ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/272381725

Separable soft shadow mapping

ARTICLE *in* THE VISUAL COMPUTER · FEBRUARY 2015

Impact Factor: 0.96 · DOI: 10.1007/s00371-015-1062-6

READS

52

3 AUTHORS:

Jose Maria Buades Rubio

University of the Balearic Islands

26 PUBLICATIONS **77** CITATIONS

SEE PROFILE

Jesús Gumbau

Universitat Jaume I

38 PUBLICATIONS **72** CITATIONS

SEE PROFILE

Miguel Chover

Universitat Jaume I

133 PUBLICATIONS **458** CITATIONS

SEE PROFILE

Available from: Jesús Gumbau
Retrieved on: 03 February 2016

Vis Comput
DOI 10.1007/s00371-015-1062-6

ORIGINAL ARTICLE

Separable soft shadow mapping

Jose María Buades · Jesús Gumbau · Miguel Chover

© Springer-Verlag Berlin Heidelberg 2015

Abstract We propose an efficient technique for rendering visually plausible real-time soft shadows in screen space. First, we propose a novel blocker estimation technique based on a separable filter. Second, our technique performs a separable Gaussian blur in screen space over the hard shadows produced by the standard shadow mapping technique. Although blurring the hard shadows with a separable filter was done before in the literature using bilateral filtering, we use an alternative approach that minimizes artifacts. Since separated calculation is not possible for all cases, we provide data reutilization criteria based on two user-defined error thresholds called α and β. As a consequence of using separable approaches for both stages of the light visibility estimation, our technique is able to improve rendering performance, especially when high-resolution shadow maps and filtering kernels are used.

Keywords Rendering · Real time · Soft shadows

1 Introduction

Shadows are important in real-time scenes, since they greatly contribute to both the realism and the perception of the scene. However, while the so-called "hard shadows" can be efficiently calculated using the standard Shadow Mapping technique [23], rendering visually plausible soft shadows is an expensive task.

Soft shadows are physically produced in the real world by area lights which emit light from all the points of their surface. Area lights can be approximated by distributing a set of point lights over their surface (the more the better) and combine them using the standard shadow mapping technique for each point light. However, this method is not practicable for real-time applications since it would very expensive for smooth penumbrae.

To solve this problem, Fernando [12] presented a simple yet effective technique for computing soft shadows in real time. This method is divided into two different steps: the blocker search stage (for computing the size of the penumbra) and the shadow filtering stage (used to compute smooth shadow edges depending on the visibility factor). Both stages filter the shadow map, requiring a high amount of texture samples to be fetched (the higher the better), introducing a high-performance penalty cost.

We propose a new technique for minimizing the performance cost of existing soft shadowing algorithms. Our technique is based on converting the expensive filters used for both the blocker search and the shadow blurring stages into *separable filters*. For the first step, we use a separable filter which is able to determine whether the samples are reusable or not using novel data reutilization criteria, falling back to a safe non-separable filter when needed. For the penumbra generation stage, we apply a separable Gaussian blur to the hard shadows in screen space. This blurring algorithm is similar to the method used in [15,24], but it has been improved to avoid the need of storing and processing multiple depth layers and to solve problems not addressed in the original papers introduced by the use of separable bilateral filtering.

J. M. Buades
Universitat de les Illes Balears, Palma, Spain
e-mail: josemaria.buades@uib.es

J. Gumbau (✉) · M. Chover
Universitat Jaume I, Castellón de la Plana, Spain
e-mail: jgumbau@uji.es

M. Chover
e-mail: chover@uji.es

Published online: 12 February 2015

2 Previous work

Shadow rendering is a prolific research field in real-time computer graphics. However, it is still an open research topic, since the complexity of simulating physically realistic shadows is a complex task for real-time applications, such as games. In this section, we give a short overview of the publications in the literature that have mostly contributed to this topic.

Shadow Mapping is an image-based method introduced in 1978 by [23] that determines whether a point is being directly lit by a light source or not also introducing aliasing and undersampling artifacts when projecting the shadow map into the scene. To alleviate these artifacts, Reeves et al. [18] propose a new technique called Percentage-Closer Filtering (PCF), which is based on performing a number of shadow tests and averaging the results. PCF allows for smooth shadow transitions but it is very time consuming since several texture accesses are required. To provide more efficient filtering approaches, some authors have proposed alternative techniques like Variance Shadow Maps [10], Convolution Shadow Maps [2] or Exponential Shadow Maps [3], which attempt to reduce aliasing by blurring the shadow edges. However, these techniques are not able to produce visually plausible soft shadows, since they use a uniform filter for blurring the shadow edges.

To overcome this limitation, some authors [6,7] propose geometry-based techniques which extend the scene with new geometry elements that are used to simulate inner and outer penumbra effects. However, performance in these approaches heavily depends on scene complexity and the techniques are usually very complex and hard to implement compared to pure image-based methods.

In contrast, image-based methods are independent of the complexity of the rendered scene and are easier to implement and deliver better rendering performance. Fernando [12] proposes an easy-to-implement approach (Percentage-Closer Soft Shadows) by extending PCF with an additional stage that estimates the size of the penumbra at a given point. PCSS assumes that the light source, occluder and shadow receiver are parallel surfaces to simplify the light visibility estimation as shown in Eq. 1.

$$w_{\text{penumbra}} = w_{\text{light}} \frac{z_{\text{receiver}} - z_{\text{avg}}}{z_{\text{avg}}} \tag{1}$$

where w_{light} is the size of the light source; z_{receiver} and z_{avg} are the distances from the light source to the shadow receiver and the shadow casters, respectively; and w_{penumbra} represents the size of the penumbra at that point. First, in the blockers search step, the distance to the occluders z_{avg} is estimated by averaging the depth values in the shadow map that are smaller than the current pixel depth in light space. Then, PCF is performed using the resulting estimation. PCSS requires a high amount of texture lookups for both the blocker search and the shadow blurring stages, highly affecting performance when large kernel sizes are used and suffer from occluder fusion artifacts.

Backprojection techniques [4,5,13,14,19] use the shadow map as a discretized representation of the scene and estimate the per-pixel visibility factor by backprojecting the shadow map texels onto the light source and then computing the amount of occlusion. Although these approaches allow for perceptually plausible soft shadows (even with correct multiple occluder fusion), they are costly and introduce artifacts that are difficult to eliminate. Annen et al. [1] improve CSM by performing variable size penumbrae including a SAT-based blocker search step, which can be applied to environment light sources. Recent works like VSSM [9] and ESSM [21] propose to extend prefiltering techniques with a blocker search estimation to generate variable-sized penumbrae. Although these techniques offer performance improvements over PCSS, they introduce the usage of SAT which becomes a bottleneck that increases with the size of the shadow map.

