strategy. Any of these sampling techniques can be combined with any interleaved sampling pattern. For instance, Figure (f) illustrates shadows generated with a* 4×4 *interleaved sampling pattern of 64 samples distributed with the Poisson disk strategy.*

3.2. Depth Complexity Sampling

The Depth Complexity Sampling (DCS) algorithm [FBP08] is a generalization of the penumbra-wedge approach. For each visible surface point **p**, it computes the number of occluders (*i.e.* the depth complexity) lying between **p** and a set of light samples. This information is then used to either derive an artifact free physically plausible v_coef or solve the visibility queries to numerically evaluate the direct lighting.

The integration of the STSV algorithm into the DCS framework is straightforward (Figure 4(b)). For common occluders, the DCS algorithm evaluates the depth complexity between each visible point **p** and the light samples. Then, the STSV algorithm updates the depth complexity for the visible points **p** as follows. First, for each **p** lying into the STSV of a perforated triangle **t**, Equation 1 allows us to retrieve the transmittance values affecting its visibility from the set of light samples. Then the depth complexity counter of the corresponding light sample **s** is incremented if the retrieved transmittance value indicates that **s** is occluded by **t**. After the depth complexity evaluation we finally perform the direct lighting step as proposed in the DCS algorithm.

4. Implementation

In this Section we detail the GPU implementation of our Soft Textured Shadow Volume algorithm. We propose an algorithm (Algorithm 2) targeting the graphics processors taking benefit of *Vertex Programs*, *Geometry Programs*, and *Fragment Programs*.

As in the shadow volume algorithm [Cro77], we first render the scene to compute an approximation of the indirect illumination and to initialize the Z-buffer (line 2). Then we separate common occluders from perforated triangles to perform the direct lighting passes (lines 5 and 6). Completely opaque meshes are treated with object-based soft shadow algorithms (lines 8 and 12) while we generate the shadows cast by perforated triangles with our STSV algorithm (lines 9 and 13).

Algorithm 2 render_scene(Scene w, RenderView v)

```
Require: Pre-computed sample pattern
 1: set_up_camera(v);
 2: (color-buffer, Z-buffer) ← draw_ambient_lighting(w);
 3: for all l ∈ w.lights do
 4:     clear_shadow_buffers();
 5:     tri_o ← w.get_opaque_triangles();
 6:     tri_p ← w.get_perforated_triangles();
 7:     if Depth Complexity Sampling then
 8:         DCS(tri_o, l);
 9:         STSV_DCS(tri_p, l);
10:         color_buffer += DCS_direct_lighting(w, l);
11:     else
12:         PWedge(tri_o, l);
13:         STSV_bitmask(tri_p, l);
14:         add_v_coef();
15:         color_buffer += PWedge_direct_lighting(w, l);
16:     end if
17: end for
```

4.1. Sample Distribution

In order to accurately compute the visibility interaction between a visible point **p** and an extended light source, it is necessary to distribute the light samples using an adapted probability distribution function [PH04]. To avoid visible sampling patterns we use a decorrelated sampling strategy, *i.e.*, a sample set is generated independently for each visible point **p** according to a specific sample distribution. A good sample distribution is given by the stratified sampling approach (Figure 5(b)). However, better results are obtained with more sophisticated methods such as Poisson disk [Coo86] (Figures 5(c) and 5(e)) or low-discrepancy [Nie92] (Figure 5(d)) sampling strategy. Finally, any of these sample distributions can be combined with the interleaved sampling strategy [KH01, SIMP06] (Figure 5(f)) in order to reduce the decorrelation noise.

Depending on the light type and the sampling strategy, we store a set of pre-computed light samples in a texture. We reduce memory consumption and the number of texture fetches using the data representation of the sample posi-

tions described in the Depth Complexity Sampling algorithm [FBP08]. The decorrelated sample distributions are generated by randomly rotating per pixel the pre-computed sample positions. According to the desired performance/quality ratio, we propose to distribute either 16 or 64 samples onto the light source. Note that any number of samples can be chosen with respect to hardware limitations. The low-discrepancy sample distribution is based on the $(0, 2)$-sequence while with the Poisson disk sampling strategy we experimentally fix the minimum distance constraint to 0.2456139 or 0.1076681 for respectively 16 or 64 samples.

