4.1

Cascaded Shadow Maps

Wolfgang Engel, Rockstar, San Diego

Cascaded shadow maps are useful in a large outdoor environment, where shadows should be visible across huge distances and where changing lighting conditions such as different times of day require dynamic shadows over the whole scene.

Background

Cascaded shadow maps were first mentioned by John Carmack in his QuakeCon 2004 speech. They are utilized in 3DMark06 [3DMark06], and it seems like they are also used in several upcoming games.

The main idea of cascaded shadow maps is to slice the view frustum into multiple slices (usually three to six) and write shadow data for each of these slices into a dedicated shadow map. Figure 4.1.1 shows a view frustum sliced into four pieces.

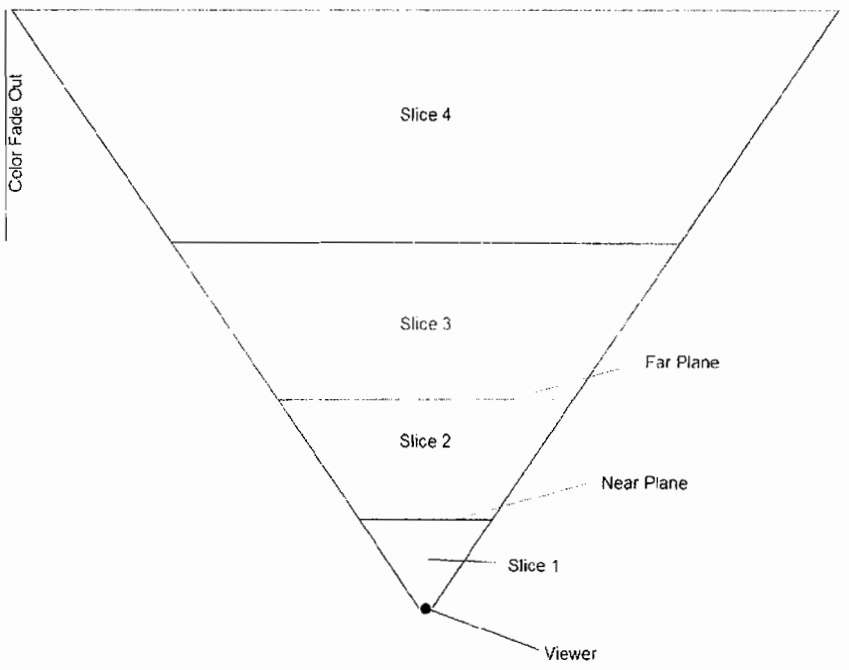

FIGURE 4.1.1 Cascaded shadow maps: sliced view frustum.

The distance between the far plane of each view frustum slice and the far plane of the previous view frustum slice is usually doubled or tripled each time. For example, the first map might go out to 10 meters, the second map up to 30 meters, the third map up to 90 meters, and the fourth map up to 270 meters or more. The exact values depend on the world units, shadow map resolution, and image quality settings. The advantages of using cascaded shadow maps are:

- Large shadow distance
- Scalability
- Reduced dueling frusta problems
- Robustness

Large Shadow Distance

It is possible to cover very large distances with just four or five shadow maps. Because each slice covers a depth range proportional to its distance from the viewer, texels are distributed to give a consistent texel-to-screen pixel ratio. Each additional shadow map triples the distance of the furthest shadows, so relatively few are required to cover a huge view distance.

Scalability

Cascaded shadow maps scale with hardware capabilities very well. Based on the required shadow map distance, the number of view frustum slices/shadow maps can be increased or decreased per level. Additionally, the near and far plane of each slice's frustum, which represents a frustum, can be adjusted during game play. The near and far plane can be moved in for cut scenes or if the viewer approaches a building or an object that is nearer than the first or second shadow map far plane.

The near and far planes can be moved outward if it is not necessary to have sharp shadows close to the viewer. This might be the case in a racing game that uses a tunnel motion blur at speed. Because the tunnel motion blur softens near shadows, shadow map resolution can be dedicated to objects that are farther away than the motion blur effect, and the application can dynamically adjust the shadow resolution along the view frustum or use other algorithms to do so [Zhang].

