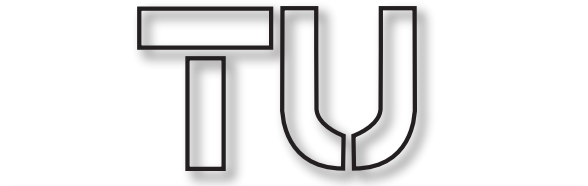


Dissertation

Applications of Temporal Coherence in Real-Time Rendering

ausgeführt
zum Zwecke der Erlangung des akademischen Grades
eines Doktors der technischen Wissenschaften

unter der Leitung von
Associate Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer,
Institut für Computergraphik und Algorithmen E186,
eingereicht
an der Technischen Universität Wien,
Fakultät für Technische Naturwissenschaften und Informatik,

von
Dipl.Ing. Mag.rer.soc.oec. Daniel Scherzer,
Matrikelnummer 9727072,
Sonnenuhrgasse 1/14,
A-1060 Wien, Österreich,
geboren am 4. September 1978 in München

Wien, 12. Januar 2010.

Applications of Temporal Coherence in Real-Time Rendering

Daniel Scherzer

Abstract

Real-time rendering imposes the challenging task of creating a new rendering of an input scene at least 60 times a second. Although computer graphics hardware has made staggering advances in terms of speed and freedom of programmability, there still exist a number of algorithms that are too expensive to be calculated in this time budget, like exact shadows or an exact global illumination solution. One way to circumvent this hard time limit is to capitalize on *temporal coherence* to formulate algorithms incremental in time.

> The main thesis of this work is that *temporal coherence* is a characteristic of real-time graphics that can be used to redesign well-known rendering methods to become faster, while exhibiting better visual fidelity.

To this end we present our adaptations of algorithms from the fields of exact hard shadows, physically correct soft shadows and fast discrete LOD blending, in which we have successfully incorporated *temporal coherence*. Additionally, we provide a detailed context of previous work not only in the field of *temporal coherence*, but also in the respective fields of the presented algorithms.

Kurzfassung

In der Echtzeitgraphik gilt es eine gegebene Szene mindestens 60 mal in der Sekunde darzustellen. Obwohl die Computergraphik-Hardware in den letzten Jahren unglaubliche Fortschritte in Bezug auf Geschwindigkeit und Programmierbarkeit vorzuweisen hat, existiert dennoch eine Reihe von Algorithmen, die zu zeitaufwändig sind, um in solch einem kurzen Zeitraum berechnet werden zu können. Als Beispiel seien an dieser Stelle nur exakte Schattenberechnungen oder eine korrekte globale Beleuchtungslösung genannt. Eine Möglichkeit, dieses Zeitlimit zu umgehen, ist die den meisten Animationen inhärente *zeitliche Kohärenz* zu nutzen, um Algorithmen inkrementell in der Zeit zu reformulieren.

> Die Hauptthese dieser Arbeit ist, dass zeitliche Kohärenz in der Echtzeitgraphik dazu verwendet werden kann, um bekannte Darstellungsalgorithmen zu beschleunigen und mit besserer visueller Qualität auszustatten.

Zu diesem Zweck präsentieren wir neue Algorithmen aus den Bereichen pixelgenaue harte Schatten, physikalisch korrekte weiche Schatten und eine neue Methode für schnelles Überblenden von diskreten LOD-Stufen, die alle erfolgreich zeitliche Kohärenz einsetzen. Darüber hinaus bieten wir eine detaillierte Übersicht über verwandte Arbeiten aus den Bereichen der zeitlichen Kohärenz, harte sowie weiche Schatten-Berechnungen.

Contents

1 Introduction

Computer graphics is concerned with the rendering of synthetic images. One of the driving forces of computer graphics is to become more and more physically correct. This means that more complex properties of the interaction of light and environment are taken into account when rendering. Therefore the used algorithms also have become more complex and more computationally demanding.

Real-time rendering has the conflicting goal of creating a sequence of such images fast enough to still allow for continuous animation and user interaction. Here a limit of at least 60 frames per second is considered as sufficiently smooth for the human observer, which is not coincidentally also the refresh rate used in most monitors. This means the time available for one frame is about 16 milliseconds. All calculations necessary to create a frame have to fit into this time budget. This not only includes all the rendering algorithms we are concerned with in computer graphics, but may also contain the domain specific code of an application, artificial intelligence, input processing and sound rendering.

For our purpose the execution of an algorithm can be described as the evaluation of a function f on an input set $\mathbb{I}$ onto an output set $\mathbb{O}$: $f : \mathbb{I} \to \mathbb{O}$. The result of the algorithm on a concrete input data set $\boldsymbol{i} \in \mathbb{I}$ is written as $\boldsymbol{o} := f(\boldsymbol{i}) \in \mathbb{O}$.

Example 1.a: *Let f be the application of the shadow mapping algorithm [Wil78] to a scene, an image space algorithm. In this case an input $\boldsymbol{i} \in \mathbb{I}$ consists of a scene description $\boldsymbol{s} \in \mathbb{S}$, a light source $\boldsymbol{l} \in \mathbb{L}$ and camera $\boldsymbol{c} \in \mathbb{C}$, and probably some parameters like shadow map resolution and targeted frame buffer size. So $\mathbb{I} = \mathbb{S} \times \mathbb{L} \times \mathbb{C} \times \mathbb{N}^2 \times ....$ The output set $\mathbb{O}$ in this case is the set of images and $\boldsymbol{o} \in \mathbb{O}$ is one rendered image of the input scene with shadowing information stored for every pixel.*

For each frame $\boldsymbol{i}$ will be different. Consider for instance a moving camera or a dynamic environment. We denote this as $\boldsymbol{i}_k$, were $k \in \mathbb{N}$ is the frame

number. Therefore the output $\boldsymbol{o}_k := f(\boldsymbol{i}_k)$ will also have to be evaluated each frame.

If we change some of the parameters mentioned in Example 1.a, like use a bigger target frame buffer size or make the scene description more complex to create a more realistic looking image, the computation of $f(\boldsymbol{i}_k)$ will take more time. This is exactly the conflict mentioned above: One the one hand we want to create more realistic images, but on the other hand we have to adhere to a strict time budget in creating these images.

1.1 Motivation

One way out of this dilemma is to modify algorithms to be incremental in time: Results $\boldsymbol{o}_k$ from previous frames can be reused to speed up or increase correctness of calculations for the current frame $\boldsymbol{o}_{cur}$. If we use results from the previous n frames we can write the incremental formulation of a given algorithm f_{inc} as

$$f_{inc} : \mathbb{I} \times \underbrace{\mathbb{O} \times \cdots \times \mathbb{O}}_{n-times} \to \mathbb{O}. \tag{1.1}$$

The benefit in speed or quality gained here depends upon the relevance of prior results $\boldsymbol{o}_{cur-n}, \cdots, \boldsymbol{o}_{cur-1}$, onto $f_{inc}(\boldsymbol{i}_{cur}, \boldsymbol{o}_{cur-n}, \cdots, \boldsymbol{o}_{cur-1})$. This is called *temporal coherence*. If we assume that f_{inc} is stable – small changes[1] in $\boldsymbol{i}_k$ produce only small changes in the output $\boldsymbol{o}_k$ – we can assume that the prior results become more relevant when the current input $\boldsymbol{i}_{cur}$ is similar to all the inputs used to calculate $\boldsymbol{o}_{cur-n}, \cdots, \boldsymbol{o}_{cur-1}$, namely $\boldsymbol{i}_{cur-n}, \cdots, \boldsymbol{i}_{cur-1}$. Continuous animation and very small n are likely to validate this condition.

Please note that also a very different $\boldsymbol{i}_{cur-n}, \cdots, \boldsymbol{i}_{cur-1}$ can have the same result when applying f_{inc}. Reconsider for instance Example 1.a: If the differences in the $\boldsymbol{i_k}$'s only concern parts of the scene that are not visible to the light source and the camera, then the difference will have no effect on the outcome of the algorithm and we of course would also like to detect these cases.

Example 1.b: *Reconsider Example 1.a in a static scene with a moving first person camera. Shadows are calculated for a fixed light source. This is a rather common setting for games, especially first person shooters. Each frame the camera moves a small fraction, so the $\boldsymbol{i}_k$ stay exactly the same except for the part that stores the camera parameterization. Also the $\boldsymbol{o}_k$ are similar (see Figure 1.1): Most of what was visible in the previous frame*

[1] Of course we still have to define the meaning of "small" in this context.

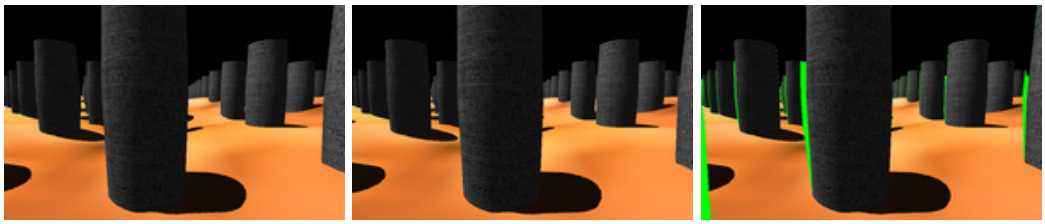


Figure 1.1: *On the* left *two consecutive frames from a strafe-left motion are shown. Please not the strong frame-to-frame coherence. The* right *image shows the incoherent parts (2% – mainly due to dissocclusion) marked in green.*

will still be visible. No shadow caster/receiver moves, so shadows should also stay the same over the course of the camera movement. All in all the images produced are rasterizations of slightly different coordinate spaces, due to camera movement. This means we can reuse pixels with stored shadowing information from previous frames by reprojecting – and thereby accounting for camera movement – them into the current frame.

As can be seen in Figure 1.1, temporal coherence between frames is high and therefore 98% of the previous frame can potentially be reused. So in theory only 2% of the calculations for the current frame have to be performed, which can mean huge savings in processing time.

Example 1.c: *An example for object-space temporal coherence is the following acceleration algorithm for ray-tracing: A predefined animation with fixed camera (so only objects move) is given. As a preproseesing step the scene is subdivided into an object-space voxel grid. For each frame of the animation each voxel is checked for changes. A change is when an object moves inside, into or out of a voxel. For each ray it is determined through which voxels it passes. During the calculation of the individual frames, only those rays that pass through changing voxels need to be recalculated. All other rays can be reused.*

1.2 Goals and Challenges

Temporal coherence can be applied with three main goals in mind:

- *Speed up*: A given algorithm can be accelerated by redesigning it to be incremental in time, thereby distributing the total workload over several frames.

- *Increase in quality*: The results of a given algorithm can be augmented by taking into account calculations done in previous frames.

- *Reducing temporal aliasing*: When rendering frames, for each frame independent rasterizations are produced. This causes temporal aliasing and can result in strong flickering artifacts. Temporal coherence can be applied to avoid this by introducing knowledge of previous rasterizations into to calculations for the current one, thereby allowing for temporal smoothing by dissallowing sudden changes in coherent regions.

These goals have in common that for a change in $\boldsymbol{i}_{cur}$ some latency in the output may be introduced, thereby forcing an algorithm to converge to a stable solution over a number of frames. This means that over several frames only part of the solution is visible to the observer. On the one hand it is desirable that the convergence is unnoticeable or at least smooth and not disturbing to the beholder. On the other hand the convergence should be as quick as possible to avoid visible artifacts. Ongoing animation also causes information from previous frames to be outdated. This has to be accounted for to avoid temporal artifacts, like after-images or tailing.

Aside from the fact that the redesign of algorithms to account for temporal coherence can be challenging, special care has to be taken to fit these algorithms to the massively parallel nature of modern graphics architectures: For instance in Figure 1.1 the incoherent parts (marked in green) are disjoint and distributed all over the output image. To detect and render only these parts, modern hardware may take the same amount of time than for rendering the whole image.

1.3 Dissertation Thesis

This work focuses on rendering algorithms using temporal coherence in image space. The resulting algorithms are very fast and executed completely on a state-of-the-art GPU, thereby freeing the main processor for other tasks.

> The main thesis of this work is that temporal coherence is a characteristic of real-time graphics that can be used to redesign well-known rendering methods to become faster, while exhibiting better visual fidelity.

We will underline this statement by providing detailed discussions of successful adaptations in the remainder of this thesis.

1.4 Contributions

We target three very specific problem sets: pixel correct hard shadows, physically correct soft shadows and faster and more robust discrete LOD blending. In each of this areas we present new or improved algorithms.

Pixel correct hard shadows: Shadow mapping suffers from spatial aliasing (visible as blocky shadows) as well as temporal aliasing (visible as flickering). Several methods have already been proposed for reducing such artifacts, but so far none is able to provide satisfying results in real time.

We extend shadow mapping by reusing information of previously rasterized images, stored efficiently in a so-called history buffer. This buffer is updated in every frame and then used for the shadow calculation. In combination with a special confidence-based method for the history buffer update (based on the current shadow map), temporal and spatial aliasing can be completely removed. The algorithm converges in about 10 to 60 frames and during convergence, shadow borders are sharpened over time. Consequently, in case of real-time frame rates, the temporal shadow adaption is practically imperceptible. The method is simple to implement and is as fast as uniform shadow mapping, incurring only the minor speed hit of the history buffer update. It works together with advanced filtering methods like percentage closer filtering and more advanced shadow mapping techniques like perspective or light space perspective shadow maps. The results [SJW07] have been published in:

- Daniel Scherzer, Stefan Jeschke, and Michael Wimmer. **Pixel-correct shadow maps with temporal reprojection and shadow test confidence.** In Jan Kautz and Sumanta Pattanaik, editors, *Rendering Techniques 2007 (Proceedings Eurographics Symposium on Rendering)*, pages 45-50. Eurographics, Eurographics Association, June 2007.

Physically correct soft shadows: A vast amount of soft shadow map algorithms have been presented in recent years. Most use a single sample hard shadow map together with some clever filtering technique to calculate perceptually or even physically plausible soft shadows.

On the other hand there is the class of much slower algorithms that calculate physically correct soft shadows by taking and combining many samples of the light.

We present a new soft shadow method that combines the benefits of these approaches. It samples the light source over multiple frames instead of a single frame, creating only a single shadow map each frame. Where temporal

coherence is low we use spatial filtering to estimate additional samples to create correct and very fast soft shadows. The results [SSMW09] will be published in:

- Daniel Scherzer, Michael Schwärzler, Oliver Mattausch, and Michael Wimmer. **Real-time soft shadows using temporal coherence.** *Lecture Notes in Computer Science 2009 (LNCS)*, November 2009.

Discrete LOD blending: We present a method for automatic interpolation between adjacent discrete levels of detail to achieve smooth LOD changes in image space. We achieve this by breaking the problem into two passes: We render the two LOD levels individually and combine them in a separate pass afterwards. The interpolation is formulated in a way that only one level has to be updated per frame and the other can be reused from the previous frame, thereby causing roughly the same render cost as with simple non interpolated discrete LOD rendering, only incurring the slight overhead of the final combination pass. Additionally we describe customized interpolation schemes using *visibility textures*.

The method was designed with the ease of integration into existing engines in mind. It requires neither sorting nor blending of objects, nor does it introduce any constrains in the LOD used. The LODs can be coplanar, alpha masked, animated, impostors, and intersecting, while still interpolating smoothly. The results have been published in [SW08]:

- Daniel Scherzer and Michael Wimmer. **Frame sequential interpolation for discrete level-of-detail rendering.** *Computer Graphics Forum (Proceedings EGSR 2008)*, 27(4):1175-1181, June 2008.

1.5 Overview

The remainder of this work is structured as follows: Chapter 2 gives an overview of the related work of this thesis. This comprises papers from the areas of temporal coherence, shadow- and soft shadow mapping, as well as LOD techniques. Chapter 3 presents our approach to pixel correct hard shadows. Here we use temporal coherence to reuse shadowing solutions from previous frames to increase the quality of the current frame. In Chapter 4 we extend the ideas described in Chapter 3 to the rendering of physically correct soft shadows by presenting a light source area sampling method that works in real-time. Our discrete LOD blending algorithm is discussed in detail in Chapter 5. Finally, in Chapter 6 we investigate the validity of our main thesis on the presented algorithms, thereby summarizing our findings.

2
Related Work

This chapter gives an overview of the literature of the areas touched by our research: We start by investigating methods using temporal coherence in Section 2.1. Section 2.2 outlines related work in hard shadow mapping, while Section 2.3 focuses on work in the area of soft shadow mapping. Finally Section 2.4 gives a brief overview of LOD techniques relevant to our work.

2.1 Temporal Coherence

The term *frame-to-frame coherence* was first introduced by Sutherland et al. [SSS74] in his seminal paper "Characterization of Ten Hidden-Surface Algorithms", in which he describes various versions of coherence, like scan-line or area coherence that allow for more efficient rendering.

In an early paper by Hubschman and Zucker [HZ81], frame-to-frame coherence is investigated in animation sequences for static scenes and a continuously moving camera in scan-line rendering. They derive a number of geometrical constrains for the case of closed, convex and non-intersecting polyhedra. With these they predict changes in visibility and by this accelerate visibility detection, by only processing the parts of the scene where a visibility change can occur. Coorg and Teller [CT96] build on this work. They introduce a data structure that helps to predict imminent visual events. Temporal coherence allows them to maintain this data structure dynamically, only containing the data relevant for the current view port.

2.1.1 Ray-tracing

Especially in ray-tracing, a number of algorithms exist that make use of information stored over time:

Badt [BJ88] introduced reprojection to accelerate ray-tracing for animations. His forward reprojection algorithm uses object space information stored

from the previous frame. This allows him to approximate ray-traced animation frames of diffuse polygons. Adelson and Hodges [AH95] later extend his approach to ray-tracing of arbitrary scenes.

Adelson and Hodges [AH92] also use frame-to-frame coherence not in the temporal dimension, but in-between the two images of stereoscopic views when ray-tracing. They found that up to 95% of the pixels of the left-eye view can be reused for the right-eye view by reprojection.

Jevans [Jev92] and Davis [Dav98, DD99] use temporal coherence in ray-tracing in similar ways: They divide the scene into an object-space grid. All voxels where changes occur (objects move in/out/around) during a predefined animation are identified. Rays that pass through these changing voxels have to be recalculated in all the frames where these changes occur. While Jevans' algorithm is formulated in a serial manner, Davis' algorithm accelerates parallel ray-tracing on a distributed system by rendering different sub-regions of the output image on different nodes and employing temporal coherence locally on each node.

Havran et al. [HBS03] use temporal coherence to reuse ray/object intersections in ray casted walkthroughs. They do this by reprojecting and splatting visible point samples from the last frame into the current, thereby avoiding the costly ray traversal for more than 78% of the pixels in their test scenes.

2.1.2 Image-based Rendering

Many publications focus on using temporal coherence for replacing parts of a scene with image-based representations: Gröller [Grö92] and Schaufler [Sch96] reuse image data generated from previous frames. Complex distant geometry is replaced by impostors.

Lengyel and Snyder [LS97] employ frame-to-frame coherence to guide factorization of a scene into layers. Image warping is used to rerender layers over multiple frames (see Figure 2.1). Differences in perception of fore-/background objects, as well as differences in the motion of objects are factors to redistribute rendering resources adaptively. More rendering resources are spent for fast moving foreground objects, which thereby gain in fidelity, while a slowly changing background receives less rendering resources. They targeted their implementation on a hardware termed Talisman [TK96] that supports rendering of layers natively.

Reagan and Pose [RP94] propose an address recalculation hardware for head mounted displays. This hardware allows for orientation viewport mapping after rendering, which minimizes latency caused by head rotations. With this hardware they implement priority rendering: The scene is divided into layers with increasing distance to the eye. Layers that are farther away are

Figure 2.1: *Frame-to-frame coherence can be used to aid in factorizing a scene into coherent layers.Image courtesy of [LS97]..*

updated less often than nearer layers.

Bishop et al. [BFMZ94] introduce frameless rendering. This approach minimizes lag by rendering each pixel independently based on the most recent input. Pixels stay visible for a random time-span. To avoid image tearing pixels are rendered in a random order. This approach dose not use the object coherency so important for many polygon renderers, but relys on temporal coherence to achieve sensible output.

2.1.3 Image Warping

Another class of approaches that use temporal coherence is image warping. Here images are used as a cache to be reused and warped into different views.

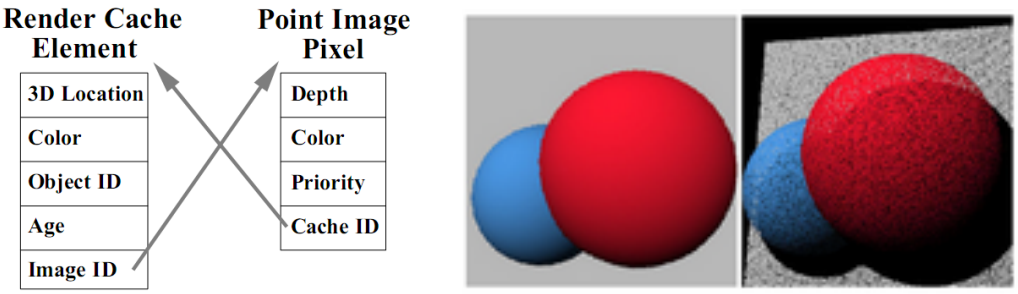
Chen and Williams [CW93] calculate in-between views by morphing a number of reference images. Similarly McMillan and Bishop [MB95] use an image warping approach for the creation of new views from reference images. This technique is used for the generation of arbitrary views for stereoscopic displays with only two hyper-stereo reference images as input. Mark et al. [MMB97] build on McMillan and Bishop's image warping algorithm to render a new view from two stored views. For each view color and depth are stored. Both images are warped into the new view to allow for small camera movements. Then the two images are composited together to compensate for

most dissocclusions. The authors report an increase in apparent rendering speed of about one order-of-magnitude to regular rendering.

Simmons and Sequin [SS00] introduce a mesh-based reconstruction called a tapestry for dynamically sampled environments. It allows the reuse of radiance values across views by reprojecting them into the new view.

Stamminger et al. [SHSS00] augment interactive walkthroughs (calculated by rendering hardware) with photorealistic results calculated by an asynchronous process. This has the advantage that interactivity of the walkthrough is not hindered by the processing time of the high quality calculations. The differences between high quality and interactive results are stored in so-called *corrective textures* and applied to the scene objects with projective texturing. This has the advantage that the temporal coherence of such scenes allows for lazy updates of *corrective textures*. Additionally the texture resolution can be adapted to the number of available samples and hardware texture filtering can be used to resolve under- as well as oversampling.

Wimmer et al. [WGS99] accelerate the rendering of complex environments by using two different rendering methods for the near and the far field: The near field is rendered using the traditional rendering pipeline, while ray casting is used for the far field. To minimize the number of rays cast, they use a panoramic radiance cache and horizon estimation.



(a) Left: *data elements of one render cache element;* right: *data stored for each pixel in the point image (reprojected image)*

(b) *Reprojection between viewpoints:* Left: *original viewpoint and* right: *resulting viewpoint*

Figure 2.2: *The* render cache *introduced by Walter et al. is intended as a general acceleration structure for arbitrary non-interactive renderers. Image courtesy of [WDP99].*

Walter et al. [WDP99] introduce the *render cache*. It is intended as an acceleration data structure for renderers that are too slow for interactive use. The *render cache* is a point based structure, which stores previous results, namely 3d coordinates and shading information. By using reprojection, image

space sparse sampling heuristics and by exploiting spatio-temporal image coherence these results can be reused in the current frame (see Figure 2.2). Progressive refinement allows decoupling the rendering and display frame rates, enabling high interactivity. Walter et al. [WDG02] later extend this approach with predictive sampling and interpolation filters. Finally Velázquez-Armendáriz et al. [VALBW06] and Zhu et al. [ZWL05] accelerate the *render cache* on the GPU.

Smyk et al. [SKDM05] explore temporal coherence to speed up irradiance calculations using a cache structure called anchor. The idea is to permanently store and update (if needed) all the incoming radiance samples used to estimate the irradiance of an irradiance record. Thereby they not only accelerate the process, but also reduce temporal artifacts, like flickering in methods that render each frame independently.

In a related approach, Gautron et al. [Gau08] present a temporal caching scheme for glossy global illumination: *temporal radiance caching* of animated environments (camera, objects and light sources move). They reuse part of the global illumination solution of previous frames by introducing temporal gradients, which estimate contribution of a record within its lifespan.