4.2. Soft Textured Shadow Volume Extrusion

The Soft Textured Shadow Volume of each perforated triangle is robustly extruded onto GPU with a *Geometry Program* implementing the procedure presented in Section 2.2. In this *Geometry Program*, we compute and transmit the per-STSV vertex parameters required by the interpolation algorithm [HAM07] (Section 2.3). In order to ensure their correct interpolation we clamp the extrusion of the STSV at the level of the scene bounding box. We also transmit per STSV vertex the matrices S and M' of Equation 1. These matrices are constant for each S-triangle.

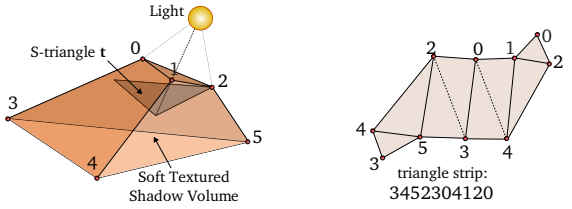


Figure 6: *Triangle strip used for the Soft Textured Shadow Volume extrusion. The resulting extruded volume is capped and it implicitly defines a coherent normal orientation required for its robust z-fail rendering.*

Our STSV construction (Section 2.2) minimizes the number of generated vertexes rather than builds a tight bounding volume of the shadow cast by the S-triangle. This reduces both the computational complexity of its generation and the geometric amplification bottleneck of the geometry processors. Specifically, we generate each STSV using the optimized triangle strip configuration depicted in Figure 6 which requires only *ten* vertices.

After its extrusion, the STSV is sent to the rasterization stage where it is rasterized according to the failure of the Z-buffer visibility test.

4.3. Transmittance Sampling

For each fragment q of the STSV, we retrieve its corresponding visible scene point $\mathbf{p}$ from the Z-buffer. In order to determine if $\mathbf{p}$ is included into the STSV, we apply the efficient interpolation algorithm of [HAM07]. Then for each light sample $\mathbf{s}$, we retrieve with Equation 1 the texture transmittance value affecting its visibility from $\mathbf{p}$ (Appendix A). We use these transmittance values to compute the visibility properties according to the chosen implementation (penumbra wedge or DCS). In both cases the fix Raster OPeration stage (ROP) updates the light sample buffer with respect to these visibility informations.

5. Results

Our implementation is based on the OpenGL API. We measured performance on a 64 bit Linux workstation with a Core^{TM} 2 Duo 3GHz with 4GB of DDR2, and a NVIDIA GeForce GTX280. Our work targets direct soft shadow generation and the indirect lighting is orthogonally evaluated with any existing technique. Here, it is approximated using an irradiance map [RH01] and a screen-space local ambient occlusion [SA07].

In order to produce meaningful results, each frame is rendered from scratch, in a purely forward manner, without frustum culling, light sorting, precomputed visibility set, or scissor/depth bound tests. In practice, such specific optimizations usually lead to a huge performance enhancement [Len05].

We also compare our algorithm to the efficient contour based soft shadow mapping (SSM) technique [GBP07]. For these comparisons we use a slight variation of authors's original implementation. We have disabled the adaptive precision strategies because they reduce the quality, and we have optimized the computation of tight occluder search areas using the MSSM data structure [SS07]. We have also selected the maximal shadow map resolution leading to reasonable memory consumption and performance: the shadow map resolution is 1024^2.

5.1. Memory Consumption

The treatment of S-triangles by our STSV algorithm neither requires pre-computed parameters nor additional per-mesh texture. Thus, memory consumption of our algorithm is independent of the geometric complexity of the scene.

When it is included in the penumbra wedge framework, the STSV algorithm requires both a set of pre-computed light samples and a light sample buffer storing the visibility bit mask. Thus, for a 1024×1024 frame buffer and 64 samples distributed onto the light source, the memory cost of the STSV implementation is:

$$\overbrace{64 \times 1024^2 \times 1bit}^{LS\_buffer} + \overbrace{64 \times 16bits}^{precomputed\ sample\ pattern} \approx 8MB$$

Note that the STSV algorithm does not require any specific memory allocation when it is included in the DCS framework. Indeed, the precomputed light samples are shared with

those of the DCS and the light sample buffer is a reference to the depth complexity buffer.

