Robust Shadow Maps for Large Environments

Daniel Scherzer*

Institute of Computer Graphics
Vienna University of Technology
Austria

Abstract

One of the most demanding challenges for real-time shadow algorithms is their application to large-scale, polygon-rich and dynamic environments. In this paper, we discuss the major problems encountered in applying shadow maps to such an environment and provide practical and robust solutions to the appearing problems. We tackle projection aliasing with the aid of an eye space blur. We compare the major biasing methods to remove incorrect self-shadowing of polygons. Finally we are providing some advancements to the recently published light space perspective shadow mapping method to resolve projection aliasing problems.

Keywords: shadow algorithm, real-time

1 Introduction

Shadows give important visual cues for perceiving the geometric relationship between objects. This means a shadow can help to clarify position, size and geometry of the shadow casters and the geometry of the shadow receiver as well as the distance to the light source (see Figure 1).

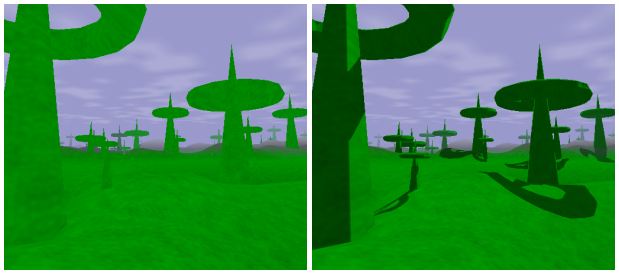

Figure 1: The same image is shown once with and once without a shadow. With the shadow attached the objects interrelationship is far easier to understand.

Consider a tree: Without a shadow, the tree just floats above the ground, missing the firm connection to the terrain in reality provided by his shadow. Additional shadows help to create a certain atmosphere and add immensely to the realism of a scene.

*scherzer@cg.tuwien.ac.at

A point is said to be in shadow, when it cannot be seen from the viewpoint of the light source. The object that is the cause for not seeing this point is called the shadow caster, occluder or blocker, which blocks the light rays from reaching the point. The object on which the point in shadow lies is called the shadow receiver.

1.1 The Problem

Why is the shadowing of a large-scale, polygon-rich and dynamic environment so demanding for shadow algorithms? Let's take a look at each of the three properties of the environment and the demands that these properties place on any shadow algorithm:

A *dynamic* environment is a scene that may vary from frame to frame. For example some objects move or the position or direction of the light changes. This makes it necessary to regenerate the shadow each frame because of the possible changes in shadow casters or lighting conditions that affect the resulting shadow. This demands an algorithm that can calculate a new shadow each frame in a matter of milliseconds.

A *polygon-rich* environment, in our context and at this point of hardware speed, is a scene that contains roughly 100,000 or more visible triangles. This makes the use of classical geometry-based shadow algorithms, for example the classical shadow volume algorithm [7], difficult. The shadow-volume algorithm needs a silhouette search and consumes fill-rate proportional to the number of silhouette edges. Approaches to use simpler geometry for the shadow generation are possible, but may result in incorrect shadows. Additionally the generation of the simpler geometry is a non-trivial task. There simply is no robust and universal algorithm currently known that handles all configurations correctly.

A *large-scale* environment is a scene that contains near as well as far off objects, in arbitrary positions and sizes. In these environments, point, spot and directional lights are common. Especially directional lights, used to simulate the sun for instance, often cover large parts of the terrain. This setup means no additional problems for the shadow-volume approach because the geometrical nature of the algorithm makes it independent of the scales that occur in the scene. However we had to rule this approach out because of its performance deficit in polygon-rich scenes. The image space approaches, namely shadow

mapping, have problems with such constellations. Solutions to these problems exist. We will use light space perspective shadow maps [20] to overcome these problems.

The rest of the paper is structured as follows: Section 2 gives an overview of the basics of shadow generation. Section 3 discusses related work on shadow mapping with an emphasis on common problems of shadow maps. Section 4 describes practical solutions we used for solving the problem of shadowing large-scale, polygon-rich and dynamic environments. Section 5 describes the results we found on implementing various approaches and Section 6 finally gives a summary and states open problems for future research.

2 The Basics

Shadow generation was and is a hot research field in computer graphics. Two main algorithms, shadow volumes by Crow [7] and shadow mapping by Williams [20] can handle scenes with general sets of shadow casters and receivers, including self-shadowing, in real time.

