Adaptive Depth Bias for Shadow Maps

Hang Dou[1] Yajie Yan[1] Ethan Kerzner[2] Zeng Dai[3] Chris Wyman[4]

[1]Washington University in St. Louis [2]SCI Institute [3]University of Iowa [4]NVIDIA

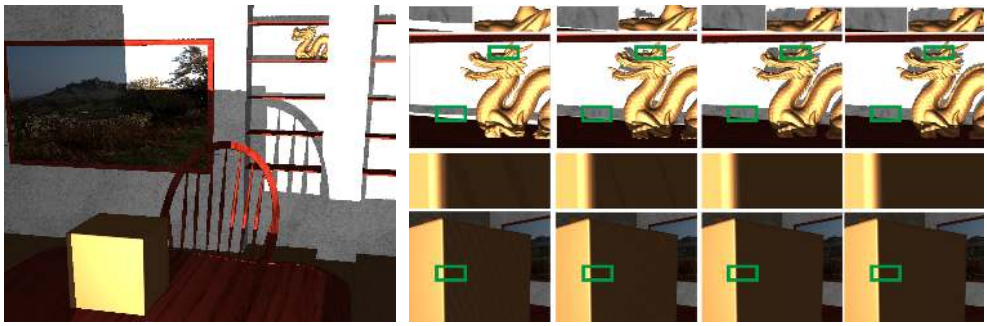

Figure 1. The overview (left) of an interior scene illuminated by traditional shadow mapping and details (right). Both constant depth bias (column 1) and slope-scale depth bias (column 2) suffer from shadow acne and shadow detachment. Our method (column 3) has no acne and preserves more shadow details. Dual-depth-layers depth bias (column 4) shown for reference.

Abstract

Unexpected shadow acne and shadow detachment due to limited storage are pervasive under traditional shadow mapping. In this paper, we present a method to eliminate false self-shadowing through adaptive depth bias. By estimating the potential shadow caster for each fragment, we compute the minimal depth bias needed to avoid false self-shadowing. Our method is simple to implement and compatible with other extensions to the shadow mapping algorithm, such as cascaded shadow map and adaptive shadow map. Moreover, our method works for both 2D shadow maps and 3D binary shadow volumes.

1. Introduction

Shadow mapping is still the most widely used technique to generate surface shadows in interactive applications, such as games and preview of off-line rendering. However, directly using traditional shadow mapping can easily generate images with intense shadow acne or false self-shadowing because of under sampled geometry in light space. There are two common ways to remove shadow acne. One is to avoid false

 ISSN 2331-7418

self-shadowing by storing extra geometry information and the other is to add a depth bias before shadow testing. Recording extra geometry information usually introduces extra storage, construction and look up cost. Adding a constant depth bias does not work for every part of the geometry. Our method computes an adaptive depth bias for every pixel based on local geometry features.

Woo et al. [1992] records the average depth of the closest and second closest surface in the shadow map. However, false self-shadowing remains an problem at silhouettes. Weiskopf and Ertl [Weiskopf and Ertl 2003] reduce the artifacts at silhouettes by adding a constant bound to the midpoint depth bias. Dual depth layers based methods produce much less artifacts or acne than existing bias based methods but they introduce extra storage, construction and look up cost, which makes those methods less scalable. Our method results in comparable quality to dual layer depth map but better performance.

Williams [1978] used a constant depth bias before shadow testing for each pixel. A constant bias can remove incorrect self-shadowing but also evokes false unshadowing or shadow detachment. Later, non-constant depth bias were proposed to relieve shadow detachment by restricting the bias amount based on surface slope scale. King [2004] introduces a slope-scale depth bias which is computed based on the fragment's depth slope relative to light view. Gautron et al. [2013] weight the depth bias by the angle between the fragment's surface normal and the incident light ray. These two methods do not provide minimal bias for every fragment and thus still produce false unshadowing to some extent. Also, the bias is still constant for fragments with the same surface slope. Compared to our approach, they have more false positive and true negative errors.

We propose a novel approach to remove false self-shadowing by generating adaptive depth bias based on local geometry features for each fragment. A minimal depth bias for each fragment is computed by estimating the fragment's potential occluder. We also introduce an adaptive epsilon to make sure that the false shadowed fragment is shifted just above its estimated occluder. Our adaptive depth bias is easy to compute, comes with small overhead and works for both 2D shadow maps and 3D binary shadow volumes in fully dynamic scenes. To test our method, we apply the algorithm to traditional shadow map [Williams 1978], paraboloid shadow map [Brabec et al. 2002] and voxelized shadow volumes (VSVs) [Wyman 2011].

2. Adaptive Depth Bias

As a start point, we address the problem with traditional shadow map. Extension of our method to other shadow mapping algorithms is straight-forward and will be discussed in detail in the following sections.

