Hard Shadow Anti-Aliasing for Spot Lights in a Game Engine

Márcio C. F. Macedo[*] Almir V. Teixeira Antônio L. Apolinário Jr. Karl A. Agüero

Federal University of Bahia, Department of Computer Science, Brazil

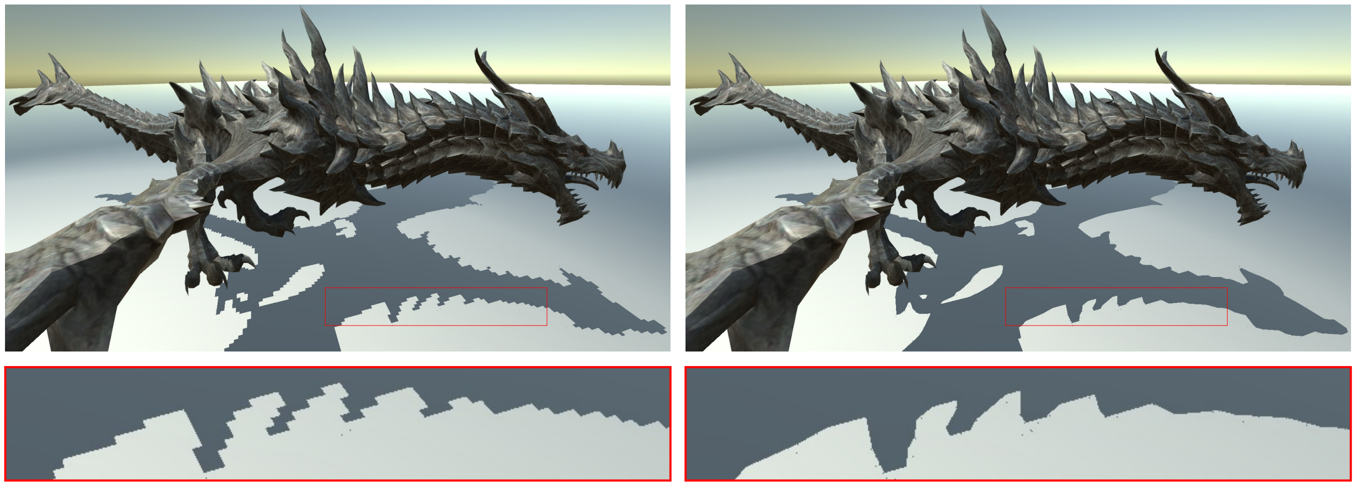

Figure 1: Unity generates real-time shadows with aliasing artifacts along the shadow boundary because of the shadow mapping technique (left). In this paper, we show how these shadows can be anti-aliased for spot lights using the concept of shadow revectorization (right).

Abstract

Real-time shadow rendering is desirable in several computer graphics applications, such as games. The most popular game engines typically provide support to this feature through the traditional shadow mapping algorithm. Unfortunately, shadow mapping is well known to generate aliasing artifacts along the shadow boundary, decreasing the realism of the rendered virtual scenes. In this paper, we aim to minimize these artifacts by implementing an anti-aliasing hard shadow technique in a game engine. Even with the limited source code access provided by some game engines, we demonstrate how to implement an improvement of the shadow mapping technique for shadow anti-aliasing. We validate our approach by implementing the revectorization-based shadow mapping technique for spot lights in a commercial and popular game engine, the Unity 3D. We show that, in this case, we are able to improve the shadow visual quality at the cost of only $\sim$1.2% of loss, in average, in the frame rate. Finally, the results indicate that the proposed implementation is ready to be integrated into Unity projects which use spot lights as a source of direct illumination.

Keywords: Real-time shadows, shadow mapping, revectorization.

1 Introduction

Shadow rendering plays an important role for games, since they augment the realism of virtual characters and scenarios, further improving the visual perception of the player with respect to the relative disposition of the virtual objects into the game.

Shadows, like many other computer graphics effects in games, must be rendered in real-time and with high quality to provide high user interactivity with the application. In this case, shadows must consume a few milliseconds (ideally, less than 10 milliseconds, as pointed in [41]), of the total frame time in a game, since many other effects are expected to run in real-time and increase the frame time as well. Meanwhile, the shadows must be visually convincing, enabling the rendering of realistic virtual scenarios.

Accurate shadow rendering is not much used for games because it requires the computation of the direct illumination part of the rendering equation [17, 19], which states that the visibility of each one of hundreds or thousands of rays directly emitted from an area light source to the virtual scene must be taken into account to compute photorealistic shadows. Even with the modern graphics hardware, the task of computing accurate shadows is still impractical for real-time, dynamic applications.

The most traditional way to compute shadows in real-time dates back to 1978, when the shadow mapping technique was proposed [38]. However, the aliasing artifacts introduced by such a technique trade off high frame rate by low rendering quality. Since then, numerous practical algorithms (e.g., [22, 20, 32, 24]) have been proposed to reduce those artifacts efficiently. Unfortunately, most of them were not incorporated into game engines, leading the developers to implement their own algorithms if they want more accurate shadows. For the game engines which use the shadow mapping and do not give sufficient source code access for one to change its algorithm, the only option left to the developer is to keep rendering scenes with aliased shadows.

In this paper, we present a new implementation of an anti-aliasing hard shadow technique for a game engine. Inspired by revectorization-based shadow mapping (RBSM) [24], which provides shadow anti-aliasing in real-time, we show how this technique can be implemented for spot lights in Unity 3D, one of the most popular game engines nowadays, and that has limited source code access and does not provide much support for real-time shadow anti-aliasing. We show that RBSM indeed offers real-time frame

[*]e-mail: marciocfmacedo@gmail.com

rates, while improving the visual quality of the rendered virtual scenes.

In this work, our main **contributions** are:

1. An evaluation of the developed shadow anti-aliasing technique with respect to visual quality and rendering time for different scenes loaded into Unity;

2. A description of how to implement variants of the shadow mapping algorithm in Unity;

3. A practical implementation of an anti-aliasing shadow technique for Unity;

4. A review of the support provided by the most popular game engines with respect to shadow rendering;

2 Related Work

To understand the proposed implementation of the shadow anti-aliasing technique for Unity, we first review the existing techniques for shadow rendering, with a focus on shadow anti-aliasing. Then, we present a review of how the existing game engines support shadow rendering. We restrict our review to techniques which handle hard shadows, as soft shadows (i.e., shadows with penumbras) are beyond the scope of this paper. A more detailed, comprehensive review of the existing shadow techniques can be found in [11, 40].

2.1 Shadow Rendering

Ray tracing [37] is one of the most traditional algorithms able to compute accurate shadows. In this technique, a view ray is traced from the camera viewpoint to the virtual scene through each pixel in the image plane. If the view ray hits a surface point in the scene, a new shadow ray is traced from the hit point to the light source. If the shadow ray hits an opaque object before reaching the light source, the surface point hit by the view ray is in shadow. Otherwise, the surface point is directly visible by the light source and is lit. Ray tracing is not only able to simulate shadows, but also many other effects (e.g., reflection, refraction), which makes it a powerful tool for computing global illumination effects. Unfortunately, even with the recent advances in literature [36, 15, 31], ray tracing does not provide real-time performance for dynamic scenes. Therefore, it is mostly used for applications that use offline rendering (e.g., movie production) or for the rendering of precomputed effects for static or dynamic scenes in games [28].

A faster alternative to ray tracing that has also found applicability in games is the shadow volume technique [9]. A shadow volume consists of a set of polygons formed by an extrusion of the vertices located at the silhouette of the objects presented in the scene in the direction of the rays emitted by the light source. A surface point is in shadow if it is located inside the shadow volume and is lit otherwise. Shadow volume is faster than ray tracing and generates accurate shadows. However, despite the recent advances towards adapting the use of shadow volume to compute real-time shadows [16, 27], shadow volume still demands high memory footprints and quite involved implementations, which makes it less practical for games nowadays.

A faster, less accurate alternative than both ray tracing and shadow volume is the shadow mapping technique [38]. For this technique, the virtual scene is rendered twice, one from the viewpoint of the light source, where the depth buffer as seen from the light source is stored as a shadow map, and other from the viewpoint of the camera, where the scene is projected into the shadow map and a depth comparison determines whether a surface point lies in shadow. Shadow mapping has several advantages which makes it largely used in games: easiness for implementation, real-time performance, scalability with respect to geometry complexity, hardware support, among others [11]. Unfortunately, all those advantages come at the price of aliasing artifacts which are generated due to the finite resolution of the shadow map.