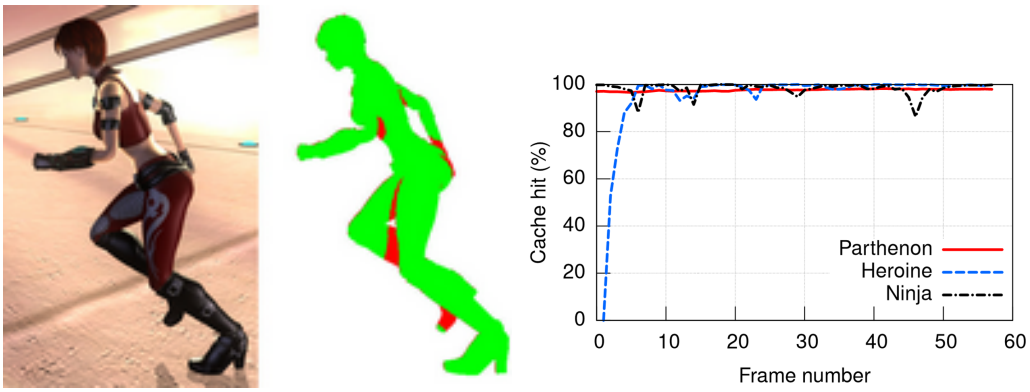


Figure 2.3: *Images and statistical data of the animation sequence "Heroine" used by Nehab et al. in their paper;* Left: *shaded model;* middle: *visualization of the frame-to-frame coherence;* right: *coherence statistics (cache hits); Image courtesy of [NSL+07].*

Similar to Walters' render cache idea is Nehab et al.'s [NSI06, NSL+07] so-called *reprojection cache*, which is introduced as a way to accelerate real-time pixel shading in hardware raterization renderers (see Figure 2.3). The main difference to the *render cache* is that the *reprojection cache* does not contain points but visible pixels in screen space (with additional data like depth). This is a very hardware friendly approach as this cache is just a viewport-sized off-screen buffer and can therefore reside in graphics memory

without causing traffic between GPU and CPU. Another difference that fits perfectly to hardware is that this method uses reverse reprojection and therefore can use hardware texture filtering capabilities for sample retrieval. For distinguishing between a cache hit and miss when reprojecting current fragments into the cache, the depth is used. If a cache value has a depth equal ($\pm\epsilon$) to the current fragment's depth, a cache hit is assumed. The authors accelerated several common algorithms using their temporal caching method, including precomputed global lighting effects, shadow mapping, stereoscopic rendering, motion blur and depth of field.

The *reprojection cache* approach described in this paper is similar to our concurrent work [SJW07], where we specialized in improving shadow quality (see Chapter 3 for details). Yang et al. [YNS+09] use a related approach for amortizing supersampling: They maintain several samples from previous frames and combine them in the current frame using reprojection. In the majority of cases they can thereby avoid the computational cost of calculating multiple samples for each fragment.

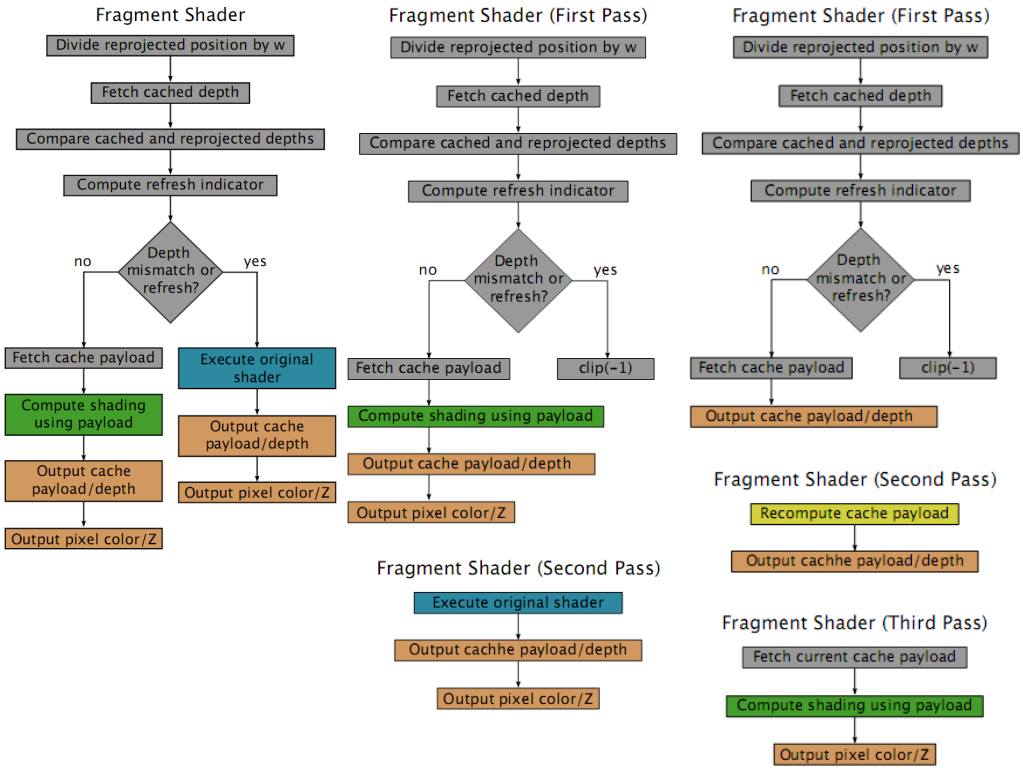


Figure 2.4: *3 algorithms for implementing temporal reprojection in the fragment shader: 1-pass* (left)*, 2-pass* (middle) *and the faster 3-pass* (right)*. Image courtesy of [SaLY+08a].*

Sitthi-Amorn et al. [SaLY+08a] analyse the potential performance gain achievable with Nehab et al.'s respective Scherzer et al.'s approach. They find that a 3-pass algorithm (in contrast to the single pass or 2-pass algorithms used before) is more efficient to execute on current hardware (see Figure 2.4), because it:

- does not need MRTs: The authors found that for pixel bound scenes MRTs cause more overhead than rendering in multiple passes.
- it produces a more coherent branching behavior: The two subbranches of the presented 3-pass algorithm have nearly equal execution time.

The problem when and how to update the *reprojection cache* (refresh policy) is investigated in [SaLY+08b]. They present automatic methods to select when and which samples to refresh using a parametric model that describes the way possible caching decisions affect the visual fidelity and the shader's performance. They train their model by using sample rendering sessions and an abstract syntax tree (AST) representation of the pixel shader. The use of an interactive profiler allows user control. The user can explore possible accuracy/speed trade-offs and choose the most suitable.

2.2 Shadow Mapping

One of the application domains where we employ temporal coherence is shadow mapping [Wil78]. This is a real-time approach for calculating shadows, which can handle arbitrary shadow caster and receiver constellations. Here the shadow computation is performed in two passes: first, a depth image of the current scene (the *shadow map*) as seen from the light source (in *light space*) is rendered and stored. Second, the scene is rendered from the viewpoint (in *view space*) and each 3D fragment is reprojected into the light space. If the reprojected fragment depth is farther away that the depth stored in the shadow map, the fragment is in shadow and shaded accordingly.

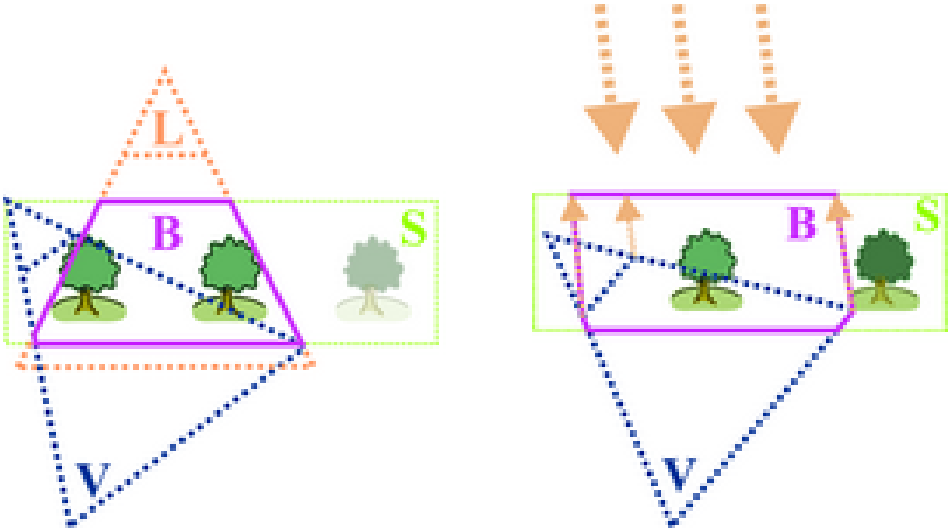


Figure 2.5: *Shadow map focusing better utilizes the available shadow map resolution by combining light frustum* **L**, *scene bounding box* **S** *and view frustum* **V** *into the bounding volume* **B**. *Here shown on the* left *for point lights and on the* right *for directional light sources.*

Unfortunately, shadow mapping suffers from spatial and temporal aliasing, visible as blocky pixels and flickering, respectively. The primary problem is *undersampling* due to insufficient shadow map resolution. To use the shadow map resolution more efficiently, Brabec et al. [BAS02] proposed to focus the shadow map to the visible scene parts, which makes shadow maps useful for larger scenes (see Figure 2.5). The downside is that now the shadow map extents are recalculated each frame and therefore the rasterization is likely to differ each frame, thus creating an abundance of temporal aliasing (flickering).

Removing the inherent drawbacks of shadow maps is an active research field and therefore a great number of different approaches exist. We divide

the approaches into *filtering-based* and *reparameterization-based* solutions and name the most relevant representatives for our work. This division underlines the fact that these approaches are orthogonal to each other and can be combined.

2.2.1 Filtering

The standard shadow map test results are binary: Either a fragment is in shadow or not, creating hard jagged shadow map edges for undersampled portions of the shadow map. These edges can be substituted with high-frequency noise, which is much more pleasing to the human visual apparatus. Traditional bilinear filtering is inappropriate for shadow maps, because interpolating depths creates incorrect results on discontinuous portions of the shadow map, which means that more efficient prefiltering techniques are not applicable. Reeves et al. [RSC87] had the insight that filtering can also be done after the shadow test is evaluated and proposed percentage closer filtering *(PCF)* (see Figure 2.6). *PCF* uses Poisson disk sampling in light space to approximate the reprojected eye fragments. This is in essence a reconstruction filter for magnification of the shadow map. Another, simpler approach is to substitute the high frequency noise with smooth shadow map boundaries by using (bi)linear or cubic filtering of shadow map test results, which is currently easier to implement on hardware.

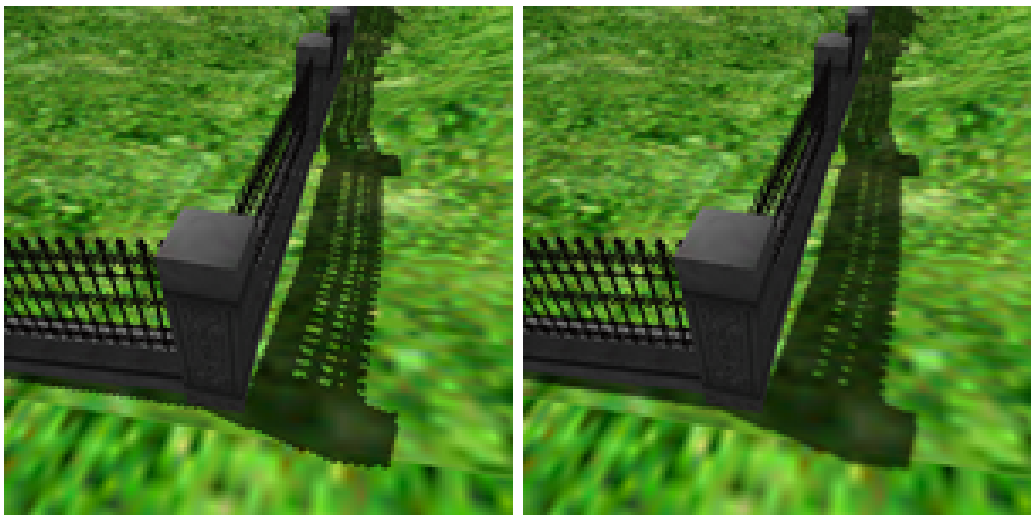


Figure 2.6: *Undersampled unfiltered shadow maps on the* left *suffer from hard jagged edges. These can be removed by filtering. On the* right PCF *with a 3x3 kernel is applied.*

A very sophisticated off-line filtering approach are *deep shadow maps* [LV00].

Here each texel contains a compressed piecewise linear representation of the visibility function – a weighted average of n piecewise linear transmittance functions taken at samples on the texel's area. This representation can be prefiltered and allows high quality shadows for complex cases such as hair or clouds (see Figure 2.7). Hadwiger [HKSB06] presented an interactive version for volume ray-casting on the GPU.

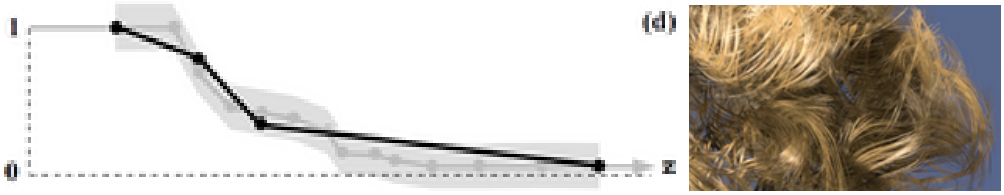


Figure 2.7: *Deep shadow maps store a picewise linear representation of the transmittance function gathered from various samples at every texel* (left). *This allows shadow mapping of challenging cases like hair* (right). *Image courtesy of [LV00].*

Donnelly and Lauritzen introduce variance shadow maps [DL06, Lau07]. They propose to store not only the depth, but also the depth squared, which allows to reconstruct the mean and variance of the depth distribution of each shadow map texel. This allows for prefiltering of shadow maps, allowing to use mip mapping to speed up the calculations for large filter kernels. The problem with this approach is that high variance in the depth distributions (high depth complexity) can lead to light leak artifacts (see Figure 2.8) and high-precision (32bit) texture filtering hardware is needed for satisfying results. A solution to both was presented by Lauritzen and McCool [LM08] by partitioning the depth range of the shadow map into multiple layers.

Annen et al. introduce *convolution shadow maps*, which also remove those artifacts by approximating the binary shadow map test with a weighted summation of Fourier basis terms [AMB+07] (see Figure 2.8). For an approximation using m terms, m sine and m cosine textures have to be stored. In general, shadow maps require a resolution of 24 or 32bit per depth value for satisfying results. Convolution shadow maps require only a resolution of 8bit per texture. Nevertheless, memory considerations require a restriction of the Fourier expansion to a small number of terms, which introduces ringing artifacts (Gibb's phenomenon). A faster solution to the prefiltering problem are exponential shadow maps [AMS+08]. The shadow map test is here approximated by using an exponential function.

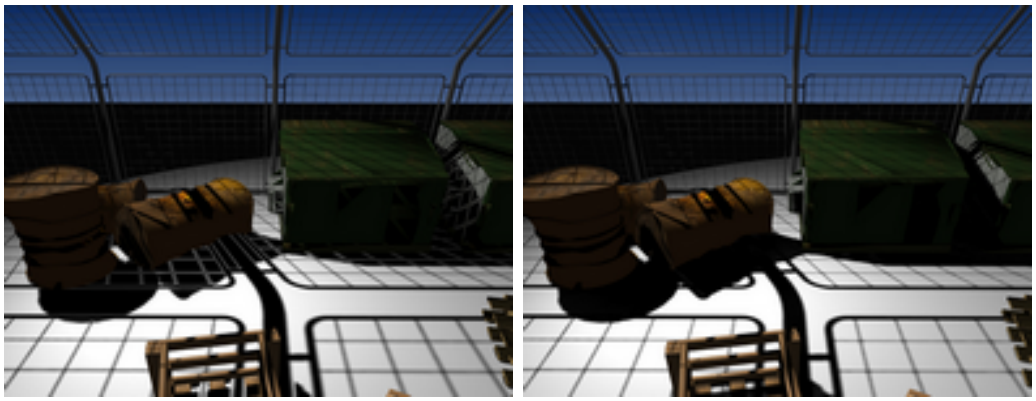


Figure 2.8: *Variance shadow maps* (left) *in contrast to convolution shadow maps* (right) *suffer from light leaks. Image courtesy of [AMB+07].*

2.2.2 Reparameterization

Shadow map aliasing artifacts due to undersampling can be divided into projective aliasing and perspective aliasing artifacts. Projective aliasing is caused by the mismatch of the projection of surface area in light space and eye-space, while perspective aliasing is caused by the mismatch between uniform shadow map resolution and the non-uniform resolution that is caused by perspective eye views (see Figure 2.9). It follows that projective aliasing is highly scene dependent, while perspective aliasing depends strongly on the camera parameters, like near plane distance and depth range. Reparameterization algorithms change the sampling on the shadow map plane to minimize the aforementioned aliasing artifacts. Due to the huge amount of literature in this area we divide these approaches further into *warping-based*, *subdivision-based* and *irregular sampling-based* algorithms.

Warping

Warping methods use a nonuniform parameterization of the shadow map to redistribute the samples. They primarily target perspective aliasing, as here reparameterizations can be found without having to undergo a slow scene analysis.

Stamminger and Drettakis introduced shadow map warping in their *perspective shadow maps* [SD02] paper. They use the post perspective view space to warp the shadow map. This has the benefit that just a simple perspective transformation is used, which can be represented by a 4×4 matrix. This maps well to hardware and is fast to compute. The main problem of this approach is that the achievable quality of this method is strongly dependent

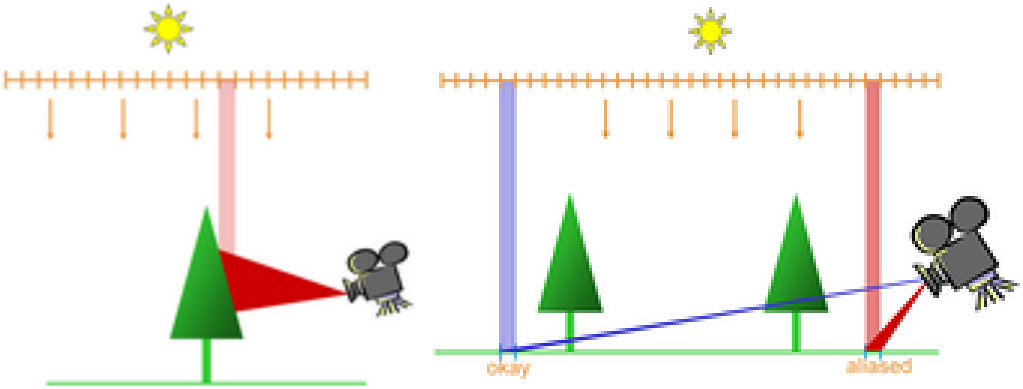


Figure 2.9: *In the figure on the* left *side the cause for projection aliasing is the orientation of the trees surface: It projects to a small area in the shadow map, but projects to a big area in camera space. Perspective aliasing* (right side) *occurs because the shadow map is indifferent to perspective foreshortening and distant as well as near areas (in respect to the camera) are therefore stored with the same resolution, but project to very different sizes in camera space.*

on the near-plane of the eye-view, because the error is distributed unevenly over the available depth range. With a close near plane most of the resolution is used up near the eye and insufficient resolution is left for the rest of the shadow map. The authors suggest to analyze the scene to push the near plane back as far as possible to alleviate this problem.

This is circumvented by decoupling the perspective transformation from the eye and using an independent near plane distance for this new transformation. This is the main idea of *light space perspective shadow maps* [WSP04], which warp the light space with a light and view aligned transformation. Additionally the near plane distance is chosen in a way to distribute the error equally over the available depth range, creating homogeneous quality (see Figure 2.10). A very similar approach are Martin and Tan's *trapezoidal shadow maps* [MT04], which use a heuristic to choose the near plane distance. Chon and Gortler [Cho03, CG04, CG07] optimize shadow map quality for small numbers of planes of interest by using multiple shadow maps.

A more involved method that does not use a perspective transformation, but the theoretically optimal logarithmic warp, is investigated by Lloyd et al. [LGMM07, LGQ+08].

Subdivision

In contrast to warping methods, subdivision methods try to approximate the ideal sample distribution with multiple shadow maps (see Figure 2.11).

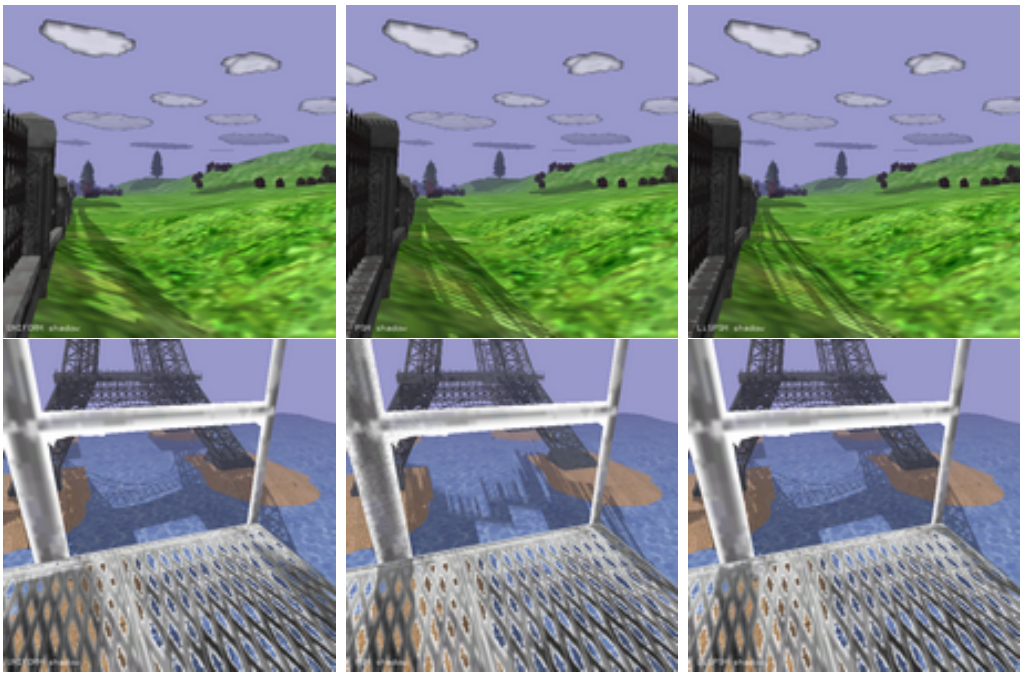


Figure 2.10: *Comparison of uniform* (left)*, perspective* (middle) *and light space perspective shadow maps* (right)*, each using a* 1024^2 *shadow map.*

Tadamura et al. [TQJN99] and Engel [Eng07] partition the frustum along the view vector into geometrically increasing sub-volumes. The same approach is used by Zhang et al. [ZSXL06]. Here the sub-volumes do not strictly increase geometrically, but a combination of geometrical and linear increase is applied.

The post perspective eye space used in *perspective shadow maps* can also be partitioned along the view frustum faces. These can be mapped to a cube map [Koz04], which reduces artifacts.

Tiled shadow maps[Arv04] tile the light view (here fixed resolution shadow map) to change the sampling quality according to a heuristical analysis based on depth discontinuities, distances and other factors, which allows setting a hard memory limit, thereby trading speed against quality. Forsyth [For06] clusters object with a greedy algorithm into multiple shadow frusta.

The main observation in *adaptive shadow maps* [FFBG01] is that a high quality shadow map only needs high resolution at shadow edges. Therefore the shadow map is stored in a hierarchical grid structure (quad-tree). Each quad-tree node has a fixed resolution shadow map attached to it. Each frame the nodes can be split iteratively to increase the shadow map resolution available. Lefohn et al. [LSO07] adapt this method by eliminating the edge detection phase in favor of generating all shadow map texels that are needed

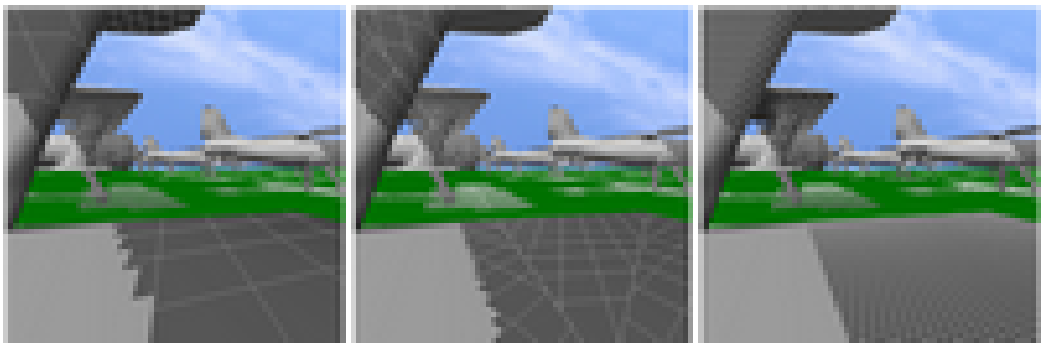


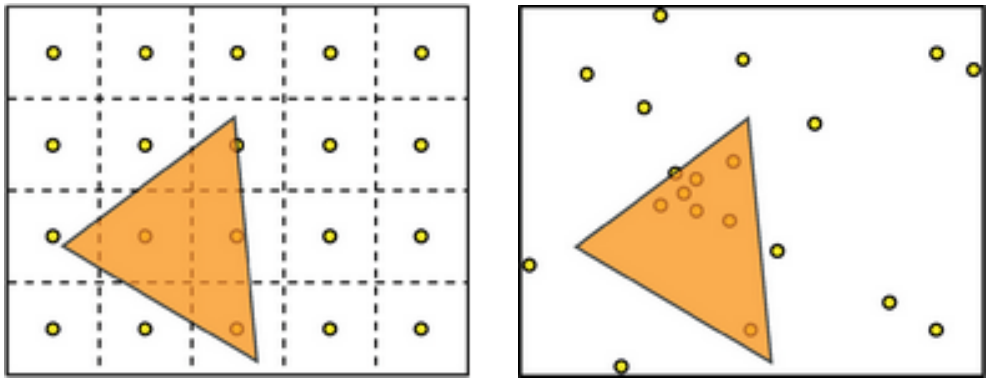
Figure 2.11: Left: *warping with a single shadow map;* middle: *face partitioning with warping (here 3 shadow maps are used);* right: *z-partitioning with warping (here 4 shadow maps are used); All images use a total of* 1024^2 *texels. Image courtesy of [LTYM06].*

to resolve the shadowing of screen-space pixels (*resolution-matched shadow maps* (RMSM)). To make this approach feasible, the authors use coherence between eye-space and light space: They assume that surfaces that are continuously visible in image space, are also so in light space and employ a connected-components analysis to find these surfaces and then request shadow map pages for each of those. A similar method are *queried virtual shadow maps* [GW07b]. Here again hierarchical shadow map refinement is used, but in a more hardware friendly manner: A shadow map is split into tiles and for each tile refinement is performed. This process is repeated as long as the refinement increases quality. The quality is checked by using an occlusion query for each tile. If the result of the refinement equals the previous result the refinement was unnecessary and therefore further refinement can be omitted. What makes this method practical is that only very little data (the result of each occlusion query) has to be sent back to the CPU. The downside here is that tiles are created that subsequently have to be split again. Giegl and Wimmer [GW07a] improve upon this in *fitted virtual shadow maps* (FVSM), where they use eye space fragments to predict were refinement will be needed, thereby generating only the tiles with sufficient resolution. This is similar to resolution matched shadow maps, but RMSM compute the required subdivision levels on the GPU, while FVSM do this on the CPU, but on lower resolution buffers.