Shadow acne in traditional shadow map is mainly due to shading samples from camera view mismatching samples in shadow map. The amount of depth bias needed

to eliminate false self-shadowing differs among fragments. As shown in Figure 2 (left), suppose F_1 and F_2 lie on the same planar surface. F_1 is shadowed by F_2 which is sampled in the shadow map. We refer to F_2 as the occluder of F_1 or occluder of the corresponding texel. The minimum depth bias or optimal depth bias needed to

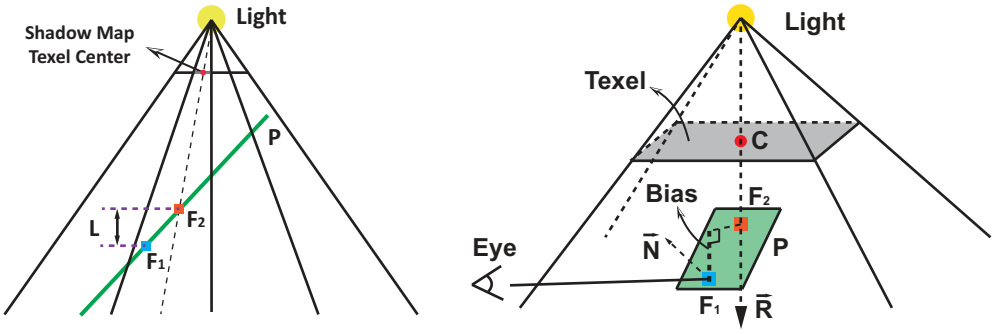

Figure 2. (left) 2D illustration of false self-shadowing. F_1 and F_2 are two fragments on planar surface P. L represents the optimal bias for F_1. (right) 3D Illustration of optimal depth bias computation for a traditional shadow map.

remove false shadowing for F_1 is L. Besides optimal depth bias, a small epsilon value is needed to move F_1 just above its occluder F_2. The formula to compute adaptive depth bias is simply:

$$adaptiveDepthBias = optimalDepthBias + adaptiveEpsilon. \quad (1)$$

2.1. Optimal Depth Bias

To compute the optimal depth bias for a given fragment, we first locate its potential occluder. As depicted in Figure 2 (right), under the assumption of planar occluders, given a fragment F_1, its potential occluder F_2 can be computed as the intersection of $\vec{R}$ and P, where $\vec{R}$ is the ray traced from the light source through the texel center C, and P is the tangent plane defined by F_1 and normal N. The optimal bias is then the depth difference between F_1 and F_2. In practice, rather than explicitly computing *optimalDepthBias*, we use the depth of the intersection point of the light ray through the shadow map texel center with the tangent plane of the fragment to do visibility checking. Assuming fragments lying on planar surfaces provides a good approximation to the common situation in many real scenes.

2.2. Adaptive Epsilon

To shift a fragment just above its potential occluder, a proper epsilon value is needed besides the optimal depth bias. However, a constant epsilon does not work since the depth value is often compressed non-linearly. Therefore, instead of using a constant epsilon directly, we transform the constant epsilon adaptively based on the depth compression function.

The adaptive epsilon is computed based on a constant epsilon and the depth compression function:

$$\varepsilon = f'(x)\Delta x, \tag{2}$$

$$\Delta x = sceneScale \times K, \tag{3}$$

where ε denotes the adaptive epsilon we use for biasing, x represents the unnormalized depth value of the shaded fragment, Δx represents the unnormalized epsilon value measured in world space coordinate, $f(x)$ represents an arbitrary depth compression function which maps depth values from near and far clip plane to $[0,1]$, *sceneScale* denotes the length of scene's bounding box diagonal, and K is a constant. In practice, we use standard OpenGL depth compression function.

$$a = -\frac{lf+ln}{lf-ln}, \tag{4}$$

$$b = -\frac{2\times lf\times ln}{lf-ln}, \tag{5}$$

$$f(x) = \frac{-a\times x+b}{2\times x}+\frac{1}{2}, \tag{6}$$

where ln and lf represents the light near and far plane distance, x denotes the real depth value ($x \in [-lf,-ln]$) and $f(x)$ denotes the compressed depth value ($f(x) \in [0,1]$). From equation 6 , we obtain:

$$f(x)' = \frac{-b}{2\times x^2}, \tag{7}$$

$$x = \frac{b}{2f(x)+a-1}, \tag{8}$$

Combining equation 2, 7 and 8, we obtain the formula for adaptive epsilon:

$$\varepsilon = \frac{(2f(x)+a-1)^2}{-2\times b}\times \Delta x. \tag{9}$$

Replacing a and b with equation 4 and 5, we present adaptive epsilon as follow:

$$\varepsilon = \frac{(lf-depth\times(lf-ln))^2}{lf\times ln\times(lf-ln)}\times sceneScale\times K, \tag{10}$$

where $depth$ represents normalized depth value for the given fragment. We set $K = 0.0001$ in all our experiments.

3. Implementation

We now describe and implement our method in GLSL for a traditional shadow map, paraboloid shadow map, and voxelized shadow volume.

3.1. Adaptive Depth Bias for Traditional Shadow Map

For each fragment in traditional shadow map, we locate its potential occluder, compute the adaptive epsilon and shift the fragment just above its potential occluder before visibility checking. The pseudocode in algorithm 1 and concrete GLSL implementation in listing 1 show the details. The input parameters are the fragment normal in light space, fragment position in world space, and shadow map resolution.

Algorithm 1 Pseudocode for shadow map sampling.

```
SM ← generateShadowMap(LightPosition)
for each fragment F with normal N do
    P ← defineTangentPlane(F, N)
    C ← locateTexelCenter(SM, F)
    R ← defineLightRay(LightPosition, C)
    F′ ← planeRayIntersect(R, P)
    ε ← calcAdaptiveEpsilon(F′)
    isLit ← checkVisibility(SM, F′, ε)
    outputColor ← isLit × shadeFrag(F)
end for
```

```
uniform mat4 lightView,lightProj; // Light view and projection matrix
uniform float epsilon; // Constant epsilon based on scene scale
// Left bound and near plane of light view frustum
uniform float lightLeft,lightNear;
uniform sampler2D shadowMap;
bool VisibilityCheckInTSM(vec3 lsFragNormal, vec3 wsFragPos, float smRes) {
    // defineTangentPlane:  Light space fragment normal
    vec3 n = normalize(lsFragNormal.xyz);

    // LocateTexelCenter:  Obtain the light space shadow map grid center
    vec2 lsGridCenter = GetLightSpaceCenter(wsFragPos,lightView,
                                            lightProj,lightLeft);

    // defineLightRay:  Light ray direction in light space coordinate
    vec3 lsGridLineDir = normalize(vec3(lsGridCenter, -lightNear));

    // FindPotentialOccluderByPlaneRayIntersection:  Compute the
    // depth of potential occluder for given fragment
    float actualDepth = GetOccluderDepth(wsFragPos,lightView,
                                         lightProj,lsGridLineDir);
    // Look up depth in shadow map
    float SMDepth = GetFragDepth(wsFragPos,lightView,lightProj,shadowMap);
    // Compute adaptive depth bias
    float adaptiveEpsilon = GetAdaptiveE(lightProj,SMDepth,epsilon);
    // checkVisibility
    bool isLit = CheckVisibility(SMDepth,actualDepth,adaptiveEpsilon);
    return isLit;
}
```