Some authors [15,17,24] propose to perform the shadow filtering stage in screen space rather than in the shadow map. These techniques work as follows: first, the "hard shadows" are computed using standard Shadow Mapping. Then, these shadows are blurred using a separable Gaussian filter in screen space with per-pixel variable size depending on the estimation used in Eq. 1. Since these methods operate in screen space, they can take advantage of splitting the filtering process in two separate linear filters, reducing the computational cost from $O(n^2)$ to $O(n+n)$ and improving performance. These techniques usually exploit cross-bilateral filtering [8] to preserve discontinuities in a given domain. However, these techniques incur into additional problems. On one hand, there are situations in which there is not enough information in screen space to apply the filter because the needed samples are occluded by other elements. This causes [17,24] to produce incorrect penumbrae in some circumstances (as shown in the results section). To overcome this problem, Gumbau et al. [15] propose to use a number of layers that contain shadowing information for occluded objects. On the other hand, they still rely an computationally expensive "blocker search" estimation.

More recently, Schwrzler et al. [20] propose to exploit temporal coherence to determine which shadows need to be recalculated assuming typical viewer and objects' movements in scenes. This technique is complementary to other shadowing methods for improving rendering times by minimizing shadow computations. Sintorn et al. [22] present a method for compressing massive shadow maps ($256K \times 256K$) using a sparse voxel octree. Although this technique allows for rendering high-quality soft shadows in real time,

it only works for precomputed shadows and hence it is not suitable for fully dynamic scenes.

To further explore the most significant soft shadowing techniques, we recommend the reader the survey by Hasenfratz et al. [16], and the book Real-Time Shadows by Eisemann et al. [11] which greatly offers both theoretical and implementation details.

3 Algorithm overview

The improvements of our algorithm over the classical PCSS are based on applying the divide and conquer paradigm. Since the PCSS is based on the application of two heavy-weighted filters (for computing the blocker search and for filtering the shadows), we aim to improve performance by converting those filters into separable filters, thus reducing the computational complexity by one order of magnitude from $O(n^2)$ to $O(n+n)$ and, hence, improving performance. As discussed later, due to the fact that the process is not always separable for all cases, two user-defined thresholds are specified, defining the maximum allowed error. These error thresholds are called α and β. Algorithm is carried out in four conceptual steps:

1. *Horizontal Blocker Computation* Computes horizontal blocker distance only in the shadow map X axis.
2. *Blocker Computation* Computes blocker distance, using data computed in the previous step with threshold α to determine data reutilization.
3. *Horizontal Soft Shadow Computation* Computes soft shadows horizontally from hard shadow map in screen space.
4. *Soft Shadow Computation* Computes soft shadows, using data computed in the previous step and threshold β to determine data reutilization.

To minimize the amount of different rendering operations, the conceptual stages 2 and 3 are carried out in one single rendering step, performing all the calculations in only 3 steps in practice (Fig. 1). See algorithm in Table 1 for more details.

4 Separable soft shadow mapping

Just like PCSS, our algorithm is divided into two steps. First, the average distance to the blockers ($\text{blocker}_{\text{average}}$) is computed as follows. Let p be the pixel being shaded and let d be the distance to the light source. Also, let $\text{d}m(s,t)$ be the function used to access to the depth map at coordinates (s,t). Then, the blocker average is calculated as:

$$\text{blocker}_{\text{average}}(p) = \frac{\text{blocker}_{\text{sum}}(p)}{\text{blocker}_{\text{count}}(p)}$$

$$\text{blocker}_{\text{sum}}(p) = \sum_{y=-n/2}^{n/2} \sum_{\substack{x=-n/2 \\ \text{d}m(x,y)<d}}^{n/2} \text{d}m(x,y)$$

$$\text{blocker}_{\text{count}}(p) = \sum_{y=-n/2}^{n/2} \sum_{\substack{x=-n/2 \\ \text{d}m(x,y)<d}}^{n/2} 1$$

The evaluated area is determined by n, which depends on the size of the light source and the distance d. For a specific point p, given n neighboring pixels in the vertical axis $(p_{-n/2}, p_{-n/2+1}, \ldots, p_{0-1}, p_0, p_{0+1}, \ldots, p_{n/2-1}, p_{n/2})$, all of them located at the same distance d, we can rewrite the blocker search computation for a point p_0 as:

$$\text{blocker}_{\text{average}}(p_0) = \sum_{t=-n/2}^{n/2} \frac{\text{partial\_blocker}_{\text{sum}}(p_t)}{\text{partial\_blocker}_{\text{count}}(p_t)}$$

$$\text{partial\_blocker}_{\text{sum}}(p_t) = \sum_{\substack{x=-n/2 \\ \text{d}m(x,t)<d}}^{n/2} \text{d}m(x,t)$$

$$\text{partial\_blocker}_{\text{count}}(p_t) = \sum_{\substack{x=-n/2 \\ \text{d}m(x,t)<d}}^{n/2} 1$$

This reformulation allows to divide the problem in two sequential steps (horizontal and vertical) reducing the computational cost from $O(n^2)$ to $O(n+n)$, in the same way that a split Gaussian filter applied to an image reduces cost. However, this filter separation is only applicable for points that share the same distance to the light source and, therefore, would be only applicable to a limited amount of cases in prac-

Fig. 1 Hard Shadow Map, intermediary steps and final image. From *Left to Right* HardshadowMap, Horizontal Blur, Soft Shadow Map and Final image. Right image shows final Soft Shadow results without combining with ColorMap, uses left images as input and a vertical blur has been applied. In this case $m = 14$, so 41×41 samples are used, and weights are computed from a gaussian $N(0, 0.5)$

tice. To relax this condition and then increase performance, we define a threshold α that determines the amount of error between the distance d (from p_0 to light source) and the distance d_t (from other points p_t to light source). Then, if $|d - d_t| < \alpha$, we consider that the data are reusable. Increasing α allows for more data to be reused and then better performance is achieved; however, less accurate results are obtained. Once $blocker_{average}$ is computed, penumbra of size m is computed in the second step using Eq. 1. Given a pixel p and its distance to the light source d_l, this step computes the percentage of depth fragments that occlude the pixel p in the screen-aligned rectangle area centered at p with size m.

$$\text{soft\_shadow}(p) = \sum_{y=-m/2}^{m/2-1} \sum_{\substack{x=-m/2 \\ \text{d}m(x,y)<d_l}}^{m/2-1} 1/(m*m)$$

Once again we can rewrite this formulation as:

$$\text{soft\_shadow}(p) = \sum_{\substack{t=-m/2 \\ \text{d}m(x,t)<d_l}}^{m/2-1} \text{partial\_soft\_shadow}(p_t)/m$$

$$\text{partial\_soft\_shadow}(p_t) = \sum_{\substack{x=-m/2 \\ \text{d}m(x,t)<d_l}}^{m/2-1} 1/m$$

This step is split into two sub-steps that compute the horizontal and vertical soft shadow contributions. The vertical soft shadow step uses the horizontal soft shadow values if and only if the distance between pixels (camera depth distance) is lower than a user-defined threshold β. This distance is modified taking into account the normal of the pixel p. If it is not possible to reuse data, then it is mandatory to access the depth map to get the appropriate depth values.

