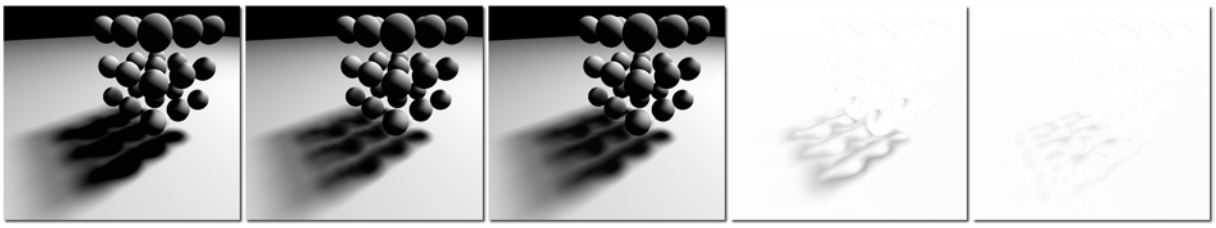
Graphics Hardware (2006)
M. Olano, P. Slusallek (Editors)

Realistic Soft Shadows by Penumbra-Wedges Blending

Vincent Forest, Loïc Barthe, Mathias Paulin

IRIT-UPS-CNRS, University of Toulouse, France
Vincent.Forest/Loic.Barthe/Mathias.Paulin@irit.fr



(a) *Penumbra wedge* (b) *Our algorithm* (c) *1024-sample shadows* (d) *Difference between (a) and (c)* (e) *Difference between (b) and (c)*

Figure 1: *A cube composed of* 9×9 *spheres. This scene illustrates the penumbra blending generated by different soft-shadow algorithms: the penumbra-wedges, our method and a reference image.*

Abstract

Recent real-time shadow generation techniques try to provide shadows with realistic penumbrae. However, most techniques are whether non-physically based or too simplified to produce convicing results. The penumbra-wedges algorithm is a physical approach based on the assumption that penumbrae are non-overlapping. In this paper, we propose an algorithm that takes the advantages of the penumbra-wedges method but solves the "non-overlapping" limitation. We first compute the light occlusion regions per fragment. Then we use this information to detect the areas where penumbrae are overlapping and we perform a realistic penumbra blending.

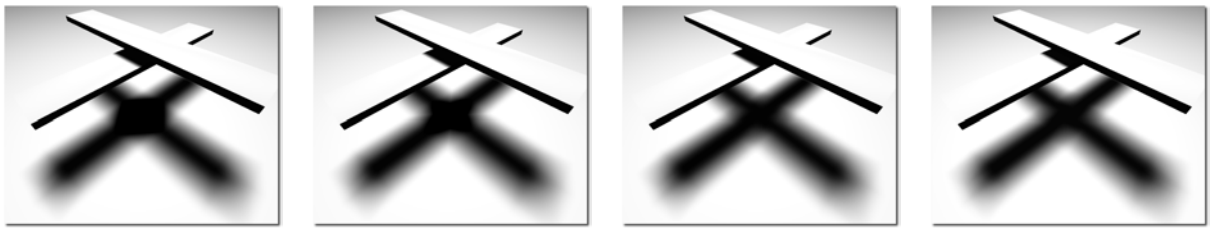
Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Line and Curve Generation

1. Introduction

Hard shadows are now standard in real-time rendering applications. Even though hard shadows give clues on objects relationship, they are based on the restrictive approximation that lights are points. Real light sources are in general extended and they generate an area of penumbra on the shadow boundaries. Recent algorithms try to enhance shadow realism by producing realistic soft shadows.

In this paper we present a physically based real time soft shadow algorithm for rectangular lights. This algorithm is based on the penumbra-wedges method presented by Assarsson and Akenine-Möller [AMA02]. The penumbra-wedges algorithm computes a per-fragment coefficient that represents the light occlusion percentage. The light occlusion computation is independent for each silhouette and hence the result is only a very coarse approximation as soon as silhouettes are overlapping. In order to minimize the approximation error, we first find which fragments can produce overlapping error. Then we approximate the error made by the conventional penumbra-wedges algorithm and use it to derive a physically plausible visibility coefficient.