Listing 1. Bias for sampling a traditional shadow map

3.2. Adaptive Depth Bias for Paraboloid Shadow Map

For hemispherical and omni-directional light sources, the whole field of view of the light needs to be mapped; cube and paraboloid are two common mappings. When using cube map for hemispherical or omni-directional illumination, adaptive depth bias described in section 3.1 can be used directly. When a paraboloid mapping is chosen, only small change is needed. The pseudocode of visibility checking for adaptive depth bias with paraboloid shadow map is the same as the pseudocode for traditional shadow map. The only difference lies in how we locate the potential occluder.

As shown in Figure 3 (left), F_1 is incorrectly shadowed by F_2 in a paraboloid shadow map. For F_1, L is the optimal depth bias. Similar to the case of traditional shadow map, for each fragment, we first locate its corresponding texel in the paraboloid shadow map. Second, we obtain the potential occluder inside that texel by intersecting the fragment's local tangent plane with the shadow sample ray of that texel. Finally, we move the fragment just above the shadow caster with an adaptive epsilon.

As shown in Figure 3 (right), given a fragment F_1, we project it into the paraboloid shadow map and locate the corresponding texel center $C(x_c, y_c, 0)$. $H(x_h, y_h, z_h)$ represents the hit point of shadow sample ray on paraboloid. $\vec{N}$ represents the surface

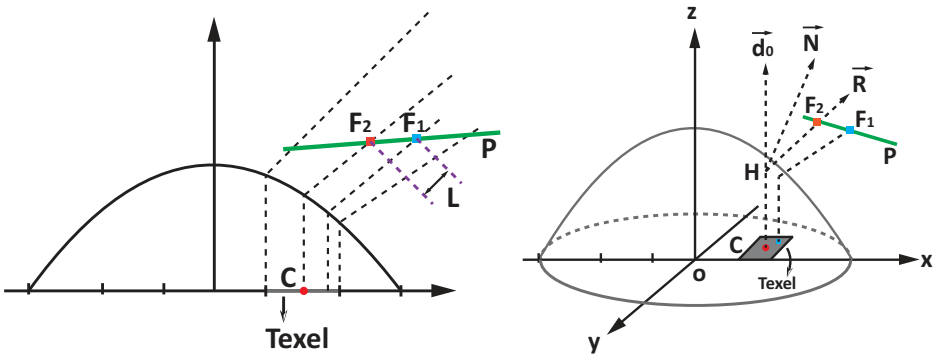

Figure 3. (left) 2D illustration of incorrect self-shadowing for paraboloid shadow map. Fragment F_1 and F_2 lie on the planar surface P. C denotes the corresponding shadow map texel center. F_1 is incorrectly shadowed by F_2. L represents the optimal depth bias. (right) 3D illustration of adaptive bias computation for paraboloid shadow map.

normal at H. We can obtain the sample shadow ray $\overrightarrow{R}$ as follow:

$$\overrightarrow{R} = 2 \times (\overrightarrow{d_0} \cdot \overrightarrow{N}) \times \overrightarrow{d_0} - \overrightarrow{d_0}$$
$$\overrightarrow{d_0} = [0,0,1]^T$$
$$\overrightarrow{N} = [\frac{x_c}{z_h}, \frac{y_c}{z_h}, \frac{1}{z_h}]^T$$
$$z_h = \frac{1}{2} - \frac{1}{2} \times (x_c^2 + y_c^2).$$

The potential occluder of F_1 inside texel T is F_2, the intersection of $\overrightarrow{R}$ and F_1's local tangent plane P. To remove the false self-shadowing, we move F_1 just above F_2 with an adaptive epsilon described in section 2.2. Below shows the shader code. The input parameters are fragment position, light position and shadow map texture resolution. The same as traditional shadow map, light rays across texel centers in paraboloid shadow map can also be precomputed.

```
// Near and far plane light view frustum
uniform float lightNear,lightFar;
uniform sampler2D shadowMap;
bool VisibilityCheckInPSM(vec3 wsFragPos,vec3 wsLightPos,float smRes) {
    /**** LocateTexelCenter ****/
    // Obtain the world space shadow map grid center for given fragment
    vec3  wsGridCenter = GetWorldSpaceCenter(wsFragPos,wsLightPos,smRes);

    /**** defineLightRay ****/
     // Light ray direction in world space coordinate
    vec3 wsGridLineDir = GetLightRayDir(nlsGridCenterN);

    /**** FindPotentialOccluderByPlaneRayIntersection ****/
    // Compute the depth of potential occluder for given fragment
    float actualDepth = GetOccluderDepth(wsGridLineDir,wsFragPos,
                                         wsLightPos);

    // Look up depth in shadow map
    float SMDepth = GetFragDepth(wsFragPos,wsLightPos,shadowMap);
    // Compute adaptive depth bias
    float adaptiveEpsilon = GetAdaptiveE(wsFragPos,wsLightPos,
                                lightNear,lightFar,SMDepth,epsilon);

    /**** checkVisibility ****/
    bool isLit = CheckVisibility(SMDepth,actualDepth,adaptiveEpsilon);
    return isLit;
}
```