4.1 Blocker search

PCSS computes blocker distance by brute force. Figure 2 illustrates this process for a given pixel **x**. In this case, a bias of 1 has been taken into account, 20 samples are used (which depth is lower than 14), that sums 163, so blocker distance estimation is $163/20 = 8.15$, then penumbra $= w_{\text{light}} * (15 - 8.15)/8.15 \approx w_{\text{light}} * 0.84$.

We propose to split the Blocker Search stage into two steps (horizontal and vertical). The first step determines the occluders for each fragment and computes the average of the depth values horizontally on a $n \times 1$ area. The second step (vertical) filters the information of $1 \times n$ reusing these values to calculate the average for the whole filtering area. However, in some cases, this information is considered as not useful, since fragments are discarded depending on their relative distances. In these cases, the results of the horizontal step are discarded and the corresponding $n \times 1$ area is entirely evaluated. Finally, this information is used in Eq. 1 to estimate the size of the filtering area, and hence the size of the penumbra. Figure 3 illustrates this process for a pixel **x**. The resulting buffer is filled with three values: light depth, sum of blockers and count of blockers. Horizontal blocker search is computed for every pixel, but results are only shown for relevant pixels.

Then, a second stage computes final blocker distance (see Fig. 4). Blocker distance is computed from horizontal blocker buffer. First, for each pixel, it computes coordinates in screen space, and determines which pixels must be used to compute the blocker distance. Second, it retrieves data from these pixels and determines if data are reusable evaluating distance $|d - d_t| < \alpha$, else retrieves the appropriate data from light depth map. For example, upper element is discarded due to its distance, two units from pixel **x**, and light depth map is accessed. Finally, 17 samples are used, sums 130, and blocker distance estimation is $130/17 = 7.65$ penumbra $= w_{\text{light}} * (15 - 7.64)/7.64 \approx w_{\text{light}} * 0.96$.

For the given pixel, PCSS estimation is penumbra $= 0.84$, applying our algorithm penumbra $= 0.96$. Penumbra Error Computation section presents a study of penumbra error.

4.2 Blurring the hard shadows

The information calculated in the previous step determines the size of the filtering area. The closer the potential blockers

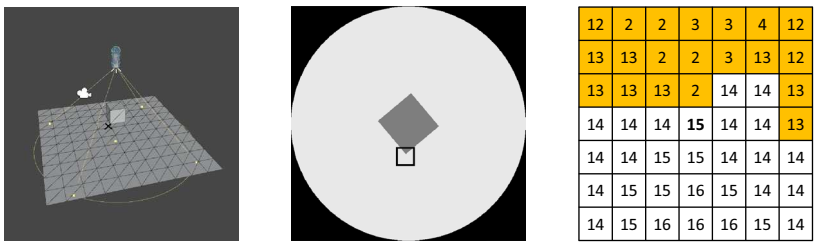


Fig. 2 PCSS blocker search calculation sample. *Left* scene used as example, marked with a **X** the pixel for which the blocker search is computed. *Middle* light-space depth map, *brighter* pixels represent far distances, *darker* pixels represent near distances, the *rectangle* represents the area that must be evaluated for the given pixel. *Right* depth map values used by the centered pixel

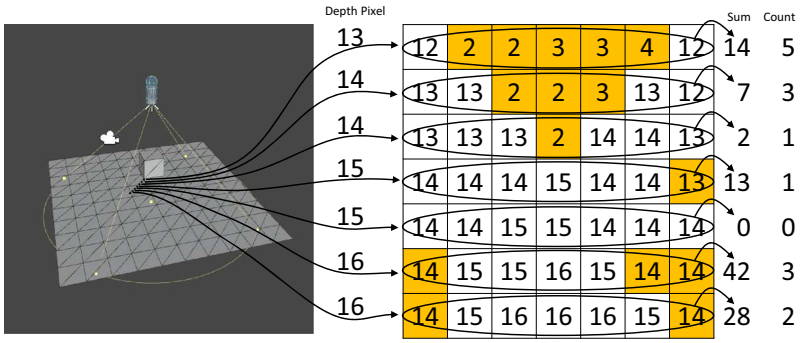


Fig. 3 Separated PCSS blocker horizontal stage sample

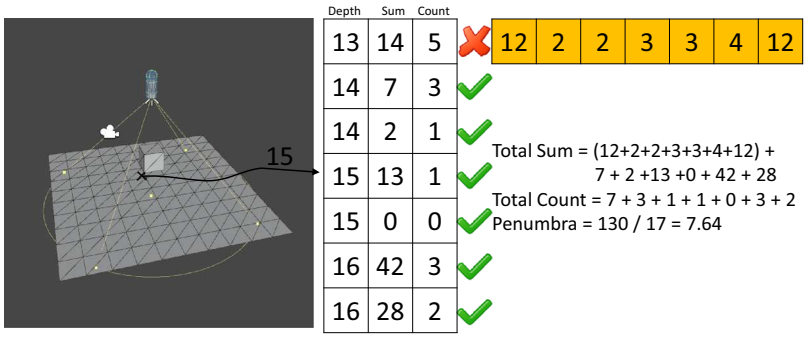


Fig. 4 Separated PCSS blocker vertical stage sample, α is set to 2

are to the current depth, the smaller and harder the soft shadows, and viceversa. In the case that no potential blockers were found in the previous step, it is considered as being totally illuminated. Blurring the hard shadows is done in screen space.

We propose to separate the hard shadow blurring step into two separate steps: horizontal and vertical, similarly to the method used in Blocker Search step. However, this stage operates on the screen buffer rather than on the shadow map. The first step calculates the average of fragments in the horizontal direction performing m samples. The second step searches information on vertical area performing another m samples on the results of the horizontal step. If the current fragment falls inside the allowed threshold β, then the information from the former step is reused, otherwise the shadow map is directly sampled to retrieve the missing data to complete the filtering step. In some cases, it is not possible to perform the filtering operations, such as in the upper and bottom borders of the screen, which would need information that falls out of the screen. This problem is inherent to the family of screen-space algorithms. Figure 1 shows blurring process steps.

Although the conversion of both stages into separable filters provides a theoretical reduction of the computational complexity in one order of magnitude from $O(n^2)$ to $O(n+n)$, this is not true for all cases. Depending on the threshold error β, we can balance the quality-performance tradeoff.

4.3 Algorithm implementation

Our algorithm computes the soft shadows in four different rendering steps (see Table 1). First, the scene is rendered from the light source and the results are stored in the depth map. In the next step, the blocker search is computed horizontally ($partial\_blocker_{sum}$ and $partial\_blocker_{count}$). The third rendering step finishes computing Blocker Average applying the reutilization schema based on the α value. Penumbra values are computed for every pixel. The Hard Shadow map is filtered horizontally using the previously calculated penumbra size. The next step computes Soft Shadow map reusing information coming from previous step, and using threshold β. The last rendering step mixes Soft Shadow map with Color map (computed in step 1) to produce the final image (Fig. 5).