This paper is structured as follow : after discussing previous works, we present our algorithm. Then, we describe



(a) (b) (c) (d)

Figure 2: *Penumbra blending comparison. From left to right : (a) standard penumbra-wedges algorithm [AMA02], (b) penumbrae blending by the probabilistic approach proposed by Assarsson et al. [ADMAM03], (c) our algorithm, (d) reference image using 1024-sample shadows*

its hardware implementation and some GPU optimizations, followed by an analysis of the results and performance. We conclude with a discussion and our future works

2. Previous works

We present a synthetic overview of the main shadow techniques. For a complete overview, see Woo *et al.* [WPF90] for standard shadow algorithms and Hasentfratz *et al.* [HLHS03] for real-time shadows.

The two main algorithms generating hard shadows are shadow-volumes [Cro77] and shadow-map [Wil78]. The shadow-map algorithm first renders the scene as seen from the light and captures a depth image named shadow-map. To determine whether a fragment is in shadow or not, its depth from the light is compared to the depth stored in the shadow-map. The shadow-map approach is image-based and hence the quality of the shadows mainly depends on the shadow-map resolution. In order to reduce aliasing, Fernando *et al.* [FFBG01] propose an adaptive-algorithm that increases the shadow-map resolution only where necessary. With the same goal, Stamminger *et al.* [SD02] introduce the perspective shadow-maps which are shadow-maps generated in normalized device coordinate space, i.e., after perspective transformation.

There are many algorithms based on shadow-map that simulate soft shadows. Heidrich *et al.* [HBS00] generate soft shadows for linear lights. They interpolate the visibility of a fragment using several shadow-maps. Brabec and Seidel [BS02] use one sample of the light for rendering "soft edges" of the shadow. Wyman and Hansen [WH03] compute only outer-penumbra. They modulate the shadow-map by a penumbra-map that stores the penumbra intensity of the first visible polygon. The same idea is in Chan and Durand's approach [CD03]. They use geometry primitives called *smoothies* to modulate the shadow-map. Atty *et al.* [AHL*05] propose a physically based algorithm. They sample the light and accumulate the masked area for each sample. More recently, Guennebaud *et al.* [GBP06] use a shadow map as a simple and uniform discretized representation of the scene, thus allowing them to generate realistic soft shadows in most cases. Several algorithms also try to simulate soft shadows by filtering hard shadow boundaries [RSC87]. However, this is an approximation only correct in very specific cases

Even though some algorithms reduce the shadow-map aliasing, the shadows quality remains dependent of the shadow-map resolution. On the other hand, shadow-volumes [Cro77] generate very accurate shadows. This algorithm builds a mask that defines which fragments are in shadow. The first pass computes ambient lighting contribution. Then each silhouette is extruded from the light source to build a shadow-volume quadrilateral. The collection of these quadrilaterals represents a shadow-volume which is rendered in the eye space to update the stencil buffer. This update was initially done by the non-robust Z-pass algorithm, less fillrate intensive than the most recent and robust Z-fail approach [BS99] [Car00] [EK02]. The fillrate can be reduced by applying the Z-fail algorithm only where necessary. In that case, the detection is done by object or better, by fragment [Lai05]. After this step, fragments that have non-zero stencil value are considered in shadow. Even though shadow-maps are more popular, shadow-volumes have the important advantage of producing exact and accurate shadows. The drawback is an increase of the fillrate due to the creation of the shadow-volumes geometry. However, we demonstrate in this paper that an accurate occlusion evaluation placed on top of an efficient shadow-volumes algorithm (the penumbra-wedges algorithm) provides very realistic soft shadows in real time, even on complex scenes.

The penumbra-wedges algorithm [AMA02] is a physically based soft shadow-volumes algorithm which can be summarized as follow:

1. Compute specular and diffuse lighting contribution
2. **Compute a visibility buffer**
3. Modulate specular-diffuse image with the visibility buffer

4. Add ambient lighting contribution

The visibility buffer (vBuffer) stores coefficients representing the light occlusion percentage. First, the buffer is initialized by a shadow-volumes pass. During this step, coefficients one and zero are affected respectively to shaded and unshaded fragments. The second pass modulates the coefficient for fragments in the penumbra. The silhouette edge generated during the shadow-volumes pass is extruded into a quadrilateral that splits the penumbra-wedge in two parts (figure 3). The inner and the outer half-wedge bound respectively the inner and outer-penumbra and they are treated independently. In both cases, the silhouette edge is projected onto the light plane to compute the light occlusion area. This area is subtracted or added to the visibility buffer depending if the fragment is treated as an inner or an outer-penumbra fragment. In short, this approach performs the integration of the light occlusion area from the center of the light (figure 4). Assarsson and Akenine-Möller [ADMAM03] [AMA02] [AAM03] have implemented this algorithm on GPU to reach interactive performances. More recently, Lengyel [Len05] improves their implementation with fillrate optimizations, a more accurate detection of the fragments in penumbra and a more optimized fragment-program.