Listing 2. Bias for Paraboloid Shadow Map

3.3. Adaptive Depth Bias for Voxelized Shadow Volume

Voxelized Shadow Volumes [Wyman 2011] enable computation of both volume shadows in participating media and surface shadows. Similar to traditional shadow algo-

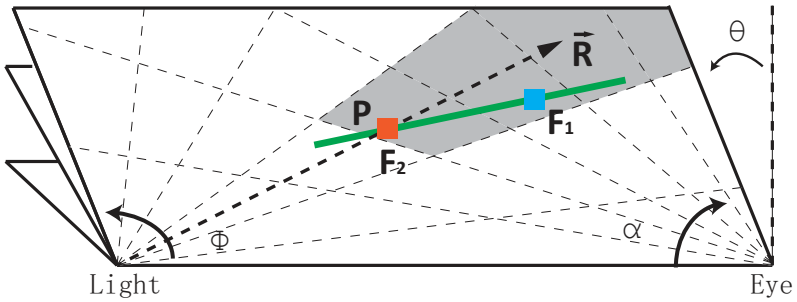

Figure 4. Incorrect shadowing in the epipolar coordinate system for VSVs. Gray voxels mean occluded. The voxel containing fragment F_1 is incorrectly shadowed by the voxel containing fragment F_2.

rithms, computing surface shadows with VSVs suffers from visual artifacts caused by discretization of geometry. Unlike traditional shadow maps VSVs represent shadows with binary information. A voxel is occluded only if it contains a occluding fragment or it lies in the shadow of another occluding object.

VSVs represent shadows with a binary epipolar voxel-grid. As shown in Figure 4, epipolar space is defined relative to an epipole, a line connecting the eye and light. Three angles define an epipolar point. Thus a point in epipolar space is: $(\alpha,\phi,\theta) \in ([0,\pi],[0,2\pi),[0,\pi])$. The angle θ defines an epipolar plane relative to some vector (by convention, the camera's up vector). Points on an epipolar plane are defined relative to the eye and light point. The angle α determines an axis parallel to view rays and the angle ϕ defines an axis parallel to light rays. As shown in Figure 4, given a fragment F_1, we project the given fragment into the corresponding voxel $V_1(\alpha,\phi,\theta)$ and obtain V_1's voxel center $C_{V_1}(\alpha_c,\phi_c,\theta_c)$ as described in [Wyman 2011]. Instead of converting C_{V_1} to Cartesian coordinate to generate the shadow sample ray, we transform the vector *light to eye* in the plane defined by θ_c to generate shadow sample ray:

$$\overrightarrow{V} = Eye - Light$$
$$\overrightarrow{R} = Rotate(\overrightarrow{V},\phi_c)$$

Then we intersect $\overrightarrow{R}$ with F_1's local tangent plane P to obtain F_1's potential occluder F_2. To remove false self-shadowing, we need to shift F_1 just above the voxel which potentially cast the shadow. However, directly using F_2's corresponding voxel V_2 as the potential shadow caster will cause false positive error as shown in Figure 5.

The pseudocode and GLSL shader follow. The input parameters are fragment position, fragment normal, light position, and the resolution of the texture which holds VSVs. All parameters are in camera space.

Assume V_2's adjacent voxel which is closer to light source is V_3. If F_2 is behind or

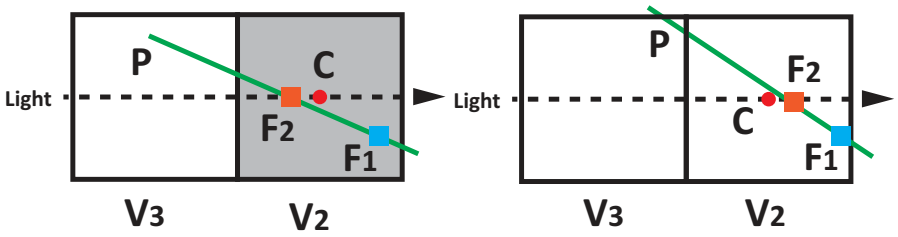

Figure 5. V_3 and V_2 are two adjacent voxels. C is V_2's voxel center. F_1 and F_2 lie on the planar surface P. (left) V_2 is occluded. Use V_2 for visibility checking will result in false self-shadowing of F_1. (right) V_2 is lit. Use V_2 for visibility checking won't cause false self-shadowing.

on the voxel center, we use V_2 for visibility checking. If F_2 is above the voxel center, we useV_3 for visibility checking. Below shows the pseudocode for clarity.