Dueling Frusta

If the light view frustum is facing the light source, stretched shadow map texels often become visible. This is called *dueling frusta*. Cascaded shadows have numerous texels distributed along the view frustum. In other words the texel-to-pixel ratio decreases more slowly over the view distance, and the cascaded shadows do not suffer as much from this problem as shadow map approaches with a lower number of texels.

Robustness

All the approaches that warp shadows, such as in [Kozlow] and [Wimmer], show huge quality differences between best and worst cases. This quality change can even happen very abruptly. The uniform transformation used for cascaded shadow maps offers a well-balanced quality without noticeable worst-case scenarios.

Implementation

Cascaded shadow maps are mostly like any common shadow map implementation as covered in many introductory texts on shadow maps [Engel]. The major difference is the challenge of dynamically generating efficient light view frusta that capture the view frustum the player sees. Additionally, this chapter will introduce a way to use several large shadow maps in a hardware-cache friendly way and discuss the cascaded-shadow-map-specific challenges for common shadow map problems such as surface acne, soft penumbra, and shadow caster culling.

Constructing a Light View Frustum

Cascaded shadow maps provide a shadow solution for outdoor environments with a visible change in time of day. They partition a scene with several light view frusta and follow the principle of shadow buffer frustum partitioning as described in [Forsyth]. The main improvement is to provide a near-optimal partitioning for large complex scenes and a distant light source with far less computation.

To construct all the light view frusta, the viewer's frustum is first sliced into several pieces. Each of these slices represents a piece of the viewer frustum but at the same time is itself a frustum. To construct such a frustum, the same methods are applicable as for any frustum construction.

Figure 4.1.2 shows a frustum that covers the third slice of the viewer's frustum. It has a near and a far plane that are parallel to the near and far plane of the viewer's frustum and are scaled with a sensitive value to catch shadow casters that are outside of the viewer's frustum on the left and right side of the viewer and a small overlap with the previous slice to simplify the transition phase between the two.

Each of the frusta is then transformed into light space, and the extrema points in light space are used to construct a bounding box. This bounding box represents the orthographic frustum. Figures 4.1.3 and 4.1.4 show how the extrema points of these view frusta are used to create a light view frustum in light space.

The extrema points will be the same from four light directions, because without the y axis, the 2D view should look the same.

Figures 4.1.3 and 4.1.4 also show how using multiple shadow maps focuses on the viewer's frustum more closely than a single one and therefore uses the available texture memory more efficiently. This effect increases with the number of shadow maps used.

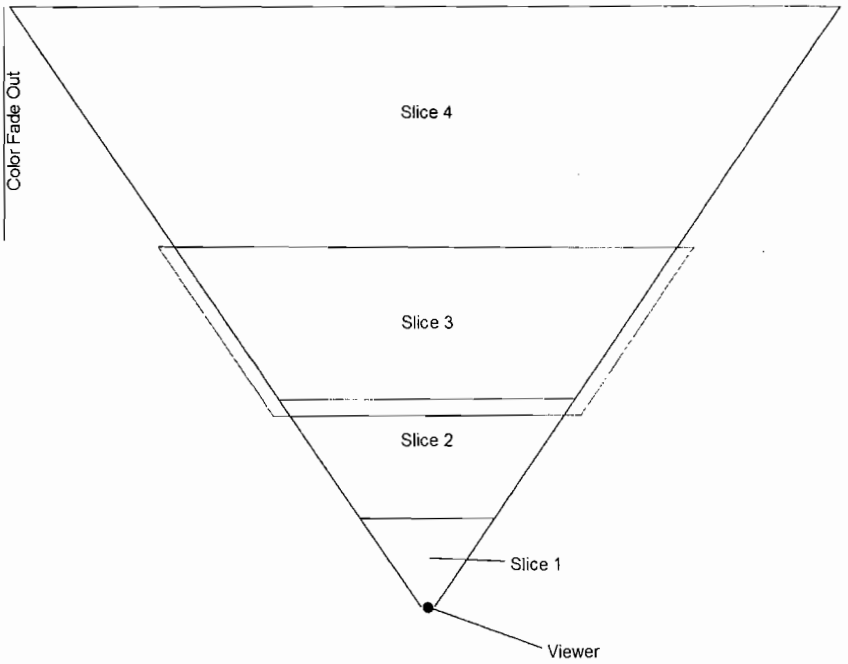

FIGURE 4.1.2 View frustum slice.