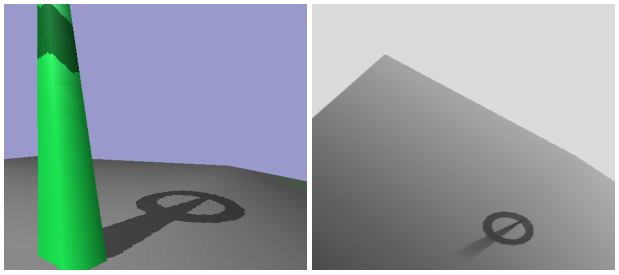

Figure 2: The shadowed eye view (left). The light depth view (shadow map) (right).

The use of shadow mapping has gone a long way since its introduction by Williams [19] and its fully hardware-accelerated use today. His basic idea was to render the scene from the light's point of view into a z-buffer and save the so gathered depth values in the so called shadow map. For a point light, these depth values give us the distance each point visible to the light has to the light source. Upon rendering the scene from the eye point, each pixel is transformed into light space and its depth compared to the depth stored in the shadow map. If the stored depth is nearer to the light, the pixel must be in shadow and the lighting of the pixel can be altered to reflect this (see Figure 2).

One important benefit of shadow mapping is that it is independent of the scene geometry, which makes it suitable for highly tesselated scenes. This independence comes from the fact that shadow mapping is an image space technique.

With the use of projective texture mapping as described by Segal [15], shadow mapping can be easily matched to hardware, with one additional render pass to create the light view depth image, the *shadow map*.

An excellent overview of shadow algorithms can be found in Möller and Haines' Real-Time Rendering book [12].

3 Problems of shadow mapping

Shadow mapping, when used with the z-buffer, suffers from undersampling, which causes various types of aliasing artefacts (see Table 1), because of the finite and regular structure of the z-buffer. This basic form of the technique will be called standard shadow mapping (SSM) in the rest of the paper. Numerous solutions to these problems exists. Johnson [9] proposes the use of an irregular z-buffer that can eliminate this problems. A similar approach was chosen by Aila in [1]. Another solution is to use a hierarchical grid as a shadow map structure [8]. These solutions don't map well to current hardware and software implementations are not competitive to todays hardware methods in terms of speed.

Error	**Type**	**Cause**
insufficient resolution on polygons almost parallel to the light direction	local	Projection aliasing (undersampling)
insufficient resolution near the observer	global	Perspective aliasing (undersampling)
moiré-patterns	local	Self-(un)shadowing (resampling; depth precision)

Table 1: The different errors of shadow mapping. Global errors affect easily predictable or all parts of the scene, while local errors appear on parts of the scene that are more complex to identify and are therefore harder to countermeasure in real-time.

3.1 Self-(un) shadowing

One shadow map problem originates in the transformation of the regularly spaced pixels from eye to light space. In light space these transformed pixels may fall between the regular sample values the shadow map provides. Together with the finite precision of depth values, this can lead to self-shadowing artefacts and depth quantization. Without biasing, moiré patterns are visible (see Figure 6). Self shadowing artefacts are local effects because they depend on the orientation of the concerned polygons relative to the light direction (see Table 1). A workaround for this problem exists in using a manually defined depth bias. A depth bias is a small increment added to the shadow maps depth values to move them further away in order to avoid an incorrect shadowing of the corresponding shadow casters. This needs user intervention and provides no automatic solution for an arbitrary scene. An additional problem of the bias is that the shadow is moved in light space z-direction. This can lead to noticeably incorrect shadows.

Second-depth shadow mapping, as proposed by Wang and Molnar [17], is a solution that can handle most configurations well. The idea is to assume solid shadow casters. With this assumption, the depth test can be transfered from the nearest surface as seen from the light source to the second nearest surface. Later Weiskopf and Ertl, influenced by this idea and by work of Woo [21] (the midpoint shadow maps) proposed dual shadow maps [18]. The first shadow map contains the depth of the first surface visible form the light source, as any normal shadow map does. The second shadow map contains the depth of the second surface, the surface you would encounter, if you had stripped away the surfaces stored in the first depth map. These two depths can be combined to get a robust bias.

For large-scale environments with thousands of objects, an automatic method seems indispensable. One offset value rarely suffices to solve the biasing problems of the whole scene, and manually choosing offset values would be too time consuming. Dual shadow maps are problematic to use because the generation of the shadow map per se is already expensive. A lot of objects are visible for the common directional lights and have to be rasterized. The second shadow map we would need to generate would nearly double the costs of this step.