Even though the penumbra-wedges approach generates non-aliased and physically plausible soft shadows, this algorithm is based on the assumption that the silhouettes are non-overlapping. This assumption is seldom verified in practice and the method often generates very approximative soft shadows where penumbrae overlap (figure 2(a)). Assarsson minimizes the overlapping artifacts by a probabilistic approach [ADMAM03]. As illustrated in figure 2(b) this improvement still produces unrealistic soft shadows. The accurate combination of several silhouettes remains in fact a fundamental and open problem.

Note that even though smooth imperfections in the penumbra are seldom perceptible, the overlapping artifact produces over-shadowed areas that attract attention (figure 1(a)). Our algorithm uses penumbra-wedges to generate non-aliased physically based soft shadows for rectangular lights. In the next section, we propose a new technique for suppressing artifacts when penumbrae are overlapping, at interactive frame rates.

3. Realistic soft shadows by penumbra-wedges blending

When two silhouettes are overlapping the overlapping occluded area is counted twice (figure 5) resulting in an over-shadowed region. For a realistic penumbra blending we need to define the geometry of light occlusion per fragment. This is done by computing a visibility buffer per silhouette (section 3.1) and then blending it within a final visibility buffer (section 3.2).

3.1. Silhouette visibility buffer computation

The accurate blending of soft shadows requires the knowledge of the geometry of the occluded light region for each fragment. On the other hand, on current hardware, the number of parameters that can be stored per fragment is very limited and thus the representation of the occluded light region has to be compact. This compact representation is obtained by dividing the light area in four parts using a radial subdivision and by computing for each subdivision a rectangle that bounds the occluded region. This produces a more accurate occluded light region detection as shown in figure 6. The occlusion area is now integrated independently from the center and per radial part. The following algorithm describes the computation of the silhouette visibility buffer for a single radial part:

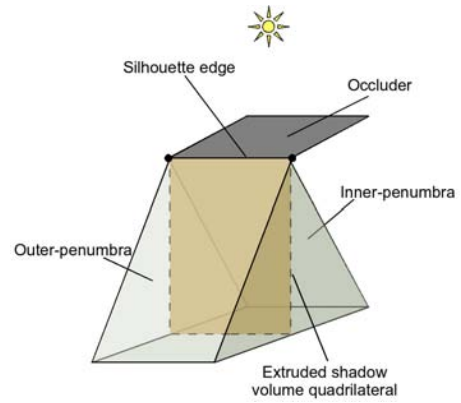


Figure 3: *Illustration of a penumbra-wedge.*

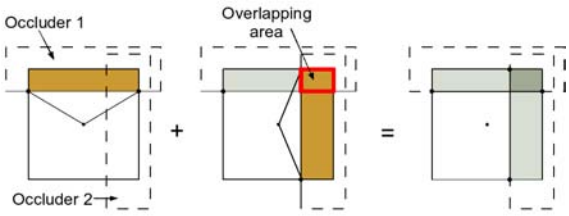


Figure 5: *When the light occlusion area of the second occluder is added to the visibility buffer, the previous bounding area(in red) is already stored. The overlapping causes an over-estimation of the light occlusion.*

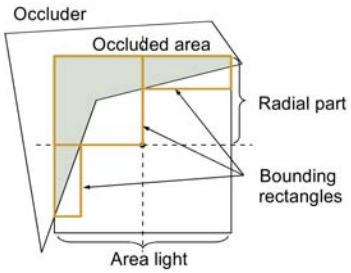


Figure 6: *The light area is subdivided into radial parts. Note that this gives better determination of the occluded light region than a non-subdivide approach*