In contrast, our reference SSM algorithm requires 4MB for the shadow map, plus 80MB for the MSSM data structure.

5.2. Performance Analysis

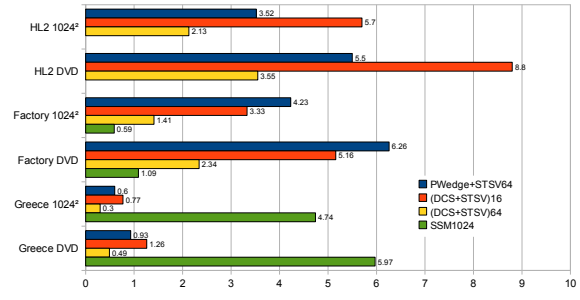


Figure 8: *Average frame rates on our three test scenes. The shadows are computed with our Soft Texture Shadow Volume algorithm (STSV) combined with either the Penumbra Wedge (PWedge) or the Depth Complexity Sampling (DCS) approach. Performances are reported for two image resolutions:* 1024 × 1024 *and* 720 × 576 *(DVD).*

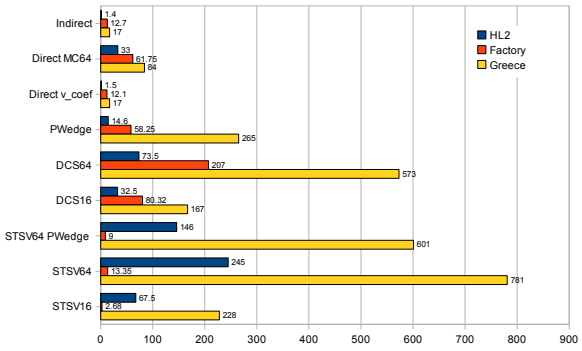


Figure 9: *Time in milliseconds of the rendering steps for an image resolution of* 1024 × 1024. *The numbers next to the acronyms give the number of per-light samples used for each pixel. The direct lighting can be either attenuated with a v_coef or numerically solved by Monte-Carlo sampling with the DCS framework (Direct MC64). Except for our indirect lighting pass (that is independent of the number of lights), the given times are the average of the per-light computation times.*

Our benchmarks are based on the *three* test scenes of Figure 10. Figure 8 gives the average frame rates along a camera path defined for each scene. The costs of the different rendering steps are given in Figure 9 for a representative frame.

The HL2 scene (Figure 7(a)) is composed of 4,002 perforated triangles casting large visible shadows. In such situation, rendering time is computationally bounded by the Soft Textured Shadow Volume evaluation. Due to the limited shadow map resolution, the SSM algorithm is not able to generate many of the shadows produced by small distant objects of this outdoor scene. Eventhough we observe performance around 34/48 fps, shadows are of low quality (Figures 10(a) and 10(b)) and a fair comparison would require to combine this SSM approach with, for instance, parallel-split shadow maps [ZSXL06].

In the high poly-count factory scene (Figure 7(b)), highly detailed and thin geometries are represented with few S-triangles (14 perforated triangles). Thus, even though they cast large visible shadows, their computational cost is negligible compared to the overall rendering time. In this scene, our object based algorithm outperform the SSM implementation both in terms of performance (5 times faster), and quality (Figure 2). These lower performances of the SSM algorithm are due to the light sources that are closely surrounded by geometries. In this case, even the use of the MSSM optimization cannot reduce the size of the occluder search areas which are around 100^2 texels for each pixel.

The Greece scene (Figure 7(c)) is composed of 26,150 S-triangles (*two* per leaf) and it is thus far more demanding for our algorithm. However, despite the computational resources required by the STSV evaluation, the object-based frameworks generate robust direct soft shadows (Figure 10(d)) of this complex fully dynamic environment in about a second. For this scene, the SSM algorithm is limited by the rendering into the shadow maps, yielding performance in the same order of magnitude, but with lower quality (Figure 10(c)).