4.4 Mean filter vs. Gaussian filter

Both PCSS and the prefiltering methods use the mean for computing the soft shadows. ESSM performs a mean filter over a circular area but PCSS and VSSM use a rectangular one in contrast. In these cases, the generated shadow appears aligned to the depth map, which is justified by assuming a rectangular light source. Our algorithm can be configured to use any separable filter; however, since SSSM is screen aligned, a rectangular filtering would produce unpleasing visual results. Therefore, we have chosen to use a Gaussian filter, because

Table 1 SSSM algorithm

```
Create DepthMap
Create DepthMap rendering from light source
Step1: Render Scene. Compute partial blocker
Render Scene, computing:
    HardShadowMap, accessing DepthMap
    Others maps (Normal, WorldPosition and Color)
    Blocker Info Map, computed horizontally:
        partial_blocker_sum
        partial_blocker_count
Step2: Compute Blocker Average. Blur Horizontally
Compute Blocker Average
    for each (y point)
        get Blocker Info
        if distance <α
            uses it
        else
            for each (x point)
                get data from DepthMap
Compute Penumbrae Size
if (penumbra == 0)
    soft_shadow_value = hard_shadow_value
Compute Blur HardShadow horizontally
Step3: Compute Soft Shadow. Show final image
Compute Soft Shadow
    for each (y point)
        get Blur HardShadow
        if distance <β
            uses it
        else
            for each (x point)
                get data from HardShadowMap
Render final image mixing SoftShadow and ColorMap
```

it is still separable (like the mean filter) and it emulates a circular area light source, removing the undesired alignment issues (see Fig. 5). Using a Gaussian filter involves weighting the samples accordingly and does not affect performance (Figs. 6, 7).

5 Results

This section shows the results of our method by analyzing the obtained performance comparing it to other methods while evaluating the visual error introduced by the user-defined thresholds (α and β). All tests use 1,024 $\times$ 1,024 shadow maps (Fig. 1) rendered to a 1,280 $\times$ 720 HD viewport, running on a GeForce GT 555M (drivers v314.07), and an Intel Core I7-2670QM @2.20 GHz CPU.

5.1 Performance

We have chosen the plant scene (Fig. 8) for all test comparisons (performance, penumbra error, image quality). The performance achieved with our algorithm is better than the base algorithm (PCSS). Moreover, using the appropriate thresholds, our algorithm is able to get better performance than ESSM without compromising the visual quality of the shadows.

In the test scene, PCSS ($29 \times 29/41 \times 41$) obtains 4.52 fps while ESSM ($H = 4$) runs at 8.02 fps. In the same scene, SSSM ($29 \times 29/41 \times 41$) obtains the same rendering performance as ESSM by setting the thresholds to $\alpha = \beta = 0.02$ (8.12 fps). However, using more reasonable values for the thresholds ($\alpha = \beta = 0.10$), the performance is greater to both ESSM and PCSS (see Fig. 6). Table 2 shows the results for a 1,024 $\times$ 1,024 depth map, while Table 3 shows the results for 4,096 $\times$ 4,096 a depth map. Table 4 shows performance comparison between PCSS and SSSM. Note that SSSM achieves higher performance when high-quality filters are used.

Our performance results show how performance depends on kernel sizes for each different approach. Using small kernels is the worst scenario for our technique, since the cost of adding different rendering steps is greater than the benefits we obtain. In this case, our cost is comparable with the brute force approach (PCSS). SSABSS on the other hand is the winner in this scenario, since it requires less rendering steps. With medium-sized kernels, our rendering performance is comparable with SSABSS depending on the reutilization parameters used. However, the usage of large kernels fully exploits the benefits of our technique, being able to provide high-quality smooth penumbrae while maximizing rendering performance. Figure 9 shows the differences in rendering quality using large filtering kernels compared to small kernels. The necessity of using large filtering kernels depends on each single scene/camera/light configurations. Large light sources would require larger penumbrae and larger filters. Also, certain camera/light configurations would require high-quality shadow filtering when rendering penumbrae at grazing angles.

Finally, it has be mentioned that, according to Tables 2 and 3, rendering step 2 is more computationally expensive than the other steps. This is produced as a consequence of putting together two different steps in the same rendering step as shown in the algorithm in Table 1.

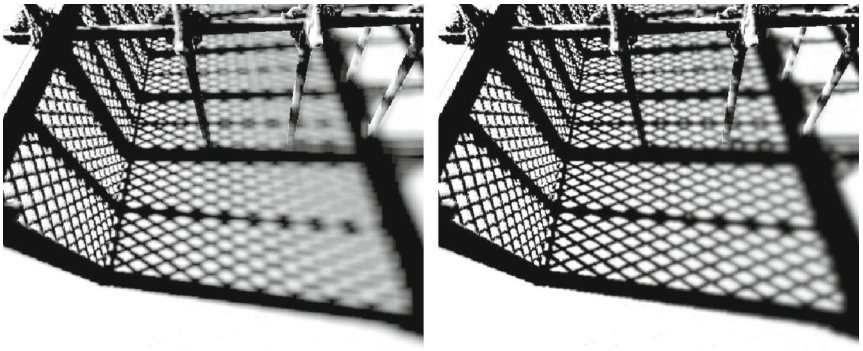
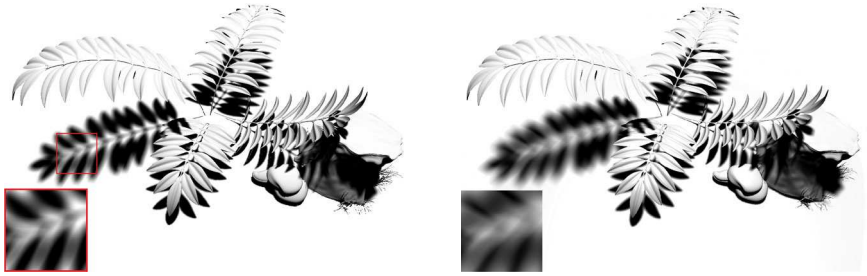
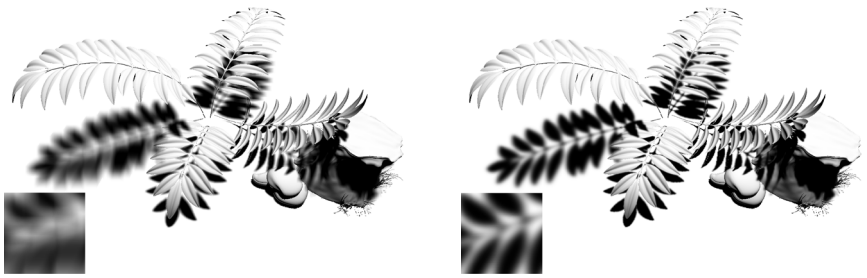