Another solution which influences the depth biasing, but is mainly a solution to the resampling problem was proposed by Reeves, Salesin and Cook [14]. The paper describes a new filtering technique called percentage closer filtering. An additional benefit of this filtering approach is that it provides smooth shadow boundaries, which lessens the jagged effect of the borders of undersampled shadows. A simplified version of this technique is incorporated into today's consumer level hardware.

3.2 Projection aliasing

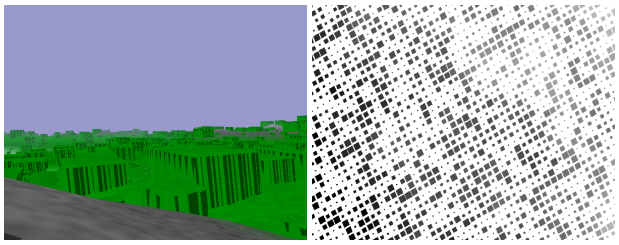

Figure 3: The projection aliasing artefacts, the black stripes, in the eye view (left) are caused by too few samples of the cubes sides as seen from the light view (right).

Following Stamminger and Drettakis [16], we further divide the aliasing artifacts into projection and perspective aliasing. Projection aliasing is a local phenomenon (see Table 1). This phenomenon is primarily caused by surfaces almost parallel to the light direction. These surfaces are therefore very sparsely sampled because little of the area of the surface is visible from the point of view of the light source (see Figure 3). A solution to this problem cannot be found with a simple method that operates on the whole scene, but requires a detailed analysis of the scene geometry. Projective aliasing remains an open problem for any real-time shadow mapping approach. For large-scale environments, this problem is especially complicated. The usual count of objects in such scenes is very high, and therefore some objects will probably have polygons that are almost parallel to the light direction. Hierarchical shadow maps [8] or an irregular z-buffer [9] [1] can solve this problem, but these methods are not competitive in terms of speed to real-time methods. At this point, only a careful design of the scene or clever application of a blur filter on the shadows can alleviate this problem. See Section 4.2 for details.

3.3 Perspective aliasing

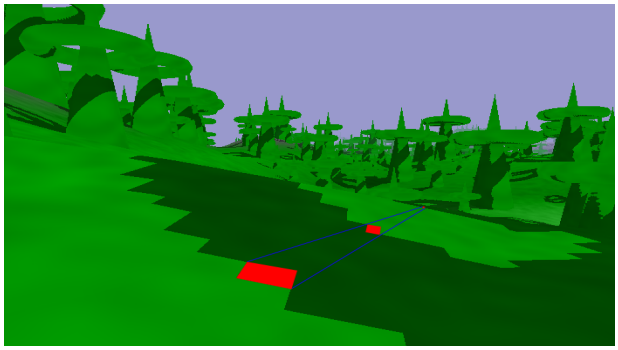

Figure 4: Insufficient shadow map resolution for shadows near the view point caused by perspective aliasing.

Fast approaches exist that mainly handle perspective aliasing. Perspective aliasing is common with standard shadow maps for a perspective eye view. A perspective view shows nearby objects larger than distant objects. The light space in which the standard shadow map is calculated does not incorporate this information. So an object is stored with a fixed resolution in the shadow map, regardless of the distance to the eye. The outcome is a shadow resolution that is too low for nearby objects and too high for distant objects in eye space (see Figure 4). A simple solution is to use multiple shadow maps. For example one for the near objects and one for the far objects. This means a partitioning of the view frustum into subsets and each subset is rendered into its own shadow map. Another idea, common to most of the fast approaches, is to redistribute the resolution of the shadow map in a better way. Perspective aliasing is aggravated for large scale environments. The shadow map resolution must be used for the shadowing of the whole visible scene, therefore the resolution with which a single object is saved to the shadow map decreases with the increase of the area to cover with the shadow map. This makes perspective aliasing the number one problem for this type of environment.

The first completely hardware implemented technique that used the redistribution approach was perspective

shadow mapping (PSM) [16]. The main idea is to construct the light space in the post perspective space of the eye. This should create an even distribution of the depth map samples in the eye space. Unfortunately in real world scenarios, serious robustness and quality issues emerge [20]: Various special cases have to be considered, which complicates implementation. The shadow quality is hard to predict and usually bad for scenes with very near and far off objects. Shadow quality changes rapidly from frame to frame on view-point changes, which leads to flickering shadows when animating the scene.