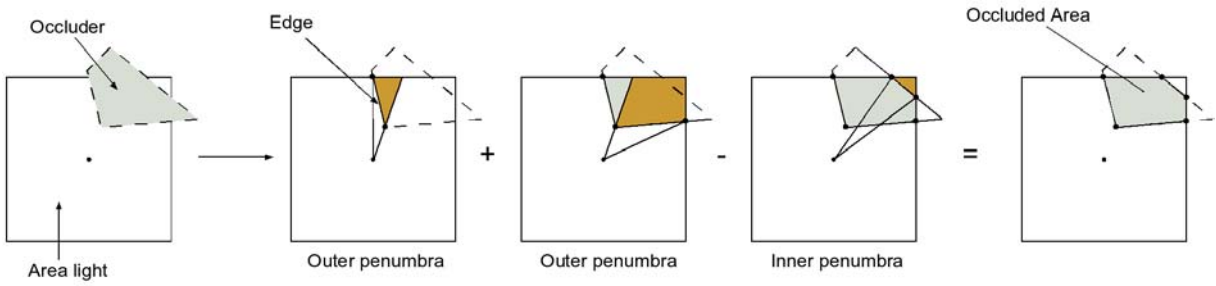


Figure 4: *Light occlusion area computation. For a fragment, each edge of the occluder is projected and clipped on the light plane. The occluded area is added or subtracted if the edge is part of respectively the outer or the inner-penumbra.*

for each silhouette-edge **do**
1. Compute the occluded light percentage and, as in the penumbra-wedges algorithm, add or substract it to the visibility buffer
2. Compute a bounding rectangle of the occluded light region and combine it with the bounding rectangle computed previously
end for

The bounding rectangle gives a per-fragment approximation of the occluded light region that allows us to detect where occluders are overlapping.

3.2. Silhouette visibility buffer blending

The blending of a silhouette visibility buffer is done using a per radial blending function defined as follow:

for each fragment **do**
1. Accumulate the light occlusion percentage as in the penumbra-wedges algorithm
2. Identify the overlaps in the occluded regions
3. Approximate the effective overlapping area and compute its corresponding occluded light percentage
4. Substract it from the visibility coefficient computed in the first step
5. Compute a new bounding rectangle of the occluded light region which is the combination of the silhouette and the final vBuffer bounding rectangle
end for

During the first step, the blending function proceeds as in the penumbra-wedges algorithm. In the second step, we identify where the occluded regions can overlap by computing the bounding rectangles intersection. Since bounding rectangles region can be partially occluded, we weight their intersection by a coefficient of occlusion as follow:

$$\varepsilon = \frac{A_f}{B_f} \cdot \frac{A_s}{B_s} . \beta$$

with :

- ε : effective overlapped area
- β : bounding rectangles intersection area
- $_{s,f}$: indices identifying respectively silhouette and final vBuffer
- A_x : light occlusion area
- B_x : bounding rectangle area

The last step of our blending function combines the two bounding rectangles in a new one, used as the input for the next blending computations.

4. Hardware implementation

Our algorithm is a per-silhouette 3 passes algorithm:

1. Initialize the silhouette vBuffer by a shadow-volumes pass
2. Compute occluded light percentage by the penumbra-wedges pass
3. Blend the silhouette occluded light percentage with the final vBuffer

As we can see, our approach is more GPU intensive than the previous real-time implementation of the penumbra-wedges algorithm [AMA02] [ADMAM03] [AAM03]. Therefore, we first focused on an optimized real-time implementation of the penumbra-wedges algorithm.

4.1. Shadow volumes

Our shadow-volumes implementation is based on the Carmack's approach [Car00]. Because the silhouette edges selection is CPU intensive, we use a half-edge structure both accelerating the edge selection and providing explicitly the connectivity of the silhouette. Hence, shadow-volumes can be stored as triangle strips and the extrusion of the shadow-volumes quadrilaterals can be made on the GPU (freeing the CPU for other tasks). In the case of attenuated light sources, the fillrate cost is reduced using a per-light scissor and depth-bound test according to Lengyel's presentation [Len05].

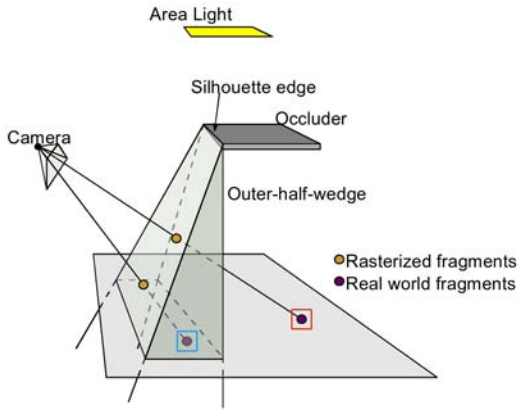


Figure 7: *Quick test defining whether a viewport space fragment is outside an half-wedge or not. The rasterized fragment is rejected if its corresponding real-world fragment (bounded by the red rectangle) is outside any plane of the half-wedge. However the real-world fragment bounded by the blue rectangle is inside the half-wedge and thus the rasterized fragment is concerned by the light occlusion computation.*