6. Discussion and Future Works

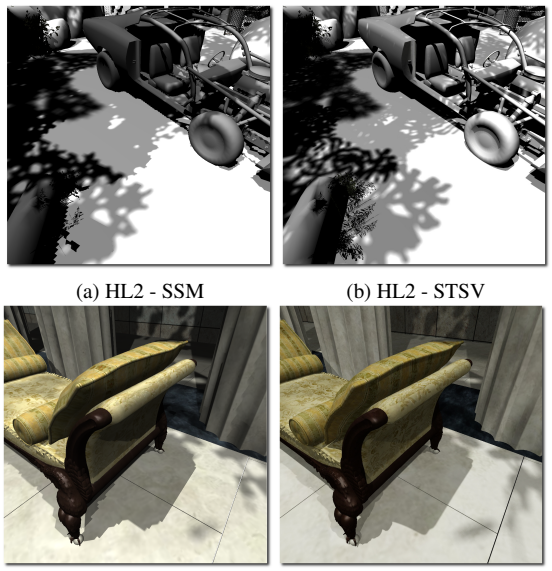
We build the Soft Textured Shadow Volume according to the orientation of the corresponding S-triangle as seen from the light center. This leads to the well-known single light sample artifact when both sides of the triangle are lit by the area light. This artifact can be very simply corrected by extruding a STSV for each side of the S-triangle.

In order to get a fair performance/quality ratio, we distribute at most 64 light samples with sampling strategies that drastically reduce variance. We have empirically set this maximum number of samples but the true limitation is given by the hardware constraints of the implementation. Nevertheless, thanks to the Monte-Carlo sampling nature of the STSV algorithm, the average of several runs would lead to a result statistically very close to the exact solution.

Unlike image-based approaches, the STSV algorithm does not exhibit the shadow map discretization aliasing. However, as any rendering technique, the viewport discretization is still prone to image aliasing. This cannot be naively corrected by a multi-sampling approach since the

(a) HL2 (b) Factory (c) Greece

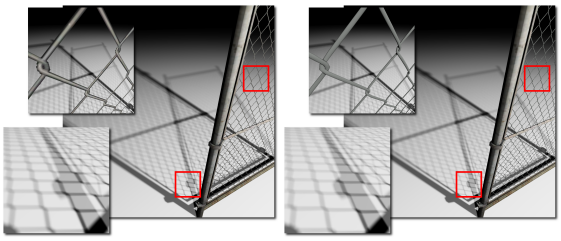
Figure 7: *Representative frames of the three test scenes used for our benchmarks. (a)* 84,712 *triangles,* 4,002 *perforated triangles, and* 2 *directional lights. (b)* 582,510 *triangles,* 14 *perforated triangles, and* 4 *omni-directional lights. (c)* 1,396,078 *triangles,* 26,150 *perforated triangles, and* 3 *omni-directional lights.*



(a) HL2 - SSM (b) HL2 - STSV

(c) Greece - SSM (d) Greece - STSV

Figure 10: *Comparison of the shadows generated with (a, c) SSM* 1024^2 *and (b, d) DCS+STSV.*



(a) STSV (b) DCS

Figure 11: *Comparison between shadows cast by a fence represented by (a) perforated triangles (*2 *perforated triangles, STSV:* 90*fps) and (b) by its corresponding mesh (*233,358 *triangles, DCS16:* 3.3*fps).*

STSV fragment program has to be evaluated for each sub-pixel sample. Thus, the LS_buffer must be super-sampled as the Z-buffer and the stencil buffer in common object-based approaches. A more efficient solution would consist in combining a conservative rasterization [HAMO05] and a judicious anti-aliasing algorithm that super-samples the STSV only where the aliasing occurs.

Common object-based techniques extrude primitives only at the silhouette edges of the occluders whereas our STSV algorithm generates a STSV for *each* triangle. This could be seen as a bottleneck. However, perforated triangles are used as an alternative representation for detailed geometries (leafs, fence, *etc.*) and the generation of an analogous shadow quality cast by an explicit mesh representation would require much more computational resources (Figure 11).

Also, perforated triangles are useful in a wide range of applications such as sprite/billboards [DDSD03, PMDS06, Ris07] or distant objects in a LOD hierarchy. The STSV algorithm is an efficient object-based approach allowing the generation of robust soft shadows on such representations.