Algorithm 2 Pseudocode for sampling a voxelized shadow volume.

```
VSVs ← GenerateVSVs(LightPosition, eyePosition)
for each fragment F with normal N do
    P ← defineTangentPlane(F, N)
    C_V ← locateVoxelCenter(VSVs, F)
    R ← defineLightRay(LightPosition, C_V)
    F' ← planeRayIntersect(R, P)
    C'_V ← locateVoxelCenter(VSVs, F')
    C''_V ← adjacentCloserToLightVoxel(C'_V)
    if F' is behind or on C'_V then
        isLit ← checkVisibility(VSVs, C'_V)
    else
        isLit ← checkVisibility(VSVs, C''_V)
    end if
    outputColor ← isLit × shadeFrag(F)
end for
```

```
uniform float epsilon;  // constant epsilon based on scene scale
bool VisibilityCheckInVSV(esFragPos, esFragNormal, esLightPos, textureRes) {
    /**** Define tangent plane ****/
    vec3 n = normalize(esFragNormal);

    /**** Locate the voxel center in VSVs ****/
    vec3 voxelCenter = GetVoxelCenter(esFragPos, esLightPos, textureRes);

    /**** Compute the light ray passing through the voxel center ****/
    vec3 lightRay = VoxelCenterLineDir(voxelCenter, esLightPos);

    /**** Plane ray intersection to obtain potential occluder****/
    float t_hit = dot(n, esFragPos.xyz - esLightPos.xyz) / dot(n, lightRay);
    vec3 potentialOccluder = esLightPos.xyz + t_hit*lightRay;

    /**** Compute alpha ****/
    float alpha = ComputeAlpha(potentialOccluder, esLightPos);
    // Compute the corresponding voxel center of the potentialOccluder
    float centerAlpha = GetCenterAlpha(alpha, textureResolution);
    // Adjacent voxel's alpha
    float adjCenterAlpha = GetAdjacentCenterAlpha(centerAlpha, textureRes);

    /**** Choose the voxel to do visibility checking ****/
    float alphaOut = alpha > centerAlpha ? centerAlpha : adjCenterAlpha;

    /**** Check visibility with constant epsilon ****/
    // epsilon is based on scene scale
    bool isLit = VisibilityCheck(alphaOut, epsilon);
    return isLit;
}
```

Listing 3. Sampling a voxelized shadow volume

4. Results and Discussion

We implement our method with OpenGL/GLSL and C++. All the test scenes are rendered on a machine with Intel(R) Cores(TM) i7 CPU @2.93GHz and a NVIDIA graphics card GTX580. All the output images have the resolution of 1024×1024. In our test scenes, VSVs are generated by applying shadow map resampling and prefix scan as in Wyman et al.'s work [2011].

Figure 6 and Figure 7 compare our method with constant depth bias and slope scale depth bias [King 2004] for traditional shadow map and paraboloid shadow map in a complex scene. We use dual depth layers method [Weiskopf and Ertl 2003] to generate reference images. With constant and slope scale depth bias, objects close to the light have shadow acne while objects far from the light suffer from heavy shadow detachment. In the test scenes, our method gives results equivalent to dual depth shadow mapping but with significant improved performance.

As illustrated in Figure 9 (right), our adaptive bias suffers from over bias when the shaded fragment's local tangent plane is almost parallel to the light ray, which results

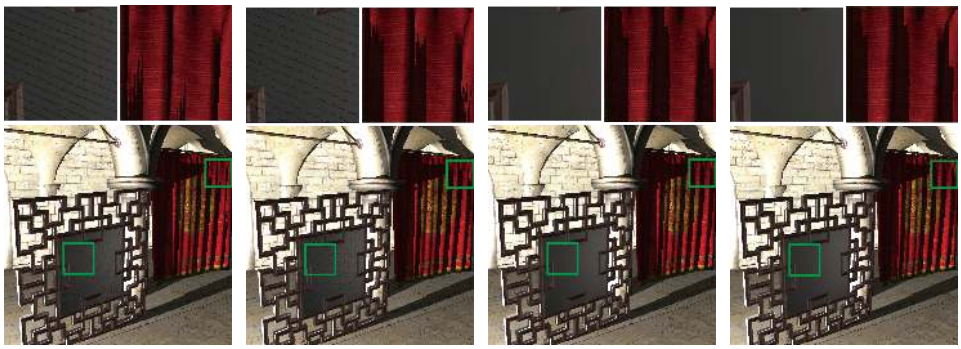

Figure 6. Sponza illuminated by a traditional light source with a shadow map resolution of 1024×1024. (left) Constant depth bias and slope scale depth bias (center left) suffer from shadow acne and shadow detachment to some extent. Our method (center right) has no visible false shadows. Dual depth layers depth bias (right) serves as a reference image.

Method	Shadow Map	Final Shading	Overall
Constant	2.095ms	4.232ms	6.327ms
Slope Scale	2.112ms	4.535ms	6.647ms
Ours	2.108ms	5.211ms	7.319ms
Dual Layer	4.716ms	5.174ms	9.890ms

Table 1. Performance measure of Sponza scene (20K polygons). The scene is lit through traditional shadow mapping with a shadow map resolution of 1024×1024.

in unexpected noise. However, this only happens to the fragments whose local tangent plane is almost parallel to light rays, which transports almost no radiance to the viewer with common materials like Lambertian and Phong. Therefore, this problem will be gone after shading is applied.