4.2. Optimized penumbra-wedges

The visibility buffer computation algorithm is based on the implementation of Assarsson, Akenine-Möller and Lengyel. First, following Lengyel, we compute the three outside bounding planes of each half-wedge during penumbra-wedges construction. These planes are transformed in viewport-space by a vertex program and sent to the fragment program which performs a quick test to know whether a viewport-space fragment is outside the half-wedge or not (figure 7). In addition, according to the Lengyel's approach, we use a depth test to reject fragments that are on the wrong side of the shadow-volume quadrilateral (figure 8). Finally, the silhouette edge is projected onto the light plane.

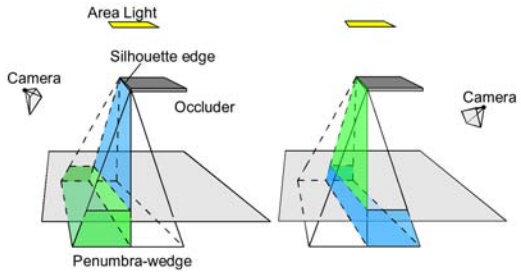


Figure 8: *On the left, the positive side penumbra-wedge and on the right, the negative side penumbra-wedge. In blue, the inner-half-wedge rasterized fragments and in green, the outer-half-wedge rasterized fragments.*

Still following Lengyel, we clip the edge to a local "near plane". This is necessary for silhouette edges that have a point that does not lie between the fragment and the light plane (figure 9). Then we clip in 2D the edge to the light borders. Note that "world to light space" transformation assures that the left and bottom borders are equal to -1 while right and top borders are equal to 1. Earlier implementation performs the clipping in 3 dimensions resulting in 36 fragment program instructions. By clipping edges in 2 dimensions and using a more advanced fragment program instructions set, Lengyel limits the clipping part to 23 instructions. In our implementation, we rather use the instruction *MOV_SSAT* to clip the x and y to the [-1..1] range and we compute their corresponding y and x coordinates using the linear equations of the edges. Thus, we obtain a 16 instructions clipping step. Moreover, many of our instructions (such as the MOV and MAD instructions) are "non-ALU intensive", in contrast with the CMP instruction frequently used in previous implementations.

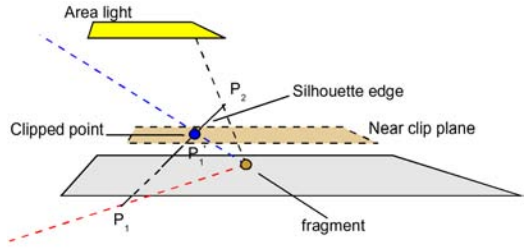


Figure 9: *The point P_1 is clipped to the local "near plane" in P_1' because it cannot be projected onto the area light plane.*

4.3. Light Occlusion geometry computation

At this stage, we need to compute for each radial part, the occluded light percentage and the bounding rectangle of the occluded light region. To do so, we first clip the edge in the radial part (a dynamic branching is used to avoid computation when the edge does not intersect the current radial part). For the edge clipped in the radial zone, we compute the occluded light percentage as described in the Assarsson and Akenine-Möller's approach [AMA02]. Finally, we add it (outer-half-wedges) or subtract it (inner-half-wedges) to the silhouette visibility buffer.

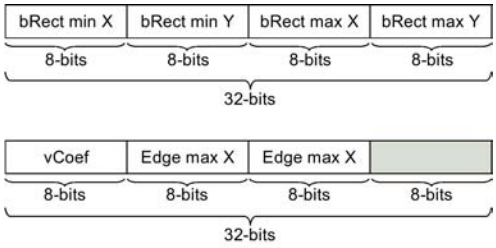


Figure 11: *Packed parameters representation. The bounding rectangle coordinates is stored in a 32-bits scalar. The visibility coefficient and the maximum in X and Y of the occluding edges coordinates are packed in another 32-bits scalar.*

The bounding rectangle computation requires more attention. First, we compute a rectangle that bounds the occluded light region defined by the outer-half-wedges. Then