Finally, the combination of the DCS and the STSV algorithms provides the visibility information between each visible point and an arbitrary set of light samples. The proposed sampling strategies distribute the samples according to the light importance. Then the DCS direct lighting pass uses this sample distribution and the computed depth complexity to numerically solve the direct lighting integral. However, we could achieve better real-time direct illumination results on specular surfaces using the DCS+STSV framework and a multiple importance sampling strategy [VG95]. Indeed, the samples would be distributed according to the importance of both light *and* Bidirectional Scattering Distribution Function.

7. Conclusion

We have presented a Soft Textured Shadow Volume algorithm that addresses one of the main issues of the object-based shadow generation: the *accurate* computation of the soft shadows cast by triangles with a spatially varying transmittance. We have demonstrated that our approach can be naturally integrated into common object-based soft shadow algorithms. This provides efficient object-based frameworks computing soft shadows on fully animated scenes. Thanks to its object-based nature, our algorithm does not suffer from shadow discretization aliasing nor shadow popping (Figure 2). In addition, it handles omni-directional area lights without any additional performance penalty or specific treatments.

References

[AAM03] ASSARSSON U., AKENINE-MÖLLER T.: A geometry-based soft shadow volume algorithm using graphics hardware. *ACM Transactions on Graphics, Proc. SIGGRAPH 22*, 3 (2003), 511–520.

[AHL*06] ATTY L., HOLZSCHUCH N., LAPIERRE M., HASENFRATZ J.-M., HANSEN C., SILLION F.: Soft shadow maps: Efficient sampling of light source visibility. *Computer Graphics Forum 25*, 4 (2006), 725–741.

[AHT04] ARVO J., HIRVIKORPI M., TYYSTJÄRVI J.: Approximate soft shadows with an image-space flood-fill algorithm. *Computer Graphics Forum, Proc. EUROGRAPHICS 23*, 3 (2004), 271–280.

[AMA02] AKENINE-MÖLLER T., ASSARSSON U.: Approximate soft shadows on arbitrary surfaces using penumbra wedges. In *Proc. EG Workshop on Rendering Techniques* (2002), Eurographics, pp. 297–306.

[AMB*07] ANNEN T., MERTENS T., BEKAERT P., SEIDEL H.-P., KAUTZ J.: Convolution shadow maps. In *Proc. EG Symposium on Rendering* (2007), vol. 18, Eurographics, pp. 51–60.

[Bad90] BADOUEL D.: An efficient ray-polygon intersection. *Graphics gems* (1990), 390–393.

[BS99] BILODEAU B., SONGY M.: Real time shadows. Creative Labs sponsored Game developer Conference, unpublished slides, May 1999.

[Car00] CARMACK J.: An email from john carmack to mark kilgard outlining z-fail algorithm. Id-Software, 2000.

[Coo86] COOK R. L.: Stochastic sampling in computer graphics. *ACM Trans. Graph. 5*, 1 (1986), 51–72.

[Cro77] CROW F. C.: Shadow algorithms for computer graphics. In *Proc. SIGGRAPH* (1977), ACM Press, pp. 242–248.

[DAM*08] DONG Z., ANNEN T., MERTENS T., BEKAERT P., SEIDEL H.-P., KAUTZ J.: Real-time, all-frequency shadows in dynamic scenes. *ACM Transactions on Graphics, Proc. SIGGRAPH 0*, 0 (2008), to appear.

[DDSD03] DÉCORET X., DURAND F., SILLION F. X., DORSEY J.: Billboard clouds for extreme model simplification. In *Proc. SIGGRAPH* (2003), ACM Press, pp. 689–696.

[DL06] DONNELLY W., LAURITZEN A.: Variance shadow maps. In *SIGGRAPH Symposium on Interactive 3D Graphics and Games* (2006), ACM Press, pp. 161–165.

[ED06] EISEMANN E., DÉCORET X.: Plausible image based soft shadows using occlusion textures. In *Proc. of the Brazilian Symposium on Computer Graphics and Image Processing* (2006), Conference Series, IEEE Computer Society, pp. 155–162.

[EK02] EVERITT C., KILGARD M.: *Practical and robust stenciled shadow volumes for hardware-accelerated rendering*. Tech. rep., NVIDIA Cooperation, 2002.