Recently various methods to cure the remaining problems of perspective shadow mapping were published. Chong's senior thesis [6] presented a reparameterization of PSM into a more general frame-work and provided a thorough analysis of the 2D case. Based on this Chong and Gortler presented in [5] a shadow algorithm that can calculate a shadow map that gives optimal results for a chosen plane of interest. After this more theoretical approaches, Kozlov [10] investigated practical advancements of PSM to solve the problems of PSM. Trapezoidal shadow maps (TSM) [11] and light space perspective shadow maps (LispSM) [20] are concurrent approaches to use the main advantages of PSM, but not its weaknesses. The shadow quality of both approaches is similar. The first one uses a new space, the trapezoidal space, and 2D transformations. This is used together with an iterative process that determines a perspective transformation that should minimize projection aliasing. The paper additionally describes the use of a linear z-depth distribution to minimize self-shadowing artefacts. The second approach uses an additional perspective transformation that is applied after the light space is determined. The free parameter of this transformation determines the strength of the perspective warp. This parameter is calculated with a formula, derived from a perspective aliasing error analysis. The small overhead for calculating the additional perspective warp, the ease of tweaking the shadow quality and the simplicity of implementation, makes LispSM an ideal algorithm for our desired *large-scale*, *polygon-rich* and *dynamic* environments.

4 Solving the problem

In this section we will look at practical solutions to the afore-mentioned problems of shadow mapping. In Section 4.1 we will tackle the problem of focusing the light space to the volume of space that can cast visible shadows. Section 4.2 compares different methods of biasing to avoid self-shadowing artefacts. Section 4.3 investigates into blurring to hide projection aliasing and finally Section 4.4 discusses various extensions to LispSM to make it more robust in real-world scenarios.

4.1 Focusing the light space

To increase the amount of useful information that is stored in the shadow map, we only want to consider the parts of the light space for the shadow map that can cast a visible shadow into the view frustum. Brabec [4] showed the importance of focusing the shadow map to the visible parts of the scene. This step seams obvious, but some intricacies are involved. The geometrical solution is to calculate the convex hull of the view frustum and the light position (for directional lights this position is at infinity) and afterwards clip this body with the scene bounding volume and the light frustum (see [16] for details). Clipping to the scene bounding volume is necessary because today very large view frusta are common and they frequently extend outside the scene borders. We call the resulting body the intersection body B (see Figure 5).

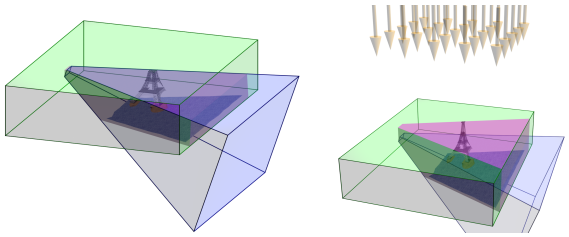

Figure 5: The view frustum is blue. The scene bounding box is green. *Left:* The clipping of the view frustum with the scene bounding box decreases it's size considerable. *Right:* The final intersection body B (violet) with a light direction from above (orange arrows)

If we use a visibility algorithm, O'Rorke's article [13] gives various practical hints for using this visibility information to decrease the volume we have to consider for the shadow map generation.

4.2 Self-shadowing artefacts

The cause for self-shadowing artefacts is the resampling that takes place when eye space pixel coordinates are transformed into light space to get the respective shadow map samples. The use of a bias is the common method to resolve this problem. If we use a constant bias, all depth values are moved the same amount into or out of the screen. This leads to problems for polygons with different depth slopes. For a depth slope near zero hardly any biasing is needed, while for a polygon that is almost parallel to the light direction (large depth slope) a big bias is appropriate. The handling of this problem is the purpose of slope-scale biasing. However, slope-scale biasing has problems with the non-linear distribution of z-depth values of PSM, TSM and LispSM. This non-linear distribution of depth values is generated by the perspective transformation that involves an $1/w$ term, generating a hyperbolic depth value distribution. Therefore the false self-shadowing problem is increased for these algorithms. For TSM, the biasing problem is so great

that the authors of the paper recommend omitting the z-coordinate from the perspective transformation, actually generating linearly distributed depth values. Kozlov [10] proposes to use slope-scale biasing in world-space for PSM and transforms the results into post-projective space. LispSM has less problems with self-shadowing artefacts. The methods [17] [21] [18] mentioned in Section 3.1 are unsuited for real-time applications because of the performance penalty introduced by the generation of the second depth image needed by all of them.