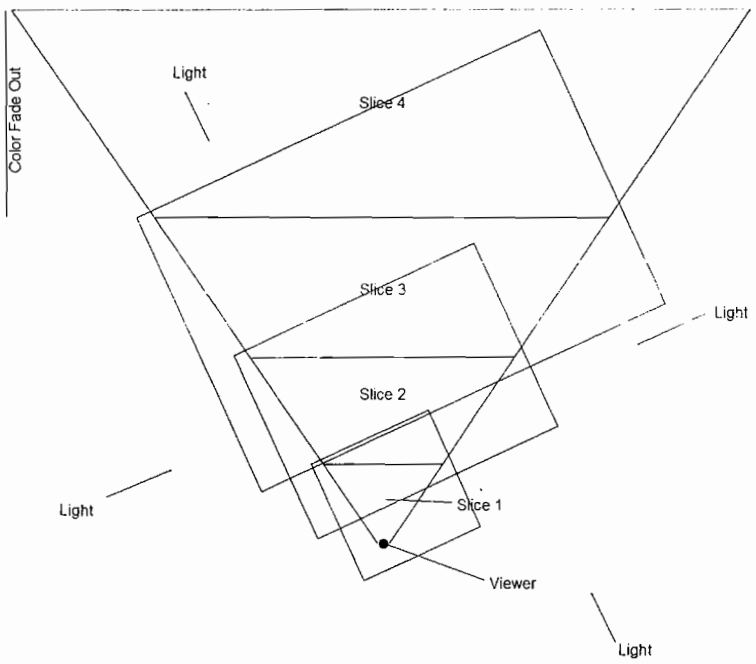

FIGURE 4.1.3 Bounding box around view frusta in light space.

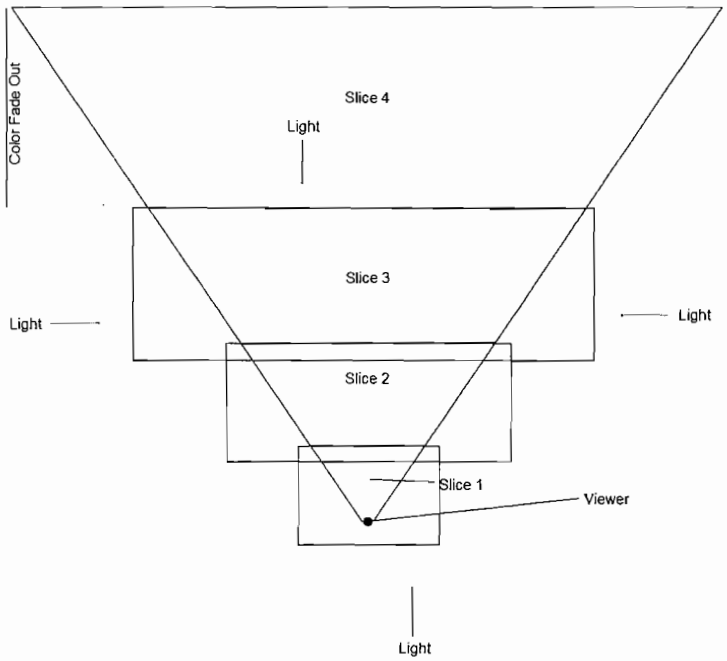

FIGURE 4.1.4 Bounding box around view frusta in light space.

ON THE CD

The code snippet in Listing 1 of the file CascadedShadowMaps.txt on the CD-ROM shows how to construct the frustum for each slice, how it is transformed into light space, and how the view and projection matrices are set up.

Shadow Prepass

Using several large shadow maps can substantially reduce the performance of a game. One reason for this might be that the GPU cannot efficiently cache several large shadow maps being fetched in a random pattern. Shadow map texels are significantly larger than those in most standard texture formats and cannot be compressed well, if at all. If also reading diffuse, normal, and other standard texture maps, the GPU texture cache is often too small to work effectively.

To relieve this problem, the solution proposed here renders all shadow-map-related data in a preprocess (also called deferred shadowing). Like all other shadow map approaches, a light-space pass is done by drawing the depth values from the point of view of the light source into the shadow maps. Depending on the underlying hardware platform, this might be a depth-buffer-only write. The scene is then rendered from the camera's point of view, calculating whether each pixel is in shadow or not, using the shadow maps and writing out the resulting shadow factor into a screen-sized texture. Finally, the scene is rendered again using this texture along with the diffuse, specular, and normal textures performing full lighting.