[FBP08] FOREST V., BARTHE L., PAULIN M.: Accurate shadows by depth complexity sampling. *Computer Graphics Forum, Proc. EUROGRAPHICS 27*, 2 (2008), 663–674.

[Fer05] FERNANDO R.: Percentage-closer soft shadows. In *SIGGRAPH Sketches* (2005), ACM Press, p. 35.

[GBP07] GUENNEBAUD G., BARTHE L., PAULIN M.: High-quality adaptive soft shadow mapping. *Computer Graphics Forum, Proc. EUROGRAPHICS 26*, 3 (2007), 525–533.

[HAM07] HASSELGREN J., AKENINE-MÖLLER T.: Textured shadow volumes. *Journal of Graphics Tools 12*, 4 (2007), 59–72.

[HAMO05] HASSELGREN J., AKENINE-MÖLLER T., OHLSSON L.: *GPU Gems 2 - Programming Techniques for High-Performance Graphics and General-Purpose Computation*. Addison Wesley, 2005, ch. Conservative Rasterization, pp. 677–694.

[JHB*09] JOHNSON G., HUX A., BURNS C., HUNT W., MARK B., JUNKINS S.: Soft irregular shadow mapping: Fast, high-quality, and robust soft shadows. In *SIGGRAPH Symposium on Interactive 3D Graphics and Games* (2009), ACM Press, pp. 57–66.

[KH01] KELLER A., HEIDRICH W.: Interleaved sampling. In *Proceedings of the 12th Eurographics Workshop on Rendering Techniques* (2001), Eurographics, pp. 269–276.

[Len05] LENGYEL E.: Advanced stencil shadow and penumbra wedge rendering. Game developer Conference, unpublished slides, 2005.

[LTYM06] LLOYD B., TUFT D., YOON S., MANOCHA D.: Warping and partitioning for low error shadow maps.

In *Proc. EG Symposium on Rendering* (2006), Eurographics, pp. 215–226.

[mi] MENTAL IMAGES: mental ray renderer. http://www.mentalimages.com/products/mental-ray.html.

[MT97] MÖLLER T., TRUMBORE B.: Fast, minimum storage ray-triangle intersection. *Journal of Graphics Tools 2*, 1 (1997), 21–28.

[MT04] MARTIN T., TAN T.-S.: Anti-aliasing and continuity with trapezoidal shadow maps. In *Proc. EG Symposium on Rendering* (2004), Eurographics, pp. 153–160.

[Nie92] NIEDERREITER H.: *Random number generation and quasi-Monte Carlo methods*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1992.

[PH04] PHARR M., HUMPHREYS G.: *Physically Based Rendering: From Theory to Practice*. Morgan Kaufmann, 2004, ch. Monte Carlo Integration I: Basic Concepts, pp. 631–660.

[PMDS06] POPESCU V., MEI C., DAUBLE J., SACKS E.: Reflected-scene impostors for realistic reflections at interactive rates. *Computer Graphics Forum, Proc. EUROGRAPHICS 25*, 3 (2006), 313–322.

[RH01] RAMAMOORTHI R., HANRAHAN P.: An efficient representation for irradiance environment maps. In *Proc. SIGGRAPH* (2001), ACM Press, pp. 497–500.

[Ris07] RISSER E.: *GPU Gems 3*. Addison Wesley, 2007, ch. True Imposters, pp. 481–490.

[RSC87] REEVES W. T., SALESIN D. H., COOK R. L.: Rendering antialiased shadows with depth maps. In *Proc. SIGGRAPH* (1987), vol. 21, ACM Press, pp. 283–291.

[SA07] SHANMUGAM P., ARIKAN O.: Hardware accelerated ambient occlusion techniques on gpus. In *SIGGRAPH Symposium on Interactive 3D Graphics and Games* (2007), ACM Press, pp. 73–80.

[SD02] STAMMINGER M., DRETTAKIS G.: Perspective shadow maps. In *Proc. SIGGRAPH* (2002), ACM Press, pp. 557–562.

[SEA08] SINTORN E., EISEMANN E., ASSARSSON U.: Sample based visibility for soft shadows using alias-free shadow maps. In *Proc. EG Symposium on Rendering* (2008), Eurographics, p. to appear.