Due to the popularity of shadow mapping, many techniques have been proposed to improve the visual quality of the shadows by minimizing the aliasing artifacts. Rather than generating the shadow map by using a uniform sampling of the depth buffer, warping strategies [35, 39, 25, 22] use non-uniform projections to improve the sampling resolution of the shadow map for the regions which are near the camera viewpoint, while lowering the sampling resolution for the other parts of the shadow map. This kind of strategy helps on the minimization of the aliasing artifacts at little additional cost.

Aliasing artifacts are severely magnified for large-scale virtual environments when using a single shadow map of insufficient resolution, since the shadow map will not be able to capture the details of the entire scene. Partitioning strategies [12, 43, 42, 20] have been proposed to generate multiple shadow maps and associate each one of them with a specific part of the 3D space. Partitioning techniques, combined with warping techniques, are able to greatly reduce aliasing artifacts. However, this strategy must be used with care, because the generation and management of multiple shadow maps incur the increasing of memory usage and processing time.

One alternative to shadow anti-aliasing relies on the filtering, or blurring, of either the aliased hard shadows [33] or the shadow map itself [10, 6, 32]. This kind of strategy is effective to simulate penumbra, requires low processing time and is able to minimize the aliasing artifacts. Unfortunately, details of the shadow boundary may be lost due to the blurring of the shadow and light leaking artifacts may be generated due to the shadow map pre-filtering.

To effectively remove the aliasing artifacts, some techniques [34, 29, 21] store additional geometric information (e.g., triangle position, normals) of the objects into the shadow map in order to change the shadow mapping visibility function and increase the quality of the shadow rendering. Unfortunately, these techniques increase memory consumption and processing time to generate high-quality anti-aliasing. Without relying on additional geometric information, revectorization-based shadow mapping [7, 24] detects the surface points located in the aliased shadow boundary, performs a traversal over the shadow boundary to estimate the distance of these points to the origin of the aliased shadow boundary, and finally uses a set of linear comparisons to determine whether a surface point must be put in shadow (i.e., revectorized). As we show in the rest of this paper, this shadow revectorization technique is able to recover an approximate accurate shadow boundary and achieve real-time frame rates, while keeping memory consumption as low as the one obtained with the shadow mapping.

2.2 Shadows in Game Engines

Since shadows play an important role in games, shadow rendering is an important feature for game engines.

Unity 3D Engine [4] uses shadow mapping to generate shadows for spot and point light sources. For directional light sources, cascaded shadow maps [12] take advantage of the partitioning strategy to reduce aliasing artifacts. Area light sources are supported for the Unity Pro version and require precomputed lighting conditions for the scene. Although not natively supported, shadow volumes are available as a free plugin for older versions of Unity in the Unity Asset Store[1]. While these techniques may work well for some scenarios, the main problem with Unity is that both free and professional versions of the engine give limited source code access for developers (the entire source code is available through an additional payment for Unity Pro users). Hence, it is difficult for one to implement its own improvements in the shadow algorithms using the available source code.

[1]https://www.assetstore.unity3d.com/en/content/1861

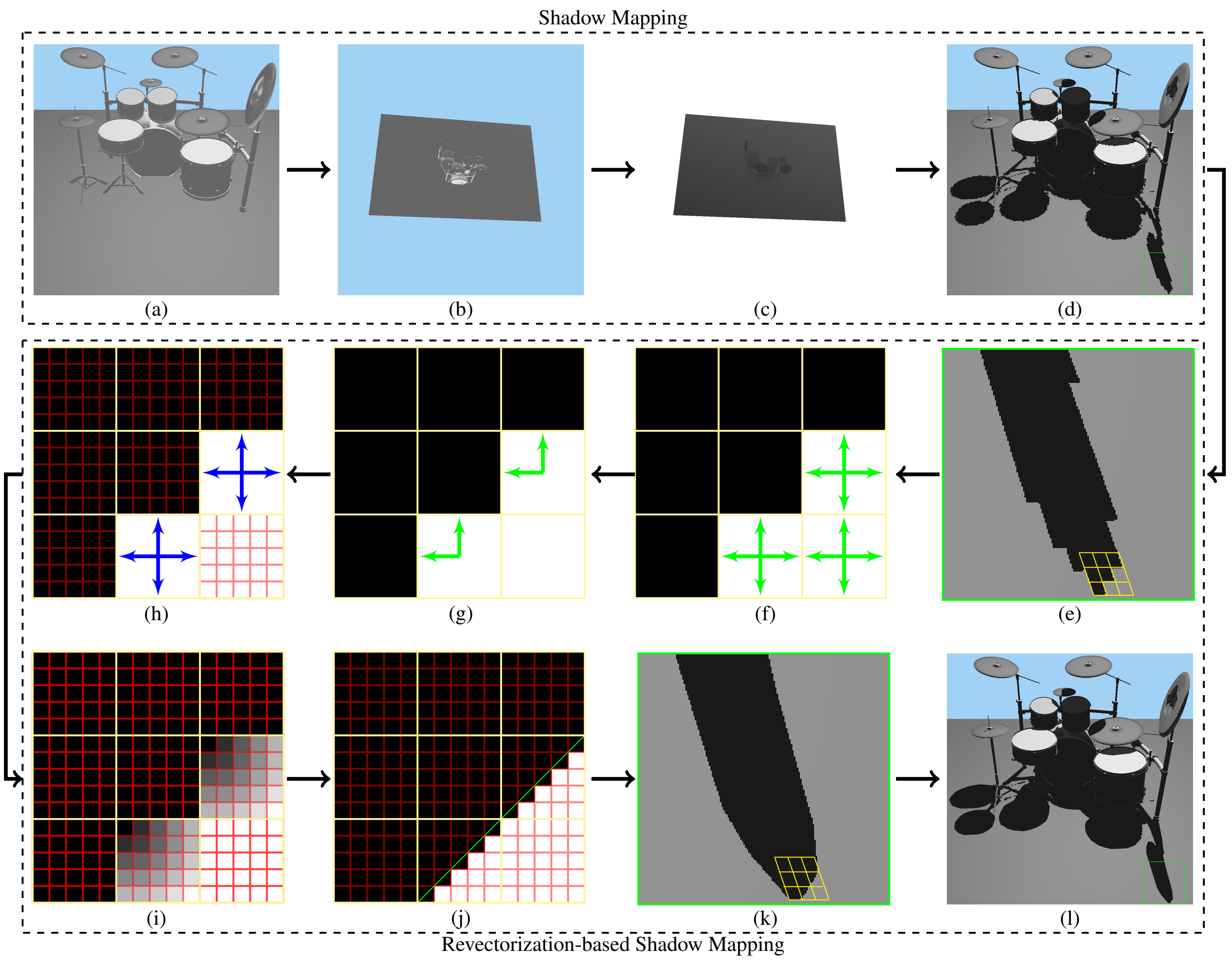

Figure 2: An overview of the revectorization-based shadow mapping technique. Shadow mapping renders the scene from the camera (a) and light source viewpoints (b) to generate a shadow map (c) and compute real-time shadows (d) with aliasing artifacts (e). Then, for each shadow map texel (yellow grid in (e)) lit by shadow mapping, revectorization-based shadow mapping evaluates the neighbourhood (f) to detect the discontinuity directions (green arrows in (g)) where the aliasing is located (g). A traversal over the shadow boundary (h) is performed to determine the size of the aliasing (blue arrows in (h)) and the normalized relative distance of each fragment in the camera view (represented by the red grid in (h, i, j)) to the aliased boundary (i). With such data, the algorithm is able to fit a revectorization line (green line in (j)), which allows the revectorization (j, k, l) of the shadow boundary.

Similarly to Unity, CryEngine [1] and Unreal Engine [5] support shadow mapping for point and spot light sources, cascaded shadow maps for directional lights and ray tracing for static scenes. Additionally, Unreal Engine supports an original technique called ray traced distance field soft shadows, which improves the visual quality of the cascaded shadow maps, further reducing the aliasing artifacts, but increasing processing time.