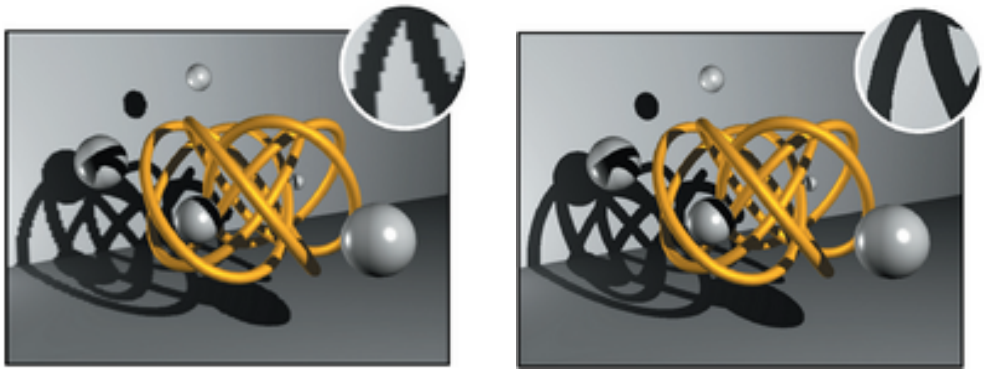
Irregular Sampling

In the second pass of shadow mapping, all screen-space fragments are reprojected into the shadow map to be tested. The irregular sampling methods

try to sample the light-space at these reprojected (probably quite irregular) positions, thereby giving each screen-space fragment the best sample for the shadow map test and removing all aliasing artifacts. The results are as good as ray-tracing would provide, but the downside is that this approach does not map well to rasterization hardware, which requires regular grid rasterization. Johnson at al. [JMB04, JLBM05] propose a hardware extension (they use a list of samples at each regular grid element) to allow for irregular sampling: the irregular z-buffer and show its application to shadow mapping (see Figure 2.12).



(a) *Regular z-buffer* (b) *Irregular z-buffer*



(c) *Shadow mapping* (d) *Irregular shadow mapping*

Figure 2.12: *Regular grid sampling versus irregular sampling in shadow mapping. Image courtesy of [JLBM05].*

Alias-free shadow maps [AL04] provide a hierarchical software implementation using an axis-aligned BSP tree to efficiently evaluate shadow information at the required sample points. This approach is later mapped to graphics hardware by Arvo [Arv07] and Sintorn et al. [SEA08] (details in Section 2.3.5).

2.3 Soft Shadow Mapping

Soft shadows are a very active research topic and the second domain where we apply temporal coherence methods. We can therefore only give an overview of the publications in this field. A still valuable survey of soft shadowing methods is due to Hasenfratz et al. [HLHS03].

There are two major paradigms: methods that use shadow volumes (object-based algorithms) and methods that use shadow maps (image-based algorithms). We concentrate on algorithms that employ shadow mapping due to their higher performance in real-time applications and there relevance to our work.

Since physically based soft shadow mapping requires many light source samples, it was previously considered too costly for real-time rendering. Therefore a number of algorithms were proposed that offer cheaper approximations.

2.3.1 Single Sample Soft Shadows

The algorithms discussed in this subsection divide the penumbra into an inner and an outer part, a physically incorrect assumption. Nevertheless, this approach can create acceptable soft shadows for small circular light sources. The inner penumbra is the part of the penumbra that is inside the hard shadow of the center sample point and the outer penumbra is the part of the penumbra outside of the before mentioned hard shadow.

The first single light sample method for creating convincing soft shadows was presented by Parker et al. in [PSS98]. While this work only calculates the outer penumbra a modification for hardware with shadow map support by Brabec and Seidel [BS02] also calculates the inner penumbra. Here we discuss the modifications for shadow maps by Brabec and Seidel.

The idea is to create a standard shadow map at the center of the light source. On rendering the scene from the point of view of the eye, for each pixel that is illuminated we search the shadow map for the nearest texel that is in hard shadow (see Figure 2.13). To make this search robust for most cases, the authors use object ids [HN85] to omit incorrect self shadowing of arbitrary receiver occluder configurations. This makes soft self shadowing impossible. If a maximum search radius r_{max} is reached, the pixel is considered completely lit, otherwise an attenuation factor f is calculated:

$$f = \frac{r}{r_{max}} \tag{2.1}$$

where r is the distance where the nearest blocked pixel was found. f rises from 0, which means that the pixel is in shadow, to 1, which represents a

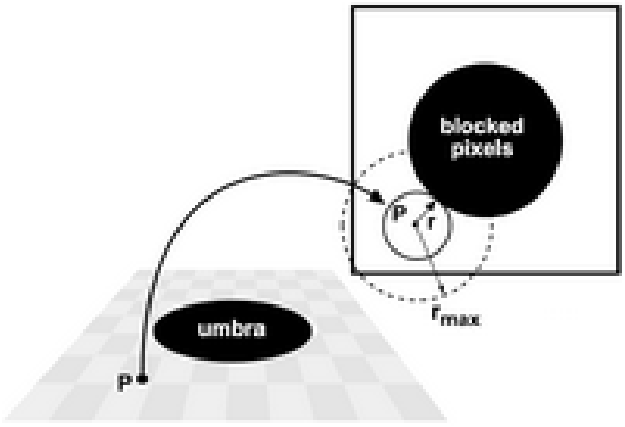


Figure 2.13: *A circular search is performed to find the nearest blocked pixel.* r_{max} *is hereby the maximum search distance. Image courtesy of [BS02].*

fully illuminated pixel.

To incorporate the effect of the distance between the receiver and the light source, i.e., the penumbra should be larger the farther away the receiver is from the light source, the authors vary r_{max} according to this distance. The distance between receiver and occluder is more difficult to incorporate. The authors choose to rescale r_{max} in Equation 2.1. After the search, they take the distance between the search start point and the found occluder point to shorten r_{max}. In a similar manner, this neighborhood search can also be used to calculate the inner penumbra.

Figure 2.14: *Brabec and Seidel's* single sample soft shadows *with varying distances between shadow caster and receiver. Image courtesy of [BS02].*

The visibility of the light source is, due to the single light source sample, very approximat and therefore only acceptable for small light sources. Nevertheless the results of this algorithm are very convincing as long as the light source stays sufficiently far away from the occluder or is small enough (see

Figure 2.14). The greatest performance loss of this approach is incurred by the neighborhood search.

A faster realization of this method without the neighborhood search was proposed by Kirsch [KD03]. Here the search is replaced by a dependent texture lookup into a *shadow width map*. The shadow width map is created by repeated smoothing of an inverted occlusion map of the shadow blockers (see Figure 2.15), in effect only calculating the inner penumbra.

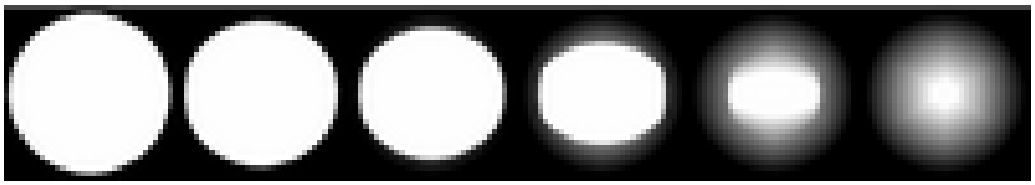


Figure 2.15: *The* shadow width map *is created by repeated texture compositing of a binary occluder map. Image courtesy of [KD03].*

The downside of this method is that shadows are smaller than physically expected. The primary source for artifacts are overlapping blockers in light space. Only one depth value per depth map texel is stored, which results in shadow depth discontinuities if the depth values of the blockers exhibit high differences (see Figure 2.16).

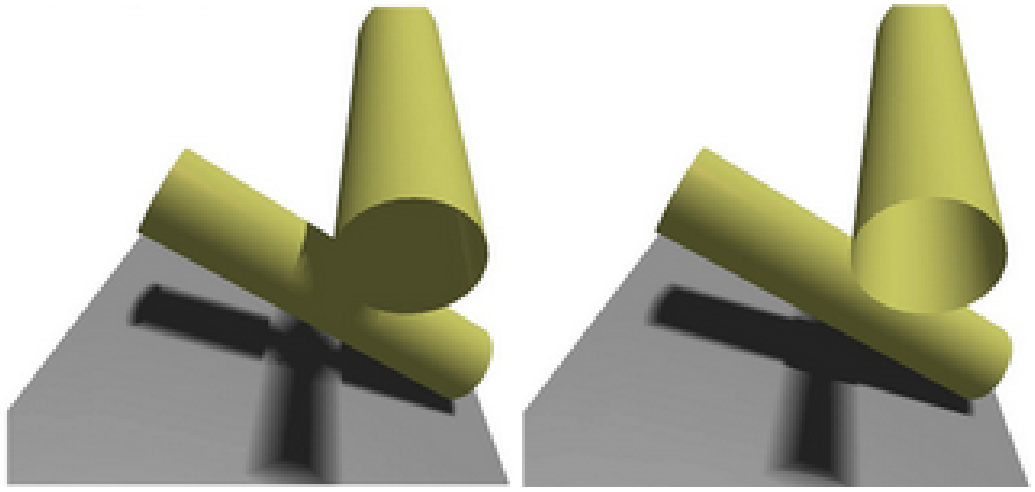


Figure 2.16: *Artifacts caused by overlapping blockers* (left) *can be lessened by partitioning blockers and receivers, thereby losing self-shadowing capabilities. Image courtesy of [KD03].*

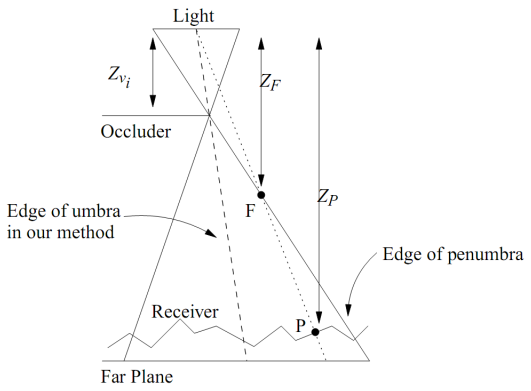
2.3.2 Hybrid Approaches

The hybrid approaches discussed in this subsection either perform shadow edge detection, or use additional geometry for penumbra rendering.

Penumbra maps introduced by Wyman and Hansen [WH03] are a three pass algorithm: In the first pass, a shadow map from the center of the circular light source is created. In the second pass the penumbra map is calculated. The third pass uses the intensity information from penumbra map and the depth information from the shadow map to render the scene. The penumbra map is a texture that contains the shadow intensity of the corresponding shadow map pixel, which equals the penumbral intensity on the foremost polygons visible form the light source. When creating the penumbra map, additional geometry made of sheets and cones is attached to the umbra to simulate the penumbra. This idea is based on *penumbra wedges* [AMA02], a soft shadow volume approach. The intensity calculation of each penumbra map pixel can be done as follows: with Z_{v_i} the current vertex depth from the light, Z_F the depth of the current cone or sheet fragment, Z_P the corresponding shadow map depth, the intensity I is defined as:

$$I = 1 - \frac{Z_P - Z_F}{Z_P - Z_{v_i}} = \frac{Z_F - Z_{v_i}}{Z_P - Z_{v_i}} \tag{2.2}$$

and can be evaluated with vertex and fragment programs (see Figure 2.17).



(a) *Geometry for the calculation of a fragments penumbral intensity.*

(b) *Input shadow map* (top-left)*; penumbra map* (top-right)*; results* (bottom)

Figure 2.17: *Construction of the* penumbra map. *Image courtesy of [WH03].*

The main limitations of this method are the lack of an inner penumbra and unnatural looking shadows for big light sources or far off occluders. Also

the silhouette calculation can decrease the performance in polygon-rich scenes. The fill-rate bottleneck associated with shadow volumes is avoided because of the use of shadow maps. The authors report frame-rates of 20 frames per second (2003), but with a third of the time spend for the silhouette calculation.

Chan and Durand's [CD03] rendering fake soft shadows with *smoothies* uses the basic shadow mapping algorithm, but generates 2D primitives, called smoothies, out of the extrusion of the silhouette edges of the shadow caster objects to simulate the penumbra. Smoothies are planar surfaces perpendicular to the surface of the occluder. The algorithm proceeds in 5 steps:

- First a standard shadow map from the center of the light source is created.
- Afterwards the silhouettes of the shadow casters in object space are found [McC00].
- Then the smoothie edges are constructed by extruding the silhouette edges outward and connecting them. The size of the extrusion specifies the size of the penumbra.
- Now the depth of the smoothies is rendered into a depth map. Together with the *smoothie* depth a ratio of distances between the light source, blockers and receiver are stored into a smoothie alpha map. These two maps are combined into a depth-alpha texture, called the smoothie buffer.
- Finally the rendering takes place by calculating the depth comparison to the standard shadow map and to the smoothie buffer and using the smoothie alpha texture to add the penumbra attenuation (see Figure 2.18).

This amounts to a total of 3 render-passes: shadow map generation; smoothie depth and alpha generation and the final rendering pass.

The authors state the performance of this algorithm with 39 frames per second (2003) for a moderately complex scene with 10,000 triangles and a map resolution of 1024x1024 pixel. The edge connection is a problem for big light sources, which introduces noticeable artifacts. An extension of smoothies which calculates the inner penumbra is described in [Cha04].

Smoothies are very similar to penumbra maps. Both render soft shadows with attached geometry to simulate the penumbra and use the resulting intensity to gather the final image. One difference between the two approaches

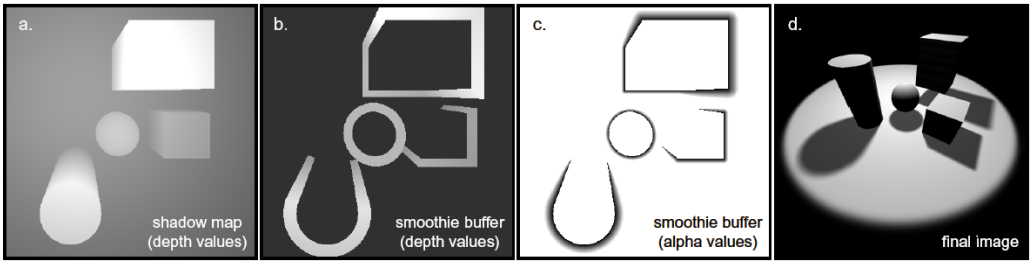


Figure 2.18: *From* left *to* right*: shadow map; smoothie; depth values; smoothie alpha values; final rendering. Image courtesy of [CD03].*

is the used geometry: For penumbra maps sheets and cones are used and smoothies are constructed with quads. Secondly penumbra maps store the depth information of the occluders only, while in the smoothies algorithm the depth information of the occluders and of the smoothies are stored, which results in extra storage cost and an additional depth comparison. But on the plus side smoothies can handle surfaces that only act as receivers with this additional information.

De Boer [dB06] introduced a real-time image space algorithm that incorporates inner and outer penumbra. The light source has a circular shape and the main advantage over most other algorithms presented so far is that all steps are done in image space, so the algorithm is largely independent of scene complexity. The approach is based on the image space construct *skirt*. A *skirt* is associated with an shadow caster edge and contains attenuation and depth information (see Figure 2.19).

The first step is to create a shadow map. To find the silhouettes of the shadow casters, an edge detection filter is applied to the shadow map and the edges are saved with an attenuation value of one in an otherwise black *skirt* buffer. The *skirt*'s attenuation values are then created by applying a smoothing filter repeatedly. The depth value saved in the *skirt* buffer for each pixel is the minimal depth value of all neighboring pixels the smoothing kernel is applied to. The iteration-count is proportional to the radius of the light source and limits the maximum size of the penumbra.

The approach is realized in 3 passes: shadow map creation, skirt buffer creation and final image rendering. In the final step a fragment shader incorporates the relative distances of occluder, light and receiver and calculates the appropriate attenuation. For the inner penumbra, an additional disc search (neighborhood search in Brabec and Seidl [BS02]) is necessary. Nevertheless claims the author real-time frame-rates (2004).

The main problems of this approach are the not entirely smooth penumbra transitions and performance degredations for larger penumbras (because of

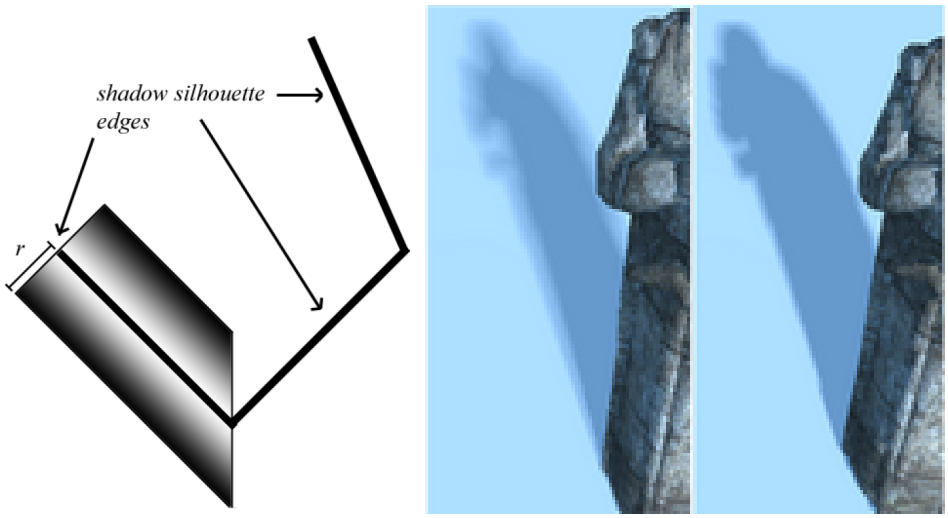


Figure 2.19: *A skirt is associated with a shadow caster edge and contains attenuation and depth information* (left). *Skirt results* (middle)*; smoothies* (right)

the disc search) and large light sources (because of the higher iteration count for the skirt generation).

2.3.3 Filtering

The filtering-based approaches try to estimate the soft shadowing result by evaluating neighboring texels in the shadow map. Here filtering, normally used for reconstruction or blurring, is extended to distance dependent kernels, to account for the changing penumbra size of soft shadows.

An early approach involving convolution was presented in [SS98]. The idea is to use an image of the light source and convolve it with the blocker image to obtain a *shadow map* (see Figure 2.20). This texture is later used to modulate the illumination function. This is only valid if the light source, the blockers and the receivers lie in parallel planes. To use this approach for general scenes, the authors propose an error driven algorithm which subdivides the set of blockers if needed. Note that this subdivision doesn't work for all cases. Especially polygons elongated in the light direction cannot be broken down easily.

Drawbacks of this approach are its restriction to a special case and the low performance when the subdivision algorithm is used for more general configurations. Although the soft shadows are not physically correct, the results are very smooth and the apparent quality therefore is high.

A variation of this approach was presented by Eisemann and Decoret [ED06b, ED08]. Here the scene is split into layers and the occlusion information of

Figure 2.20: *The light source image* (left)*, the blocker image* (middle) *and the convolved result* (right)*. Image courtesy of [SS98].*

each layer is calculated by projecting the layer into a so called *occlusion texture*. The layering is performed in one pass by using a fast scene voxelization technique [ED06a]. Each of the occlusion textures is prefiltered using convolution. The combined layers result in plausible soft shadows. The greatest benefit of this method is that its speed is independent of the size of the penumbra. The major disadvantage is the dependence of this approach on the depth complexity of the input scene: Higher depth complexity requires a larger number of occlusion textures, each of which need to be convolved and combined.

Another very fast and simplified adaptation was presented by Fernando [Fer05]. Percentage Closer Soft Shadows (PCSS) estimate the soft shadow from a single sample and employ a blocker search to estimate the penumbra width and then use PCF-filtering accordingly to soften the shadows. The penumbra with $w_{penumbra}$ is given by

$$w_{penumbra} = \frac{(d_{receiver} - d_{blocker})w_{light}.}{d_{blocker}}$$

were $d_{receiver}$ is the distance between light source and receiver, $d_{blocker}$ is the distance between light source and blocker and w_{light} is the width of the light source (see Figure 2.21). Very fast speeds can be achieved by using a kernel with fixed sample count and only varying the kernel width (sample spacing) to account for the penumbra width estimation. Unfortunately this introduces artifacts for large penumbrae (see also Figure 2.21).

Using a variation of *convolution shadow maps* and the average blocker search from PCSS, Annen et al. [ADM+08] extract area light sources from environment maps, which allows for approximated environment lighting by a number of area lights. The extraction is managed by starting from the texels with highest energy in the environment map. These seed areas for area light

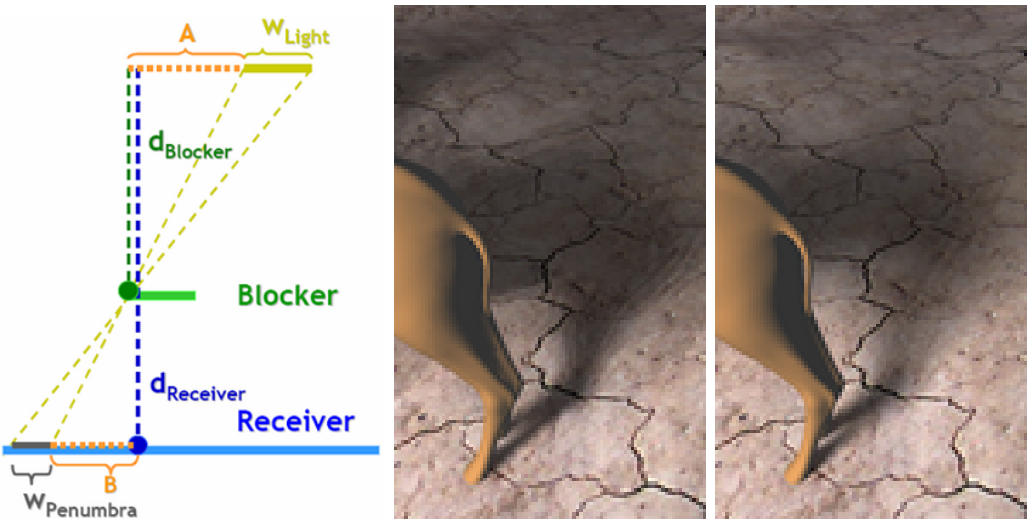


Figure 2.21: *Percentage closer soft shadows assume parallel blocker, receiver and light source constellations to simplify the calculation of the penumbra width* (left Image courtesy of [Fer05].)*, but produce errors for large penumbras* (middle) *when compared to the ground truth* (right)*.*

sources are enlarged outward as long as the ratio between energy increase (due to the enlargement) and the total light source energy is above a certain threshold. They also reformulate the estimation of the average blocker depth as a convolution operation, thereby making it efficient to evaluate. Because the soft shadow calculations are based on PCSS, the approach shares the same principal shortcomings as PCSS. But the possibility of prefiltering allows for more sophisticated filtering, using larger kernels or summed area tables to allow for improved results.

2.3.4 Back Projection

Back projection methods [GBP06, AHL+06, ASK06] treat the shadow map as a discrete approximation of the blocker geometry. By back projecting shadow map texels onto the light source, an accurate calculation of the percentage of light source visibility can be attained. However, one shadow map can only capture the scene visibility from one point on the light source (ignoring occluders not visible from this point on the light source), so this is still an approximation. Nevertheless, convincing and physically plausible results are produced. The downside of these methods is that a region (related to the size of the penumbra) of the shadow map has to be sampled for each fragment, which becomes costly for large penumbrae and which has to be limited to allow

real-time performance, leading to artifacts for large penumbrae. Additionally the discretization causes artifacts: First, the overlap in the back projected shadow map texels is not taken into account. And second, gaps in the back projection can lead to light leaks (see Figure 2.22).