[SIMP06] SEGOVIA B., IEHL J.-C., MITANCHEY R., PÉROCHE B.: Non-interleaved deferred shading of interleaved sample patterns. In *Proc. SIGGRAPH/EUROGRAPHICS conference on Graphics hardware* (2006), Eurographics, pp. 53–60.

[SS07] SCHWARZ M., STAMMINGER M.: Bitmask soft shadow. *Computer Graphics Forum, Proc. EUROGRAPHICS 26*, 3 (2007), 515–524.

[TQJN99] TADAMURA K., QIN X., JIAO G., NAKAMAE E.: Rendering optimal solar shadows using plural sunlight depth buffers. *Computer Graphics International* (1999), 166–173.

[VG95] VEACH E., GUIBAS L. J.: Optimally combining sampling techniques for monte carlo rendering. In *Proc. SIGGRAPH* (1995), ACM Press, pp. 419–428.

[Wil78] WILLIAMS L.: Casting curved shadows on curved surfaces. In *Proc. SIGGRAPH* (1978), ACM Press, pp. 270–274.

[WSP04] WIMMER M., SCHERZER D., PURGATHOFER W.: Light space perspective shadow maps. In *Proc. EG Symposium on Rendering* (2004), Eurographics, pp. 143–151.

[ZSXL06] ZHANG F., SUN H., XU L., LUN L. K.: Parallel-split shadow maps for large-scale virtual environments. In *Proc. of the Virtual reality continuum and its applications* (2006), ACM press, pp. 311–318.

Appendix A: Transmittance Value

`NV_gpu_program4` fragment program sub-routine. Retrieves the transmittance value of the perforated triangle affecting the visibility of the light sample `s_pos` as seen by the treated pixel.

```
# texture[0].w = transmittance texture of the STSV
get_transmittance_value:

    DP3.CC0 r0.x,     J_row0,  s_pos; #r0.xyz=s*_h = W·M'·s
    DP3.CC0 r0.y,     J_row1,  s_pos;
    DP3     r0.z,     J_row2,  s_pos;
    MOV     transm.w, 0;
    ADD     r1.z,     r0.x,    r0.y;  #r1.z=s*_h.z-s*_h.x-s*_h.y
    SUB.CC0 r1.z,     r0.z,    r1.z;
    RET     (LT.xyzz);                #if(s*_h.z-s*_h.x-s*_h.y<0 ||
                                      #   s*_h.xy<0) return;
    DP3     r1.x,     S_row0,  r0;    #r1.xy,r0.z=s'_h=S·s*_h
    DP3     r1.y,     S_row1,  r0;
    DIV     r0.xy,    r1,      r0.z;  #r0.xy= s' =s'_h.xy / s'_h.z
    TXL     transm.w, r0,      texture[0], 2D;
RET;
```

Appendix B: v_coef from Visibility Bit Mask

`NV_gpu_program4` fragment program. Fast vectorized and parallel bit count routine. Compute the v_coef from the bit mask visibility of a set of 64 uniformly distributed light samples.

```
# texture[0] = RGBA16UI bit mask texture
ATTRIB f_pos = fragment.position;
TEX.U    bitmask, f_pos,  texture[0], RECT;

AND.U    r0,      bitmask, 0x5555;
SHR.U    bitmask, bitmask, 1;
AND.U    bitmask, bitmask, 0x5555;
ADD.U    bitmask, r0,      bitmask;
AND.U    r0,      bitmask, 0x3333;
SHR.U    bitmask, bitmask, 2;
AND.U    bitmask, bitmask, 0x3333;
ADD.U    bitmask, r0,      bitmask;
AND.U    r0,      bitmask, 0x0F0F;
SHR.U    bitmask, bitmask, 4;
AND.U    bitmask, bitmask, 0x0F0F;
ADD.U    bitmask, r0,      bitmask;
AND.U    r0,      bitmask, 0x00FF;
SHR.U    bitmask, bitmask, 8;
AND.U    bitmask, bitmask, 0x00FF;
ADD.U    bitmask, r0,      bitmask;

I2F      bitmask, bitmask;
DP4      v_coef,  bitmask, 0.015625;   # 0.015625 = 1/64
```