Fig. 5 Mean vs. Gaussian filter comparison. *Left* image rendered with mean filter produces screen space alignment artifacts. *Right* image rendered with gaussian filter, which forces precomputed weights



(a) PCSS $29 \times 29/41 \times 41$ (4.52 fps)

(b) ESSM $H = 4$ (8.02 fps)



(c) SSSM $29 \times 29/41 \times 41$ Mean Blur (12.51 fps)

(d) SSSM $29 \times 29/41 \times 41$ Gaussian Blur (12.51 fps)

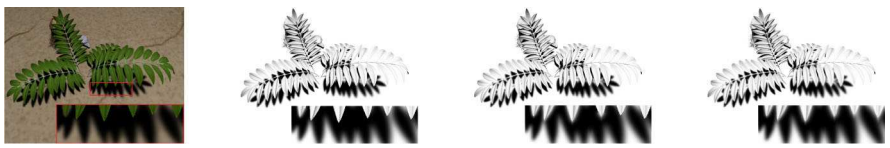


Fig. 7 Artifacts produced by a high threshold. From *Left to right* Colored $\alpha = \beta = 0.10$. Soft shadow map $\alpha = \beta = 0.10$, $\alpha = \beta = 0.15$ and $\alpha = \beta = 0.20$. Increasing threshold produces artifacts at the border of floor with leaves. Leaves from floor are also affected when threshold is set at 0.20

5.2 Penumbra error computation

In this section, we evaluate the penumbra error generated using different values for the α parameters. Table 5 shows the average error in texels^2 introduced at the penumbra calculation step, which is around half texel for a 1,024 × 1,024 shadow map and $\alpha = 0.10$. We consider this error very low taking into account the benefits obtained by the method. Figure 10 shows different reutilization levels for the plant scene.

5.3 Image quality analysis

To determine the quality of the shadows generated by our method, we compare our results with PCSS (because it is a

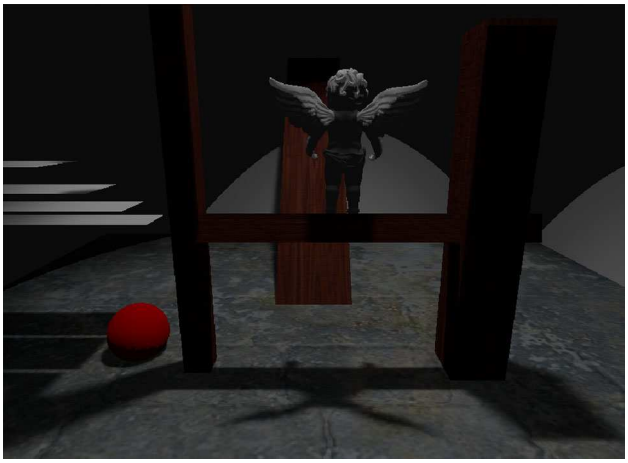


Fig. 8 Scene used to evaluate performance, penumbra error and image quality

Table 2 Rendering times in milliseconds for plant scene, and varying α and β values

α β	Depth map size 1,024 × 1,024				
	0.01	0.10	0.15	0.20	0.50
DM	0.46	0.46	0.46	0.46	0.46
Step1	13.67	13.71	13.92	13.81	13.60
Step2	41.23	34.70	32.42	30.53	24.86
Step3	73.29	33.03	23.88	19.30	9.19
Total	128.21	81.45	70.22	63.65	47.66
fps	7.80	12.27	13.84	15.71	20.98

SSSM configuration is 29 × 29 for blocker search and 41 × 41 for blur shadows

Table 3 Same configuration previous table, but increasing depth map size to 4,096 × 4,096

α β	Depth map size 4,096 × 4,096				
	0.01	0.10	0.15	0.20	0.50
DM	2.04	2.05	2.04	2.04	2.06
Step1	22.72	22.54	22.77	22.48	22.77
Step2	52.98	39.38	36.37	34.37	26.36
Step3	86.28	37.41	26.93	20.04	10.06
Total	161.99	99.34	86.08	76.89	59.20
fps	6.17	10.06	11.61	13.05	16.89

Frame rate does not decrease drastically increasing depth map size

Table 4 Performance comparison in FPS between PCSS, SSSM and SSABSS varying filter size

	PCSS		
	7/15	11/21	29/41
	47.52	23.90	4.85
	SSABSS		
β	7/15	11/21	29/41
0.01	81.75	37.55	6.74
	SSSM		
α β	7/15	11/21	29/41
0.01	39.59	17.81	9.15
0.10	48.36	28.80	15.67
0.15	49.34	31.50	17.90
0.20	49.71	33.52	19.20
0.50	51.12	34.75	21.88

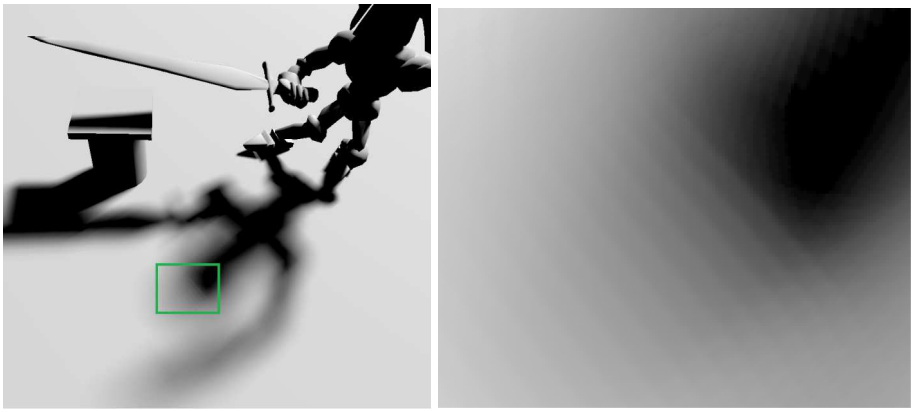
Depth map size is 1,024 × 1,024 and rendering viewport is 1,280×960

well-known method), ESSM (because it is the latest proposal in the literature for prefiltered soft shadows) and ground truth (most realistic). Since our technique is compatible with any separable filter, we have tested mean filter (because it is used on most shadow techniques) and Gaussian filter (because it produces visually pleasing results). However, as discussed in Sect. 4.4, it would be preferable to choose the Gaussian filter for best results when working with SSSM (see Fig. 6). A comparison is shown in Fig. 13. Resulting images are similar except at the edges, where increasing α and β values also increase these errors. Table 6 shows numerical data for these images (light size = 0.4) and for an extra large light size (0.10). This error analysis has been carried out computing PSNR and SSIM taking as reference ground truth image. From PSNR and SSIM analysis, we can conclude than SSSM gets worse results than PCSS, but the final image has enough quality.