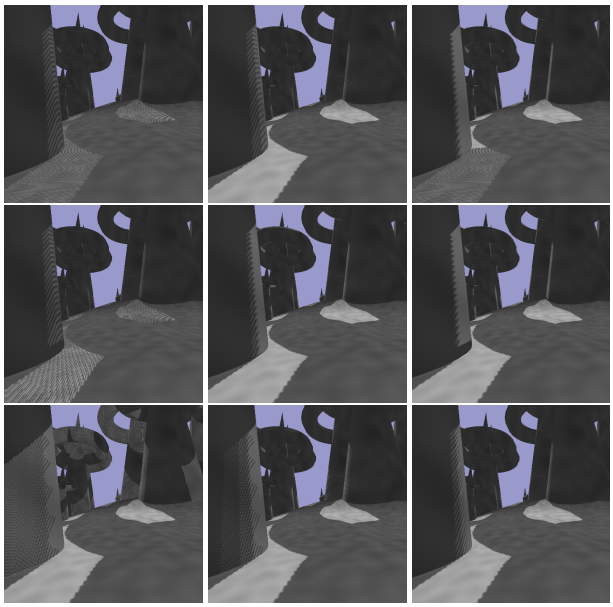

Figure 6: *Left:* no biasing; *Middle:* constant biasing; *Right:* slope-scale biasing; *Top:* hyperbolic z-distribution; *Center:* linear z-distribution; *Bottom:* back-side rendering

We present extensive experiments with different biasing methods for LispSM. We found that a simple slope-scale biasing with the hyperbolic z-distribution of LispSM, as for example provided by the *polygon offset* interface of Open GL, gives satisfying results for common configurations. The needed bias can be much smaller than the value needed for a constant bias. The resulting shadows are therefore less shifted respectively more correctly positioned than with the bigger constant bias. For example in our test scene, we could avoid most self-shadowing artefacts with a relatively small slope-scale bias of 2.0/4.0. The use of a linear z-distribution as proposed in the TSM paper is also easily incorporated into LispSM with the aid of vertex shaders. The results with linear z biasing are better as with the hyperbolic z-distribution, but come at the cost of additional hardware requirements for the needed vertex shaders. Another form of biasing that uses the back sides of the scene geometry for the depth comparison was tried too. This method removes most self-shadowing artefacts on the ground, but in all other cases the quality is similar to normal slope-scale biasing. The following matrix of images in Figure 6 shows the results with the various combinations of the afore-mentioned methods. The performance of the different methods is equal, because of the unnoticeable penalty in rendering time these methods introduce. On most platforms no speed difference at all is perceivable.

As can be seen the version with a linear z-distribution, together with slope-scale biasing gives the best results.

4.3 Projection aliasing

We experimented with an eye-space blur of the shadow map to hide some of the remaining artefacts, especially projection aliasing. The idea is to map the shadow map in eye space onto unlit, unicoloured geometry, generating a grey-scale (grey values originate from the pcf filtering at the shadow borders) image of the mapped shadows (see Figure 7). This image can be blurred repeatedly to generate a 2D-texture that can be applied as a intensity lookup texture for the lighting calculation in the final rendering. As can be seen in Figure 8, projection aliasing can be removed with a high enough blur at the cost of shadow details.

Figure 7: The shadow map is applied to the unlit and unicoloured geometry.

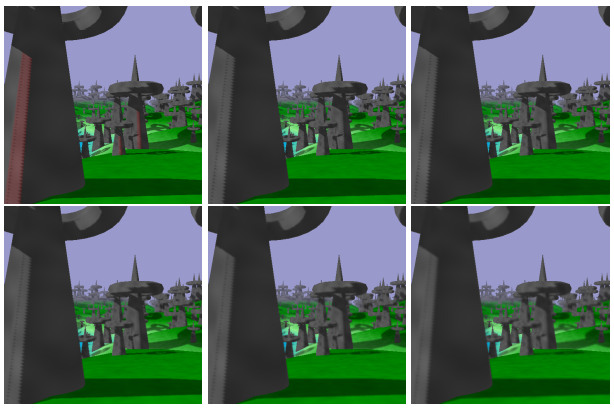

Figure 8: The same scene starting with no shadow map blur and with 1x, 2x, 4x, 8x, and 16x blur iterations. Notice the regions marked in red with projection aliasing artefacts and how the artefacts disappear with increasing blur.

The solution to the loss of shadow detail in the distance is to use a depth dependent blur. Near the viewer, the shadow map is blurred more and with increasing depth the shadows are less and less blurred, to preserve shadow

details in the distance. This is successfull because the projection aliasing artefacts in the distance are generally much smaller (in terms of the pixel area) as the ones near the view point.