Table 1 shows the corresponding performance of the scene depicted in Figure 6. Both constant depth bias and slope scale depth bias add small overhead. Dual layers depth bias has two rendering passes and thus doubles the time for shadow map creation. Besides, an extra texture look-up costs close to 0.6ms, consuming 18% more rendering time compared with constant depth bias. In the shading stage, cost for computing adaptive depth bias in our method is close to the cost for an extra texture look-up in dual layers based method. However, with an extra render pass in shadow map creation, dual layer based method costs close to 50% more rendering time compared with our method. Figure 10 shows the performance chart of the scene in Figure 7 (19M polygons). As the shadow map resolution increases, slope scale depth costs around 5% more rendering time compared with constant depth bias. Our method costs around 20% more rendering time compared with constant bias with significantly less false shadowing. Compare to our method, dual layer depth map gives

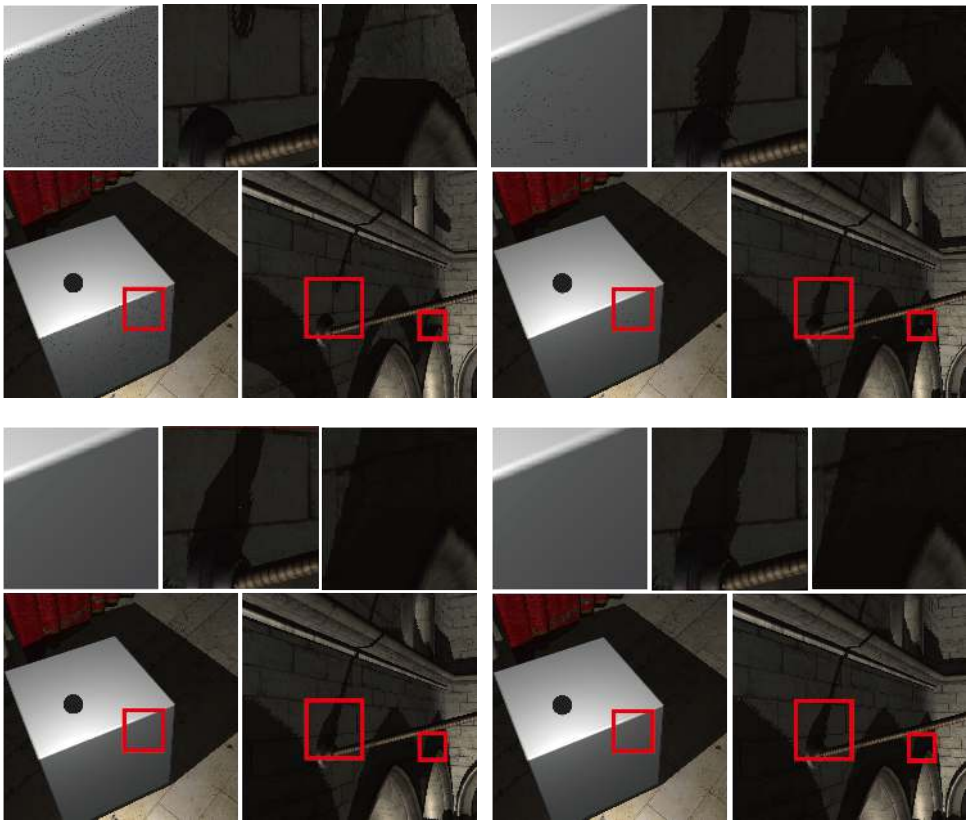

Figure 7. The scene is lit by an omni-directional light source through dual paraboloid shadow mapping with a shadow map resolution of 2048×2048. We move the camera to different parts of the scene. (left) Constant depth bias and slope scale depth bias (center left) both leave shadow acne on the cube and suffer from false unshadowing on the wall to some extent. Our method (center right) eliminates all the shadow acne on the cube without shadow detachment on the wall. Dual depth layers depth bias (right) serves as a reference image.

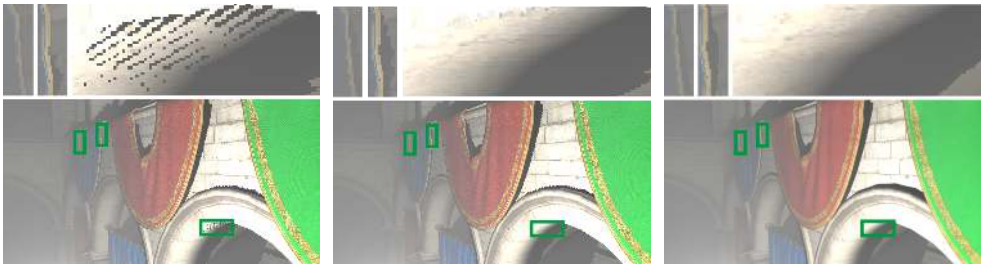

Figure 8. The scene is lit by a omni-directional light source through VSVs with a volume resolution of $1024 \times 1024 \times 512$. (left) Surface shadows are cast through VSVs with a constant bias. (center) Surface shadows are cast through VSVs with our adaptive bias. (right) Reference image. Surface shadows are cast with a 2D 8192×8192 shadow map.

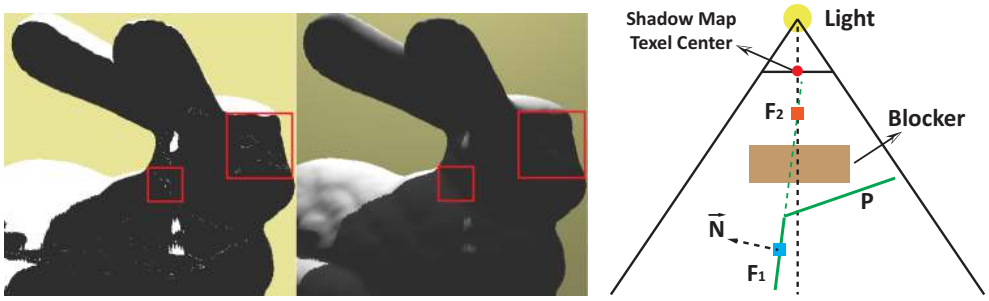

Figure 9. (left) Our adaptive bias introduces noise because of overly shifting. (center) Noise is invisible under Lambertian shading. (right) Illustration of over shifting.