While the light-space pass renders each of the shadow maps into a simple and fast texture atlas, the subsequent pass sets this atlas once and renders all objects from the viewer's perspective. To pick the right shadow map from the texture atlas, multiple light projection matrices are set, incorporating the correct offsets to address one of the shadows.

A common trick to make shadow maps more cache friendly is to mipmap them. This might be a good option on certain hardware platforms if there are only a small number of shadow maps. Generating mipmaps for a large number of shadow maps can become expensive.

The shadowing screen-space pass is fast, because it uses small pixel shaders that only do the depth comparison and calculate a shadow fade-out value based on camera distance. Additionally, only one texture is set in these pixel shaders for all objects drawn, allowing the entire GPU texture cache to be used for shadow maps.

With an upper limit of four shadow maps, we can use a conditional instruction in the pixel or vertex shader on a variable that keeps the distance of the four shadow maps in each of its channels to choose between the four light matrices without having to use any branches. The code snippet in Listing 2 of the file CascadedShadowMaps.txt on the CD-ROM shows a pixel shader that can be used with four shadow maps and provides a fade-out for the last map. The source snippet with the conditional that picks the right shadow map looks like this:


```
// start distance of 4 shadow maps
float4  zGreater = (StartShadows < camDistance);
float   mapToUse = dot(zGreater, 1.0f );

// Get pixel depth from the point of view from the light camera
float4 pos = mul(float4(WorldSpace.xyz, 1.0f),
LightMatrixArr[mapToUse - 1]);
```

One disadvantage of a shadow prepass is that tweaking the texture coordinates of shadow map fetches while rendering into the frame buffer is not possible. Thus, shadows that follow a parallax map are not as easily achieved as with the common approach.

Surface Acne

Maybe the biggest challenge with any shadow map approach is *depth aliasing* or *surface acne*. Sampling depth values happens in a stair-stepped manner because of the nature of sampling [Schueler][Schueler2] as shown in Figure 4.1.5.

Because the continuous surface of a polygon is compared to the sampled surfaces, a random point of the polygon might be behind the sample surface, and therefore a false shadow is applied. This effect is emphasized on curved surfaces.

Increasing the shadow resolution along the view frustum makes depth-aliasing artifacts look smaller, owing to the higher sampling density, but it does not remove them.

shadow casters that are not visible but cast shadows into the scene and should therefore be considered by the shadow map. Therefore, in most cases more objects will be rendered into the shadow maps than into the frame-buffer. A culling system that culls away objects efficiently for each of the shadow map light frusta is therefore essential (see [Picco] for a good overview of low-level culling mechanisms).

A good approach is to use a dynamic system that is based on an octree as a hierarchy culling method. Breaking down the world into a tree-like structure offers the ability to cull a top level node without culling lower level nodes since they cannot be visible anyway. On a lower level a sphere that surrounds the light frustum can be culled against the bounding sphere of the objects, maybe followed by the cone that surrounds the light frustum culled against the bounding sphere of objects, and then maybe the frustum itself can be culled against the bounding sphere.

An even more efficient approach might use the fact that the position of the sun is known through the time of the day to precalculate visibility. If the viewer always stays on the ground, this would open up even more opportunities to precalculate the visibility of objects.

A good culling system might even determine how many shadow maps need to be used and prevent rendering into shadow maps that would store object shadows that are not visible.

Further Considerations

A shadow prepass can be considered as a Z-prepass or even a first stage of a preprocessing pipeline that is used to achieve different depth buffer effects. Seeing the shadow prepass in the wider perspective of the preprocessing pipeline amortizes some of its cost.

Any perspective [Wimmer] or warped shadow map approach can be cascaded as long as it is stable enough to cover a full time-of-day circle with an acceptable level of depth aliasing. Interesting future approaches that utilize the cascaded shadow map idea are based on a logarithmic distribution of shadow map resolution [Lloyd].

Conclusion

This article covers a proven approach to cascade shadows around the viewer frustum with a simple uniform distribution of shadow map resolution. The main advantage of this approach is that it is very generic compared to other shadow map approaches that offer shadows at large viewing distances. The distribution of quality is quite even, and there are no obvious worst-case scenarios. Additionally, the scalability makes cascaded shadow maps a perfect solution for different scenarios in a game. Adding key-frame support to the parameters that control the view frustum construction allows the game to distribute shadow quality based on the viewer being in a certain area or the existence of occluding objects. Paired with a capable occlusion culling system, the performance will scale well with the view distance and the quality of the shadows.