Although it is not formally defined as a game engine, NVIDIA GameWorks [2] is a sample development kit which provides support for several real-time shadow techniques into the NVIDIA ShadowWorks [3] and advanced ray tracing solutions via NVIDIA OptiX [30]. NVIDIA GameWorks is open-source, has already been integrated into the Unreal Engine[2] and is extensible to work with Unity Pro[3].

Unity is the most used game engine for mobile gaming [8], is the second game engine most recommended by industry experts [14] and is also one the most popular game engines for general game production, having more than 770 millions of users over the world[4]. However, even with this huge popularity, Unity still produces shadows with aliasing artifacts, which are mostly visible for the shadows rendered from the point or spot light sources. Therefore, we present an implementation of an anti-aliasing shadow technique which uses the available source code access in the popular Unity game engine to improve the visual quality of the hard shadows generated from spot lights.

[2]https://developer.nvidia.com/nvidia-gameworks-and-ue4

[3]https://developer.nvidia.com/content/gameworks-unity

[4]https://unity3d.com/pt/public-relations

3 Revectorization-based Shadow Mapping

In this section we describe the shadow anti-aliasing algorithm chosen to be implemented in Unity: Revectorization-Based Shadow Mapping (RBSM) [24]. As we show here, an implementation of RBSM is able to use the available resources of the game engine to provide shadow anti-aliasing.

Shadow aliasing (Figure 2-(d)) is caused by the mismatch of resolutions between the shadow map (Figure 2-(c)), that is rendered from the light source viewpoint (Figure 2-(b)), and the visible fragments in the scene, as seen from the camera viewpoint (Figure 2-(a)). This artifact is generated when different fragments in the scene are projected into the same shadow map texels, obtaining the same visibility conditions.

To minimize shadow aliasing, RBSM defines a new visibility function which works per fragment in the camera view. Much like an extension of the morphological anti-aliasing [18] for hard shadows [7], RBSM detects the fragments located in the aliased shadow boundary (Figure 2-(f, g)), traverses the shadow boundary to estimate the size of the aliasing (Figure 2-(h, i)), and then determines whether a fragment in the lit side of the aliasing must be put in shadow to revectorize the aliasing boundary (Figure 2-(j)), further reducing the visual aspect of the aliasing (Figure 2-(k, l)). An overview of this process is depicted in Figure 2 and is described in details below.

The first step of RBSM consists in the generation of the shadow map (Figure 2-(c)), as proposed by the shadow mapping technique. Then, each fragment in the camera view is projected into the shadow map, where a shadow test is conducted to determine whether the fragment is in shadow, according to the depth difference between the depth stored in the shadow map and the depth of the fragment projected [38].

To formalize such an algorithm, let us define the surface point $\mathbf{p}$ distant to the light source by $\mathbf{p}_z$. Now, let us assume the shadow map texel in the position x,y as $\mathbf{t}_{x,y}$ and $z(\mathbf{t}_{x,y})$ as a function which computes the depth of the blocker of the surface point $\mathbf{p}$ stored in the shadow map. Using these definitions, the shadow test may be defined by the binary visibility function $s(\mathbf{p}_z, z(\mathbf{t}_{x,y}))$ [38]

$$s(\mathbf{p}_z, z(\mathbf{t}_{x,y})) = \begin{cases} 0 & \text{if } \mathbf{p}_z > z(\mathbf{t}_{x,y}), \\ 1 & \text{otherwise,} \end{cases} \quad (1)$$

which indicates that the surface point $\mathbf{p}$ is in shadow if its distance $\mathbf{p}_z$ to the light source is higher than the depth $z(\mathbf{t}_{x,y})$ stored in the corresponding shadow map texel. In (1), the value 0 indicates that $\mathbf{p}$ is in the umbra and 1 otherwise.

Next, once the aliased hard shadows (Figure 2-(d, e)) have been generated as a result from the shadow test, RBSM detects the fragments which are located inside an aliased shadow boundary (Figure 2-(f, g)), because only these fragments need to be anti-aliased by the algorithm. To do so, the shadow test result computed for each fragment is compared against the ones obtained for its 4-connected neighbourhood in the shadow map (Figure 2-(f)). In practice, the neighbourhood shadow test evaluation is computed as $\mathbf{N}$

$$\mathbf{N} = \left[s(z(\mathbf{t}_{x-o_x,y})), s(z(\mathbf{t}_{x+o_x,y})), s(z(\mathbf{t}_{x,y+o_y})), s(z(\mathbf{t}_{x,y-o_y})) \right], \quad (2)$$

where o_x and o_y are the shadow map texel width and height and $\mathbf{p}_z$ was omitted from (2) because the value is the same for the four shadow test evaluations.

If the shadow test states that a fragment is lit and its visibility condition is different from at least one of its neighbours in the shadow map, the fragment is located in an aliased shadow boundary and must be revectorized. In this case, we need to compute the directions where the aliasing artifacts are located (Figure 2-(g)). To store such directions, the RBSM algorithm defines a discontinuity $\mathbf{d}$ as the absolute difference in the shadow test results between a fragment and its 4-connected neighbors in the shadow map [24]

$$\mathbf{d} = \left[|\mathbf{N}_1 - s|, |\mathbf{N}_2 - s|, |\mathbf{N}_3 - s|, |\mathbf{N}_4 - s| \right]. \quad (3)$$

Similar to morphological anti-aliasing, in the next step of RBSM, a traversal is performed over the shadow boundary (Figure 2-(h)) to estimate the size of the aliasing, and compute the normalized relative distance of each fragment to the origin of the aliased boundary where the fragment is located (Figure 2-(i)). In fact, for each fragment projected into the shadow map, the algorithm traverses the shadow boundary in all the four directions of the 2D space and stops the traversal whenever the traversal steps out of the lit part of the shadow boundary. Discontinuity directions are useful in this traversal because they only exist at the shadow boundary. So, whenever the traversal does not step into a place where there is discontinuity, the traversal is out of the shadow boundary.

The traversal over the shadow boundary allows not only the computation of the normalized relative distance and position of the fragment in the shadow boundary, but also the identification of the shape of the aliasing in which the fragment is located. These data are used in the last step of RBSM, which determines whether the fragment must be shadowed or remain lit (Figure 2-(j)). In this step, a linear comparison is performed using the normalized relative position of the fragment to detect where a fragment is located in the inner-side of the revectorized boundary (green line in Figure 2-(j)). As shown in Figure 2-(k, l), RBSM is able to reduce the aliasing artifacts generated by shadow mapping.

4 Shadows in Unity

As already stated in Section 2.2, Unity uses the shadow mapping as a basis to compute shadows generated from directional, point and spot light sources.

Directional light sources try to mimic the behaviour of distant light sources (e.g., the sun) by emitting parallel light rays in a single, dominant direction to illuminate large outdoor scenes. Since these light sources are defined by their directions, rather than their positions, a shadow map is rendered using orthographic projection to keep the light rays parallel to each other. For large-scale scenes, a single shadow map is mostly insufficient to compute accurate shadows. To solve this problem, Unity uses a variant of the shadow mapping, the cascaded shadow mapping [12], to partition the 3D space into several parts, and associate a shadow map for each one of them[5]. Cascaded shadow mapping enables high-quality shadow rendering for large scenarios, at the cost of more processing time. However, even when using this technique, aliasing artifacts still can be seen in the final rendering, due to the limited resolution of the multiple shadow maps rendered.

Differently from a directional light source, a point light source has a position and works like an infinitesimal sphere which emits light rays in all directions. From a lookup on the source code, we could see that Unity builds a cube map of shadow maps to compute the shadows generated by the occlusion of the light rays. To do so, the engine renders six shadow maps, one per face of the cube, and stores the shadow maps into the cube map. While being able to compute shadows for large scenes, shadow rendering with this type of light source is relatively expensive, since six shadow maps need to be generated by the algorithm.

Spot light sources have a position, a dominant direction and are able to simulate shadows for a limited extension of the scene. Hence, a single shadow map rendered using perspective projection is able to cover the part of the scene that is illuminated by the light source. While spot light sources are not well suited for outdoor illumination and produces aliasing artifacts along the shadow boundary, they are able to simulate shadows efficiently, since only one shadow map must be rendered per light source.