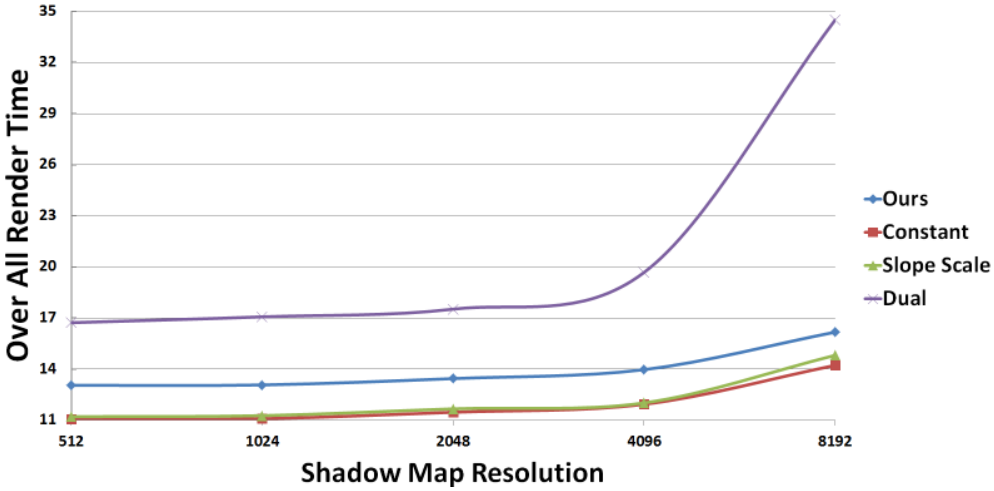

Figure 10. Performance comparison of different depth bias algorithms under different shadow map resolution. The rendering time is measured in milliseconds.

a nearly equivalent image quality but it costs 50% more rendering time when shadow map resolution is under 4096×4096 and costs more than 50% when shadow map resolution keeps increasing.

Figure 8 shows the result of applying our adaptive bias to VSVs. Since VSVs only contain binary value in each voxel, extending other depth bias algorithms, such as slope scale depth bias and dual depth layers, does not work for VSVs in a straight forward way. So we only compare the rendering result of adaptive bias with constant bias. VSVs' nature of non-uniform epipolar voxel grid and view-dependence make constant bias hard to work well. With constant bias, there remains some false self-shadowing on the arch while the distant blue curtain already suffers some shadow detachment. The adaptive depth bias reduces the false self-shadowing and in the mean time preserving more shadow details than fixed constant depth bias.

The main limitation of this technique comes from the assumption that each frag-

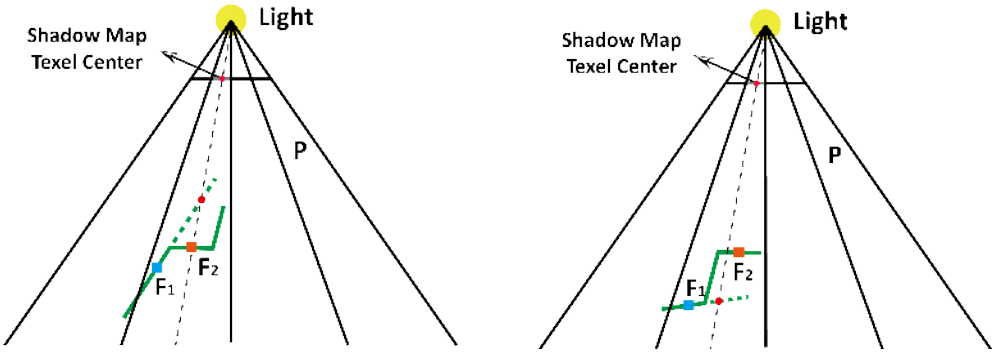

Figure 11. Illustration of the limitation of adaptive depth bias. F_1 and its potential occluder F_2 do not lie on the same planar surface.

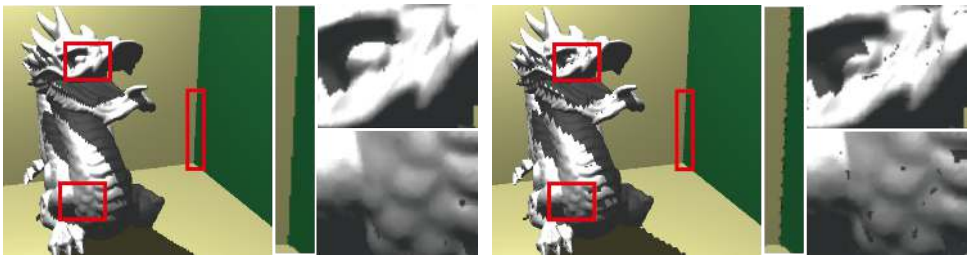

Figure 12. (left) A 1024×1024 shadow map can guarantee the planar occluder assumption. (right) A 512×512 shadow map fail to guarantee the planar occluder assumption.

ment lies on the same planar surface with its potential occluder. Figure 11 depicts a simple corner case. In the left image, although the fragment is over shifted, the false self-shadowing can still be removed. However, in the right image, the adaptive depth bias fail to remove the false shadowing.

The planar occluder assumption can break due to low shadow map resolution or large scene scale where the objects are very far from the light source. In Figure 12, the light source is far from the object. With a 1024×1024 shadow map, adaptive depth bias can eliminate most false self-shadow. With a 512×512 shadow map, false self-shadow appears in the corner of the wall and concave areas of the model and shadow detachment is visible on the dragon.

For VSVs, the nature of epipolar sampling makes the planar occluder assumption hard to keep when the light is behind the viewer. As shown in Figure 13, with a $1024 \times 1024 \times 512$ binary volume, adaptive depth bias can eliminate most false self-shadow. When the volume resolution reduces to $512 \times 512 \times 512$, shadow acne begins to appear.