Acknowledgments

I thank Tom Forsyth and Stefan Krause for reviewing the article and John Carmack for inspiring the technique. I am thankful to Alan Wasserman, Steve Reed, Derek Tarvin, David Etherton, and Eugene Foss for allowing me to write this article. A big thank you goes out to Ron O'Hara for help with the view frustum code. Additionally, I thank Mark Robinson, Raymund Kerr, Ben Padget, Christina Coffin, Alexander Ehrath, and Michael Krehan for discussing with me many aspects of the featured method and widening my view to see the various aspects of the approach.

References

[3DMark06] Futuremark Corporation. "3DMark06 Whitepaper v1.0.2." Available online at *http://www.futuremark.com/companyinfo/pressroom/companypdfs/3DMark06_Whitepaper_v1_0_2.pdf?m=v.*

[Engel] Engel, Wolfgang. *Programming Vertex and Pixel Shaders.* Charles River Media, 2004: pp. 287–301

[Forsyth] Forsyth, Tom. "Shadowbuffer Frustum Partitioning." *ShaderX*^{4}, edited by Wolfgang Engel. Charles River Media, 2006: pp. 289–297.

[Kozlow] Kozlow, Simon. "Perspective Shadow Maps: Care and Feeding." *GPU Gems.* Addison-Wesley, 2004: pp. 217–244.

[Lloyd] Lloyd, Brandon, David Tuft, Sung-Eui Yoon, and Dinesh Manocha. "Warping and Partitioning for Low Error Shadow Maps." Available online at *http://gamma.cs.unc.edu/wnp/.*

[Mitchell] Mitchell, Jason. "Poisson Shadow Blur." *Shader X*^{3}, edited by Wolfgang Engel. Charles River Media, 2005: pp. 403–409.

[Picco] Picco, Dion. "Frustum Culling." Available online at *http://www.flipcode.com/articles/article_frustumculling.shtml*

[Schueler] Schueler, Christian. "Eliminating Surface Acne with Gradient Shadow Maps." *ShaderX*^{4}, edited by Wolfgang Engel. Charles River Media, 2006: pp. 15–23

[Schueler2] Schueler, Christian. "Multisampling Extension for Gradient Shadow Maps." *ShaderX*^{5}, edited by Wolfgang Engel. Charles River Media, 2007: pp. 207–218.

[Reeves] Reeves, William, David Salesin, and Robert Cook. "Rendering Antialiased Shadows with Depth Maps." *SIGGRAPH 1987*, pp. 283–291.

[Williams] Williams, Lance. "Casting Curved Shadows on Curved Surfaces." *Computer Graphics*, *23*(3) (1978): pp. 270–274.

[Wimmer] Wimmer, Michael and Daniel Scherzer. "Robust Shadow Mapping with Light-Space Perspective Shadow Maps." *ShaderX*^{4}, edited by Wolfgang Engel. Charles River Media, 2006: pp. 313–330.

[Zhang] Zhang, Fan, Hanqiu Sun, Leilei Xu, and Lee Kit Lun. "Parallel-Split Shadow Maps for Large-scale Virtual Environments." Available online at *http://appsrv.cse.cuhk.edu.hk/~fzhang/pssm_vrcia/.*

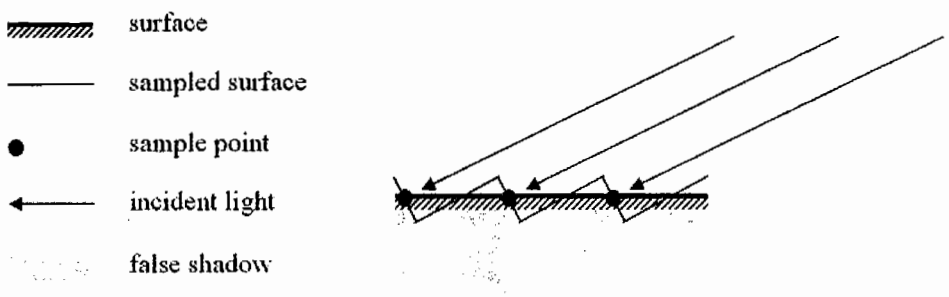

FIGURE 4.1.5 Surface acne. (Courtesy of Christian Schueler.)