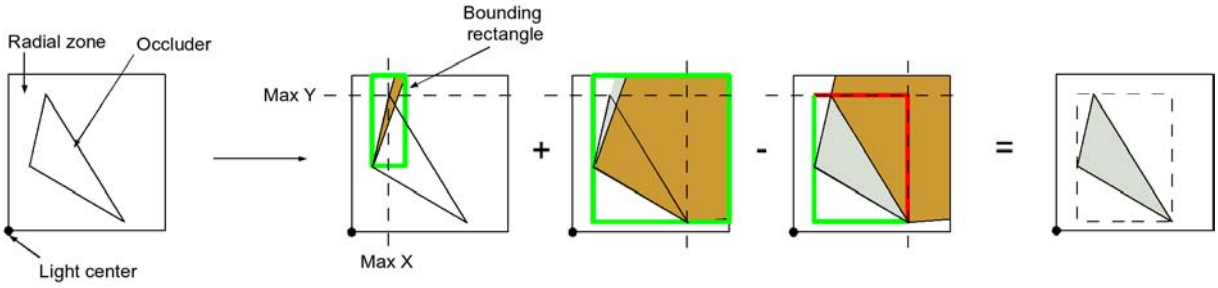


Figure 10: *Illustration of the light occlusion area and bounding rectangle computations. Note that for the inner-half-wedges bounding rectangle adjustment, we need the maximum in X and Y of the silhouette edges.*

we rasterize the inner-half-wedges that allow us to adjust the bounding rectangle. This adjustment requires the knowledge of the maximum in X and Y of the occluding edges coordinates as shown in figure 10.

We have hard constraints on the number of parameters we can store. Typically, a frame-buffer stores four values in the Red, Green, Blue and Alpha channels and our algorithm requires the storage of the following parameters for a single radial part:

- A coefficient representing the light occlusion percentage
- Top, bottom, left and right coordinates of the bounding rectangle
- The maximum in X and Y of the occluding edges coordinates

This results in seven parameters per radial part, i.e. $7*4=28$ parameters per fragment. With 8-bits of precision per parameter, we can pack top, bottom, left and right coordinates of the bounding rectangle in a single 32-bits scalar. The maximum in X and Y of the occluding edges coordinates and the occluded light percentage are stored in another 32-bits scalar. Hence, we use two 32-bits scalars by radial part (figure 11), i.e. eight 32-bits scalars per fragment. These parameters are stored using a Multi Render Target composed of two RGBA full precision float buffers.

The blending of the silhouette vBuffer with the final vBuffer consists, for one radial part, to unpack the values and perform the blending function presented in section 3.2. Dynamic branching is used to apply the blending function only if the occluded light percentage of the silhouette vBuffer fragment and the final vBuffer are not null.

In terms of memory our implementation uses two "pseudo buffers" called silhouette and final vBuffers. Each vBuffer requires 2 RGBA full float precision buffers, therefore with a 1024x768 resolution, the memory requirements for the visibility buffer computation is: $1024*768*4channels*4bytes*4buffers=48MB$ (when 512MB of available memory is now a standard on 3D-optimized graphics cards). In terms of computations, our algorithm is ALU intensive. Indeed, we think that it is the right direction for hardware algorithm implementation since GPU ALU horsepower increases more quickly than the bandwidth (as we can see with recent architectures).

5. Results

5.1. Visual results

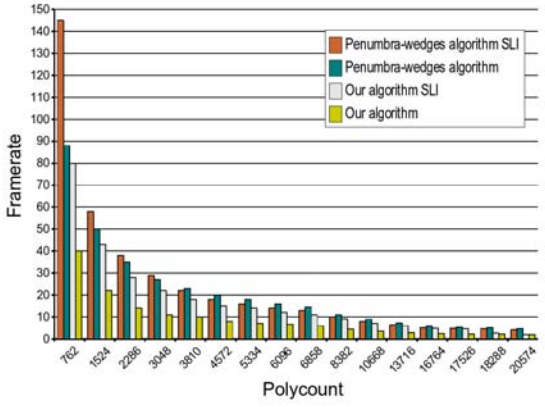
Our implementation is based on the OpenGL API. Our shaders are written with the pseudo assembly language provided by the ARB_fragment_program extension and both dynamic branchings and the instruction set introduced by the NV_fragment_program2 extension are used to avoid unnecessary computations and to increase performances. Our render contexts are based on Frame Buffer Object and Multi Render Target.

Figures 1(a) and 1(d) shows the "over-shadowed" error in the penumbra blending when several-silhouettes are overlapping. As we can see in figures 1(b) and 1(e) our algorithm significantly reduces the overlapping error and produces realistic images (figure 1(c)).