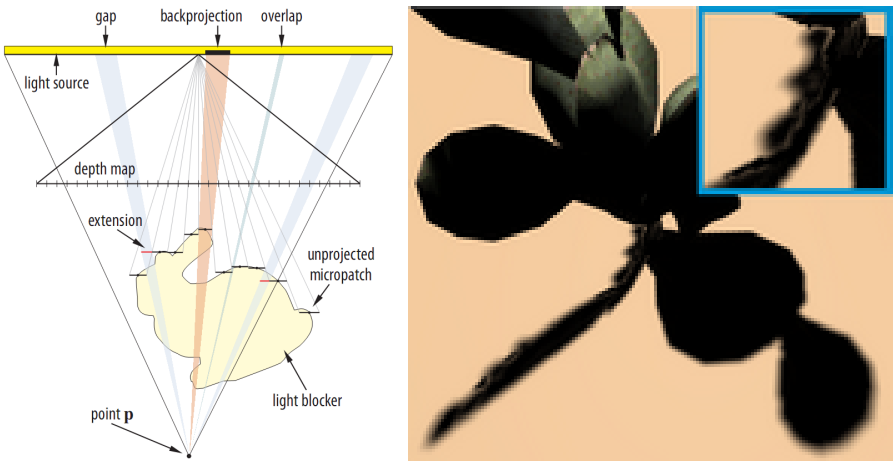


Figure 2.22: *Back projection can lead to unwanted gaps and overlaps.* Left: *Image courtesy of [SS07].*

Schwarz and Stamminger [SS07] tackle the problem of overlapping back projections. They use a number of sample points on the light source and a bit field to record which of them are occluded. They also propose a solution to unwanted gaps in the back projected texels by interpreting each texel center as a corner of a *microquad*. A *microquad* is only accepted as an occluder if all four corners pass the hard shadow test (are closer to the light source). They later extent this by using triangles *(microtris)* instead of *microquads* [SS08a] and *microrects* [SS08b], while Guennebaud et al. [GBP07] use the occluder contour.

Another approach is to use multiple samples using depth peeling [BCT08], which accounts for the lost occluders when using only a single shadow map.

2.3.5 Sampling

Maybe the most straightforward approach to computing soft shadows is by sampling an area or volume light source. Such methods are mostly targeted at off-line or interactive rendering. Heckbert and Herf [HH97] propose to sample the light source at random positions, render the scene and accumulate the results. Heidrich et al. [HBS00] reconstruct the visibility on a linear light source in high quality, while only using two shadow maps (each on one end) as samples. The linear visibility between the shadow maps is calculated by

a sophisticated interpolation method. They also briefly investigate how to extend this approach to more samples and triangular (area) light sources.

Agrawala et al. [ARHM00] create a layered attenuation map out of the shadow maps, which allows interactive frame rates. St-Amour et al. [SAPP05] combine the visibility information of many shadow maps into a compressed 3d visibility structure which is precomputed, and use this structure for rendering.

Eisemann and Decoret [ED07] show how to do visibility sampling between two surfaces in hardware and show an application of this approach for soft shadows. This method can create accurate soft shadows for planar (or heightfield) receivers. It separates casters and occluders (no self shadowing) and cannot handle more general receivers. A more general version of this method is discussed in the scope of the following approach.

Sintorn et al. [SEA08] present a hardware implementation of alias free shadows maps [AL04] extended to soft shadows by using light sampling. They realize alias free shadow maps by storing sequential lists of reprojected eye samples – one list is associated with one shadow map texel (using CUDA). Then they rasterize each scene triangle in light space (shadow casting triangles) and traverse for each fragment its list to check for occlusions. Standard rasterization only creates a sample if the location at the center of a fragment is inside the rasterized polygon. Because the eye space samples are located arbitrarily on the shadow map texel, rasterization has to be conservative to assure that a fragment is created for all locations on the shadow map texel (and not just the center). For alias free shadows (hard shadows) it is sufficient to store a binary decision for the occlusion.

To calculate soft shadows this approach has to be extended in two ways: The conservative rasterization of shadow casting triangles has to take the full influence region (umbra and penumbra) into account and the occlusion test has to be performed for multiple samples on the light source. To find the influence area the authors note the following: A point is shadowed by a triangle, if it lies in the union of the projections of the triangle from all light source points onto a plane that is either passing through the point or is farther away (as seen from the center of the light). Hence the first step is performed by projecting an approximation of the silhoutte of the light (they use a hexagon) through the vertices of each triangle onto the far plane. The convex hull of the results is the influence region of the triangle. The second step, the occlusion calculation, is realized with a light sample dependent lookup structure. For an input plane the structure returns a bit pattern of light samples behind this plane. By using each triangle edge together with a view sample, 3 planes are given. By applying an AND operation on the returned samples of each plane, the visibility of the view sample with respect to the triangle is calculated (see Figure 2.23). The differences to the

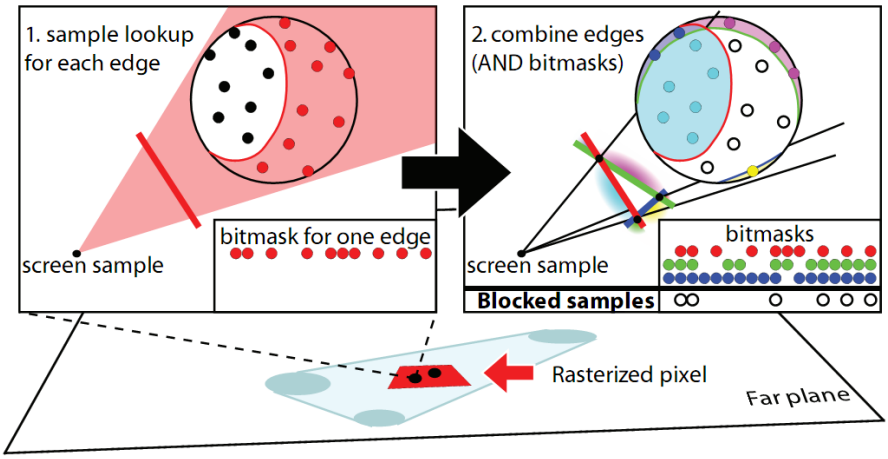


Figure 2.23: *Visibility calculations for view samples: A lookup structure returns the bitmask of light samples behind a given plane. A view sample creates with a triangle three planes (with each edge one). The resulting visibility is arrived by ANDing these bitmasks. Image courtesy of [SEA08].*

method of Eisemann and Decoret [ED07] is that this approach can handle arbitrary receivers and that it is more efficient, due to tight influence regions and conservative rasterization.

Forest et al. [FBP08] use the number of occluders between two points (depth complexity) to solve the rendering equation for direct lighting numerically. The geometrical *penumbra wedge* [AMA02] primitive is used to create surface points in penumbra regions. Then they evaluate the depth complexity function between each surface point and a set of light samples to find the amount of visible samples. For each visible sample depth complexity must equal zero. With this they are able to calculate the visibility coefficient for each viewed point, which can be later used in the rendering equation to account for soft shadows (see Figure 2.24).

Johnson et al. [JHH+09] use the much more CPU like architecture of the Larrabee processor to project light and occluders into the light view image plane and measure the area of overlap for each given receiver point. To accelerate this approach they start by storing all visible view pixels in an irregular spatial acceleration structure in light space. Using this structure, they eliminate in a first step points in the umbra of the light source form the more expensive penumbra calculations. The penumbra calculations are then performed in two steps: First a coarse (overestimated) penumbra geometry is used to reject more of the stored points. Finally the fractional area of occlusion is determined for each of the surviving points by clipping the occluders surface to the extends of the area light.

Figure 2.24: *A comparison of the original penumbra wedges* (left)*, Forest et al.'s new method* (middle) *and ray-tracing* (right)*. Image courtesy of [FBP08].*

2.3.6 Other approaches

Ren et al. [RWS+06] represent occluders by spheres and use harmonic exponentiation to calculate soft shadows. This allows the method to account for the low frequency properties of soft shadows without evaluating many light directions. This method maps well to hardware and can be computed on a modern GPU in a single pass. The major drawback of this method is that high frequency parts of the soft shadowing solution are missing.

2.4 LOD

Level-of-detail methods try to accelerate rendering by using less detailed representations for objects that are farther away. Due to the vast amount of existing techniques in this areas, only a brief overview can be given here. We therefore redirect the interested reader to the comprehensive text by Luebke et al. [LRC+02] for an in-depth discussion.

Level-of-detail methods are generally divided into two categories: *Continuous LOD* and *discrete LOD*:

2.4.1 Continuous LOD

These methods work by creating an object representation specific for each viewing condition for each frame. Since similar viewing conditions result in similar representations, the change of one representation into another is perceived as smooth. Due to the fact that there are infinitely many representations, it follows that the creation of the different representations has to be done by an automatic method, dependent on some input parameters (i.e., error metrics) [GH98]. The problem with such automatic methods is

that they do not give satisfactory results for every possible model and all viewing conditions [Hop96, JWLL05]. Therefore most of the continuous LOD literature is specialized on methods for certain types of models. A very considerable amount of literature exists in the area of terrain rendering. Here the challenge is that the level-of-detail changes over the extent of a single and huge model [LKR+96, Ste97, Ste98, Hop98, CGG+03, LH04, DS04, SW06, GM07].

2.4.2 Discrete LOD

These methods on the other hand use a small set of often hand-crafted representations and try to select the most appropriate one for a given viewing condition. The selection will be based on some metric. This metric in turn will be based on distance, size, pixel-count or even perceptual properties [FS93]. The set of representations are small because of two reasons: memory consumption would explode for many representations, and the cost of an artist designing many representations would be enormous. Due to this small number of representations, switching from one representation to another becomes visible and causes *popping artifacts* [CS06]. One method to avoid these popping artifacts is *late switching*. Here switching is performed only if the two representations have exactly the same appearance for the current viewing condition. In practice this is quite unfeasible: First of all we often do not know when two representations will be indistinguishable, because this depends on numerous factors, like lighting and the surrounding objects, which we probable do not know beforehand. And secondly, from the point-of-view of performance we want to switch as early as possible to speed up rendering as much as possible.

2.4.3 Hybrid Approach

A way out of this dilemma is so called LOD-blending [GW06]. The idea is in a way a hybrid between discrete and continuous LOD methods. Although it works by using a discrete set of LODs, it does not use hard switching to change from one representation to the next. Instead a transition phase is introduced in which the two representations are alpha blended in screen space, thereby avoiding popping artifacts. Care has to be taken how to apply blending during the transition phase: If both LODs are rendered semitransparent at the same time, objects behind become visible (see Figure 2.25). Therefore the authors suggest to use the transition shown in Figure 2.26: When a change from LOD_K to LOD_{K+1} is taking place, first the new LOD (LOD_{K+1}) is blended in. And only after this LOD is fully visible the old LOD (LOD_K) is blended

out. Note that with this approach at most one LOD is semitransparent at all times. Therefore the intersection volume of the two LODs stays opaque during the whole transition phase.

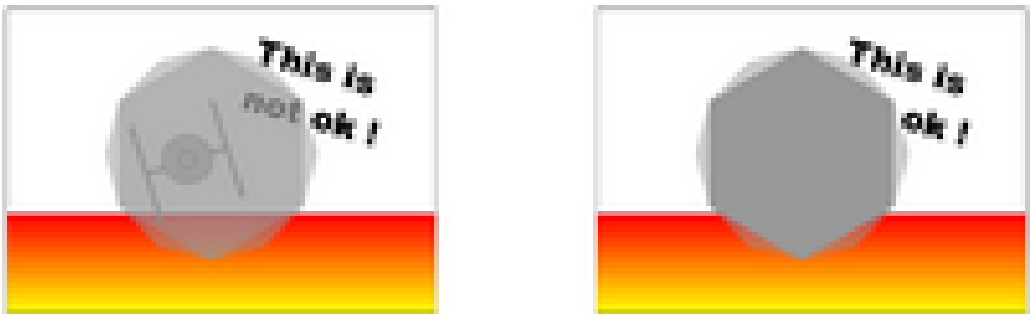


Figure 2.25: *Rendering both LOD levels semitransparent at the same time introduces incorrect visibility. Image courtesy of [GW06].*

This approach has a couple of shortcomings: First, it introduces the overhead of rendering both LOD during the transition phase, making this method slower than hard switching. Therefore transition phases should be kept short. Second, z-fighting between the two LODs can occur were the two LODs are coplanar. Third, in the middle of the transition phase one of the LODs switches from opaque to blended and the reverse is true for the other. The contents of the z-buffer are determined by the opaque LOD (because the blended one has to be drawn with depth write off) and therefore the z-buffer changes abruptly, which can still lead to popping at this point in time. We will describe our improved version of this algorithm in Chapter 5.

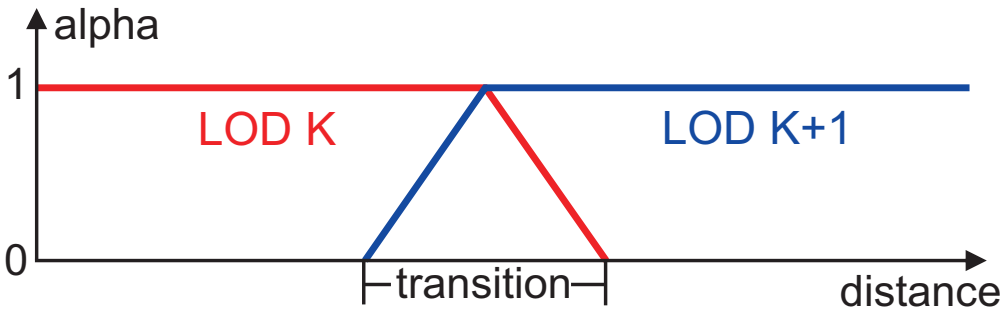


Figure 2.26: *When exchanging LOD_K with LOD_{K+1} first LOD_{K+1} is blended in and then LOD_K is blended out. Therefore one LOD is always drawn opaquely.*

3

Pixel Correct Hard Shadows

Figure 3.1: *Pixel-correct shadow maps as a result of using a shadow map (size* 1024^2*) with shadow test confidence together with a history buffer. Note that projection and perspective aliasing are completely removed.*

Due to its speed and versatility, shadow mapping is one of the most used real-time shadowing approaches. The most concerning visual artifacts of this method originate from aliasing due to undersampling. The cause for undersampling is in turn closely related to rasterization that is used to create the shadow map itself. Rasterization uses regular grid sampling for rasterization of its primitives. Each fragment is centered on one of these samples, but is only correct exactly at its center. If the viewpoint changes from one frame to the next, the regular grid sampling of the new frame is likely to be completely different than the previous one. This frequently results

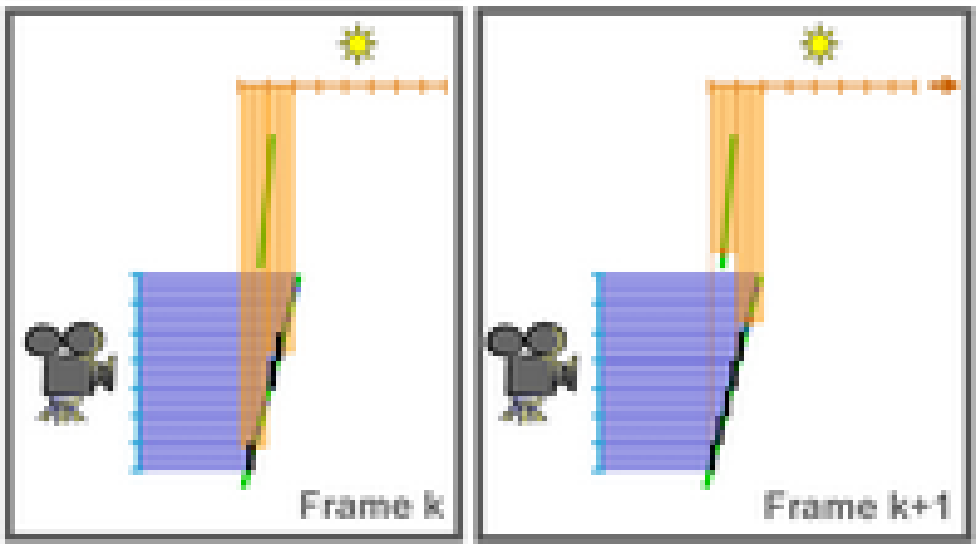


Figure 3.2: *If the rasterization of the shadow map changes (here represented by a right shift), the shadowing results may also change. On the* left *three fragments are in shadow, while on the* right *five fragments are in shadow. This results in flickering or swimming artifacts in animations.*

in artifacts, especially noticeable for thin geometry and the undersampled portions of the scene.

This is especially true for shadow maps. Due to shadow map focusing, a change in the viewpoint from one frame to the next also changes the regular grid sampling of the shadow map. Additionally the rasterized information is not accessed in the original light-space where it was created, but in eye-space, which worsens these artifacts. This frequently results in temporal aliasing artifacts, mainly flickering (See Figure 3.2).

The point addressed in this chapter is that different subsequent rasterizations of the shadow map are a vast source of information: each rasterization is a discrete approximation of the same (or at least a very similar) continuous depth image that would represent a perfect shadow map. Note that we do not need an infinite resolution shadow map for a fixed resolution view port. Instead, a depth sample in the shadow map at each reprojected view space sample position is sufficient. In practice this results in an irregular grid shadow map [AL04], which can hardly be implemented in current hardware.

In this chapter, instead of adapting the shadow map resolution, we exploit temporal coherence and the different rasterizations discussed above and accumulate the required information per pixel over time by using a so-called *history buffer* in screen space. This buffer accumulates shadow map test

results for the past few frames, reprojected to the current frame. A special confidence-based history buffer update based on the current shadow map ensures that it converges to an exact shadowing solution. Consequently, shadow borders sharpen over time until both perspective *and* projective aliasing artifacts are completely removed (see Figure 3.1). Likewise, temporal aliasing (shadow flickering) is removed, since shadows are smoothed during the convergence process. Note that in practice the convergence is typically faster than the eye adaption, so that the image quality appears consistently high.

The main contribution discussed in this chapter is a shadow mapping algorithm that quickly adapts to pixel correct (hard) shadows over time. Similarly, it eliminates image flickering by smoothing shadows in case of insufficient information. No scene analysis is required, so that the rendering speed is practically not reduced. In addition, the algorithm is simple to implement and integrate into existing rendering engines. It works together with advanced filtering methods like percentage closer filtering and more advanced shadow mapping techniques like perspective or light space perspective shadow maps.

The remainder of the chapter is organized as follows: We start by describing mathematical basics and our pixel-correct shadow map technique in Section 3.1. In Section 3.2 we present our results and limitations of our method, and in Section 3.3 we give conclusions and in Section 3.4 we discuss some possible extensions.

3.1 Our Algorithm

The main idea is to reuse shadowing information of previously rendered frames to increase the shadow accuracy of the current frame while taking into account the *confidence* of this information. We accumulate this information in a so-called *history buffer*, which will be introduced in detail in Section 3.1.1. The actual shadow mapping operation is then performed in every frame by the following steps:

1. Calculate the shadowing for all fragments of the current frame using standard shadow mapping.

2. Transform the history buffer to the current frame (see Section 3.1.2). Note that in practice we will do the reverse and transform each fragment from the current frame back into the history buffer.

3. Update the history buffer using the current shadow map (see Section 3.1.3).

4. Shadow the current frame according to the updated history buffer. This is trivial since the amount of shadowing is directly stored in the history buffer, as will become clear in Section 3.1.1.

3.1.1 Temporal Smoothing with the History Buffer

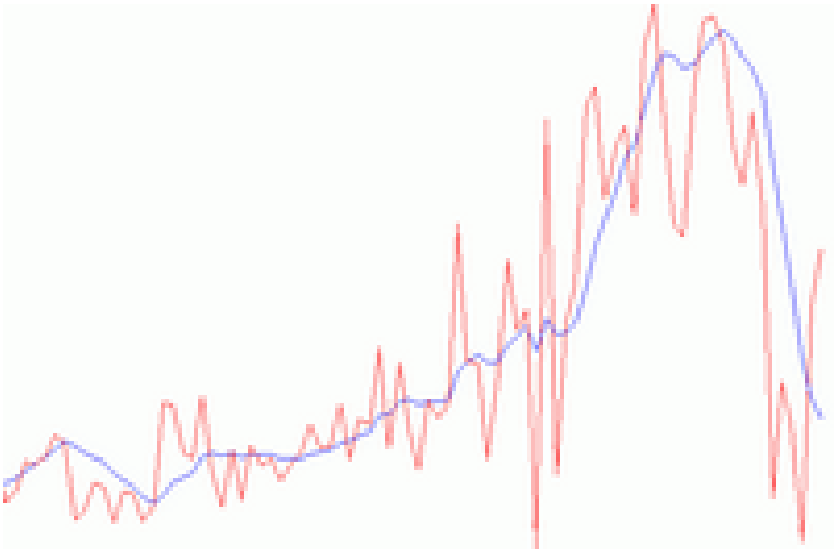


Figure 3.3: *A function, shown in red, is smoothed by applying exponential smoothing with* $w = 0.15$*. The result is shown in blue.*

In order to reduce temporal aliasing, we interpret each pixel as a separate function with the time as the input domain (usually represented by a frame number). Temporal anti-aliasing is then done by smoothing this function. An obvious way would be to use floating averages

$$s(n) = \frac{f(n) + f(n-1) + ...f(n-k)}{k} \tag{3.1}$$

where k is the number of frames we consider and f is the function we want to smooth. $f(n)$ is the function result for the current frame, $f(n-1)$ for the last frame and so forth. The formula results in averaging the last k frames. This straightforward approach raises two problems. First, we would need to store $f(x)$ for every pixel for the last k frames, which results in high storage costs. We would prefer to calculate our results incrementally, so that we would only have to store the functions of a single frame. Second, if all frames are weighted equally, arriving at the actual value of the function for the current frame would need k consecutive identical frames, which introduces a latency

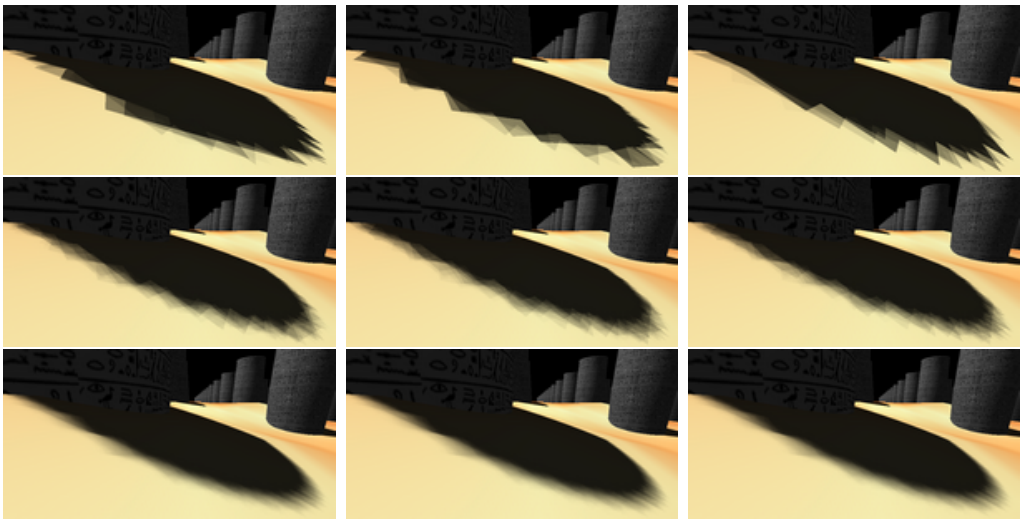


Figure 3.4: *The effect of exponential smoothing over time onto a shadow map that is resampled each frame. The* top *line uses weight 0.5, the* middle *line 0.1 and the* bottom *line 0.01.*

of k frames. Instead, we want to have increasing weights for more recent frames. This would let the function converge quickly to the current frame while maintaining some smoothing.

Fortunately, the commonly used *exponential smoothing* method circumvents these two drawbacks and is quite fast to compute. It works iteratively and allows adjustable weights:

$$s(n) = w * f(n) + (1 - w) * s(n - 1) \qquad 0 < w \leq 1 \tag{3.2}$$

Here w is a weight and $s(n-1)$ is the result of the previous evaluation. w allows balancing fast adaption of s to changing input parameters against temporal noise of the function (see Figure 3.4). With increasing w, $s(n)$ depends more on the result of the current frame function and less on older frames and vice versa. For the effect of exponential smoothing on a function see Figure 3.3. The name exponential smoothing comes from the exponential falloff of the influence of older frames implied by the recursion. $s(n)$ needs to be defined for each screen pixel

$$s_{x,y}(n) = w * f_{x,y}(n) + (1 - w) * s_{x,y}(n - 1). \tag{3.3}$$

So for instance $s_{1,1}(n)$ would be the smoothed function of the view port pixel at coordinates $(1, 1)$ taken at frame n. All the $s_{x,y}(n)$ together define the so-called *history buffer*. This buffer is simply a screen size array that allows us to capture shadowing information of a potentially infinite number of old

frames with exponential falloff. Note that in practice we need to double-buffer this array, since current hardware cannot read from and write to the same buffer in the same frame. The function $f_{x,y}(n)$ is simply the *result* of the shadow map test of the current frame for each fragment. This test returns 0 for a shadowed fragment and 1 if the fragment is lit. The history buffer has a fairly small memory footprint, containing for each fragment $s_{x,y}(n)$ and the depth. The latter is needed for a correct depth buffer lookup (see Section 3.1.2 for details). Also note that the weights w can be calculated individually for each pixel ($w_{x,y}$). This will become important in Section 3.1.3, where $w_{x,y}$ will be based on the individual pixel history.