[5]https://docs.unity3d.com/Manual/DirLightShadows.html

Name	Type	Description
_ShadowMapTexture	internal	Shadow map
_ShadowMapTexture_TexelSize	float4	Shadow map texel size / Shadow map resolution
shadowCoord	float4	Coordinates to access the shadow map for a given fragment
_LightShadowData	float4	Shadow intensity (or shadow strength, in Unity)

Table 1: List of variables available in Unity for shadow mapping with spot lights.

Name	Input	Output	Description
UNITY_DECLARE_SHADOWMAP	internal _ShadowMapTexture	void	Declares the shadow map as _ShadowMapTexture
SAMPLE_DEPTH_TEXTURE_PROJ	internal _ShadowMapTexture and float2 shadowCoord	float	Returns the depth stored in the shadow map for a given texel
UNITY_SAMPLE_SHADOW_PROJ	internal _ShadowMapTexture and float4 shadowCoord	float	Returns the shadow test for a given fragment

Table 2: List of functions available in Unity for shadow mapping with spot lights.

In Unity, the source code which contains where the shadow maps are generated is private. Therefore, the techniques (e.g., [34, 10, 6, 29, 21, 32]) which require a modification of the structure of the shadow map to provide shadow anti-aliasing cannot be implemented in the game engine without source code access. Also, directional and point light sources use an additional structure (a list of shadow maps for directional light sources, and a cube map for point light sources) to store the multiple shadow maps rendered. However, Unity does not give access to the individual shadow maps generated with those light sources. Instead, the engine encapsulates these structures into a variable named **_ShadowMapTexture**, which simply returns whether a given fragment in the camera viewpoint is located in shadow, or the depth of the blocker of a given fragment. This imposes another restriction on the implementation of shadow anti-aliasing, since most of the techniques proposed in the literature require the access per shadow map to provide shadow anti-aliasing. Fortunately, spot lights produce a single shadow map, that corresponds exactly to the **_ShadowMapTexture** variable. That is why we have chosen to focus the implementation of a shadow anti-aliasing technique for spot lights.

A list of the variables and functions provided by Unity for shadow mapping with spot lights is shown in Tables 1 and 2. An example of their usage to compute the aliased hard shadows is presented in Listing 1. The source code for shadow rendering using spot lights is available in the **UnitySampleShadowmap** method in the shader file **UnityShadowLibrary.cginc** of Unity. Basically, the **UnitySampleShadowmap** method receives as input the coordinates of the fragment rendered from the camera viewpoint, but projected on the shadow map (line 1), uses that coordinates to access the shadow map texture **_ShadowMapTexture** and compute the shadow test using the **UNITY_SAMPLE_SHADOW_PROJ** method (lines 4-5). Then, depending on whether the fragment is in shadow, the method returns 1.0, indicating full visibility of the fragment (line 7) or **_LightShadowData.r**, an intensity value accounting that the fragment is shadowed (line 8).

```
1  fixed UnitySampleShadowmap (float4 shadowCoord)
2  {
3
4      half shadow = UNITY_SAMPLE_SHADOW_PROJ
5      (_ShadowMapTexture, shadowCoord);
6
7      if(shadow > 0.0) return 1.0;
8      else return _LightShadowData.r;
9
10 }
```

Listing 1: Shadow mapping for spot lights in Unity.

```
1  fixed UnitySampleShadowmap (float4 shadowCoord)
2  {
3
4      half shadow = UNITY_SAMPLE_SHADOW_PROJ
5      (_ShadowMapTexture, shadowCoord);
6      if(shadow == 0) return _LightShadowData.r;
7
8      float4 d = computeDiscontinuity(shadowCoord);
9      if(d > 0.0) return 1.0;
10
11     float4 rd = computeRD(shadowCoord);
12     float2 nrd = normalizeRD(rd);
13     shadow = computeRBSMVisibility(nrd, d);
14     if(shadow > 0.0) return 1.0;
15     else return _LightShadowData.r;
16
17 }
```

Listing 2: RBSM for spot lights in Unity.

4.1 Shadow Revectorization in Unity

In Listing 2, we show the main pipeline of the RBSM implementation. First, we evaluate the shadow test for a given fragment to detect whether the fragment is shadowed or lit (line 4-5). As shown in Figure 2-(f), RBSM operates only on the lit fragments located in the aliased shadow boundary (line 6). Hence, for each lit fragment, we follow the pipeline shown in Figures 2 -(g, h, i, j): We evaluate the neighbour shadow map texels (Figure 2-(f)) to compute the discontinuity directions where the aliased boundary is located (Figure 2-(g) and line 8). If an aliased boundary has been detected (line 9), we traverse the shadow boundary to compute the size of the aliasing, as well as the relative distance of the fragment to the origin of the local aliasing (Figure 2-(h) and line 11). Next, we normalize such relative distance (Figure 2-(i) and line 12) and define a visibility function (line 13) to determine whether a fragment must be revectorized (Figure 2-(j) and lines 14-15).

In lines 4 and 5 of Listing 2, we use the **UNITY_SAMPLE_SHADOW_PROJ** function to compute the shadow test (1). In the function **computeDiscontinuity** of line 8, we compute equations (2) and (3) to estimate the discontinuity directions at the aliased boundary. It is noteworthy that the offset value o in (2) is equivalent to the value stored in the first two coordinates of **_ShadowMapTexture_TexelSize**, variable which returns exactly the shadow map texel size in terms of width and height.

```
 1  float computeRD (float4 coord, float2 dir) {
 2
 3      int maxSize = 16;
 4      float distance = 1.0;
 5      float umbra = 0.0;
 6      float4 d;
 7      float4 tempCoord = coord;
 8      float2 step = _ShadowMapTexture_TexelSize.xy;
 9      tempCoord.xy += dir * step;
10
11      for(int it = 0; it < maxSize; it++) {
12          half shadow = UNITY_SAMPLE_SHADOW_PROJ
13          (_ShadowMapTexture, tempCoord);
14          if(shadow == 0.0) {
15              umbra = 1.0;
16              break;
17          } else {
18              d = computeDiscontinuity(tempCoord);
19              if(d == 0.0) break;
20          }
21          distance++;
22          tempCoord.xy += dir * step;
23      }
24
25      return lerp(-distance, distance, umbra);
26
27  }
28
29  float lerp(float x, float y, float a) {
30      return x * (1.0 - a) + y * a;
31  }
```

Listing 3: An algorithm to compute the relative distance of a fragment to the shadow boundary.

A more in-depth view of the function **computeRD** (line 11, Listing 2) is shown in Listing 3. RBSM computes the relative distance of the fragment with respect to the shadow boundary for the four directions of the 2D space. For each direction, we use the first two coordinates of **_ShadowMapTexture_TexelSize** to compute the step to access the neighbour shadow map texel and perform the traversal over the shadow boundary (line 8, Listing 3). Also, we use the variable **maxSize** to define the maximum aliasing size that we will consider for revectorization (line 3, Listing 3). As suggested in [24], we define **maxSize** equals to 16 to keep the frame rate more constant. Then, for each shadow map texel being accessed, we check whether we step out of the lit side of the aliased boundary. This condition is fulfilled whenever we step into an umbra texel (lines 14-16, Listing 3) or step into a lit texel that does not have any discontinuity directions (i.e., the texel is not neighbour of an umbra texel) (lines 17-19, Listing 3). Otherwise, we increment the variable **distance** (line 21, Listing 3), which represents the size of the aliasing. In line 25 of Listing 3, we use a linear interpolation to make the distance value signed. We orientate a distance value to be positive towards the origin of the aliasing (the corner of the aliasing, which is located in the top-left side of the aliasing shown in Figure 2-(i)) and negative otherwise. This orientation of the distance value helps in the normalization of the relative distance, which is performed by the function **normalizeRD**.