4.4 Perspective aliasing

Recent papers have introduced practical solutions to this problem. The trick is to redistribute the shadow map resolution. Give more shadow map space to the near objects and less to the far objects. This trick makes shadow maps view dependent because the distance relationship to each object changes at each view change that includes a translation. Additionally the set of shadow casters to consider may changes with every view point transformation. This implicates a regeneration of the shadow map every single frame. The calculation time for these algorithms therefore has to be rather small because of the overhead this costs each frame.

Perspective shadow mapping [16] and its successors use a perspective transformation to do the redistribution of the shadow map texels. The implementation is very hardware friendly. The perspective transformation is done by the hardware and only the setup-costs for the transformation add to the rendering time. We focus on LispSM to resolve the perspective aliasing. The additional perspective transformation of LispSM is calculated in light space. This means that first the transformation into light space takes place. Normally this step involves a light view matrix and a light projection matrix that are multiplied together. This joined transformation gives us the transformation into light space.

The frustum P of the additional perspective transformation of LispSM has near and far planes parallel to the light direction and a view vector V parallel to the shadow map. It's projection center is at the height of the eye position (see Figure 9).

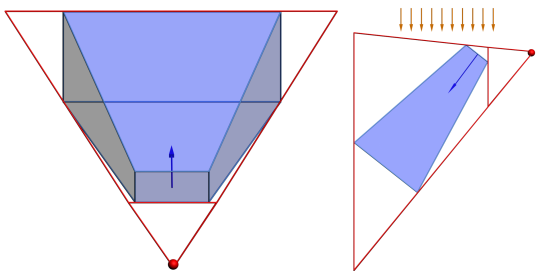

Figure 9: The view frustum is blue. The frustum defining the light space perspective transformation P is red. The red sphere is the projection center C. The light rays are orange. *Left:* Both frusta as seen from the point-of-view of the light. *Right:* A side view of both frusta with the light-rays coming from above.

The use of a perspective transformation gives the best results for perpendicular view and light directions because in this case the perspective transform can influence the whole depth range of the view frustum (see Figure 10).

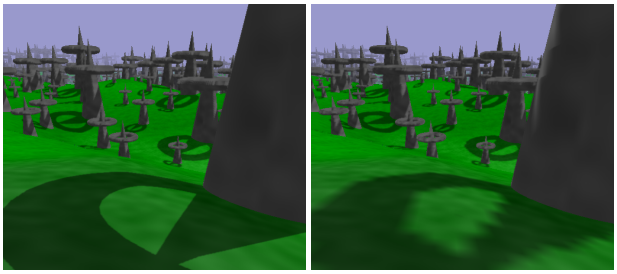

Figure 10: For near perpendicular view and light directions LispSM (left) gives the best results. Uniform shadow mapping (right)has much more perspective aliasing.

This is also the case where the most perspective aliasing is present. In the case of parallel light and view vectors, no perspective aliasing is present and the perspective transformation can only worsen the quality of the shadow map. This is called the duelling frusta case (see Figure 11). A solution exists that uses five shadow maps, which makes it, however, expensive.

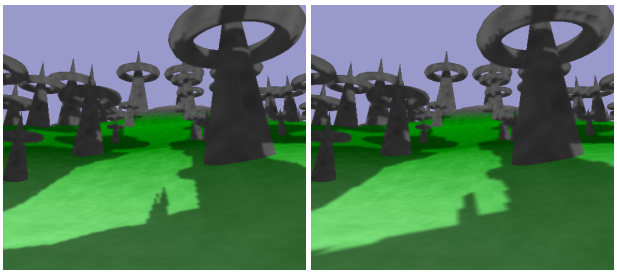

Figure 11: For near parallel view and light directions LispSM (left) converges to uniform shadow mapping (right), making the shadow borders blocky.

Recently a recipe for handling the duelling frust case with an extension to TSM was published on the internet [2]. This method uses a single shadow map and divides it into four viewports that are used to render into adaptively. An update of up to four light views is, however, unfeasible for our polygon rich scenes.

For LispSM, it is logical to decrease the perspective warp strength when the angle between light and view direction becomes smaller. In the extreme case of parallel light and view directions, no perspective warp should be present and standard shadow mapping should be reached. In the case of LispSM, this is already incorporated into the formula used to calculate the optimal warp strength presented in the paper, through the term $sin(\gamma)$:

$$n_{opt} = \frac{z_n + \sqrt{z_n(z_n + \Delta z)}}{sin(\gamma)} \qquad (1)$$

n_{opt} is the near plane distance of the perspective transformation P. Changing this parameter controls the strength of the warp. z_n is the eye frustum near plane distance. Δz is the depth extent of the intersection body B in light space and γ is the angle between light and view direction.