3.1.2 History Buffer Transformation

In practice, when evaluating Equation 3.3, we have to get the shadowing information from previous frames from the history buffer, namely $s_{x,y}(n-1)$. The history buffer represents shadowing information for 3D fragments in screen space of the previous frame $n-1$. Consequently, if the camera moves, for every currently rendered fragment we have to find the corresponding position in the history buffer (i.e., the previous frame). Since we have the 3D position of our current fragment (in the post-perspective space of the current view), we can simply use the view ($\mathbf{V}$) and projection ($\mathbf{P}$) matrices and their inverses of the current and the last frame to do the transformation (back into the post-perspective space of the previous frame):

$$\mathbf{p}_{n-1_{x',y',z'}} = \mathbf{P}_{n-1} * \mathbf{V}_{n-1} * \mathbf{V}_n^{-1} * \mathbf{P}_n^{-1} * \mathbf{p}_{n_{x,y,z}} \tag{3.4}$$

Here $\mathbf{p}_{n_{x,y,z}}$ is the fragment at position (x, y) in the post-perspective space of the current frame. This fragment is transformed by $\mathbf{P}_n^{-1}$, the inverse projection matrix of the current frame, $\mathbf{V}_n^{-1}$, the inverse view matrix of the current frame (we are now in world space), $\mathbf{V}_{n-1}$, the view matrix of the previous frame and finally by $\mathbf{P}_{n-1}$, the projection matrix of the previous frame. After homogenization we are at the position the fragment would have had in the previous frame $\mathbf{p}_{n-1_{x',y',z'}}$. For moving objects, we can additionally store the object space transformation matrices or skinning matrices to do the backprojection step. Please note that all matrix transformations can be performed in the vertex shader and only the homogenization (division by the w-coordinate) has to happen in the fragment shader.

The obtained position will normally not be at an exact fragment center in the history buffer except for the special case that no movement has occurred. Consequently, filtering the history buffer for the lookup should be done. In practice, the bilinear filtering that graphics hardware offers shows good results.

An important issue is the treatment of fragments that have no corresponding entry in the history buffer. These are fragments that project to a position outside the history buffer. For instance, such new fragments occur on the left screen border in Figure 3.5 after a translation to the left. Similarly, fragments

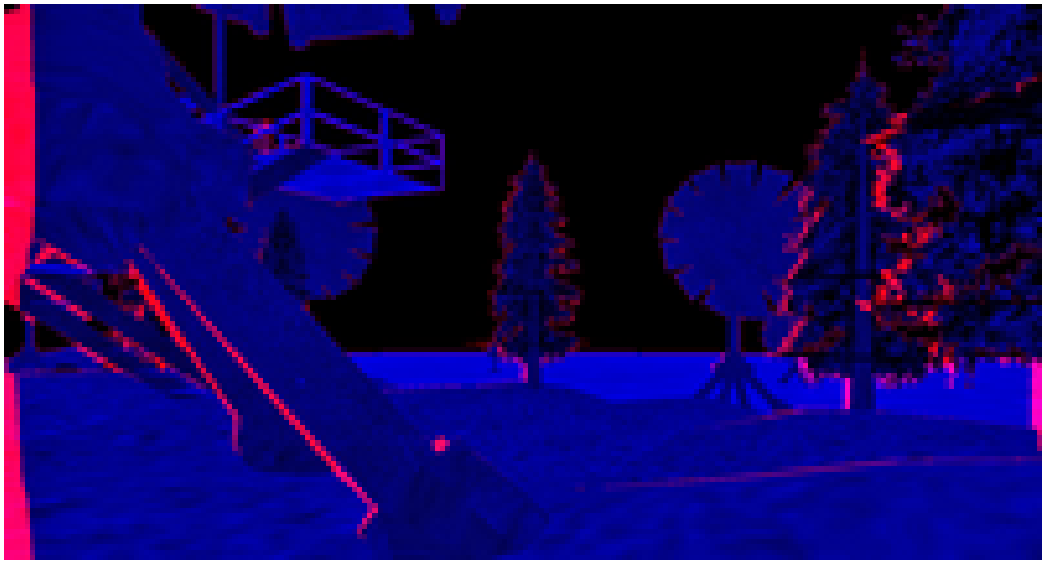


Figure 3.5: *New fragments (shown in red) without history that result from a camera translation.*

might be missed in the history buffer due to disocclusions if new scene parts appear behind occluders (as for the trees in Figure 3.5). In order to detect such cases, we check the depth difference between the current fragment and the corresponding history buffer entry.

$$\left| depth(\mathbf{p}_{n-1_{x',y',z'}}) - depth(s_{x,y}(n-1)) \right| < \epsilon \tag{3.5}$$

If this distance exceeds a certain threshold ϵ, we conclude that this fragment is new and therefore has no history. For all new fragments the original unsmoothed shadowing function is applied $s_{x,y}(n) = f_{x,y}(n)$.

3.1.3 Confidence-Based History Buffer Updates

While temporal smoothing with the history buffer reduces temporal aliasing, shadow edges are only smoothed while movement is taking place without improving their accuracy. To improve shadow edge accuracy, we take the *confidence* of a sample into account, which can be motivated as follows: The use of a simple shadow map test as source function $f_{x,y}(n)$ for the history

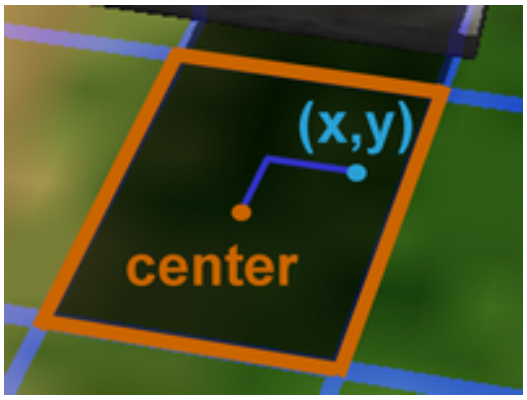


Figure 3.6: *An undersampled shadow map texel in screen-space: We define the confidence in the shadowing result at a given fragment at position* (x, y) *as* $1 - max\big(\,|x - center_x|\,, |y - center_y|\,\big) * 2$.

buffer shows the following property: If an eye fragment transformed into light space is *exactly* at the center of a shadow map pixel, the corresponding shadow map test result is the correct shadowing solution for that eye fragment. We use this property to associate with each shadow map test result a *confidence*, which is a measure for the correctness of the test in dependence on the max-norm distance of the transformed eye space fragment to the nearest shadow map sample:

$$conf_{x,y} = 1 - max\big(\,|x - center_x|\,, |y - center_y|\,\big) * 2, \tag{3.6}$$

where $conf_{x,y}$ is the confidence at the fragment position (x, y) and the pixel center is given by $(center_x, center_y)$ (see Figure 3.6). If we assume a sample size of 1, we get a highest max norm distance 0.5, which explains the factor 2.

The central idea of this work is now to use this shadow map confidence as input for the weight w for the history buffer update (Equation 3.3). In combination with the constantly updated history buffer, this allows us to compute pixel exact shadows even with very low resolution shadow maps, provided we have *different rasterizations* of the shadow map in each frame. The reason this works is that each new shadow map sample influences the current shadowing result $s(n)$ only if it has a high confidence. As long as different rasterizations are produced, it is very likely that a "good" shadow test result appears, and this will have a high influence on the shadow result, whereas "bad" shadow test results only have little influence.

Different rasterizations are provided by sub-pixel jittering the light space projection window in the light view plane based on a pseudo random sequence (Halton). Interestingly, translational jittering alone does not provide the required rasterization randomness. This is probably caused by the fact that

Figure 3.7: *Shadow adaption over time after 1* (top-left)*, 20* (top-right)*, 40* (bottom-left) *and 60* (bottom-right) *frames.*

translational jittering may or may not lead to a different rasterization of the shadow edges. Consequently, we use an additional plane rotation (with the rotation angle being a member of the Halton sequence), which ensures completely different rasterizations of shadow edges for each frame.

We apply a power function to the confidence

$$w_{x,y} = conf_{x,y}^{m}, \tag{3.7}$$

obtaining a single intuitive parameter m to balance fast history buffer adaptation against temporal noise, as mentioned in Section 3.1.1. If m is chosen relatively low (around 3), the history buffer (and thus shadowing) adaption performs quickly, but some temporal noise remains. If m is chosen relatively high (about 15), the history buffer adapts slowly, but shows no temporal noise. In this case, it is strongly biased towards more accurate shadow map samples, converging to practically pixel exact shadows. The temporal noise is visible as unsharp shadow edges that vary over time. In practice we use a low value of m if the camera is moving and start increasing m as soon as it stands still. This lets the shadow map adapt similarly to the eye, making the convergence practically invisible if the framerate is above 30Hz (see also Figure 3.7).

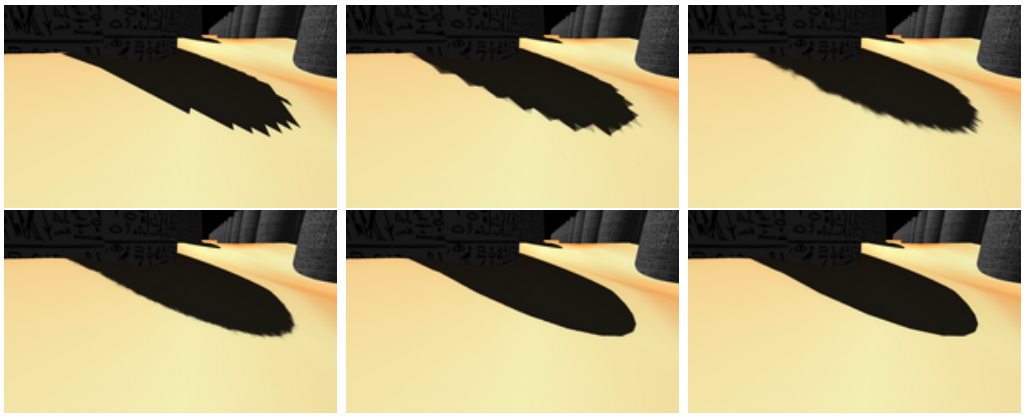


Figure 3.8: *Shadow adaption over time of an undersampled uniform shadow map after 0* (top-left)*, 1* (top-middle)*, 10* (top-right)*, 20* (bottom-left)*, 30* (bottom-middle) *and 60* (bottom-right) *frames.*

3.2 Implementation and Results

With vertex and pixel shader support, the method is efficient and simple to implement: the reprojection matrix is multiplied with the post-perspective vertex position in the vertex shader and interpolated. The only thing left to do in the pixel shader is to use this position for the lookup in the history buffer.

All images were taken with a shadow map resolution of 1024^2. Frame buffer objects were used to render the history buffer in 16bit floating point format. Depth was stored with the same precision. Two instances of the history and depth buffer are required (one for reading and one for writing), resulting in an overall memory requirement of only 4MB for a screen resolution of 1024^2. Frame buffer and history buffer are written in one pass using the multiple render target functionality. Minor speed penalties of the method occur primarily in the fragment shader, where a read and a write to the history buffer is necessary. On a Pentium4 3.2GHz with an NVIDIA GeForce 8800GTX graphics card, for a 1024^2 window we measured an average frametime of 15ms for the scene shown in Figure 3.10, of which about 1.5ms is the overhead incurred by our method.

All images shown of our method use a value of $m = 15$ for the weight calculation. For Figure 3.7 we used $m = 3$ while moving and increase m each frame of non-movement by 0.1. Figure 3.9 shows the convergence properties of our method when standing still for varying strategies of choosing m for example viewpoints. The y-axis shows the total pixel error as the percentage of pixels differing from the fully converged solution, counting only pixels with

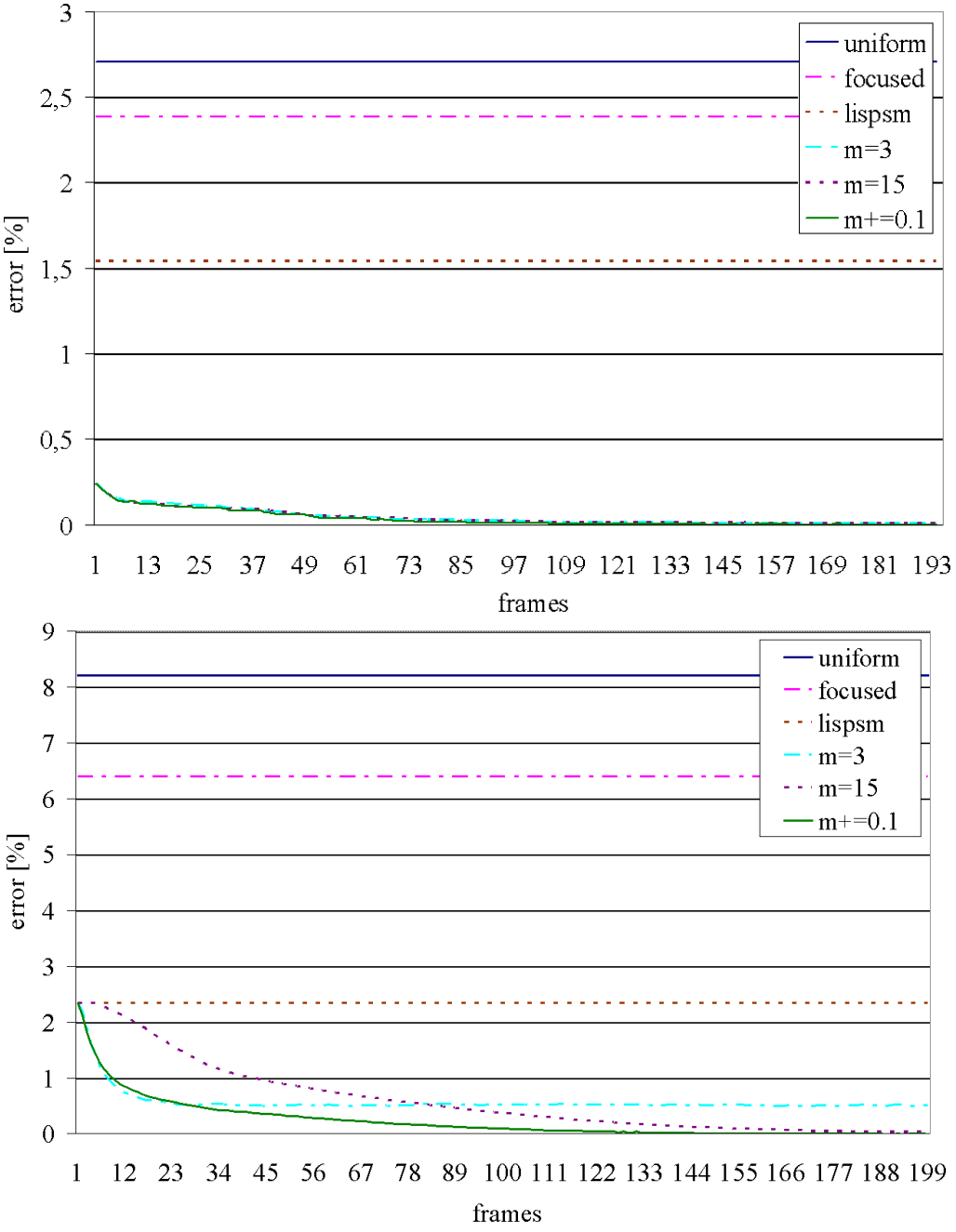


Figure 3.9: *This figure shows the convergence for two example viewpoints while standing still for different strategies of choosing* m*. The* upper *graph shows the behavior after a preceding coherent rotation of the camera (filled history buffer) and the* lower *graph shows the behavior when starting with an empty history buffer.*

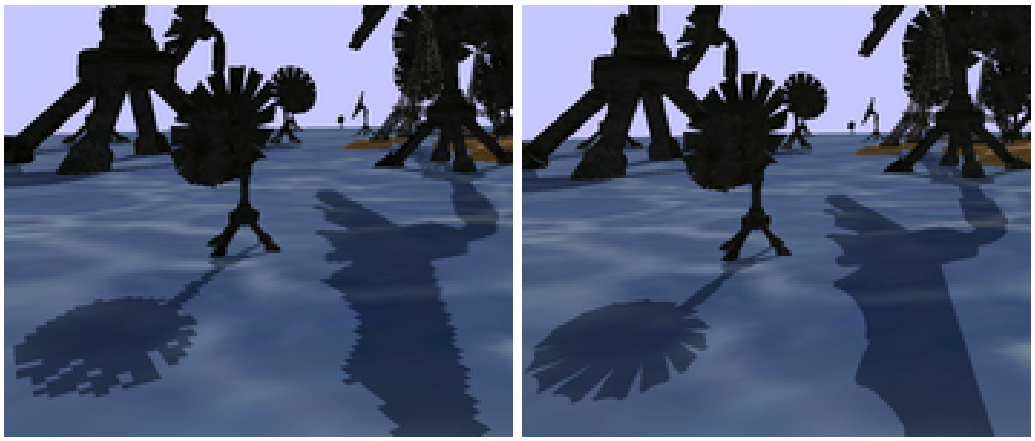


Figure 3.10: *The famous dueling frusta case where reparameterizations of the shadow map can only provide similar results as uniform shadow maps. On the* left*: light space perspective shadow maps; and on the* right*: our new method.*

a maximum difference greater than 30 in RGB (with a range from 0–255). The x-axis is in frames. At frame 0 we already have an error of only 0.25% with preceding coherent camera movement, quickly converging to 0.1% error in 20 frames. Typically this error is not noticeable anymore. Figures 3.10, 3.13 and 3.11 show comparisons of light space perspective shadow maps and our method. Note especially in Figure 3.10 and 3.13 the dueling frusta case. As Figure 3.11 shows, using PCF-filtered lookup results for $f_{x,y}(n)$ creates smoother shadow borders. If we use a largely undersampled shadow map representation, as for example a uniform shadow map in a large scene, convergence to the pixel exact solution is slow (see Figure 3.8). We greatly increased the convergence by using light space perspective shadow maps, which already reduce perspective aliasing errors. With this we can provide the pixel exact solution in a few iterations (about 10 to 60). Figure 3.7 show the convergence for an example viewpoint. It is important to note that through the exponential smoothing, we have already removed all flickering artifacts, and the convergence is extremely smooth. In our experiments, the pixel exact solution is quickly reached when an observer stops to investigate shadow map edges in detail.

3.2.1 Limitations

The main limitation of our method are dynamic scenes:

- *Moving shadows:* A moving light source is not handled by our approach because this would mean that also the light space itself would change

each frame. This in turn results in a change of the lighting solution at all shadow edges. This is exactly that part of the shadow where we can increase accuracy by using the history of a fragment. In this case there is no temporal coherence at this fragment because the state of the fragment changes abruptly from being part of the shadow to being lit. Therefore, there is no easy way of using our confidence based approach in this case. But if we accept some amount of shadow blurring (similar to normal object motion blur, but here for the shadow), we can still use temporal smoothing, by using the amount of movement as an indicator for the weight. More movement should result in more weight attributed to the new shadow result. The case of a moving shadow caster is similar to the case of a moving light source and can be handled the same way.

- *Moving objects:* A simpler case is if the shadow caster for a fragment is static, but the receiver moves. Note that the possibly changed z value (between the current frame and the previous frame stored in the history buffer) should be accounted for by using the transformation matrix of the receiver from the previous frame, when accessing the history buffer. Inside a shadow we can use our approach unchanged. More interesting is the case when a fragment on a receiver moves out or into a shadow. Here again the lighting solution changes abruptly at the shadow edge invalidating the history. Therefore, a similar approach as noted above for moving shadows should be taken.

3.3 Conclusions

In this chapter we presented a new shadowing method based on shadow maps that produces pixel accurate shadows in real time (see Figure 3.12). It resolves perspective and projective aliasing, which are present in all existing real-time shadow algorithms. Furthermore this shadow method creates temporally smooth shadow transitions, thus resolving temporal aliasing and flickering artifacts that are a major problem of shadow mapping. In addition, the algorithm is simple to implement and to integrate into existing rendering engines. It works together with advanced filtering methods like percentage closer filtering and more advanced shadow mapping techniques like perspective or light space perspective shadow maps, and has practically no performance overhead over standard shadow mapping.

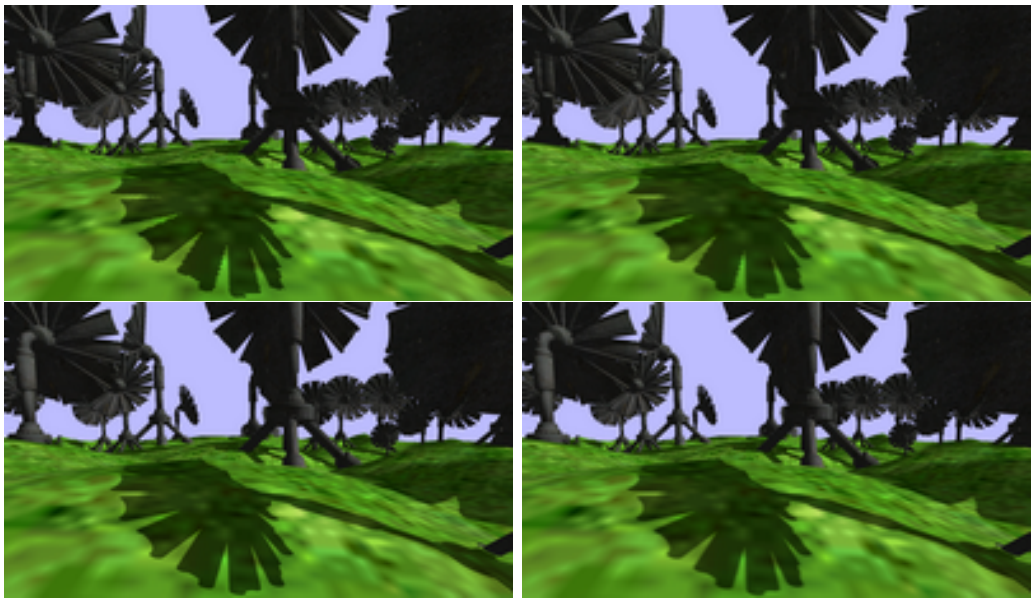


Figure 3.11: *Light space perspective shadow maps (top-left). Our new pixel correct shadow method (bottom-left). To the right both methods with 3x3 PCF filtering to produce fake soft shadow borders. LispSM (top-right) and our pixel correct shadow method below. Please note that not even PCF filtering can hide the projection aliasing artifacts and false inter-object shadows present in the scene shadowed with LispSM.*

3.4 Extensions

An important extension we want to study is the application of this method to dynamic scenes. Our optimizations of m distinguish only between a moving or a still observer. This model has to be adapted to include moving objects and light sources.

Another promising area of research is to further refine our shadow map test confidence estimator and the weighting distribution function connected with it.

The basic idea presented in this chapter has been extended by us to soft shadows in Chapter 4. Here the jittering has to be applied to positions on the area light source to create physically correct soft shadows. This approach includes the real 3D visibility solution for area/volume light sources.

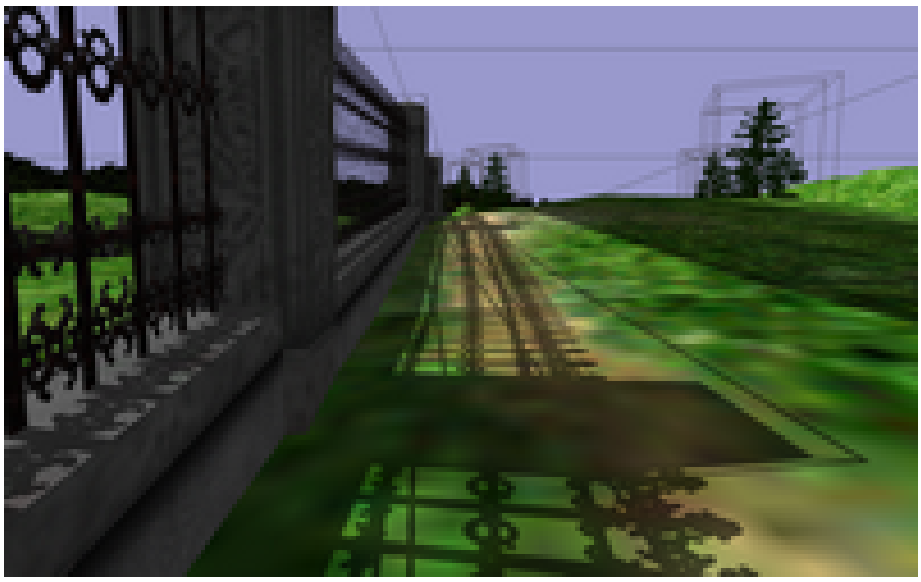


Figure 3.12: *Results of our new algorithm applied to a game scene. Please note that even the edges of bounding boxes and alpha textures like the fence cast perfect shadows with a shadow map with a resolution of only* 1024^2.