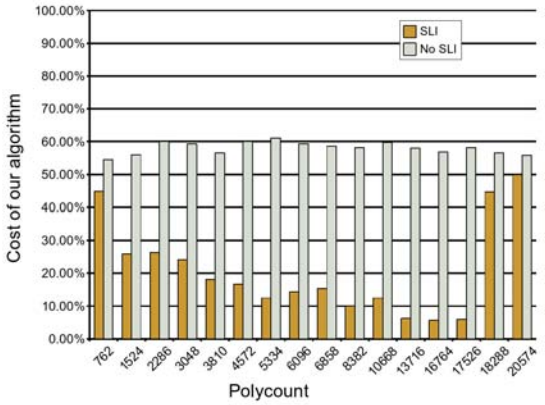
5.2. Performance

The performance was measured on a Linux workstation composed of two opteron dual-core 2Ghz with 4GB of memory. We use for the rendering two NVidia 7800-GTX configured either in the Alternate Frame Rendering SLI mode or with no SLI. The scene is composed of a collection of spheres as illustrated in figure 1. Each sphere is discretized by 762 triangles and the framebuffer resolution is 800x600. The impact of the geometry complexity of the scene is evaluated by increasing progressively the number of spheres.

As shown in figures 12(a) and 12(b), with 762 triangles the penumbra-wedges algorithm runs twice as fast as our algorithm. However, when we use more polygons the penumbra-wedges performance decreases faster than our approach because the penumbra-wedges are quickly CPU limited. We can see this limitation when we compare the



(a) *Framerate comparison*



(b) *Cost of our algorithm compare to the penumbra-wedges*

Figure 12: *Comparison between the performances of the standard penumbra-wedges and our algorithm*

penumbra-wedges performances with and without SLI enabled. Hence, with only 3048 polygons, the SLI gain is already negligible. Our algorithm is more GPU intensive than penumbra-wedges and thus it takes a strong advantage of a SLI configuration (the SLI gain is between 80% and 100%). However, when we increase the number of polygons over 17482 our algorithm becomes CPU limited and so the SLI gain is null.

Some observations can be done on these results. Following recent evolution of GPUs, the SLI configuration allows us to predict the performance of on the next GPU generation. On a SLI configuration, our algorithm is better balanced between CPU and GPU than penumbra-wedges. This will allow a better adaptation to the next GPU generation with a performance diminution of only 21% in average compared to standard penumbra-wedges, conterbalanced by the significant improvement of the soft-shadows realism (figures 1 and 2).

6. Conclusion

We have proposed a physically plausible soft shadows algorithm based on penumbra-wedges. Our technique significantly reduces the artifacts resulting of the non-overlapping silhouettes assumption. The result is a physically based soft shadow algorithm generating realistic soft shadows when penumbrae are overlapping, and this, with still an interactive framerate.

In terms of GPU evolution, functionalities allowing to construct both shadow-volumes and penumbra-wedges on the GPU would significantly enhance performances. Indeed, up to now, hardware does not permit the generation of primitives on GPU and hence, both are constructed on the CPU.

There are still some possible improvements to our algorithm. In some cases, the light occlusion area is concentrated in a part of the bounding rectangle. In this case our light occlusion position approximation can become insufficient for high quality penumbra blending and some visual artifacts can appear. A better light occlusion approximation requires a better light occlusion position approximation. The center of gravity of the light occlusion area could be used to weight the bounding rectangles intersection area. A center of gravity discretized into 256 positions per radial part would be sufficient to increase the quality of the penumbra blending. Note that in our implementation we pack in one of the two 32-bits scalars only three 8-bits values (figure 11). Hence, we could add the index of the center of gravity. In the blending function, the coordinates of the center of gravity could then be restored by indexing a two-components 1D look-up texture with the center of gravity index. Then, we can use this new information to refine the light occlusion position and so the overlapping area approximation. Finally, being shadow-volumes based, our technique is CPU limited. However, since it takes place on the top of these algorithms, it will benefit of their future evolutions and optimizations.