One possibility to tweak this formula is to use a Δz derived from a visibility algorithm: A visibility algorithm can easily calculate a more exact far plane distance. If this far plane distance is smaller than the original far plane distance, this can be used to increase the shadow quality near the viewer. The new far plane distance can be used to calculate Δz. This smaller Δz moves shadow map texels from far away and invisible objects to nearer objects.

It is important to note that this new far plane distance is not usable for the intersection body calculation because it is possible that invisible objects inside the view frustum cast a shadow inside the visible part of the view frustum (see [13] for details). The shadows of these objects would not be generated if we used this new far plane distance for all calculations. We only propose to use the new far plane distance for the calculation of n_{opt}. With this optimization, we only give these invisible shadow caster objects with visible shadows less space/resolution in the shadow map.

With the aid of n_{opt}, the projection center C_P of the perspective transformation P is given by:

$$C_P = C_{eye} - (n_{opt} - z_n) * V_P \tag{2}$$

where C_{eye} is the eye position. z_n is the eye frustum near plane distance and V_P is the view vector of the frustum of the perspective transformation.

V_P should be pointing in the direction of the eye view direction V_{eye} because we want the perspective transformation to produce the closest match between the size ratios of objects in eye space and the resolution ratios used for the objects in the shadow map. This means we want the parts of the view frustum that are close to the eye (the near plane) in eye space to be close to the projection center C_P in light space. Additionally V_P should be perpendicular to the light direction V_{light} (see Figure 12). This can be achieved by

$$V_P = (V_{light} \otimes V_{eye}) \otimes V_{light} \tag{3}$$

where $\otimes$ is the normalized vector cross product.

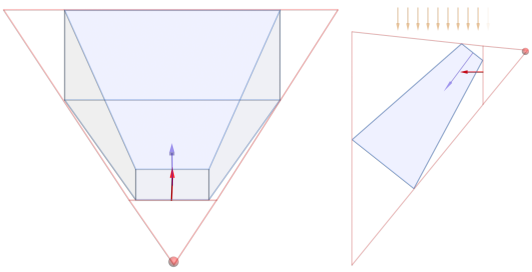

Figure 12: The view direction is blue. V_P is red. The light rays are orange. *Left:* V_P as seen from the point-of-view of the light. *Right:* A side view of V_P with the light-rays coming from above.

We discovered that in real-world scenes this is not always optimal. The problem is that we want to focus on the intersection body B, and this body can greatly differ from the original view frustum. We propose to use the direction from the eye position to the other end of the intersection body B that is in the middle of the volume of the body. We call this vector middle vector. It is clear that in certain cases this vector differs from the V_P we calculated above.

A very fast method is to treat B as a point cloud and sum up all vectors emanating from the eye position to each point. This method automatically gives more weight to points that are far away from the eye position (intersection points with the far plane), which is good because generally these points influence the volume of the body much more than the points near the view frustum near plane. Wrong results are possible if the intersection body B had had parts near the far plane intersection results with much more intersections than on other parts of the far plane intersection results. In this case, simple adding up gives wrong results. In practice we found no robustness problems with this approach that would call for the usage of a more complex method.

5 Results

We have implemented the described methods and improvements, based on the LiSPSM algorithm and used it for a scene lit by one directional light source. The scene is an outdoor environment that contains 10.000 tree-like objects and uses Coherent Hierarchical Culling (CHC) [3] for visibility determination. We implemented 2x2 percentage closer filtering with the OpenGL Shading Language because not all hardware vendors supply us with automatically applied pcf for shadow maps. Fogging was enabled. Platform was a Pentium 4 2.4GHz with 1GB RAM and an ATI Radeon 9600 with 265MB RAM. All pictures shown in this paper where captured using a 512x512 pixel viewport resolution and a 2048x2048 pixel shadow map resolution. The field of view of the view frustum was $60°$, near plane distance was 0.1 and the far plane distance was 70.

We derived the Δz from the visibility algorithm and used a slope-scale bias with linear z-distribution, when not otherwise noted. We calculated the intersection body B with the geometrical approach and used the intersection body B middle vector as V_P.

The case of our example scene shows that it is possible to use shadow mapping as a robust solution for shadow mapping of a *large-scale*, *polygon-rich* and *dynamic* environment.