The function **normalizeRD** shown in line 12 of Listing 2 takes as input the relative distance of the fragment to the ends of the shadow boundary in the four directions of the 2D space and normalizes it to a reference coordinate system which lies in the unit interval [0, 1]. As shown in Figure 2-(i), the origin of this coordinate system is located in the corner of the aliasing. Finally, the function **computeRBSMVisibility** (line 13, Listing 2) takes as input the normalized

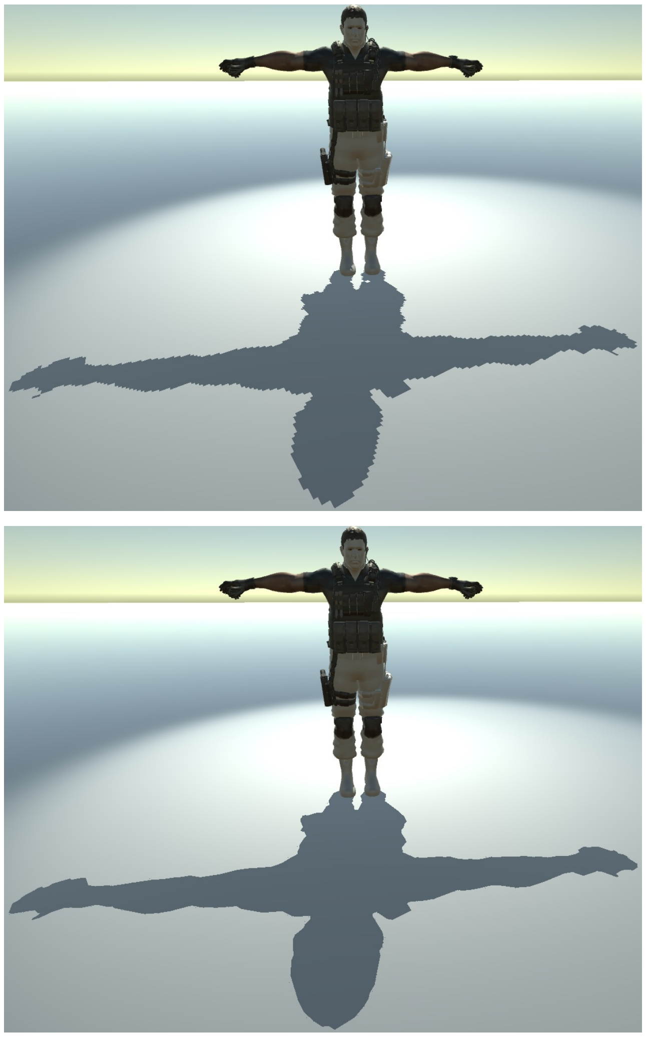
Figure 3: A visual comparison between shadow mapping (top) and RBSM (bottom) for the Chris model using a low-resolution (256^2) shadow map.

relative distance and the discontinuity directions of a given fragment and, depending on the shadow aliasing configuration, a linear comparison is used to define the new revectorized shadow boundary [24]. None of these two functions require any of the variables and functions listed in Tables 1 and 2. Therefore, they can be implemented in Unity.

5 Results and Discussion

5.1 Experimental Setup

In this section, we compare both shadow mapping and RBSM in terms of visual quality and rendering time. Similarly to Unity, we classify the shadow map resolution we have used for each figure as low, medium, high or very high. Also, most of the figures shown in this paper do not contain scenes with high-frequency textures because they could potentially mask shadow rendering irregularities. That is why we have chosen to use white textures in the ground planes and some of the other objects.

Tests were performed on a personal computer equipped with an NVIDIA GeForce GTX Titan X graphics card and an Intel Core^{TM} i7-3770K CPU (3.50 GHz), 8GB RAM. We have tested the experiments in the Unity Pro version 5.6.0.f3. The proposed shader file

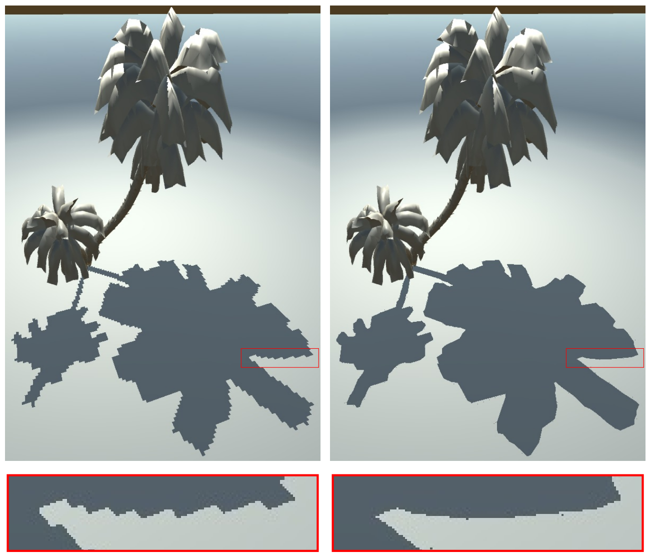
Figure 4: A visual comparison between shadow mapping (left) and RBSM (right) for the Coconut model using a medium-resolution (512^2) shadow map.

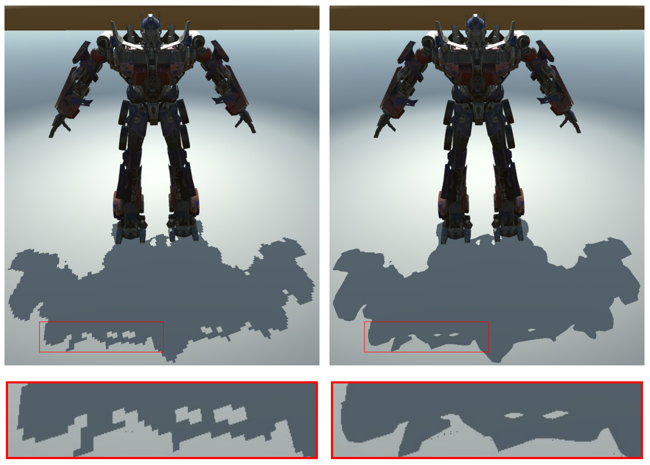
Figure 5: A visual comparison between shadow mapping (left) and RBSM (right) for the Robot model using a high-resolution (1024^2) shadow map.

was implemented using the Cg language [13] and is available in an open-source repository [23]. In the accompanying video, we show additional results, including tests with more scenarios.

For the visual quality evaluation, we wanted to compare the hard shadows generated with shadow mapping and RBSM with the ones obtained from more accurate techniques, such as shadow volumes and ray tracing. Unfortunately, Unity does not give much support for accurate hard shadow rendering techniques. We could not compare the shadows generated from area light sources with the ones obtained with shadow mapping and RBSM because area light sources generate shadows with penumbra, while we have focused on the anti-aliasing provided for hard shadows. Moreover, area light sources do not run in real-time for dynamic scenes. In terms of the available plugins for Unity, the only existing plugin that implements the shadow volume technique for Unity does not work in the current version of the game engine, as stated in the official repository

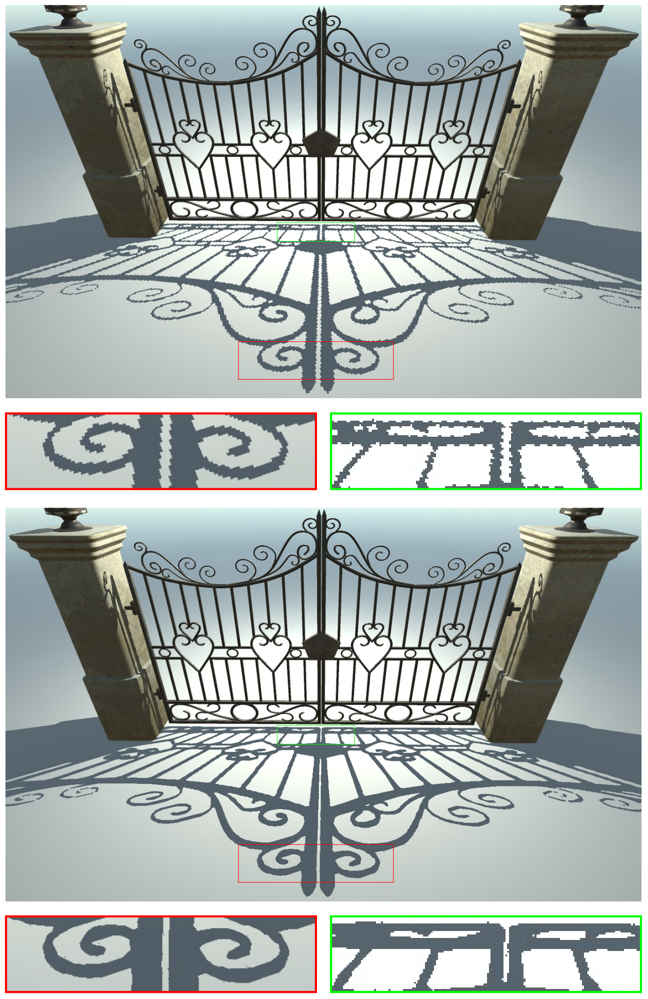
Figure 6: A visual comparison between shadow mapping (top) and RBSM (bottom) for the Gate model using a very high-resolution (2048^2) shadow map.

of the plugin[6]. The same statement is held for the available CPU implementation of ray tracing for Unity[7], which also seems to be pretty much slower than both shadow mapping and RBSM. Given these facts, in Sections 5.2 and 5.3, we evaluate the effectiveness of the RBSM implementation on the anti-aliasing of the shadows generated in Unity. For an isolated comparison between RBSM and shadow volume, we direct the reader to see previous work [24].