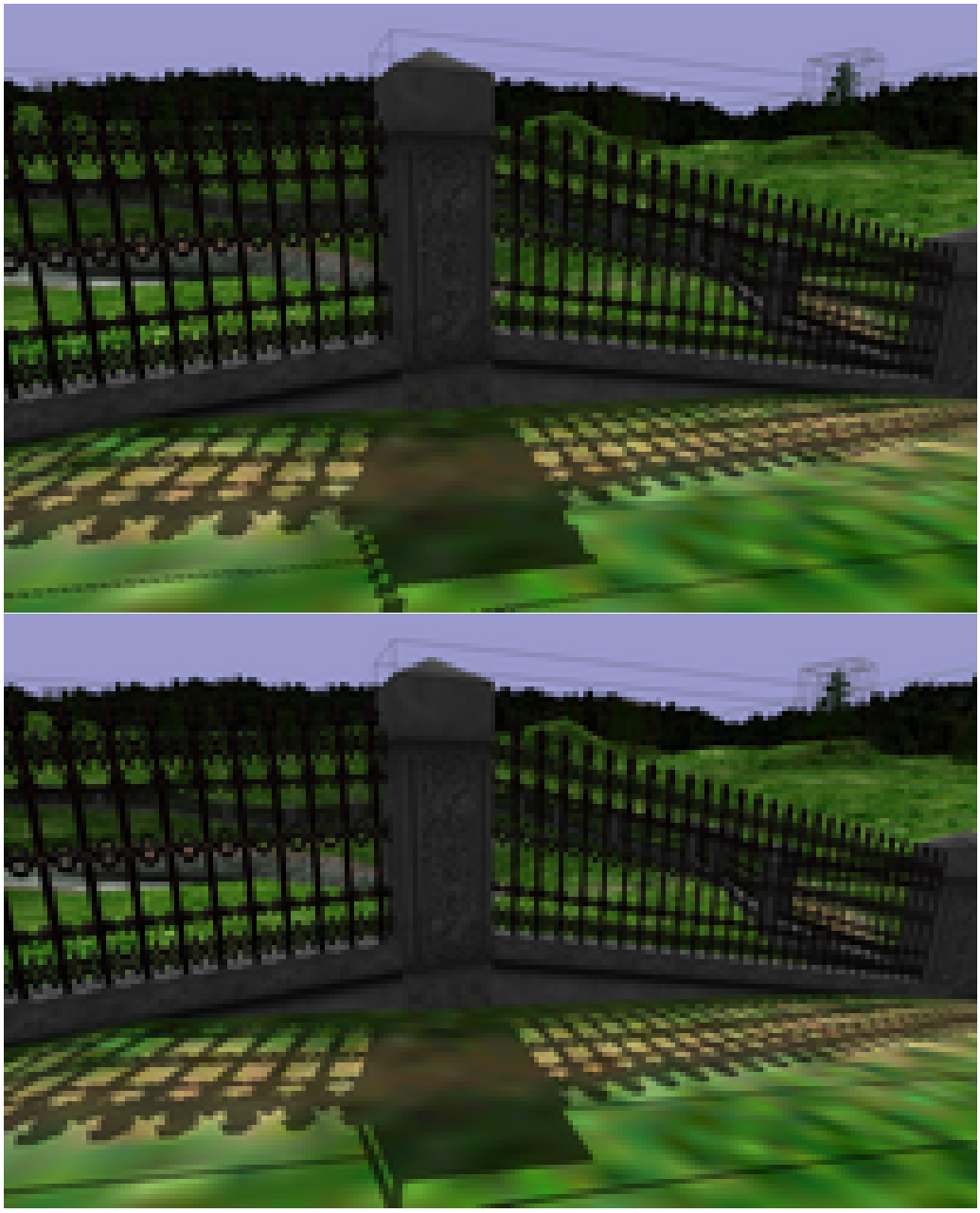


Figure 3.13: *In this figure we used our new shadow map technique to show that even scenarios where projection aliasing is the dominant source of error* (top)*, we can produce the correct shadowing solution* (bottom)*.*

4

Physically Correct Soft Shadows

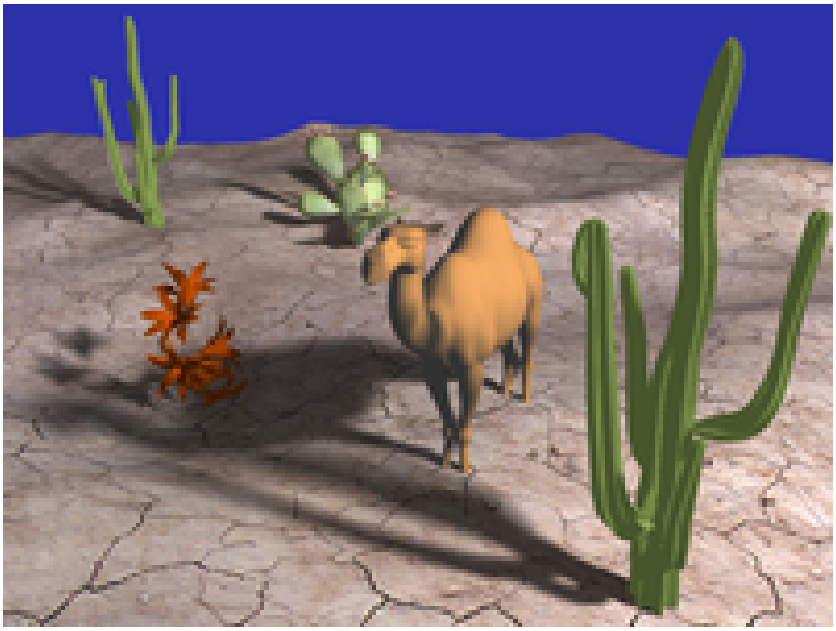


Figure 4.1: *This image was rendered with the soft-shadow method presented in this chapter and shows the difficult case of overlapping occluders. The scene consists of 70k triangles and runs at 344 FPS.*

Shadows are widely acknowledged to be one of the global lighting effects with the most impact on scene perception. They are perceived as a natural part of a scene and give important cues about the spatial relationship of objects. In reality most light sources are area light sources and hence most shadows exhibit soft borders. We should therefore consider using soft shadows for the realistic shadowing of a scene (see also Figure 4.1 and Figure 4.2). Soft shadows consist of an umbra region where the light source is totally invisible and a penumbra region where only part of the light source is visible.

Typical soft shadowing methods for real-time applications approximate an

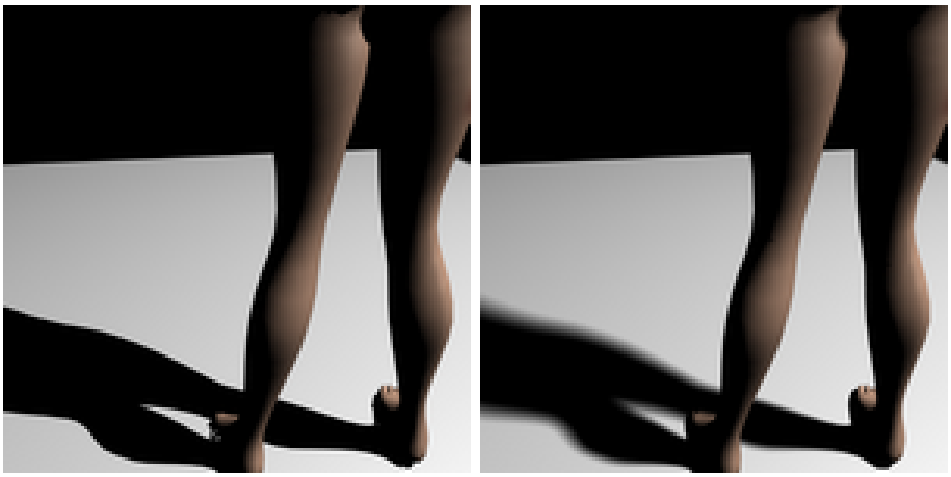


Figure 4.2: *Hard (*left*) versus soft (*right*) shadows: The blurriness of a soft shadow increases the farther the shadow caster and receiver are apart.*

area light by a point light located at its center and use heuristics to estimate penumbrae, which leads to soft shadows that are not physically correct (see the related work in Section 2.3 for details).

As the human eye is not very sensitive to the correctness of soft shadows, the results can be acceptable from a perceptual point of view or can even be physically plausible. Additionally most inherent shadow map artifacts like aliasing are often hidden through the low frequency soft shadows. Nevertheless some perceptually concerning artifacts remain: Overlapping occluders can lead to unnatural looking shadow edges, or large penumbrae can cause single sample soft shadow approaches to either break down or become very slow (see Figures 4.3, 4.9).

Accurate methods use light source sampling: The idea is to calculate soft shadows by sampling the area of the light source. Hard shadow calculations are performed for every sample and the results are combined [HH97]. The primary problem of these methods is that the number of samples to produce smooth penumbrae is huge. N samples produce $N - 1$ attenuation levels. High-quality soft shadows need 256 or more to create the 256 attenuation levels available with an $8Bit$ color channel (see Figure 4.4). If we have to render a shadow map for each sample, real-time frame rates become unlikely, even for simple scenes. But under the assumption of a light source with uniform color and dense enough sampling of the light source, the result of these approaches are correct soft shadows.

Our approach can be described as a combination of light source area sampling over time and single sample filtering:

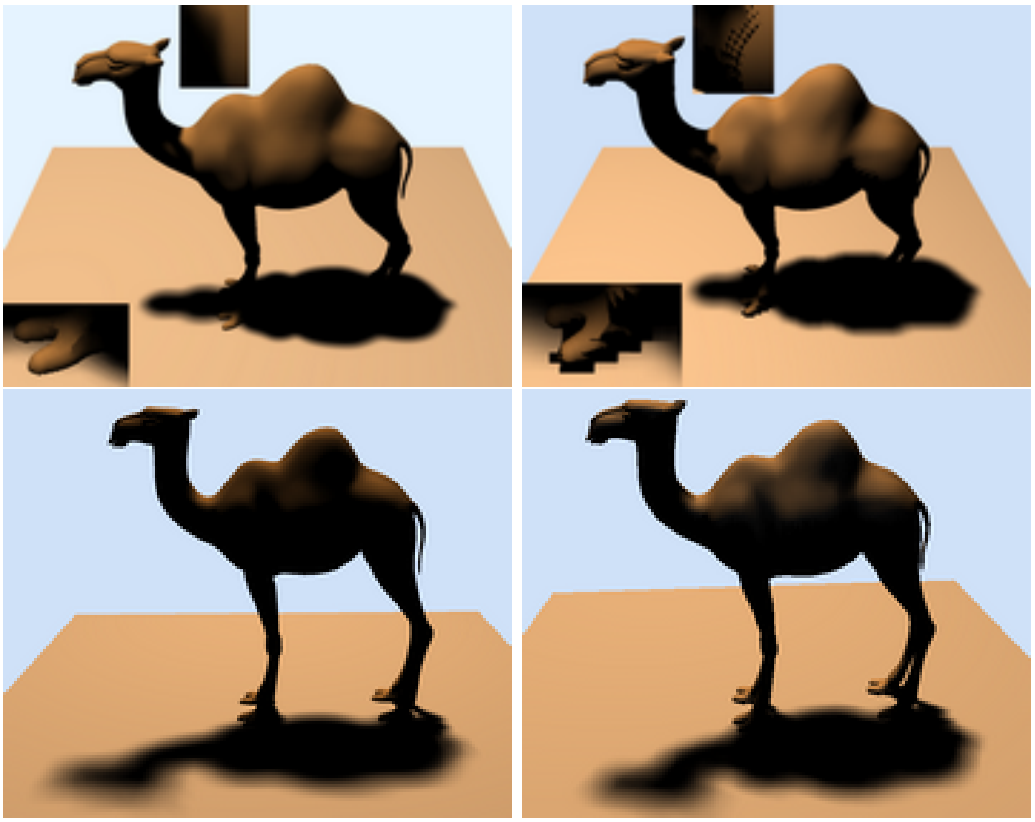


Figure 4.3: Left side*: Our Method (634FPS).* Right Side*: Bitmask Soft Shadows* upper*: 8x8 search area (156 FPS),* lower*: 12x12 search area (60FPS). Even very good single sample soft shadow methods show some artifacts, like biasing problems and contact shadow undersampling that can be avoided by using multiple samples.*

- The area sampling is done one sample per frame by creating a shadow map from a randomly selected position on the area light. For each screen pixel the hard shadow results obtained from this shadow map are combined with the results from previous frames (accumulated in a screen space buffer called the *shadow buffer*) to calculate the soft shadow for each pixel.
- When a pixel becomes newly visible and therefore no previous information is available in the *shadow buffer*, we use a fast single sample approach (PCSS with a fixed 4x4 kernel) to generate an initial soft shadow estimation for this pixel.
- To avoid discontinuities between sampled and estimated soft shadows, all the estimated pixels are augmented by using a depth-aware spatial

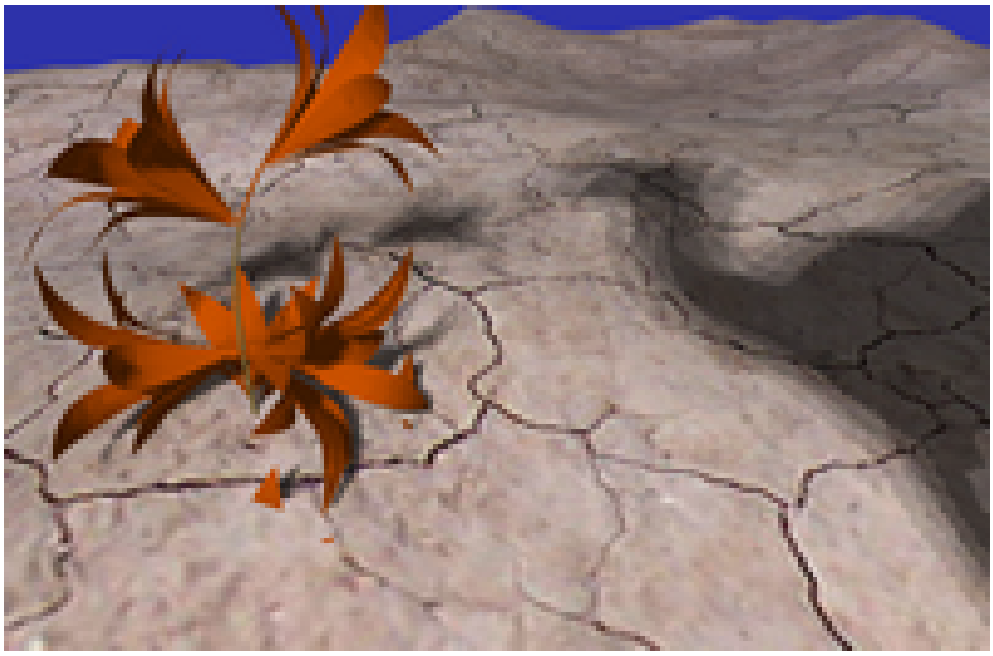


Figure 4.4: *Area sampling can lead to banding artifacts due to a insufficient number of samples. To emphasize the effect only 5 samples were used here.*

filter to take their neighborhood in the *shadow buffer* into account.

This buffer contains soft shadowing information accumulated over the previous frames for each screen space pixel. When updating a buffer pixel we start by updating the soft shadow information with the newly gathered shadow map and also check how much soft shadow information has already been accumulated. Additional spatial filtering is applied to the buffer as long as the soft shadow information is judged insufficient. For this we employ a penumbra estimation and use the soft shadow information from the neighboring pixels of the shadow buffer to allow for very fast (3-30 frames), smooth and graceful convergence to the correct solution. Finally the soft shadows can be directly rendered from the shadow buffer.

What distinguishes this approach from others is the application of *temporal coherence* (through the use of the shadow buffer) to the soft shadowing problem. This together with spatial filtering (penumbra estimation and pixel neighborhood) is combined in a soft shadow mapping algorithm that is even faster than a very fast variant of PCSS (see Section 4.2 for details), but produces *accurate real-time* soft shadows. Additionally it extends temporal reprojection to handle moving objects such as dynamic shadow casters and receivers.

The remainder of this chapter is structured as follows: We start with

Section 4.1, which explains our algorithm in detail. Afterwards our implementation and results are discussed in Section 4.2, and conclusions and ideas for future work in Section 4.3.

4.1 Our Algorithm

If we want to find the contribution of an area light source to a specific fragment, we have to calculate the fraction of the area of the light source that is visible from the fragment. For our purposes we will use the reverse value, which we call the occlusion percentage or soft shadow result $\psi(x, y)$ for a fragment at screen space position (x, y). $\psi(x, y)$ is 0 for a fragment that is illuminated by the whole area of the light source and 1 for a fragment that is not illuminated by the light source at all. Due to the fact that most calculations we perform are done per fragment and for the sake of notational simplicity, we will only use the (x, y) notation when introducing a function and will afterward omit it.

4.1.1 Estimating Soft Shadows from n Samples

To make the calculation of the soft shadow value feasible for rasterization hardware, we use sampling and shadow maps. We approximate an area light source by n different point light sources and compute a shadow map for each of them. A shadow map allows us to evaluate for every screen space fragment if it is illuminated by its associated point light.

$$\tau_i(x, y) = \begin{cases} 0 & \text{lit from point light } i \\ 1 & \text{in shadow of point light } i \end{cases} \tag{4.1}$$

$\tau_i(x, y)$ is the result of the hard shadow test for the ith shadow map for the screen space fragment at position (x, y). Under the assumption that our point light placement on the area light source is sufficiently random, the soft shadowing result ψ (i.e., the fractional light source area occluded from the fragment) can be estimated by the proportion $\hat{\psi}_n$ of shadowed samples

$$\hat{\psi}_n(x, y) = \frac{1}{n} \sum_{i=1}^{n} \tau_i(x, y) \tag{4.2}$$

This mean of n shadow map lookups is extensive to calculate in practice, but it is an approximation of the soft shadow result, that becomes more and more accurate with increasing n. Furthermore, we are interested in the quality of this approximation, which we can measure using the variance. The number of

shadowed samples $n\hat{\psi}_n$ has a Binomial distribution with variance $n\psi(1-\psi)$. We can thus give an unbiased estimator for the variance of the proportion $\hat{\psi}_n$ as

$$\hat{var}(\hat{\psi}_n(x,y)) = \frac{\hat{\psi}_n(x,y)(1-\hat{\psi}_n(x,y))}{n-1} \tag{4.3}$$

The importance of Equation 4.3 is that it allows us to estimate the quality of our soft shadow solution after taking n samples. We will later use this estimate (the standard error derived from this estimate) to judge if sampling alone will give sufficient quality. Table 4.1 shows these formulas applied to some real-world τ_i and increasing sample sizes. Please note that although the standard error decreases when the sample size is increased, the estimator for the standard error is not guaranteed to do so.

Table 4.1: *Evaluation of the presented formulas for one fragment. Increasing the sample size generally reduces variance and standard error,* $\hat{s} = \sqrt{\hat{var}}$

n	1	2	3	4	5	6	7
τ_n	1	0	1	1	1	0	1
$\hat{\psi}_n$	1.00	0.50	0.67	0.75	0.80	0.67	0.71
$\hat{var}(\hat{\psi}_n)$	0	0.25	0.11	0.06	0.04	0.04	0.03
$\hat{s}$	0	0.50	0.33	0.25	0.20	0.21	0.18

4.1.2 Temporal Coherence

We want to be able to solve Equation 4.2 iteratively, so that we only need to create one shadow map each frame. We will then use the *temporal coherence* of the current frame with previous frames to increase convergence.

The *temporal coherence* method introduced by Scherzer et al. (see Chapter 3) improves the resolution of standard hard shadow maps. It is based on the assumption that most screen space fragments stay the same from frame to frame. It can be determined which fragments of the new frame have also been present in the last frame by reprojecting (to account for camera movement) fragments from the new frame into the old and comparing their respective depths. If the depth difference is smaller than a predefined ϵ, the two fragments are considered equal and therefore fragment data from the previous frame is reused. If $d_k(x,y)$ is the depth of the fragment at position (x,y) of the current frame and $d_{k-1}(x,y)$ is the same for the previous frame,

then the test for fragment equality is given by

$$|d_k(x,y) - d_{k-1}(x,y)| < \epsilon \tag{4.4}$$

If on the other hand the difference is greater, the fragment was not present in the last frame and is therefore new (disoccluded) and no previous data for this fragment is available.

For our approach we want to be able to compute $\hat{\psi}_n$ iteratively for each frame and fragment from the information saved from previous frames together with the information gathered for the current frame. We do this by keeping $\rho_n(x,y) := \sum_{i=1}^{n} \tau_i(x,y)$ from the previous frame. ρ_n stores all the shadow map tests already performed for the n previously calculated shadow maps. We also need the sample size n, which is equal to the number of shadow map tests we have already performed for this fragment. If the fragment was occluded in the last frame, we get $n = 0$ and $\rho_n(x,y) = 0$ because no previous information is available. Therefore n can be different for each screen space fragment. We can now calculate $\hat{\psi}_{cur}(x,y)$ for the current frame as

$$\hat{\psi}_{cur}(x,y) = \frac{\tau_{cur}(x,y) + \rho_n(x,y)}{n(x,y) + 1} \tag{4.5}$$

This formula only needs access to n (the count of samples available for this fragment stored in the previous frame), ρ_n (i.e., sum of the shadow map tests up to the previous frame) and the current shadow map. We provide access to these values by storing in each frame the updated sample size $n + 1$ and $\tau_{cur} + \rho_n$ for every fragment into a screen sized off-screen buffer called the *shadow buffer* for use in the next frame. Please note that this shadow buffer is very similar to the *history buffer* introduced in Chapter 3. Now this formula can be evaluated very quickly in a fragment shader and real-time frame rates pose no problem. Note that we also have to store the depth of each fragment to be able to evaluate the test in Equation 4.4.

The disadvantage of this approach is that for newly disoccluded fragments (i.e., fragments with a small n) the results have large errors (see also Figure 4.5). This becomes clearer when we take a look at the example in Table 4.1: At $n = 2$ we have a standard error of 0.5, which means the real ψ is probably inside of 0.5 ± 0.5, so the quality of $\hat{\psi}_2$ as an estimate is really bad. A second closely connected problem that aggravates the situation even further is discontinuity in time – the difference between $\hat{\psi}_n$ and $\hat{\psi}_{n+1}$ is still large. For instance $\left|\hat{\psi}_1 - \hat{\psi}_2\right| = 0.5$ which means the soft shadow results can be 128 attenuation values apart. This is a noticeable jump and will cause flickering artifacts. In the next section we will show how to avoid this by using spatial filtering.

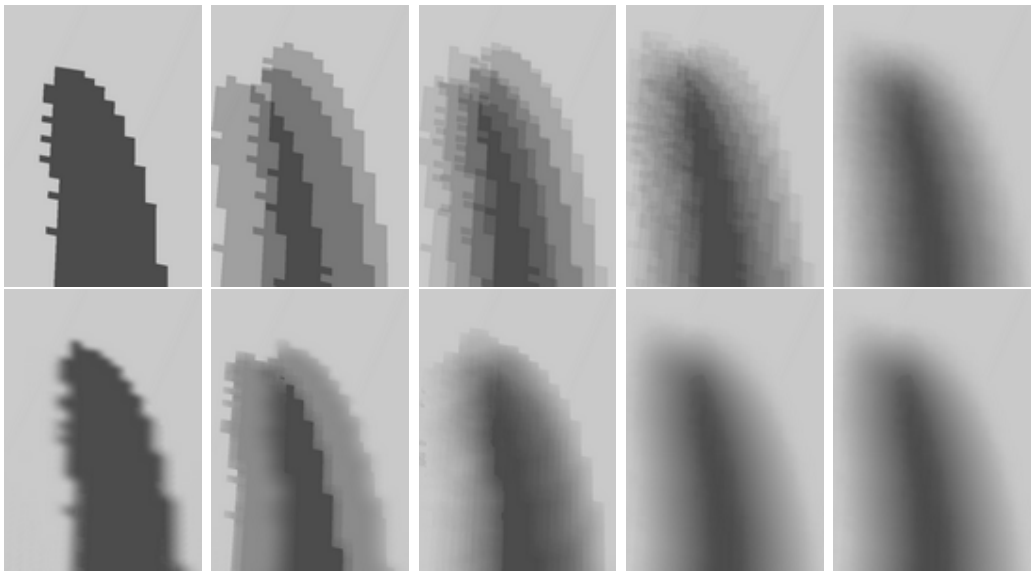


Figure 4.5: *Convergence after 1,3,7,20 and 256 frames.* Upper Row*: Sampling of the light source one sample per frame (using Equation 4.5);* Lower Row*: Our new algorithm.*

4.1.3 Spatial Filtering

Due to the use of *temporal coherence* and Equation 4.5 we have already constructed a soft shadow algorithm that takes little time to evaluate each frame, but two closely related problems remain:

- For a good estimation of ψ with $\hat{\psi}_n$ for a fragment with *temporal coherence* alone we need the fragment to be visible for many frames.

- During the frames following the disocclusion of a fragment, $\hat{\psi}_n(x, y)$ has a large standard error and may change drastically, resulting in flickering of these fragments (see also Figure 4.5, upper row).

Our first observation in this respect is that for the first few frames after a fragment has been disoccluded, a single sample soft shadow mapping approach will probably have better quality than using $\hat{\psi}_n$ (i.e., the sampled approach), which is also suggested by the high variance in this case. So our first improvement over using only Equation 4.5 is to use a very fast single sample soft shadow approach, percentage closer soft shadows, as a starting point and then refine it by using Equation 4.5 (see Figure 4.6). Note that PCSS itself is also a spatial filter.