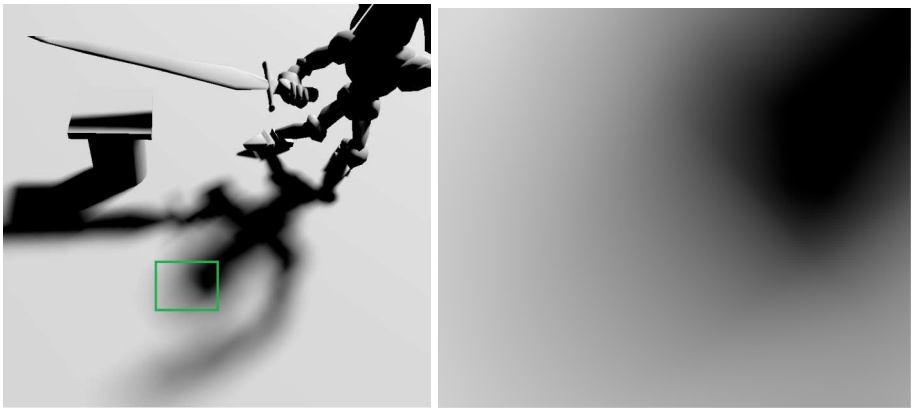
The SSSM parameters α and β determine the amount of data reutilization between the horizontal and vertical filtering steps. The higher values allow for faster but more error is introduced in the results and viceversa. As it can be seen in Fig. 7, if the β thresholds are too high, the process produces wrong results for some pixels between the boundaries of the leaves and the ground (*ghosting*) (see Fig. 13). Therefore, these values must be carefully chosen depending on the scene as a trade-off with the rendering performance.

We have tested the visual quality of the shadows generated by SSSM in different scenes, supporting even multiple light sources with different parameters (size, color and thresholds) seamlessly as shown in Fig. 11.

Finally, Fig. 12 illustrates the advantages of our technique compared to the state-of-the-art of screen space shadow map filtering techniques (SSABSS). SSABSS uses a bilateral filtering approach which is able to preserve shape borders and contours avoiding filtering unrelated shadow areas. However, as seen in this figure, SSABSS fails at rendering continuous smooth penumbrae in certain cases where the edges of hard shadows are near a depth discontinuity (object occlusion) in screen space or near the viewport limits.



(a) 15×15 filter size



(b) 41×41 filter size

Fig. 9 Image quality comparison with different kernel sizes

Table 5 Penumbra error Mean Squared Error (MSE) in texels^2 varying α

α	Penumbra filter size		
	7×7	11×11	29×29
0.01	0.452	0.436	0.433
0.10	0.540	0.532	0.534
0.15	0.572	0.568	0.575
0.20	0.614	0.630	0.647
0.50	0.815	0.986	1.131

Table 6 PSNR and SSIM values for PCSS and SSSM (different $\alpha = \beta$ values), taking as reference Ground Truth image

	PSNR		SSIM	
	0.04	0.10	0.04 (%)	0.10 (%)
PCSS	35.15	34.30	99.14	98.72
SSSM 0.01	31.23	30.82	98.05	97.44
SSSM 0.10	28.57	26.68	97.11	95.81
SSSM 0.15	27.93	25.89	96.84	95.38
SSSM 0.20	27.79	25.71	96.79	95.31
SSSM 0.50	27.78	25.73	96.80	95.41

These two metrics have been computed for two light sizes 0.04 (default light size) and 0.10 (extra large light size)

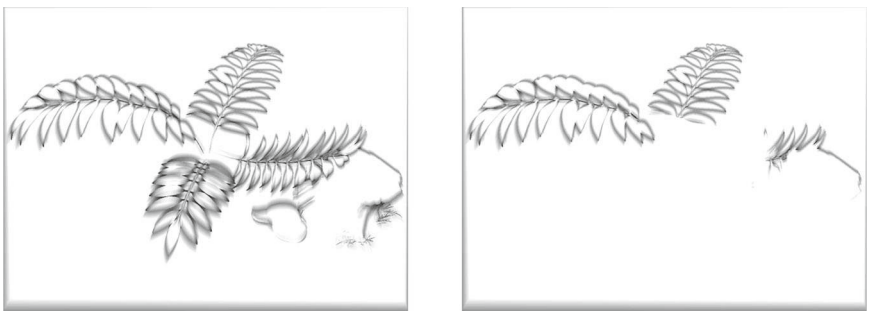


Fig. 10 Reutilization percentage representation in penumbra computation process for $\alpha = 0.01$ and $\alpha = 0.10$. *Black* means no reutilization, *white* 100 % reutilization