5.2 Visual Quality

In figures 1, 3, 4, 5 and 6, we show a visual comparison between shadow mapping and RBSM for different models and shadow map resolutions. We can see that, regardless of the shadow map resolution used, shadow mapping generates aliasing artifacts along the shadow boundary because the shadow map resolution is finite and does not match the pixel resolution of the camera view. Then, by traversing the shadow aliasing boundary, RBSM is able to minimize the artifacts and improve the shadow visual quality.

The quality of the shadow revectorization is dependent on the shadow map resolution used and the quality of the aliased boundary. RBSM works well for the scenarios shown in figures 1, 3 and 4,

[6]https://www.assetstore.unity3d.com/en/content/1861
[7]https://github.com/BenjaminSchaaf/Unity-Raytracer

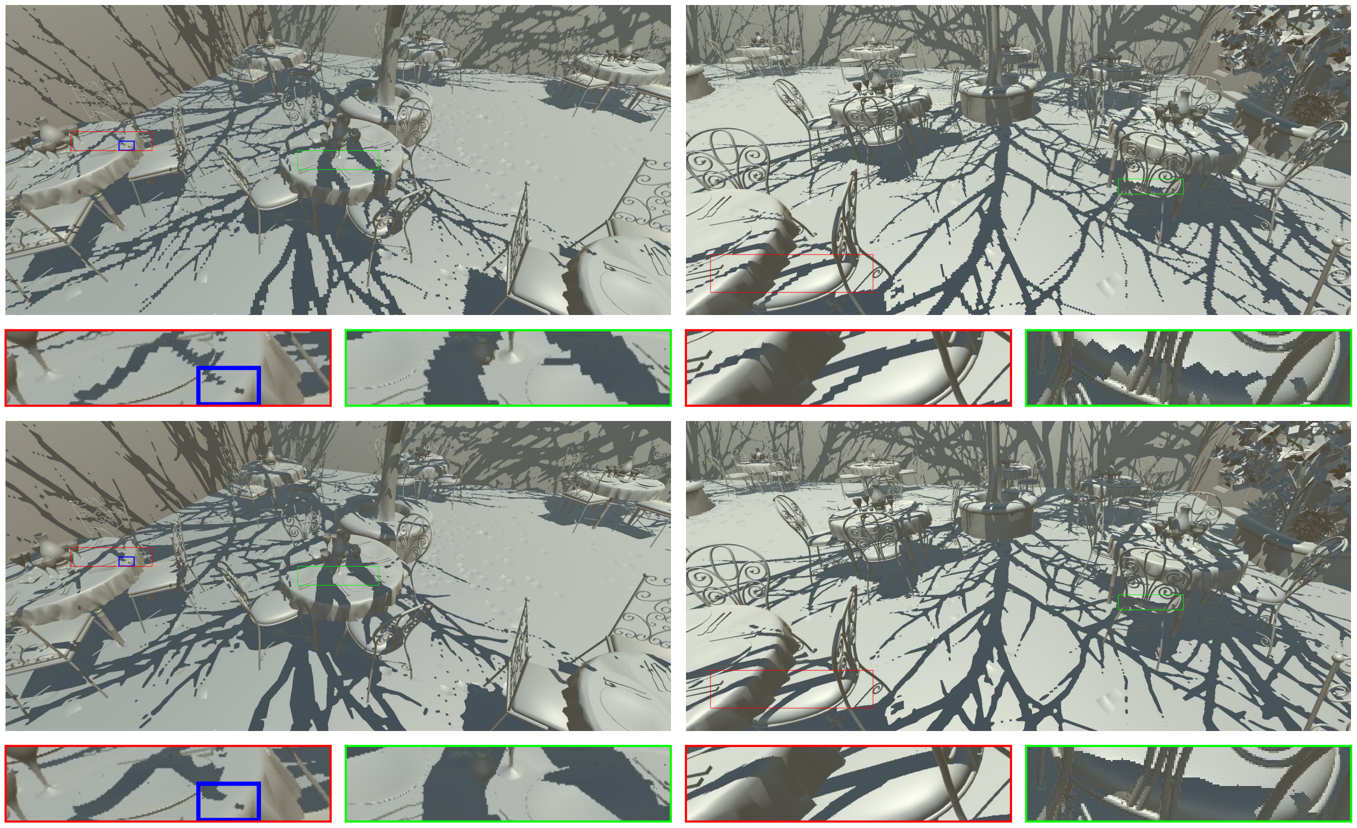

Figure 7: A visual comparison between shadow mapping (top) and RBSM (bottom) for the complex San Miguel model using a very high-resolution (2048^2) shadow map.

despite the use of a low and a medium shadow map resolutions. In these cases, the objects and their shadows are relatively simple, since there is not much intersection between shadow boundaries and a few fine details to be captured by the shadow map. Hence, the aliasing is the most noticeable artifact that prevents an accurate shadow rendering. So, by the use of RBSM, we can minimize these artifacts and improve the shadow visual quality efficiently. On the other hand, the shadows shown in the red closeup of Figure 5 and green closeup of Figure 6 have a more complex shape, with several intersections in the shadow boundary and fine details captured by the shadow map, respectively. In these cases, RBSM minimizes the aliasing artifacts at the cost of causing a shadow overestimation, suppressing some details of the original shadow boundary.

In Figure 7, we show a more complex scenario that contains many structures with fine details (trees), non-planar shadow receivers (chairs on the floor) and multiple objects that overlap each other. In this case, shadow mapping not only generates the aliasing artifacts, but also is not able to capture the fine details of the light blocker objects, causing the appearance of holes along the shadow boundary (see the blue rectangles in Figure 7). Since Unity fixes the maximum shadow map resolution that can be used in the engine, this kind of problem cannot be solved when computing real-time shadows. On the other side, RBSM helps on the minimization of the shadow aliasing artifacts (see the closeups of Figure 7), but is not able to solve the problem of the shadow holes caused by the insufficient shadow map resolution used.

In Figure 8, we show that our shadow anti-aliasing implementation works not only for indoor environments, but also for large outdoor environments with high-detailed structures, such as vegetations, streets and buildings. The closeups in Figure 8 show that the

		Shadow Map Resolution			
Scene	Method	Low (256^2)	Medium (512^2)	High (1024^2)	Very High (2048^2)
Fig. 1	SM	5.06 ms	5.09 ms	5.11 ms	5.12 ms
	RBSM	5.14 ms (1.58%)	5.16 ms (1.37%)	5.20 ms (1.76%)	5.22 ms (1.95%)
Fig. 3	SM	4.96 ms	4.99 ms	5.01 ms	5.03 ms
	RBSM	5.03 ms (1.41%)	5.07 ms (1.60%)	5.10 ms (1.79%)	5.12 ms (1.78%)
Fig. 4	SM	5.14 ms	5.16 ms	5.17 ms	5.20 ms
	RBSM	5.14 ms (0.00%)	5.19 ms (0.58%)	5.20 ms (0.58%)	5.21 ms (0.19%)
Fig. 5	SM	5.08 ms	5.11 ms	5.12 ms	5.14 ms
	RBSM	5.14 ms (1.18%)	5.16 ms (0.97%)	5.18 ms (1.17%)	5.22 ms (1.55%)
Fig. 6	SM	5.07 ms	5.08 ms	5.14 ms	5.16 ms
	RBSM	5.12 ms (0.98%)	5.14 ms (1.18%)	5.15 ms (0.19%)	5.18 ms (0.38%)
Fig. 7	SM	5.02 ms	5.03 ms	5.04 ms	5.04 ms
	RBSM	5.08 ms (1.19%)	5.10 ms (1.39%)	5.11 ms (1.38%)	5.13 ms (1.78%)
Fig. 8	SM	7.43 ms	7.46 ms	7.48 ms	7.69 ms
	RBSM	7.46 ms (0.40%)	7.51 ms (0.67%)	7.57 ms (1.20%)	7.74 ms (0.65%)

Table 3: Performance (including percentual of overhead) of shadow mapping (SM) and RBSM for varying shadow map resolution. Scenes were rendered at an output $1080p$ resolution.