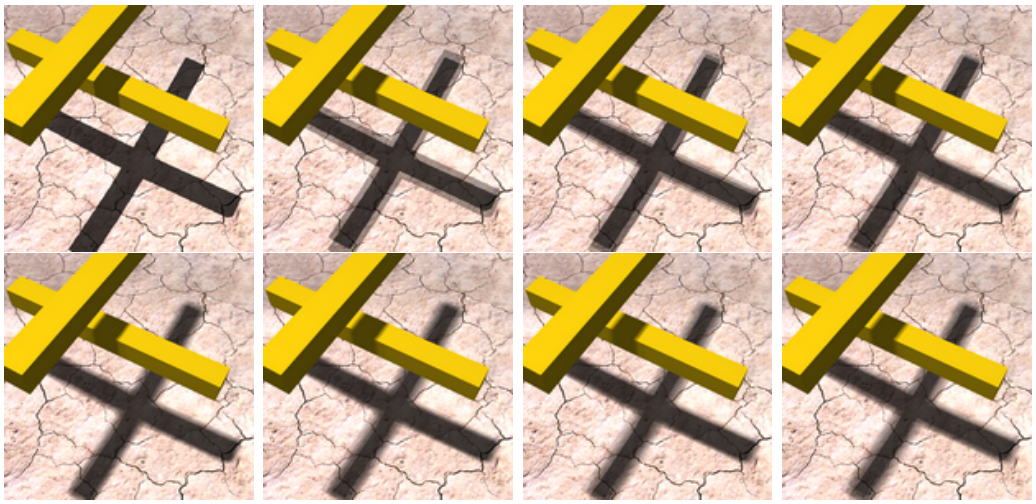


Figure 4.6: *Convergence for the first 4 frames when sampling the light source one sample per frame (*upper row*) can be greatly increased by using PCSS as its first sample (*lower row*). Here we have used $n = 4$ for the PCSS sample, so it is weighted like 4 normal samples.*

For the refinement, we have to initialize n and ρ_n for the sample generated using PCSS. The PCSS shadow test will give us a result in the range [0..1]. The most natural choice is here to use this result directly as ρ_n and set $n = 1$. Note that it can make sense to use higher values for n, meaning that the PCSS sample will be of greater weight than the following "normal" samples. Using bigger values for n can lead to faster convergence if the PCSS result is near to the correct solution (see Figure 4.6). This approach can considerably shorten the period a fragment has to be visible to achieve a good estimation of ψ with $\hat{\psi}_n$. Please note that other single sample soft shadow approaches could also be used together with our sampling approach. We have chosen PCSS mainly for its speed.

For small n, each new sample potentially causes drastic changes to the estimated soft shadow solution. On the one hand these changes need to happen to guarantee a swift convergence, but on the other hand we want to avoid the resulting flickering artifacts in the rendered shadows. Therefore we propose to introduce an additional smoothing by using a *neighborhood filter* in screen space just for the rendered shadows, without any impact on the soft shadow information (namely $n + 1$ and ρ_{n+1}) stored in the shadow buffer for the next frame (see Figure 4.5, lower row for results). Section 4.1.3 will describe in detail how this filter is constructed.

We still have to decide when to apply the neighborhood filtering. For this we use the standard error of the variance estimator of Equation 4.3. This gives us an estimated error for our sampling approach. We choose an error threshold t down to which neighborhood filtering will be used. If the error is smaller, only sampling will be applied.

Neighborhood Filter

The neighborhood filter should remove noise and flickering artifacts in the resulting shadow by smoothing. We use a box filter and only include samples within a certain depth range of the current fragment, since those are likely to have a similar $\hat{\psi}_n(x, y)$. The main question is how to set the filter width to achieve the correct amount of smoothing. Since we want to render soft shadows, a good filter size is given by the penumbra width.

Although we don't know the exact penumbra width, we can estimate it efficiently. We base our estimation on the one used in Fernando [Fer05] because it is fast and simple. It assumes blocker, receiver and light source are parallel and is given by

$$pw = \frac{near}{depth_{eye}} \frac{receiver - avg(blocker)}{avg(blocker)} light_{size} \tag{4.6}$$

where pw denotes the penumbra width (projected to screen space) we want to estimate, $receiver$ is the depth of the current fragment and $light_{size}$ is the size of the light source. $near$ is the near plane distance and $depth_{eye}$ the fragment depth for the projection.

The calculation of the average blocker depth $avg(blocker)$ is one of the costliest steps in this algorithm. It works by averaging the depths of the k nearest texels in the shadow map for each fragment each frame. In our approach, we can avoid doing this by using *temporal coherence* again: When a fragment first becomes visible we perform PCSS, including blocker depth estimation, anyway. We save this value $avg(blocker)$, generated from k depth samples, as a starting value, and in each successive frame we refine it using one additional depth sample $depth_i$ from the new shadow map:

$$avg(blocker)_i = \frac{depth_i + avg(blocker)_{i-1}(i - 1 + k)}{i + k} \tag{4.7}$$

4.1.4 Accounting for Moving Objects

The original temporal reprojection paper by Scherzer et al. (see Chapter 3) does not account for moving objects and up to now we have only accounted

for camera movement by reprojection. If we want to be able to display moving objects that cast and receive soft shadows, we have to investigate how these influence the evaluation of our algorithm. Moving objects will appear in the shadow map as potential casters of dynamic shadows and also in the scene as dynamic objects where shadows are cast upon. With moving objects disocclusions become very frequent and therefore temporal reprojection does not work as well.

Our approach is therefore to first identify moving objects and shadows that are cast by moving objects in order to handle these cases. First we change the generation of the shadow map by including for each texel the information whether it belongs to a moving object or not. Because the depth in the shadow map is always positive we can store this inside the depth by using the sign, therefore generating no additional memory cost. A negative sign means that this fragment is from a moving object (and therefore a potential dynamic shadow caster). We can now retrieve this information for each screen-space fragment when we do the shadow map test. If we have a negative depth we know that the shadow that falls on this object is cast by a moving object.

If we detect such a case we must assume that the data stored in the shadow buffer for the current fragment is probably invalid and we therefore handle it like a normal disocclusion and apply PCSS. This alone would lead to unsatisfactory results because we generate a different shadow map each frame on a different position on the light source. This would cause the PCSS shadow to jump around each frame. Therefore we additionally apply the neighborhood filter from Section 4.1.3 to smooth out any jumps and decrease discontinuities between the primarily PCSS based moving shadows with the sampled static shadow.

4.2 Implementation and Results

We implemented the algorithm in 3 passes:

1. Render shadow map

2. Render into new shadow buffer (applying algorithm) and final color buffer; use shadow buffer from previous frame as input

3. Copy final color buffer to framebuffer

For our tests, we used an Intel Core 2 Duo E6600 CPU with an NVIDIA 280GTX graphics card and DirectX 10. All images were taken using shadow maps with a resolution of 1024^2. For selecting the sample position on the area

Figure 4.7: *PCSS* (left) *versus our method* (right). *Our sampling based method removes jagged edges still visible with single sample soft shadow methods.*

light we used a Halton sequence. We also tried other quasi-random sequences but they showed similar behavior. The screen space neighborhood filter uses a Poisson disk centered at the current fragment with a fixed sample size (16 samples).

The shadow maps were rendered using standard uniform shadow mapping, a 32bit floating point texture and linear depth. Hardware 2x2 PCF filtering was not used. We used the multiple render target functionality for the second pass to render into the shadow buffer and into an 8bit RGB buffer. The shadow buffer is a 4-channel 16bit floating point texture. It contains ρ_n, n, the linear depth of the fragment and the average blocker depth $avg(blocker)$. To have meaningful neighborhood texels at the frame buffer borders, we have chosen the resolution of the shadow buffer about 5% larger than the framebuffer, so if we assume a 800x600 framebuffer, the shadow buffer would be 840x630. Please note that 5% was sufficient for our movement speeds. If faster movements occur, a larger "overscan" should be choosen. Two instances of the shadow buffer are required (for ping-pong rendering), resulting in an additional memory requirement of $2 \times 840 \times 630 \times 4 \times 2 \approx 8MB$. We used an error threshold of $t = 1/50$.

We found that the test from Equation 4.4 used in Scherzer et al. (see Equation 3.5) for determining if fragment data is available from previous frames is numerically unstable, when used in conjunction with linear depth values. We use instead

$$\left| 1 - \frac{depth_{current_{xy}}}{depth_{previous_{xy}}} \right| < \epsilon. \tag{4.8}$$

Our goal was to develop a soft shadow approach that should be faster than PCSS, but provide at the same time better quality (see also Figure 4.7), therefore we compared to a fast PCSS version using only 16 texture lookups for the blocker search and 16 texture lookups for the PCF kernel. Figure 4.8 shows benchmarks of a typical walkthrough of one of our test scenes, using a viewport of 1024x768 pixels. Timings are given in milliseconds. Our algorithm tends to be slower if there are many disocclusions, because here it has to perform the blocker search that also PCSS has to perform. Our shader is more complex (more ifs) than the PCSS shader so we can be slower then PCSS in such circumstances. The used PCSS16 always performs 16+16 lookups, while our shader only has to do those 16+16 lookups for disocclusions. Our shader performs at least one shadow map lookup and one shadow buffer lookup every frame. 16 lookups are used for the neighborhood filter, which is the case when the standard error is higher then the threshold t for every fragment were the single sample soft shadow approach is active. Figure 4.9 shows typical problems of PCSS that are solved with our approach. We also did a comparison with a more elaborate PCSS with 32 lookups for the occluder search and 32 lookups for the PCF kernel, which was considerably slower than our algorithm.

4.2.1 Limitations

Although we presented a method to handle moving objects, there is still room for improvements. Especially when dynamic shadows overlap with static shadows with large penumbrae, some flickering artifacts remain.

4.3 Conclusions and Future Work

We presented a very fast soft shadow approach based on shadow maps that uses temporal reprojection for achieving physical correctness (see also Figure 4.10). Where temporal reprojection is insufficient we use spatial filtering to allow for soft shadows on recently disoccluded fragments.

As a future direction of our research we would like to investigate multiple light sources because they lend themselves naturally to this approach: nowhere do we assume that the soft shadow data in the shadow buffer comes from the same light source, so we can extend the approach to multiple light sources simply by calculating a shadow map for each light source each frame. The values rho_n and n can then accumulate contributions from all the light sources.

Moving light sources could also be possible. We think that an approach that weights older light source samples less, together with age factors for

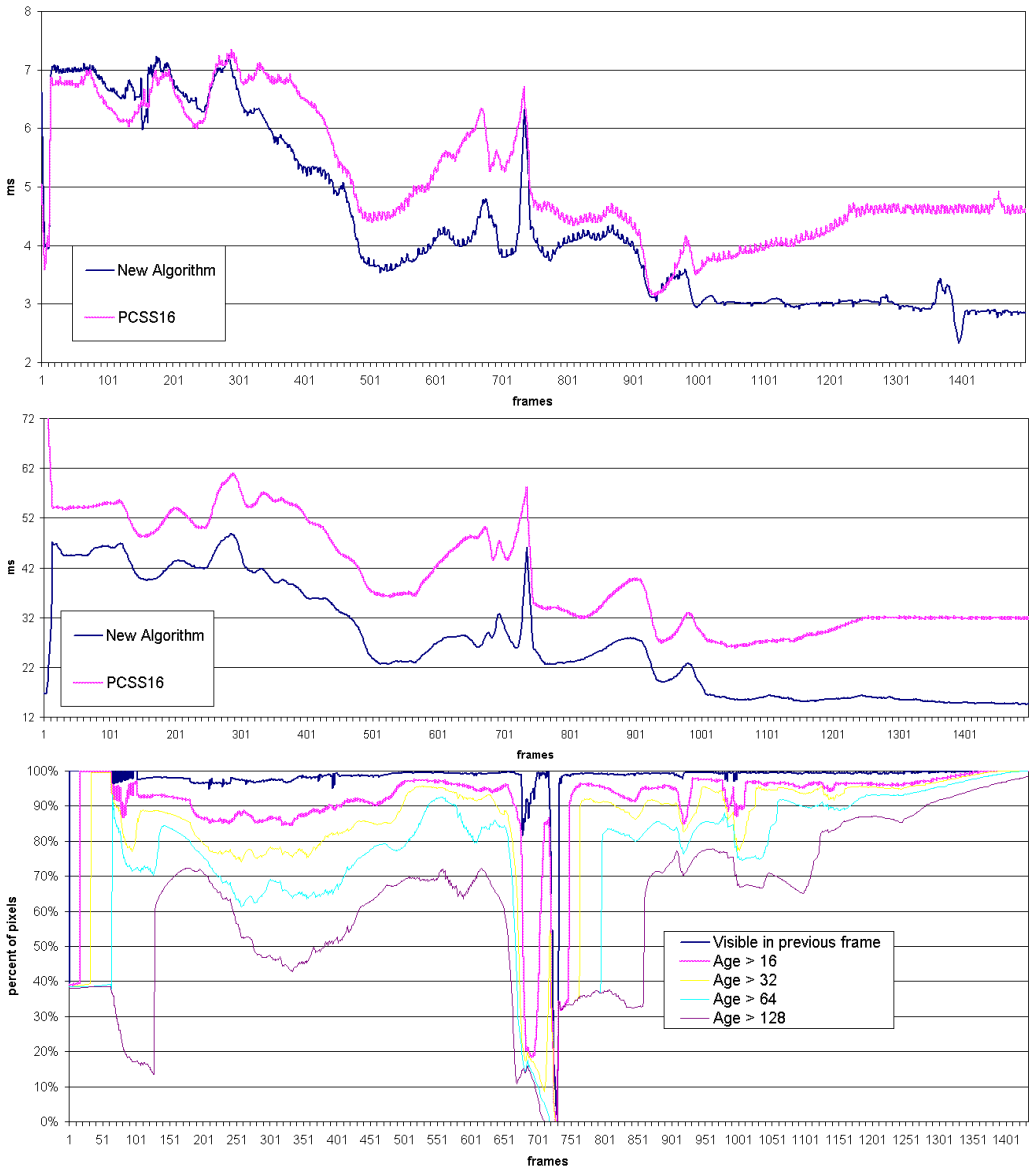


Figure 4.8: Top: *A sample walkthrough in one of our test scenes with our new method and with PCSS using 16/16 samples for blocker/PCF lookup.* Middle: *The same walkthrough, but now on a notebook with GeForce 9600M GT. Please note that the difference in speed between PCSS and our method is even bigger on this hardware.* Bottom: *Pixel age statistics for the walkthrough.*

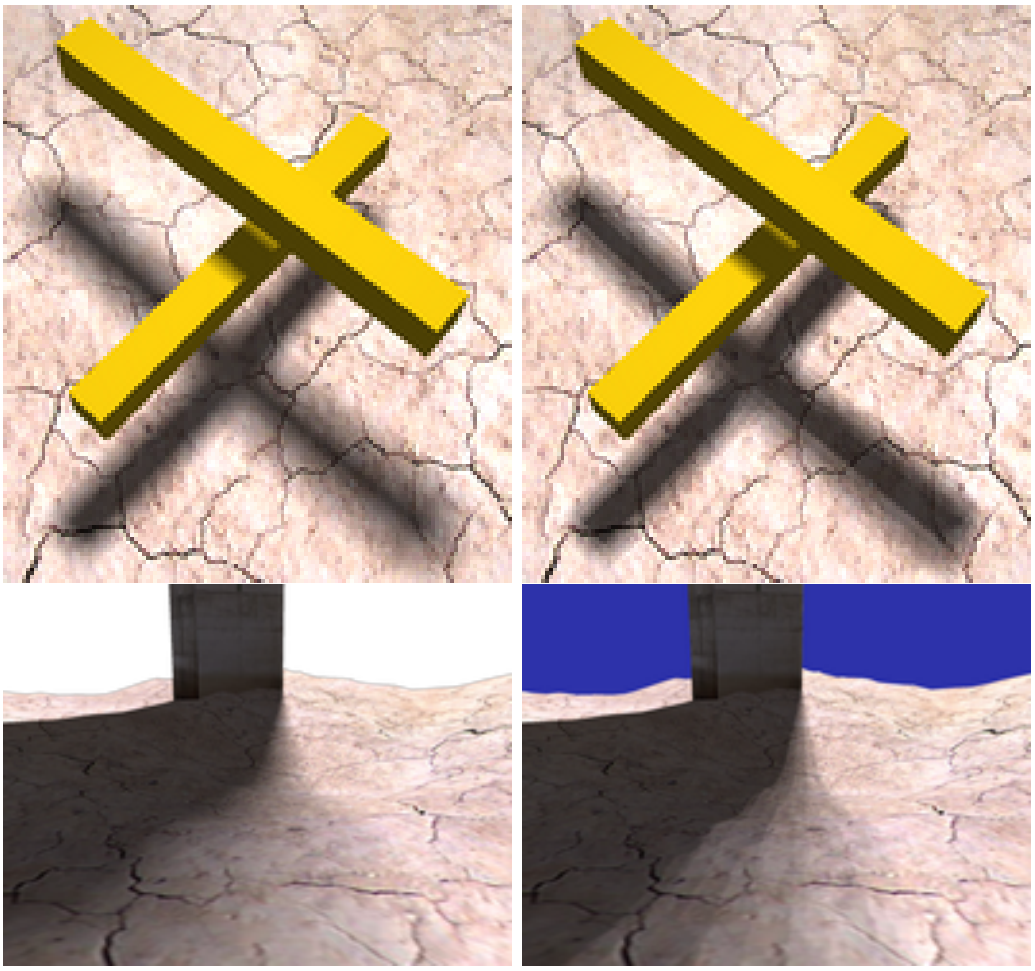


Figure 4.9: Left side*: Our Method.* Right Side*: PCSS 16/16. Overlapping occluders (upper row) and bands in big penumbras (lower row) are known problem cases for single sample approaches.*

shadow buffer fragments, could work. Moving objects would benefit from calculating each frame a second shadow map in the center of the light source. Maybe the two shadow maps could be calculated in a combined fashion. Our statistical approach is now based on uniform distributions of samples. Maybe non-uniform distributions could improve convergence.

Figure 4.10: *Overlapping occluders rendered with our method.*

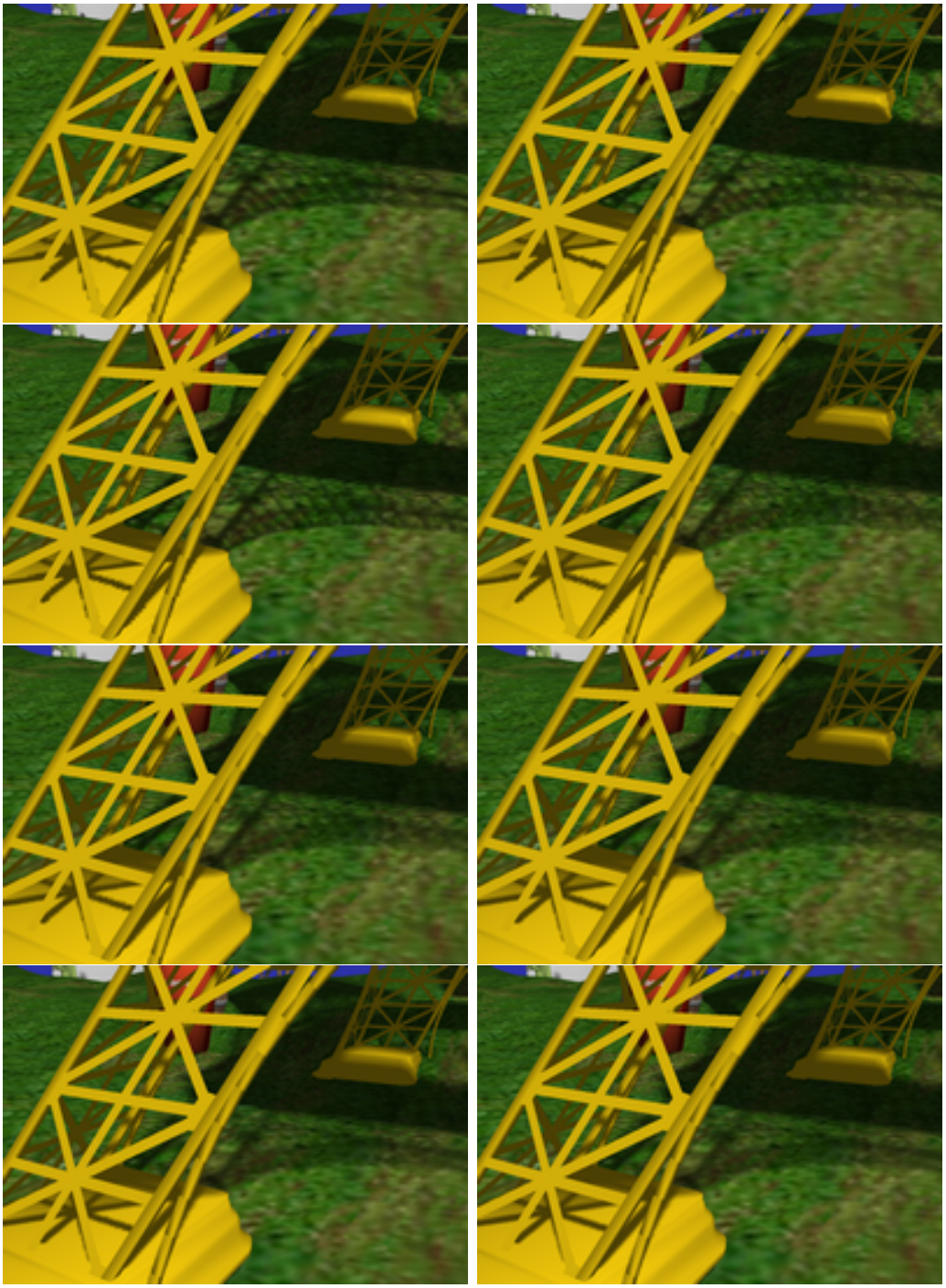


Figure 4.11: *Soft shadow adaptation after 1,2,3,4,8,13,18,38* (left to right, top to bottom) *frames starting with an empty shadow buffer.*

5
Fast LOD Blending

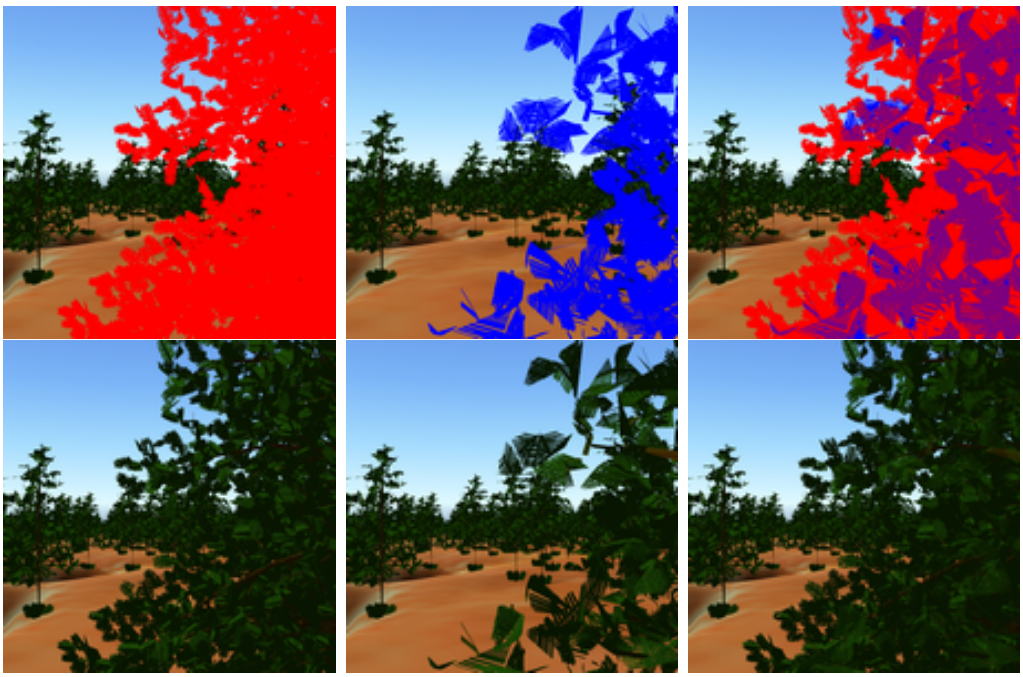


Figure 5.1: *Our technique combines two buffers containing the discrete LODs to create smooth LOD transitions.* First and second column*: buffers;* last column*: combination. The top row shows the two LODs in red and blue respectively.*

The idea of discrete level-of-detail (LOD) techniques is to use a set of representations with different levels of detail for one model and select the most appropriate representation for rendering at runtime. This selection can be based on the size of the model on screen, distance to the camera or a more elaborate metric like importance [SDC04].

The individual representations of one model can differ in a number of properties, like geometrical complexity or complexity of lighting. This can go as far as using impostors or point-based approaches for some of the

representations. Because of this diversity, automatic generation of LOD levels can often produce non-optimal results. In the industry, artists often handcraft each of the representations. This results in small numbers of LODs being used, where switching from one representation to another can lead to noticeable popping artifacts. In practice, even if the difference between two LOD levels is very small, switching is usually perceptible, because a pixel-exact match is unlikely. To be more exact this switch is noticeable exactly when the switching occurs in *supra-threshold* of the human visual system, i.e., if a (average) human can see that a switch has occurred [CS06]. Due to the detail limitations in real-time computer graphics almost all such switches are in *supra-threshold*. The reason for this is that even the LOD with the highest detail is still not detailed enough and the switches to lower detailed models should happen as early as possible.

The straightforward solution to this problem is to replace the hard switch by a transition phase. In Section 2.4.3, we have described the method by Giegl and Wimmer [GW06] which blends the two adjacent discrete levels of detail, but introduces a number of drawbacks:

1. The geometry of both representations has to be rendered in this transition phase, thereby generating a higher rendering cost than the higher quality level alone would incur.

2. Possibly complex fragment shaders have to be evaluated for the fragments of both representations.

3. State changes for enabling blending, changing geometry, depth-write mode and changing shaders necessary to blend the two representations are costly on modern architectures.

4. Where the silhouette of the two representations do not match, blending leads to semi-transparent regions of objects that should be opaque.

5. The use of blending can make it necessary to do a depth-sort of the polygons for the objects in transition. Otherwise, popping may occur due to the changing depth order of semitransparent parts (e.g., in objects represented by multiple billboards).

6. Coplanar parts of the two representations will result in z-fighting.

Due to these drawbacks, blending is not well suited for a practical implementation. In this chapter we present an algorithm that takes a completely different approach to solve the LOD transition problem. It has none of

the mentioned drawbacks while being fast, robust and straightforward to implement.