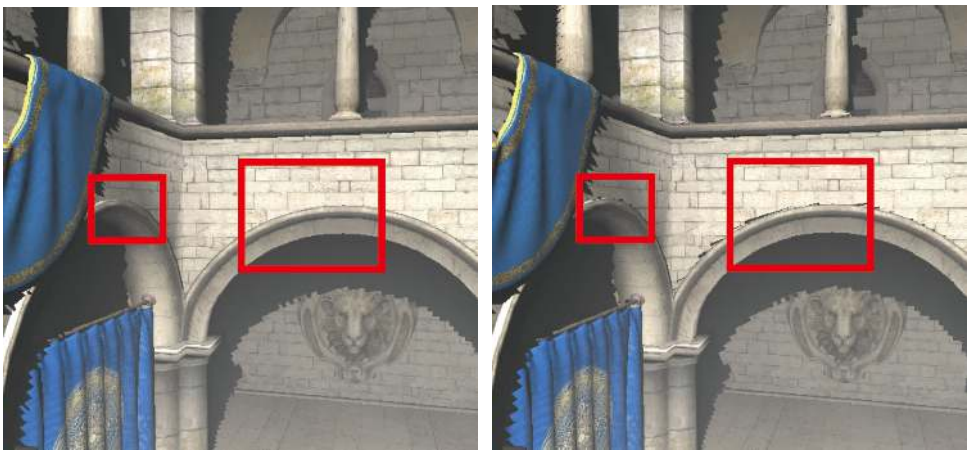

Figure 13. Illustration of the case when light source is behind the viewer. (left) $1024 \times 1024 \times 512$ VSVs. (right) $512 \times 512 \times 512$ VSVs.

5. Conclusion

We present a method to eliminate false self-shadowing for surface shadowing algorithms by producing adaptive depth bias. We compute the potential occluder for each fragment and shift the fragment just above its potential occluder before visibility checking. Our method works for 3D shadow maps as well as 3D voxelized shadow volumes. Compared to constant depth bias and slope scale depth bias, our adaptive depth bias removes more false self-shadowing and causes less false shadow detachment. Our method gives an equivalent result as dual depth layers based method but with less cost.

References

BRABEC, S., ANNEN, T., AND SEIDEL, H.-P. 2002. Shadow mapping for hemispherical and omnidirectional light sources. In *Advances in Modelling, Animation and Rendering*. Springer, 397–407. 147

GAUTRON, P., MARVIE, J., AND BRIAND, G., 2013. Method for generating shadows in an image, Jan. 23. EP Patent 2,411,967. URL: http://www.google.com/patents/EP2411967B1?cl=en. 147

KING, G. 2004. Shadow mapping algorithms. *GPU Jackpot presentation*, 354–355. 147, 155

WEISKOPF, D., AND ERTL, T. 2003. Shadow mapping based on dual depth layers. In *Proceedings of Eurographics*, vol. 3, 53–60. 147, 155

WILLIAMS, L. 1978. Casting curved shadows on curved surfaces. In *ACM SIGGRAPH Computer Graphics*, vol. 12, ACM, 270–274. 147

WOO, A. 1992. The shadow depth map revisited. In *Graphics Gems III*, Academic Press Professional, Inc., 338–342. 147

WYMAN, C. 2011. Voxelized shadow volumes. In *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics*, ACM, 33–40. 147, 152, 153, 155

Index of Supplemental Materials

In the demo package, there are two folders, "exe" and "models". Folder "models" holds all the obj models used in the demo. Folder "exe" holds the executable file and all the shaders. Shaders for traditional shadow map are in the folder "perspectiveSM". Shaders for omni-directional shadow map are in the folder "paraboloidSM".

The adaptive depth bias algorithm for traditional shadow map is implemented in the shader file "shade_perspective.frag.glsl". The adaptive depth bias algorithm for omni-directional shadow map is implemented in the shader file "shade_paraboloid.frag.glsl". Other shaders are for shadow map construction and visualization.

Author Contact Information

Hang Dou
Washington University in St. Louis
St Louis, MO 63130
hangdou@gmail.com

Ethan Kerzner
SCI Institute
Salt Lake City, UT 84112
kerzner@sci.utah.edu

Yajie Yan
Washington University in St. Louis
St Louis, MO 63130
danielyan86129@gmail.com

Zeng Dai
University of Iowa
Iowa City, IA 52246
zeng-dai@uiowa.edu

Chris Wyman
NVIDIA
Redmond, Washington
chris.wyman@acm.org

Hang Dou, Yajie Yan, Ethan Kerzner, Zeng Dai, and Chris Wyman, Adaptive Depth Bias for Shadow Maps, *Journal of Computer Graphics Techniques (JCGT)*, vol. 3, no. 4, 146–162, 2014
http://jcgt.org/published/0003/04/08/

Received:	2013-10-22		
Recommended:	2013-12-09	Corresponding Editor:	Michael Schwarz
Published:	2014-12-19	Editor-in-Chief:	Morgan McGuire

© 2014 Hang Dou, Yajie Yan, Ethan Kerzner, Zeng Dai, and Chris Wyman (the Authors).

The Authors provide this document (the Work) under the Creative Commons CC BY-ND 3.0 license available online at http://creativecommons.org/licenses/by-nd/3.0/. The Authors further grant permission reuse of images and text from the first page of the Work, provided that the reuse is for the purpose of promoting and/or summarizing the Work in scholarly venues and that any reuse is accompanied by a scientific citation to the Work.