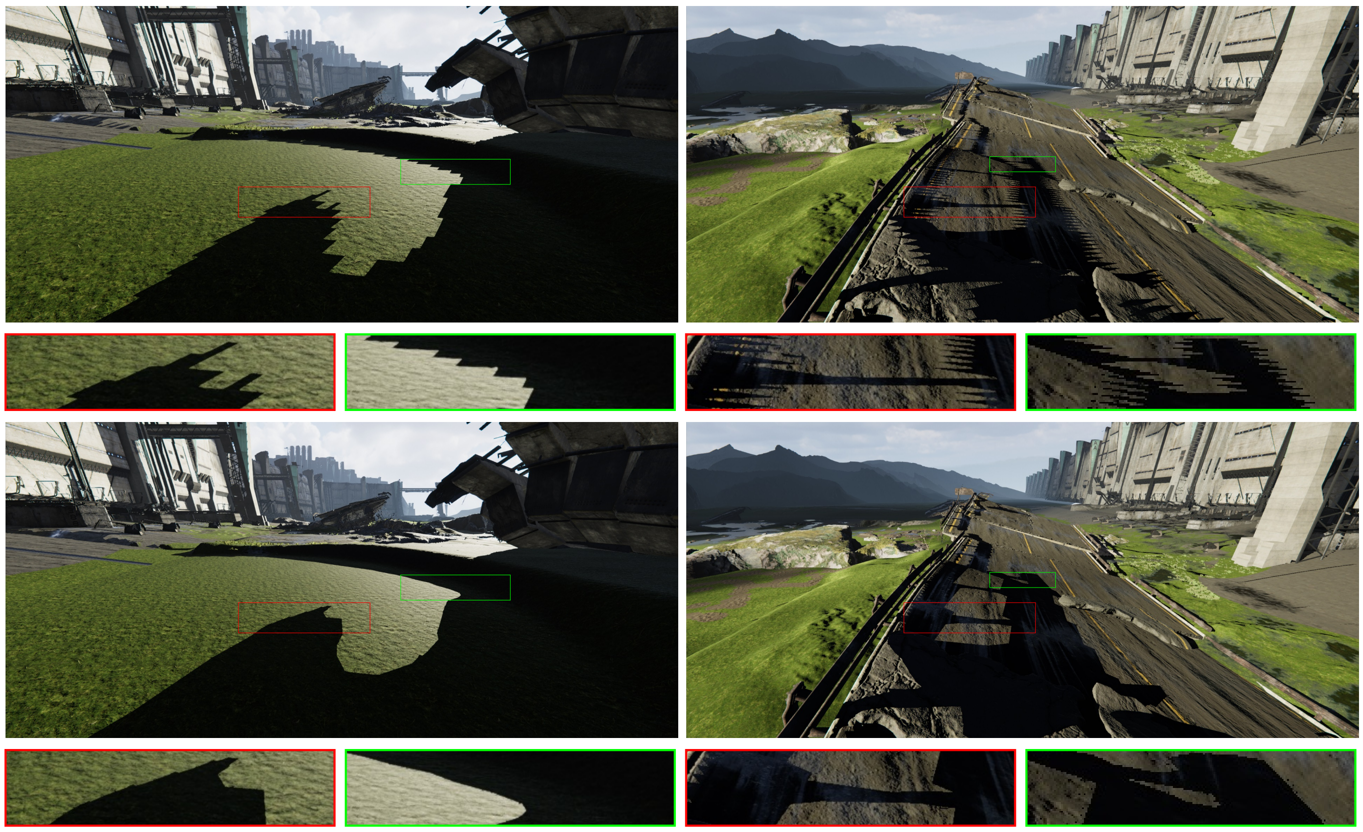

Figure 8: A visual comparison between shadow mapping (top) and RBSM (bottom) for the exterior environment of the Unity's Adam model using a very high-resolution (2048^2) shadow map.

use of spot lights for outdoor illumination clearly generates shadow aliasing artifacts, which can be effectively suppressed by RBSM.

5.3 Rendering Time

In Tables 3 and 4, we measured how much time the engine needed to generate the figures shown in this paper for varying shadow map and output resolutions. Similarly to the shadow mapping technique, RBSM becomes slower as long as both shadow map (Table 3) and output resolutions (Table 4) are increased. Fortunately, the percentage of overhead added by RBSM keeps small regardless of the shadow map or output resolution used, proving that the RBSM technique is scalable with respect to these changes in resolution.

Both Table 3 and Table 4 show that the RBSM technique is about 0.01 and 0.5 milliseconds (or between 0.1% and 3.9%) slower than shadow mapping for the hardware setup described in Section 5.1. Meanwhile, we can see from all the figures shown in this paper that RBSM greatly improves the visual quality of the shadow. Hence, we believe that RBSM proved to be an efficient hard shadow anti-aliasing technique, because RBSM can leverage the shadow visual quality, while adding a pretty small overhead of ∼1.2%, in average, in the total frame time.

6 Conclusion and Future Work

In this paper, we have presented an implementation of RBSM, an anti-aliasing, real-time hard shadow technique, for the Unity game engine. Since Unity uses the shadow mapping for shadow rendering, aliasing artifacts are generated along the shadow boundary. We have shown that, by the use of the proposed implementation of RBSM, we are able to minimize these aliasing artifacts, generating high-quality shadows at minimal additional cost.

		Output Resolution		
Scene	**Method**	720*p*	1080*p*	2160*p*
Fig. 1	SM	5.11 ms	5.12 ms	5.14 ms
	RBSM	5.16 ms (0.97%)	5.22 ms (1.95%)	5.34 ms (3.89%)
Fig. 3	SM	5.01 ms	5.03 ms	5.04 ms
	RBSM	5.11 ms (1.99%)	5.12 ms (1.78%)	5.12 ms (1.58%)
Fig. 4	SM	5.19 ms	5.20 ms	5.20 ms
	RBSM	5.19 ms (0.00%)	5.21 ms (0.19%)	5.22 ms (0.38%)
Fig. 5	SM	5.02 ms	5.14 ms	5.21 ms
	RBSM	5.18 ms (3.18%)	5.22 ms (1.55%)	5.24 ms (0.57%)
Fig. 6	SM	5.05 ms	5.16 ms	5.17 ms
	RBSM	5.10 ms (0.99%)	5.18 ms (0.38%)	5.25 ms (1.54%)
Fig. 7	SM	4.99 ms	5.04 ms	5.07 ms
	RBSM	5.03 ms (0.80%)	5.13 ms (1.78%)	5.14 ms (1.38%)
Fig. 8	SM	7.49 ms	7.69 ms	15.08 ms
	RBSM	7.63 ms (1.86%)	7.74 ms (0.65%)	15.55 ms (3.11%)

Table 4: Rendering performance (including percentual of overhead) for shadow mapping (SM) and RBSM for varying output resolution. Scenes were rendered at a very high shadow map resolution.

Due to the limited source code access provided by Unity, we were not able to implement RBSM for other types of light source, such as point and directional light sources. Moreover, we could not implement RBSM for soft shadows that simulate the penumbra effect. The implementation of RBSM using the entire Unity source code, or the implementation in other game engines, may enable one to adapt the technique to support hard and soft shadow rendering for any light source. Finally, since NVIDIA GameWorks may be used with Unity Pro, the field of shadows in game engines still needs a gentle introduction on how to integrate NVIDIA ShadowWorks with Unity Pro and an evaluation of the shadow techniques of NVIDIA ShadowWorks in the context of the Unity game engine.

Acknowledgments

We are grateful to the NVIDIA Corporation, who donated the hardware used in this work through the GPU Education Center program. Also, we are thankful to Unity for delivering the Unity Pro through the Unity Education Grant License. This research is financially supported by the scholarship program of Coordenação de Aperfeiçoamento de Pessoal do Nível Superior (CAPES).