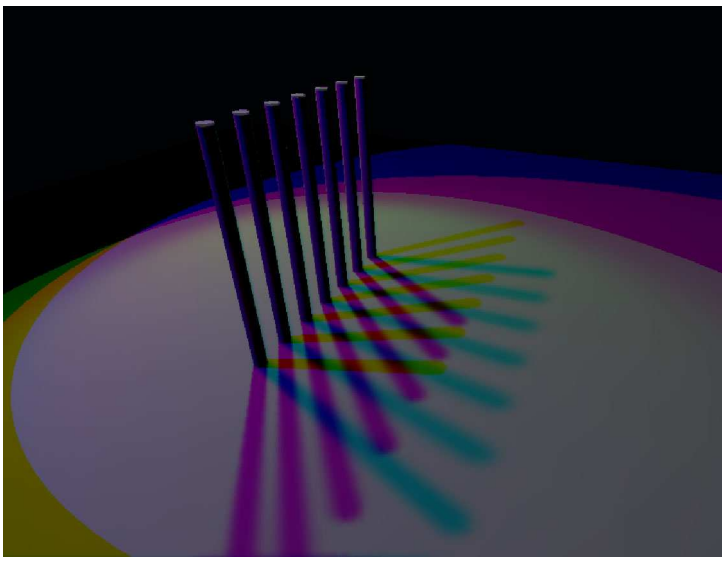
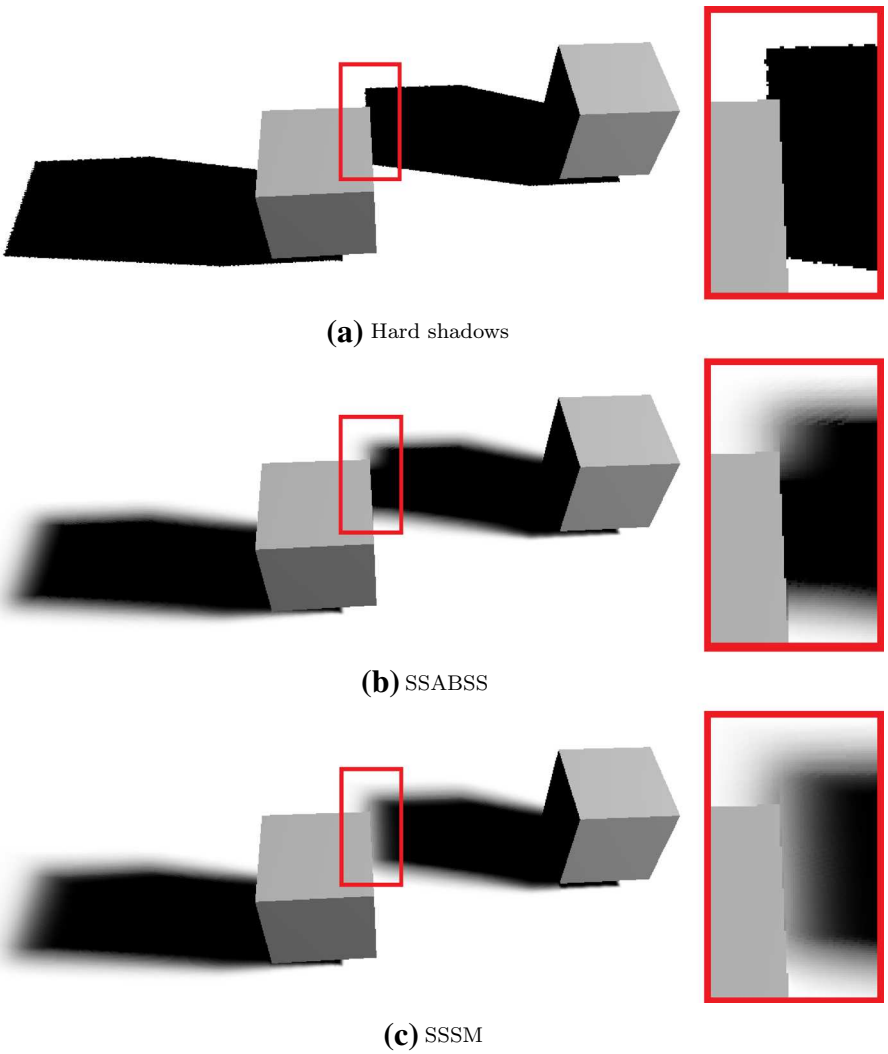


Fig. 11 Our technique working with light sources of different sizes. Rendering time is linear to the number of lights. The frame rate obtained with 1, 2 and 3 lights is 34.9, 18.7 and 12.5, respectively

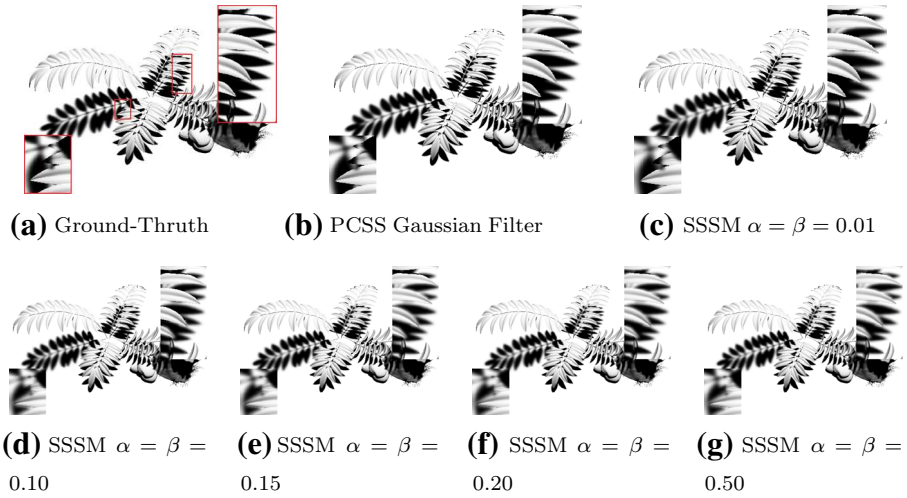


(a) Hard shadows

(b) SSABSS

(c) SSSM

Fig. 12 Failure case scene for SSABSS (*middle*). Our algorithm (*bottom*) is able to better produce smooth penumbrae in the same case. Zooms over the affected area are shown in the *right column*



(a) Ground-Thruth **(b)** PCSS Gaussian Filter **(c)** SSSM $\alpha = \beta = 0.01$

(d) SSSM $\alpha = \beta = 0.10$ **(e)** SSSM $\alpha = \beta = 0.15$ **(f)** SSSM $\alpha = \beta = 0.20$ **(g)** SSSM $\alpha = \beta = 0.50$

Fig. 13 Image quality comparison, filter size is $29 \times 29/41 \times 41$. Ground Truth image has been generated rendering 41×41 light sources weighted by the same Gaussian distribution than PCSS and SSSM

6 Limitations

Although our method improves rendering speed over other methods, as shown in the previous section, it also has some limitations. First, since the shadow blurring step is performed in screen space, it works better for flat surface rather than bumpy surfaces and, depending on the parameters, it can produce ghost artifacts around near objects as seen in Fig. 13. However, our algorithm takes into account the surface orientation and the distance of the samples to minimize rendering errors. Another limitation is the need to adjust user-defined parameters α, β that may depend on the configuration of each scene.

7 Conclusions and future work

Soft shadows are physically produced by area light sources. Rendering realistic soft shadows in real time is a costly process and it is still an open research field by itself. Approximating penumbrae using point light sources rather than using area light sources allows to reduce the computational cost of this process while generating visually plausible soft shadows. We have proposed a novel technique for estimating and rendering soft shadows that reduces the computational cost of traditional methods from $O(n^2)$ to $O(n+n)$. SSSM can be considered a different branch of soft shadowing algorithms based on PCSS, since it does not use prefiltering like CSSM, VSSM or ESSM. As a consequence of avoid prefiltering the depth-map, our technique scales better when high-resolution depth maps are used. Finally, compared to the state-of-the-art in screen space soft shadow mapping (SSABSS), our algorithm is able to better produce smooth penumbrae near depth discontinuities or near the viewport limits, where shadowing information is not present in screen space, and at the same time maximizes rendering performance for large filtering kernels.

We have analyzed how the visual error depends on the user-defined threshold and how that improves rendering performance. The SSSM algorithm allows to use larger filters while increasing the rendering performance, producing smoother penumbrae in real time.

As future work, we propose to evaluate our algorithm by merging it with other orthogonal techniques such as Perspective Shadow Maps, Adaptative Shadow Maps, Cascade Shadow Maps, Cube Shadow Map or with algorithms that exploit temporal coherence between frames. It can be also interesting to better reuse information between the blocker search and the blur shadow steps and, finally, minimize the limitations of our method as described in the previous section.

Acknowledgments This work has been supported by the Spanish Ministry of Education and Science (TIN2013-47276-C6-6-R), the Valencian Community (PROMETEOII/2014/062) and the University Jaime I (P1·1B2014-37).