6 Conclusion

In this paper we presented practical solutions to the problem of shadowing *large-scale*, *polygon-rich* and *dynamic* environments with the aid of LispSM. Our discussion was centered on directional lights because such lights often suffer, more than spot lights, from perspective aliasing. We discussed intricacies when focusing the light space

and provided different approaches to get good results. We stated the effects and problems of biasing to avoid self-shadowing artifacts and compared several solutions. We discussed perspective aliasing and introduced robust methods to implement LispSM. An open problem for future work remains projection aliasing. As a local problem it is difficult to find a fast method that can deal with it. The use of blurring is possible, but our results show that often shadow details have to be sacrificed to remove projection aliasing with this method. It would be worthwhile to investigate into more advanced soft shadow techniques to alleviate this problem.

References

[1] Timo Aila and Samuli Laine. Alias-free shadow maps. In *Proceedings of Eurographics Symposium on Rendering 2004*, pages 161–166. Eurographics Association, 2004.

[2] Graham Aldridge. Generalized trapezoidal shadow mapping for infinite directional lighting. *http://legion.gibbering.net/projectx/paper/shadow%20mapping/*, October 2004.

[3] Jiří Bittner, Michael Wimmer, Harald Piringer, and Werner Purgathofer. Coherent hierarchical culling: Hardware occlusion queries made useful. *Computer Graphics Forum*, 23(3):615–624, sep 2004. Proceedings EUROGRAPHICS 2004.

[4] Stefan Brabec, Thomas Annen, and Hans-Peter Seidel. Practical shadow mapping. *Journal of Graphics Tools: JGT*, 7(4):9–18, 2002.

[5] H. Chong and S. J. Gortler. A lixel for every pixel. In *Proceedings of Eurographics Symposium on Rendering 2004*, 2004.

[6] Hamilton Chong. Real-time perspective optimal shadow maps. Senior thesis, Harvard College, Cambridge, Massachusetts, April 2003.

[7] Franklin C. Crow. Shadow algorithms for computer graphics. In James George, editor, *Proceedings of the 4th annual conference on Computer graphics and interactive techniques*, volume 11, pages 242–248. ACM Press, July 1977.

[8] Randima Fernando, Sebastian Fernandez, Kavita Bala, and Donald P. Greenberg. Adaptive shadow maps. In Eugene Fiume, editor, *SIGGRAPH 2001 Conference Proceedings*, Annual Conference Series, pages 387–390. ACM SIGGRAPH, Addison Wesley, August 2001.

[9] Gregory S. Johnson, William R. Mark, and Christopher A. Burns. The irregular z-buffer and its application to shadow mapping. Research paper, The University of Texas at Austin, 2004.

[10] S. Kozlov. Perspective shadow maps - care and feeding. *GPU Gems*, pages 217–244, 2004.

[11] Tobias Martin and Tiow-Seng Tan. Anti-aliasing and continuity with trapezoidal shadow maps. In *Proceedings of Eurographics Symposium on Rendering 2004*, pages 153–160, 2004.

[12] Tomas Möller and Eric Haines. *Real-Time Rendering, Second Edition*. A. K. Peters Limited, 2002.

[13] J. O'Rorke. Managing visibility for per-pixel lighting. *GPU Gems*, pages 245–257, 2004.

[14] William T. Reeves, David H. Salesin, and Robert L. Cook. Rendering antialiased shadows with depth maps. *Computer Graphics (SIGGRAPH '87 Proceedings)*, 21(4):283–291, July 1987.

[15] Mark Segal, Carl Korobkin, Rolf van Widenfelt, Jim Foran, and Paul Haeberli. Fast shadows and lighting effects using texture mapping. *Computer Graphics (SIGGRAPH '92 Proceedings)*, 26(2):249–252, July 1992.

[16] Marc Stamminger and George Drettakis. Perspective shadow maps. In *Siggraph 2002 Conference Proceedings*, volume 21, 3, pages 557–562, July 2002.

[17] Yulan Wang and Steven Molnar. Second-depth shadow mapping. Technical report, University of North Carolina at Chapel Hill, 1994.

[18] D. Weiskopf and T. Ertl. Shadow Mapping Based on Dual Depth Layers. In *Procceedings of Eurographics '03 Short Papers*, pages 53–60, 2003.

[19] Lance Williams. Casting curved shadows on curved surfaces. *Computer Graphics (SIGGRAPH '78 Proceedings)*, 12(3):270–274, Aug. 1978.

[20] Michael Wimmer, Daniel Scherzer, and Werner Purgathofer. Light space perspective shadow maps. In *Proceedings of Eurographics Symposium on Rendering 2004*, 2004.

[21] Andrew Woo. The shadow depth map revisited. *Graphics Gems III*, pages 338–342, 1992.