Dragon (user deadcode3), Drum (user drumssultan), Coconut (user azlyirnizam), Robot (user ysup12) and Gate (user animated-heaven) models are courtesy of Free3D and its respective users. San Miguel model is courtesy of Morgan McGuire [26] and Guillermo M. Leal Llaguno. Adam model is courtesy of Unity.

References

[1] CryEngine. https://www.cryengine.com/. Accessed: 2017-06-27.
[2] NVIDIA GameWorks. https://developer.nvidia.com/gameworks. Accessed: 2017-06-27.
[3] NVIDIA ShadowWorks. https://developer.nvidia.com/shadowworks. Accessed: 2017-06-27.
[4] Unity 3D. https://unity3d.com. Accessed: 2017-06-27.
[5] Unreal Engine. https://www.unrealengine.com/. Accessed: 2017-06-27.
[6] T. Annen, T. Mertens, H.-P. Seidel, E. Flerackers, and J. Kautz. Exponential Shadow Maps. In *Proceedings of GI*, pages 155–161, Toronto, Ont., Canada, 2008. Canadian Information Processing Society.
[7] V. Bondarev. Shadow Map Silhouette Revectorization. In *Proceedings of the ACM I3D*, pages 162–162, New York, NY, USA, 2014. ACM.
[8] J. Chi and T. Sun. Development drivers: Third-party engines and mobile gaming. *McKinsey & Company*, February 2015.
[9] F. C. Crow. Shadow Algorithms for Computer Graphics. In *Proceedings of the ACM SIGGRAPH*, pages 242–248, New York, NY, USA, 1977. ACM.
[10] W. Donnelly and A. Lauritzen. Variance Shadow Maps. In *Proceedings of the ACM I3D*, pages 161–165, New York, NY, USA, 2006. ACM.
[11] E. Eisemann, M. Schwarz, U. Assarsson, and M. Wimmer. *Real-Time Shadows*. A.K. Peters, Natick, MA, USA, 2011.
[12] W. Engel. Cascaded shadow maps. In *ShaderX 5.0 Advanced Rendering Techniques*, pages 197–206. Charles River Media, Hingham (Mass.), 2006.
[13] R. Fernando and M. J. Kilgard. *The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.
[14] M. French, J. Batchelor, A. Boucher, and C. Chapple. The tech list. *Develop 100*, 2014.
[15] V. Fuetterling, C. Lojewski, F.-J. Pfreundt, and A. Ebert. Efficient Ray Tracing Kernels for Modern CPU Architectures. *Journal of Computer Graphics Techniques (JCGT)*, 4(5):90–111, December 2015.
[16] J. Gerhards, F. Mora, L. Aveneau, and G. Djamchid. Partitioned Shadow Volumes. *Computer Graphics Forum*, 34(2):549–559, 2015.
[17] D. S. Immel, M. F. Cohen, and D. P. Greenberg. A Radiosity Method for Non-diffuse Environments. In *Proceedings of the ACM SIGGRAPH*, pages 133–142, New York, NY, USA, 1986. ACM.
[18] J. Jimenez, B. Masia, J. I. Echevarria, F. Navarro, and D. Gutierrez. Practical Morphological Anti-Aliasing. In W. Engel, editor, *GPU Pro 2*, pages 95–113. AK Peters Ltd., 2011.
[19] J. T. Kajiya. The Rendering Equation. In *Proceedings of the ACM SIGGRAPH*, pages 143–150, New York, NY, USA, 1986. ACM.
[20] A. Lauritzen, M. Salvi, and A. Lefohn. Sample Distribution Shadow Maps. In *Proceedings of the ACM I3D*, pages 97–102, New York, NY, USA, 2011. ACM.
[21] P. Lecocq, J.-E. Marvie, G. Sourimant, and P. Gautron. Sub-pixel Shadow Mapping. In *Proceedings of the ACM I3D*, pages 103–110, New York, NY, USA, 2014. ACM.
[22] D. B. Lloyd, N. K. Govindaraju, C. Quammen, S. E. Molnar, and D. Manocha. Logarithmic Perspective Shadow Maps. *ACM Trans. Graph.*, 27(4):106:1–106:32, Nov. 2008.
[23] M. Macedo. RBSM in Unity. https://github.com/MarcioCerqueira/GlobalIllumination/tree/master/RBSMinUnity. Accessed: 2017-09-05.
[24] M. Macedo and A. Apolinario. Revectorization-Based Shadow Mapping. In *Proceedings of the GI*, pages 75–83, Toronto, Ont., Canada, 2016. Canadian Information Processing Society.
[25] T. Martin and T.-S. Tan. Anti-aliasing and Continuity with Trapezoidal Shadow Maps. In *Proceedings of the EGSR*, pages 153–160, Aire-la-Ville, Switzerland, 2004. Eurographics Association.
[26] M. McGuire. Computer Graphics Archive. https://casual-effects.com/data. Accessed: 2017-09-04.
[27] F. Mora, J. Gerhards, L. Aveneau, and D. Ghazanfarpour. Deep Partitioned Shadow Volumes using Stackless and Hybrid Traversals. In *Proceedings of the EGSR*, pages 73–83, Goslar Germany, Germany, 2016. The Eurographics Association.
[28] G. Morgan and A. Pranckevicius. Practical Techniques for Ray Tracing in Games. In *GDC Vault*, 2014.
[29] M. Pan, R. Wang, W. Chen, K. Zhou, and H. Bao. Fast, Sub-pixel Antialiased Shadow Maps. *Computer Graphics Forum*, 28(7):1927–1934, 2009.
[30] S. G. Parker, J. Bigler, A. Dietrich, H. Friedrich, J. Hoberock, D. Luebke, D. McAllister, M. McGuire, K. Morley, A. Robison, and M. Stich. OptiX: A General Purpose Ray Tracing Engine. *ACM Trans. Graph.*, 29(4):66:1–66:13, July 2010.
[31] A. Perard-Gayot, J. Kalojanov, and P. Slusallek. GPU Ray Tracing using Irregular Grids. *Computer Graphics Forum*, 36(2), 2017.
[32] C. Peters and R. Klein. Moment Shadow Mapping. In *Proceedings of the ACM I3D*, pages 7–14, New York, NY, USA, 2015. ACM.
[33] W. T. Reeves, D. H. Salesin, and R. L. Cook. Rendering Antialiased Shadows with Depth Maps. In *Proceedings of the ACM SIGGRAPH*, pages 283–291, New York, NY, USA, 1987. ACM.
[34] P. Sen, M. Cammarano, and P. Hanrahan. Shadow Silhouette Maps. *ACM Trans. Graph.*, 22(3):521–526, July 2003.
[35] M. Stamminger and G. Drettakis. Perspective Shadow Maps. *ACM Trans. Graph.*, 21(3):557–562, July 2002.
[36] I. Wald, S. Woop, C. Benthin, G. S. Johnson, and M. Ernst. Embree: A Kernel Framework for Efficient CPU Ray Tracing. *ACM Trans. Graph.*, 33(4):143:1–143:8, July 2014.
[37] T. Whitted. An Improved Illumination Model for Shaded Display. *Commun. ACM*, 23(6):343–349, June 1980.
[38] L. Williams. Casting Curved Shadows on Curved Surfaces. In *Proceedings of the ACM SIGGRAPH*, pages 270–274, New York, NY, USA, 1978. ACM.
[39] M. Wimmer, D. Scherzer, and W. Purgathofer. Light Space Perspective Shadow Maps. In *Proceedings of the EGSR*, pages 143–151, Aire-la-Ville, Switzerland, June 2004. Eurographics Association.
[40] A. Woo and P. Poulin. *Shadow Algorithms Data Miner*. CRC Press, Natick, MA, USA, 2012.
[41] B. Yang, Z. Dong, J. Feng, H.-P. Seidel, and J. Kautz. Variance Soft Shadow Mapping. *Computer Graphics Forum*, 29(7):2127–2134, 2010.
[42] F. Zhang, H. Sun, and O. Nyman. Parallel-Split Shadow Maps on Programmable GPUs. In H. Nguyen, editor, *GPU Gems 3*, pages 203–237. Addison-Wesley, 2008.
[43] F. Zhang, H. Sun, L. Xu, and L. K. Lun. Parallel-split Shadow Maps for Large-scale Virtual Environments. In *Proceedings of the ACM VRCIA*, pages 311–318, New York, NY, USA, 2006. ACM.