References

1. Annen, T., Dong, Z., Mertens, T., Bekaert, P., Seidel, H.P., Kautz, J.: Real-time, all-frequency shadows in dynamic scenes. ACM Trans. Graph. **27**(3), 1–8 (2008)
2. Annen, T., Mertens, T., Bekaert, P., Seidel, H.P., Kautz, J.: Convolution shadow maps. In: Rendering Techniques 2007: Eurographics Symposium on Rendering, pp. 51–60. Eurographics, Grenoble, France (2007)
3. Annen, T., Mertens, T., Seidel, H.P., Flerackers, E., Kautz, J.: Exponential shadow maps. In: GI '08: Proceedings of graphics interface 2008, pp. 155–161. Canadian Information Processing Society, Toronto, Ontario, Canada (2008)
4. Atty, L., Holzschuch, N., Lapierre, M., Hasenfratz, J.M., Sillion, F.X., Hansen, C.: Soft shadow maps: efficient sampling of light source visibility. Comput. Graph. Forum **25**(4), 725–741 (2006). doi:10.1111/j.1467-8659.2006.00995.x. http://hal.inria.fr/inria-00281374

5. Baoguang, Y., Feng, J., Guennebaud, G., Liu, X.: Packet-based hierarchal soft shadow mapping. Comput. Graph. Forum **28**(4), 1121–1130 (2009). http://hal.inria.fr/inria-00390541
6. Cai, X.H., Jia, Y.T., Wang, X., Hu, S.M., Martin, R.R.: Rendering soft shadows using multilayered shadow fins. Comput. Graph. Forum **25**(1), 15–28 (2006). http://dblp.uni-trier.de/db/journals/cgf/cgf25.html
7. Chan, E., Durand, F.: Rendering fake soft shadows with smoothies. In: Proceedings of the Eurographics Symposium on Rendering, pp. 208–218. Eurographics Association (2003)
8. Doidge, I., Jones, M.: Probabilistic illumination-aware filtering for monte carlo rendering. Vis. Comput. **29**(6–8), 707–716 (2013). doi:10.1007/s00371-013-0807-3
9. Dong, Z., Yang, B.: Variance soft shadow mapping. In: Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, I3D '10, pp. 18:1–18:1. ACM, New York, NY, USA (2010). doi:10.1145/1730804.1730990
10. Donnelly, W., Lauritzen, A.: Variance shadow maps. In: Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games, I3D '06, pp. 161–165. ACM, New York, NY, USA (2006). doi:10.1145/1111411.1111440
11. Eisemann, E., Schwarz, M., Assarsson, U., Wimmer, M.: Real-Time Shadows. A.K. Peters, Natick, Massachusetts, USA (2011). http://www.cg.tuwien.ac.at/research/publications/2011/EISEMANN-2011-RTS/
12. Fernando, R.: Percentage-closer soft shadows. In: ACM SIGGRAPH 2005 Sketches, SIGGRAPH '05. ACM, New York, NY, USA (2005). doi:10.1145/1187112.1187153
13. Guennebaud, G., Barthe, L., Paulin, M.: Realtime soft shadow mapping by backprojection. In: Eurographics Symposium on Rendering, pp. 227–234 (2006)
14. Guennebaud, G., Barthe, L., Paulin, M.: High-quality adaptive soft shadow mapping. In: Eurographics 2007 Proceedings on Computer Graphics Forum, vol. 26(3), pp. 525–534 (2007)
15. Gumbau, J., Chover, M., Sbert, M.: Screen space soft shadows. In: Engel, W. (ed.) GPU Pro, pp. 477–491. A.K. Peters/CRC Press, Natick, Massachusetts, USA (2010) doi:10.1201/b10648-36
16. Hasenfratz, J.M., Lapierre, M., Holzschuch, N., Sillion, F.X.: A survey of real-time soft shadows algorithms. Comput. Forum **22**(4), 753–774 (2003). doi:10.1111/j.1467-8659.2003.00722.x
17. MohammadBagher, M., Kautz, J., Holzschuch, N., Soler, C.: Screen-space percentage-closer soft shadows. In: ACM SIGGRAPH 2010 Posters, SIGGRAPH '10, pp. 133:1–133:1. ACM, New York, NY, USA (2010). doi:10.1145/1836845.1836987
18. Reeves, W.T., Salesin, D.H., Cook, R.L.: Rendering antialiased shadows with depth maps. SIGGRAPH Comput. Graph. **21**(4), 283–291 (1987). doi:10.1145/37402.37435
19. Schwarz, M., Stamminger, M.: Bitmask soft shadows. Comput. Graph. Forum **26**(3), 515–524 (2007)
20. Schwärzler, M., Luksch, C., Scherzer, D., Wimmer, M.: Fast percentage closer soft shadows using temporal coherence. In: Proceedings of the ACM SIGGRAPH symposium on interactive 3D graphics and games, I3D '13, pp. 79–86. ACM, New York, NY, USA (2013). doi:10.1145/2448196.2448209
21. Shen, L., Feng, J., Yang, B.: Exponential soft shadow mapping. In: Eurographics Symposium on Rendering, vol. 32. The Eurographics Association and Wiley (2013). doi:10.1111/cgf.12156
22. Sintorn, E., Kämpe, V., Olsson, O., Assarsson, U.: Compact precomputed voxelized shadows. ACM Trans. Graph. **33**(4), 150:1–150:8 (2014). doi:10.1145/2601097.2601221
23. Williams, L.: Casting curved shadows on curved surfaces. In: Proceedings of the 5th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '78, pp. 270–274. ACM, New York, NY, USA (1978). doi:10.1145/800248.807402
24. Zheng, Z., Saito, S.: Screen space anisotropic blurred soft shadows. In: ACM SIGGRAPH 2011 Posters, SIGGRAPH '11, pp. 75:1–75:1. ACM, New York, NY, USA (2011). doi:10.1145/2037715.2037799

Jose María Buades is an associate professor at the Computer and Mathematics Department, University of Balearic Islands, Spain. He received his B.Sc degree in Computer Science in 1999 and PhD degree in Computer Science in 2006, all from the University of Balearic Islands. His research focuses on computer graphics, real-time rendering and human–computer interaction.

Jesús Gumbau received his MS degree in Computer Science in 2004 from the Universitat Jaume I, Castellon, Spain and his Ph.D. degree from the same university in 2011. His research areas include multiresolution modelling, real-time visualisation and real-time shadow rendering.

Miguel Chover received his MS degree in Computer Science in 1992, and his PhD in Computer Science in 1996, from the Universidad Politécnica de Valencia, Valencia, Spain. Since 1992, he has been an Assistant Professor of Computer Science at the department of Computer Languages and Systems at the Universitat Jaume I, Spain. His research areas include multiresolution modelling, real time visualisation and collaborative virtual worlds. Dr. Chover is a member of Eurographics.