References

[AAM03] ASSARSSON U., AKENINE-MÖLLER T.: A geometry-based soft shadow volume algorithm using graphics hardware. *ACM Trans. Graph. 22*, 3 (2003), 511–520. 3, 4

[ADMAM03] ASSARSSON U., DOUGHERTY M., MOUNIER M., AKENINE-MÖLLER T.: An optimized soft shadow volume algorithm with real-time performance. In *HWWS '03: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware* (Aire-la-Ville, Switzerland, Switzerland, 2003), Eurographics Association, pp. 33–40. 2, 3, 4

[AHL*05] ATTY L., HOLZSCHUCH N., LAPIERRE M., HASENFRATZ J.-M., HANSEN C., SILLION F.: *Soft Shadow Maps: Efficient Sampling of Light Source Visibility*. Tech. Rep. RR-5750, INRIA, nov 2005. 2

[AMA02] AKENINE-MÖLLER T., ASSARSSON U.: Approximate soft shadows on arbitrary surfaces using penumbra wedges. In *EGRW '02: Proceedings of the 13th Eurographics workshop on Rendering* (Aire-la-Ville, Switzerland, Switzerland, 2002), Eurographics Association, pp. 297–306. 1, 2, 3, 4, 5

Figure 13: *Our algorithm applied on a scene composed of 6662 polygons. This scene is based on the models and materials of Half-life*^{2}*. Objects and textures copyright Valve Corporation: used with permission.*

[BS99] BILODEAU B., SONGY M.: Real time shadows. Creative Labs sponsored Game developer Conference, unpublished slides, May 1999. 2

[BS02] BRABEC S., SEIDEL H.-P.: Single Sample Soft Shadows Using Depth Maps. In *Proc. Graphics Interface* (May 2002), pp. 219–228. 2

[Car00] CARMACK J.: unpublished correspondance. Id-Software, 2000. 2, 4

[CD03] CHAN E., DURAND F.: Rendering fake soft shadows with smoothies. In *EGRW '03: Proceedings of the 14th Eurographics workshop on Rendering* (Aire-la-Ville, Switzerland, Switzerland, 2003), Eurographics Association, pp. 208–218. 2

[Cro77] CROW F. C.: Shadow algorithms for computer graphics. In *SIGGRAPH '77: Proceedings of the 4th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1977), ACM Press, pp. 242–248. 2

[EK02] EVERITT C., KILGARD M.: Practical and robust stenciled shadow volumes for hardware-accelerated rendering, 2002. 2

[FFBG01] FERNANDO R., FERNANDEZ S., BALA K., GREENBERG D. P.: Adaptive shadow maps. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 2001), ACM Press, pp. 387–390. 2

[GBP06] GUENNEBAUD G., BARTHE L., PAULIN M.: Real-time soft shadow mapping by backprojection. In *Eurographics Symposium on Rendering, Nicosia, Cyprus* (26-28 juin 2006), Eurographics, p. Ăă paraitre. 2

[HBS00] HEIDRICH W., BRABEC S., SEIDEL H.-P.: Soft shadow maps for linear lights. In *Proceedings of the Eurographics Workshop on Rendering Techniques 2000* (London, UK, 2000), Springer-Verlag, pp. 269–280. 2

[HLHS03] HASENFRATZ J.-M., LAPIERRE M., HOLZSCHUCH N., SILLION F.: A survey of real-time soft shadows algorithms. In *Eurographics* (2003), Eurographics Association. State-of-the-Art Report. 2

[Lai05] LAINE S.: Split-plane shadow volumes. In *Proceedings of Graphics Hardware* (2005), Eurographics Association, pp. 23–32. 2

[Len05] LENGYEL E.: Advanced stencil shadow and penumbra wedge rendering. Game developer Conference, unpublished slides, 2005. 3, 4

[RSC87] REEVES W. T., SALESIN D. H., COOK R. L.: Rendering antialiased shadows with depth maps. *SIGGRAPH Comput. Graph. 21*, 4 (1987), 283–291. 2

[SD02] STAMMINGER M., DRETTAKIS G.: Perspective shadow maps. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 2002), ACM Press, pp. 557–562. 2

[WH03] WYMAN C., HANSEN C.: Penumbra maps: approximate soft shadows in real-time. In *EGRW '03: Proceedings of the 14th Eurographics workshop on Rendering* (Aire-la-Ville, Switzerland, Switzerland, 2003), Eurographics Association, pp. 202–207. 2

[Wil78] WILLIAMS L.: Casting curved shadows on curved surfaces. In *SIGGRAPH '78: Proceedings of the 5th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1978), ACM Press, pp. 270–274. 2

[WPF90] WOO A., POULIN P., FOURNIER A.: A survey of shadow algorithms. *IEEE Comput. Graph. Appl. 10*, 6 (1990), 13–32. 2