In the case of the popular perspective shadow map approaches, depth-aliasing artifacts might be emphasized owing to the perspective distortion, because some parts of the scene will receive less shadow resolution than others.

This leaves us with the common ways to fight surface acne. If the shadow data are rendered into depth buffers, all decent hardware offers two render states that are called depth-slope scale bias and depth bias. In Direct3D they are called `D3DRS_DEPTHBIAS` and `D3DRS_SLOPESCALEDEPTHBIAS`, and in OpenGL there is a function called `glPolygonOffset()`. Both approaches use the following formula to decrease depth aliasing.

```
m * D3DRS_SLOPESCALEDEPTHBIAS + D3DRS_DEPTHBIAS
```

where $m = \max(| \partial z/\partial x | , | \partial z/\partial y |)$.

This offset is added before the depth test and before depth values are written into the depth buffer, so it only works with the depth buffer.

Kozlow shows how to calculate a depth bias value in world space based on light direction and pixel size [Kozlow].

In case these render states are not available or if the scene still shows too much depth aliasing, an approach called gradient shadow maps can be utilized [Schueler][Schueler2]. Instead of using a constant depth-bias value, Schueler scales the depth-bias value with a gradient value and adds a small constant on top of the gradient scaled bias to account for cases in which the gradient becomes zero and clamps this value at grazing angles. This clamping is chosen similarly to the depth bias value to fit the whole scene.

For a shadow map approach that offers support for shadows at large distances, any form of constant bias value will be valid only for certain distances but not for all. Setting those values separately for each of the cascaded shadow maps offers a better level of control over distance.

Soft Penumbra

The penumbra of a shadow is in real-life quite often softer with increasing distance between the shadow caster and the shadow receiver. This is due to global illumination effects affecting the shadow. In computer games the shadows are commonly softened by using a technique called percentage closer filtering (PCF) [Reeves]. This technique averages the binary result from, for example, a 3 × 3 block of texels as shown in Figure 4.1.6.

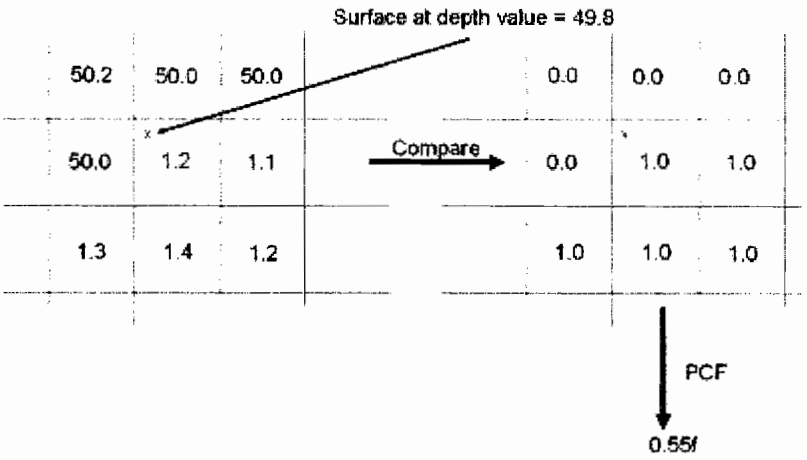

FIGURE 4.1.6 Percentage closer filtering.

Cascaded shadow maps, like any shadow map approach that tries to distribute the shadow map resolution evenly along the view frustum, need at least a visible shadow shape on distant objects that is perceived as a believable shadow. Large filter kernels with a fixed size and based on PCF or even more advanced filters based on a Poisson disk distribution of sampling points [Mitchell] might influence or destroy the shadow shape on distant objects owing to the low density of shadow samples. There are several strategies to cope with this problem. Using just a simple four-tap PCF filter on all shadow maps works very well in practice. Using an adjustable filter kernel over the whole distance should show better results but increase the cost.

Culling Shadow Casters and Other Objects

The number of objects that get rendered into one or more shadow maps is an index of the performance of any shadow map approach. Assuming a shadow map approach with a large shadow distance, the light view frusta for each of the shadow maps may cover many more objects than are visible to the viewer. Additionally, there will be