The algorithm is based on two main ideas: first, the two LODs required during an LOD transition are rendered in *subsequent frames*, thus avoiding to render two representations in a single frame. Second, we introduce the notion of *visibility textures* to replace standard blending, which allows for non-trivial interpolation between objects that can be user controlled or automatically adapted for a given object, providing more plausible transitions between two object representations than blending.

The remainder of this chapter is structured as follows: Section 5.1 explains our algorithm in detail. Results are given in Section 5.2, and conclusions and ideas for future work in Section 5.3.

5.1 Frame Sequential Interpolation

Our basic idea is to use two separate render passes to achieve the transition phase between adjacent LOD representations: Pass one renders the scene into an off-screen buffer (called *LOD buffer*). For objects in transition we use one of the two LOD representations and render only a certain amount of its fragments, depending one were in the transition (i.e., how visible) this object currently is. This is done via so-called *visibility textures*, which represent a visibility function for an object. In the next frame we will do the same using the other LOD representation and render into a second *LOD buffer*. The second pass combines these two *LOD buffers* (one from the current and one from the previous frame) to create the desired smooth transition effect.

We start by giving a high-level overview of our algorithm and discuss the details afterwards. Our new algorithm consists of two main contributions: First, the two LODs required for a transition are rendered in subsequent frames, thus avoiding the cost of having to render two representations in one frame. Second, we propose visibility textures as an alternative for alpha-blending in order to avoid many of its inherent problems at overlapping silhouette regions, coplanar regions etc.

5.1.1 LOD Transitions

An LOD transition serves the purpose of smoothly replacing LOD_k with LOD_{k+1} (or LOD_{k-1}) in order to avoid popping artifacts. While this is straightforward for continuous LOD methods through *geomorphing*, for discrete LODs it was only shown recently that it is possible to use alpha-blending for the transition [GW06]. The idea is to first "fade in" (using blending)

LOD_{k+1} until both objects are completely visible, and then "fade out" LOD_k. This approach always keeps one of the two LODs at full visibility, therefore avoiding both LODs being semi-transparent, because this would lead to the situation that we could look through both LODs onto objects that should not be visible.

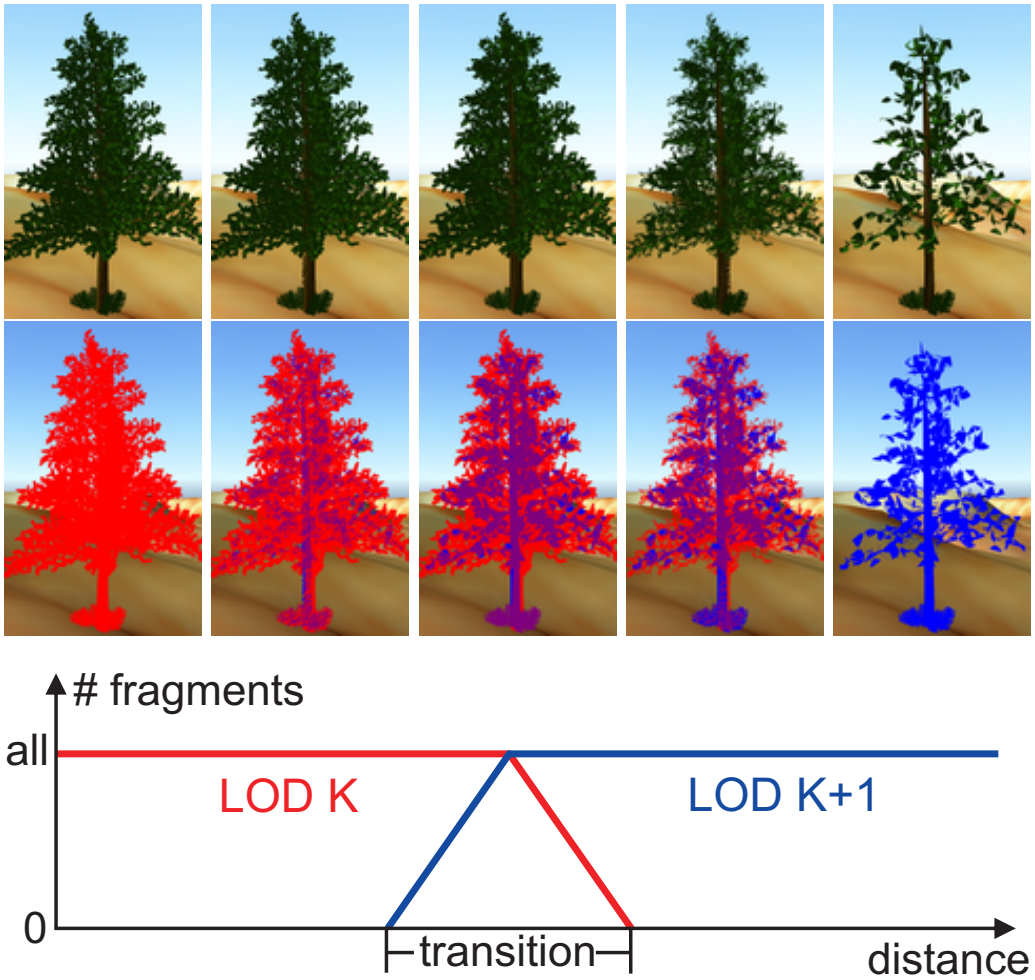


Figure 5.2: *Transition phase from LOD_k to LOD_{k+1}:* left:*LOD_k;* middle: *midway in the transition all fragments of both LODs are drawn;* right: *LOD_{k+1};* Below: *First LOD_{k+1} is gradually introduced till all its fragments are drawn. Then LOD_k is gradually removed by rendering fewer and fewer fragments. The top two rows show the result of our method and a false color illustration.*

The main difference between our method and LOD blending is that we do not use blending, but change the fragment count of one LOD, while keeping the other LOD at full fragment count (see Figure 5.2). While LOD_k stays at

full fragment count for the first half of the transition, LOD_{k+1} is gradually introduced till it too has full fragment count (*middle row*). Then LOD_k is gradually removed by reducing the count of rendered fragments step by step to 0. The visibility function (which corresponds to the alpha-function in [GW06]) used for this is depicted in Figure 5.2 in the bottom row.

5.1.2 Frame Sequential LOD Transitions

One of the main problems of using LOD transitions as shown above is that both LOD levels have to be rendered during the transition phase, causing a performance drop due to the additional geometry (which can be significant for animated meshes) and rasterization required by the second LOD. This is in contrast to the actual aim of LOD rendering, i.e., improving performance. In this section, we show how to render the two LODs in subsequent frames.

For the means of our algorithm we will distinguish two states of object instances: instances in the transition phase between two adjacent LODs (called t-instances), and all other objects (and their instances) that have either no LOD or are not currently in a LOD transition phase (called n-instances).

The central point of our algorithm is to formulate the instance transition in a way that makes it possible to split the rendering of the two LODs of every t-instance into sequential frames. Each is rendered along with the rest of the scene into one of two off-screen buffers (the *LOD buffers*) and afterwards combined to form the final image. What makes this approach fast is that we do the LOD level renderings in an *alternating fashion*: only one *LOD buffer* is rendered each frame, containing all the t-instances with one of LOD_k or LOD_{k+1}, and the rest of the scene. Note that for the combination pass, we also require a depth value and an object id per pixel for both buffers.

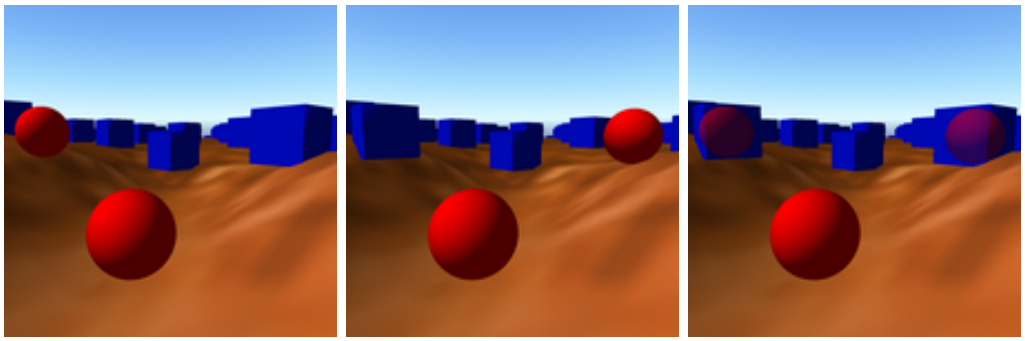


Figure 5.3: *The LOD buffers of two consecutive frames only differ in the used LODs for t-instances (*left *and* middle*). They are combined to create the final image (*right*).*

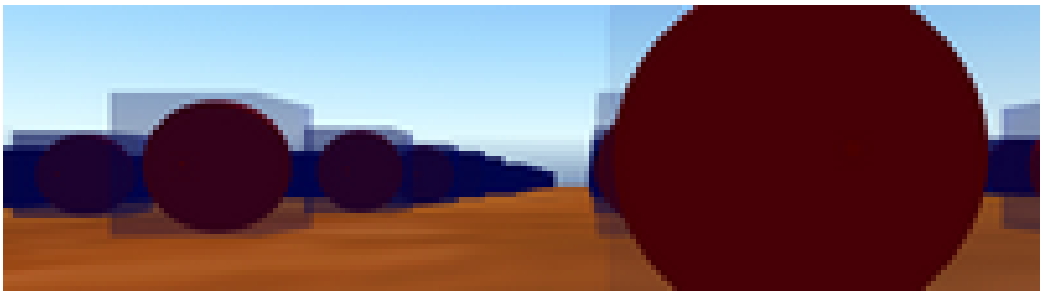


Figure 5.4: *Blending can create an arbitrary number of LOD blends per pixel.*

A sample of this is shown in Figure 5.3: In this scene only two different LODs are present – a red sphere (LOD_0) and a blue box (LOD_1). Two instances are in transition (the instances where the LODs differ in the LOD buffers *left* and *middle*), one instance is at LOD_0 (the closest red sphere) and a lot of instances are at LOD_1 (all other blue boxes). The two LOD buffers (*left* and *middle*) are identical except for the differences in the LODs used for the two t-instances.

One problem with frame-sequential rendering is that if the camera moves between these two consecutive frames, the older of the two LOD buffers will be one frame "out of date". In this case the two frames will not only differ in the LODs of the t-instances, but also slightly in the view-space. We account for that possibility by accessing the older LOD buffer via back-projection in the combination pass (see later). Similarly, if a t-instance moves between adjacent frames, its two LODs will not overlap completely. However, the artifact caused by this fact is not objectionable since it corresponds to a simple motion blur, which is expected in moving objects anyway.

Frame-sequential rendering also allows easy integration into existing rendering engines: The scene can be rendered each frame as usual. The only difference is that we now render to an off-screen buffer, and the concerned parts, the t-instances, have to be rendered differently for our algorithm. The additional combination pass discussed in the following is also self contained and only needs access to the two LOD buffers and the reprojection matrix.

5.1.3 Blending with Visibility Textures

Using alpha-blending for LOD transitions faces several problems. Blending both LODs in the same frame buffer as in [GW06] requires depth-sorting all t-instances. Objects with high depth complexity or objects that are semi-transparent themselves (e.g., billboard clouds) even require depth-sorting individual polygons in order to avoid popping due to changes in depth order.

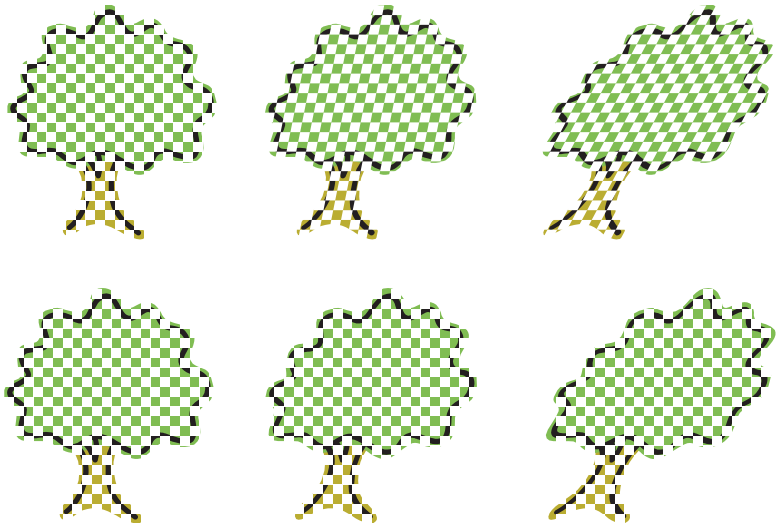


Figure 5.5: *Visibility textures (here a checkerboard pattern) are applied before transforming an object (*upper row*). If applied afterwards the pattern would not follow the movement (*lower row*).*

These problems do not appear in frame-sequential rendering of t-instances when alpha-blending is used to combine the finished LOD buffers. However, alpha-blending LOD buffers leads to problems when the silhouettes of the two LODs of a t-instance do not match for several objects appearing behind each other: in this case, several semi-transparent regions should be created, but the LOD buffers do not contain any information about the objects behind the first semi-transparent region (each LOD buffer stores only a single depth layer). See, for example, Figure 5.4. In frame-sequential LOD blending, this problem can lead to severe popping when the transition state of objects change.

We solve this problem by using *visibility textures*. The idea is to change the visibility of an object by changing the amount of rasterized fragments instead of alpha-blending. This is quite similar to the well-known screendoor transparency, where transparency is simulated using a stippling pattern. However, visibility textures are applied in object space instead of screen space, and allow more flexible visibility patterns. Note that visibility textures are already applied when rendering to the LOD buffers, since the resulting visibility is order independent.

We store a 3D visibility function per object (the *visibility texture*) and compare it to a visibility threshold to decide which t-instance fragments to discard. The visibility threshold $\tau \in [0..1]$ is given by the function depicted in Figure 5.2. Written as an equation: $\lambda : R^3 \times [0..1] \rightarrow \{true, false\}$

$$\lambda : (\boldsymbol{p}, \tau) \mapsto visTex(\boldsymbol{p}) > \tau \tag{5.1}$$

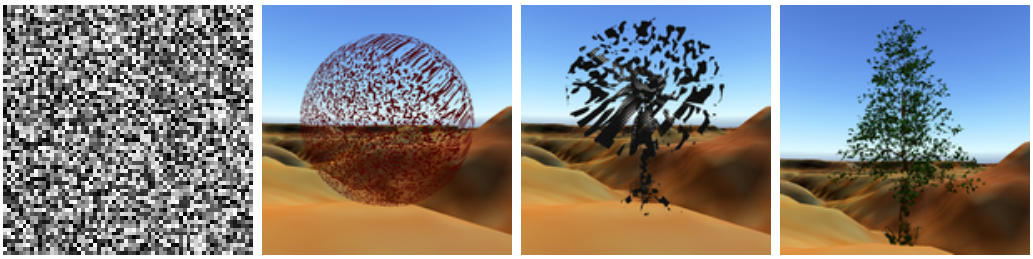


Figure 5.6: *A uniform noise visibility texture (*left*) applied to three different models with visibility* $\tau = 0.5$*).*

is the function that evaluates for each fragment if it should be discarded. Here $\boldsymbol{p}$ is the object-space coordinate (before any animation is applied) of the fragment, τ the visibility threshold and $visTex$ is the lookup into the visibility texture. Note that even though the visibility function may be continuous, the thresholding operation gives a binary result and therefore no semi-transparent pixels appear.

The visibility texture is applied in object space because in this way, the noise is perceived as part of the object rather than a world-space effect. If the lookup into the visibility texture were to take place in world space, the visibility texture would stay fixed and not move along with the object when it is animated (see Figure 5.5).

By using different visibility textures, one can control in which way the fragments of a t-instance become visible. Examples include a uniform noise pattern, a function that decreases from the center outward, or any other function best suited to a given object. This has the effect that the amount and distribution of the visible fragments of an object can be controlled (see Figure 5.6). Note that in practice, we use a 2D noise texture and access it with $(\boldsymbol{p}.x + \boldsymbol{p}.z, \boldsymbol{p}.y + \boldsymbol{p}.z)$.

5.1.4 Combination

Frame-sequential LOD rendering with visibility textures generates two LOD buffers with t-instances that appear "stippled" according to their visibility as discussed in the previous section. These two buffers need to be combined to give the final frame buffer. This is done in a simple screen-space rendering pass.

We need to determine for every pixel which of the two LOD buffer input pixels to use. Note that the older buffer is accessed using reprojection (see also Section 3.1.2). This is done by back-projecting each 3D pixel (we have its depth stored in the LOD buffer) to its position in the older LOD buffer

by applying

$$\boldsymbol{p}_{old} = \mathbf{P}_{old} * \mathbf{V}_{old} * \mathbf{V}_{new}^{-1} * \mathbf{P}_{new}^{-1} * \boldsymbol{p}_{new} \tag{5.2}$$

where $\boldsymbol{p}_{old|new}$ are the old and new pixel positions, $\mathbf{V}_{old|new}$ are the view matrices and $\mathbf{P}_{old|new}$ are the projection matrice of the old and new LOD buffers.

Pixels that back-project outside the LOD buffer have no corresponding old LOD buffer pixel and therefore we treat them as a special case and use only the new color. Note that this can create minor artifacts at the viewport borders when moving the camera. We can solve this by using a bigger off-screen viewport (as described in Section 4.2 for the shadow buffer), but we found that these artifacts were not noticeable in practice.

In addition to color, the LOD buffers also store the depth and a distinct id for each of the t-instances (for n-instances we take some default id like 0). Note that different LODs of the same instance have the same id. The ids are needed because in the final pass we only want to combine pixel pairs (one from the old LOD buffer and one from the new LOD buffer) of the same t-instance. All other fragments should come from the new frame. Three cases can occur:

1. both pixels are from n-instances
2. one pixel is from an t-instance and the other from a different t-instance or from an n-instance
3. both pixels are from the same t-instance (and thus from different LOD levels of the same object instance)

The first is the most frequent case. No interpolation should happen and therefore the pixel from the newer LOD buffer is used. In the second case, the fragment is chosen according to the stored z-values. Thus the result is the same as if the objects had been rendered to one single buffer instead of two (if no movement has taken place). The third case is the most interesting one. While it could be handled similar to the second case using z-testing, which would correspond to evaluating the screendoor-transparency between the two LOD levels of the t-instance, this is not advisable. Coplanar polygons in the two LOD would lead to z-fighting. We therefore avoid z-testing and use the average of the two colors. This has the interesting effect that halfway through the transition, when both LODs are fully visible, the colors of the two LODs will be mixed where they overlap, and thus correspond to a cross-dissolve between the two LODs.

Note that a resulting pixel only receives its color from the older LOD buffer if we are dealing with the second case and if the pixel's z-value in the

old LOD buffer is nearer (than in the new LOD buffer). In the third case the average color from the old buffer and the new buffer is taken. Altogether, most pixels of the final image will consist of the color from the new frame.

5.1.5 The Algorithm

In summary the algorithm works as follows:

1. First we render the scene into an off-screen LOD buffer containing color, object id and depth.
   (a) All n-instances are rendered normally, and with object id 0.
   (b) All t-instances are rendered using alternating LODs and an id that is unique for this instance, while using Equation 5.1 to determine for each fragment if it should be rendered.
2. In the second pass we combine the two LOD buffers in screen space. For the two fragments:
   (a) If both fragments are from n-instances, the newer fragment is kept
   (b) If one fragment is from a t-instance and the other is not from the same t-instance, the closer fragment is kept
   (c) If both fragments are from the same t-instance, the average of the two fragment colors is kept

5.2 Implementation and Results

We implemented the proposed algorithm on an Intel Core 2 Duo E6600 CPU and NVIDIA 8800GTX graphics card. We use a uniform noise visibility texture of 64×64 pixels for all the images shown in this chapter. Two screen-sized LOD buffers each consisting of color (24bit RGB), depth (16bit float and id (16bit float), 7 bytes per pixel in total are used. For a 1024^2 view port this amounts to $14MB$ of additional memory required and for a 512^2 viewport to $1.75MB$. We noticed that in practice back projection of the older LOD buffer is often not needed (and could therefore be omitted) because errors are hidden by the LOD interpolation.

We evaluated our algorithm on three different test scenes (see Figure 5.7). TREES is a scene with 4,000 trees, LODs ranging from $5,000-93,000$ triangles. WINDMILLS has 4,000 windmills, LODs ranging from $100-18,000$ triangles. SIMPLE consists of objects with two LODs: a sphere and a cube of different

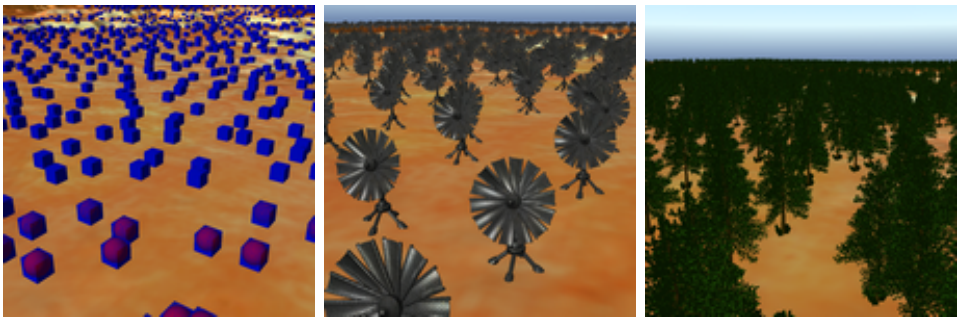


Figure 5.7: *We used three different test scenes to evaluate our algorithm. Left: SIMPLE, middle: WINDMILLS, right: TREES.*

color. SIMPLE is in a way the worst case for LOD transition algorithms in general because the objects match neither in color nor in silhouettes. While our algorithm handles it correctly, noise artifacts from the interpolation are clearly visible (see Figure 5.9). WINDMILLS is a scene that represents a common case and is well behaved for most LOD transition algorithms. Both blending and our algorithm give smooth results, while our algorithm has a higher frame rate. Note that close-up views would still reveal artifacts in the blending approach (see Figure 5.10). TREES is a scene that shows significant artifacts for the blending approach because polygons in the semi-transparent LOD, which is not z-tested, are not sorted correctly. Our approach works very well for this scene. Noise artifacts are practically invisible in this scene. In general, we conclude that the quality of the blend is sufficient even if slight noise can be visible in pathological cases (e.g. SIMPLE). Furthermore, the transition is always smooth (unlike other approaches) and does not suffer from artifacts.

Figure 5.8 shows performance measurements for the three methods with varying viewport sizes for WINDMILLS. Our method is always faster than LOD blending. Note however that in higher resolutions, when the application becomes fillrate limited, our method incurs a certain performance penalty versus discrete switching because of the final combination pass. However, the fragment shaders of the objects in this example are extremely simple and thus the penalty looks exaggerated in comparison to the more common case of realistic fragment shaders.

Given two LODs, one will have a higher rendering cost. As such, in theory each alternating frame could take longer to render (if more of the costly LODs happen to be rendered in this frame), causing uneven frame rates. However, during our tests such a case never occurred. We evaluated the number of triangles rendered each frame and encountered only differences of up to 2%

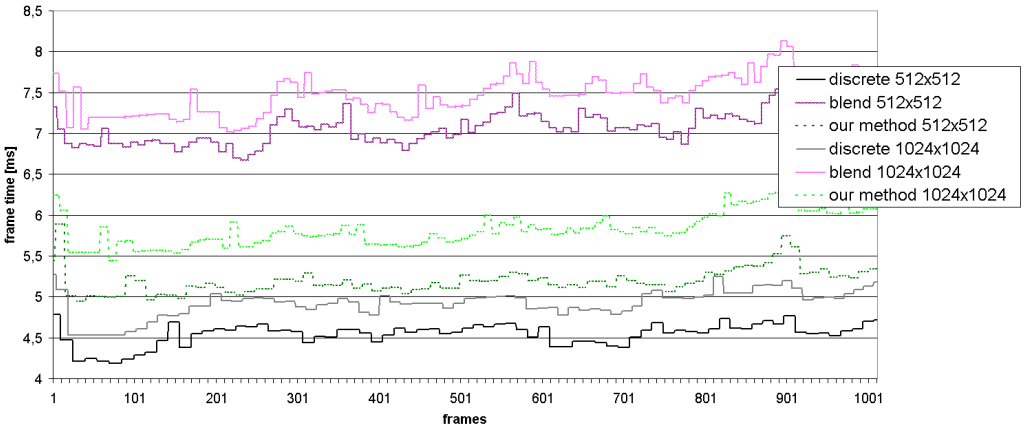


Figure 5.8: *Frame times for discrete LOD rendering, LOD blending and our method.*

between consecutive frames.

5.2.1 Integration into Applications

Integrating the algorithm into an existing rendering pipeline is not very demanding. First, we must be able to change LOD levels each consecutive frame. This is similar to standard discrete LOD switching, where LOD switches could also occur every frame. We need to insert some code into the routine that decides which LOD level to display, for example by deriving from an existing LOD object class. The code checks if the current object is in a transition phase (currently a t-instance). If not, it behaves like a standard LOD object. If it is in a transition phase, we use a toggle counter to decide which of the two LODs will be rendered in this frame and calculate τ by using the function shown in Figure 5.2. As instance id for the item buffer we use the memory address of the object. Please note that any parameters (like skinning matrices) that have to be calculated on the CPU are needed only once per frame, not for both LODs that take part in the transition phase.

Secondly, we must be able to add the shader code needed for our approach, namely the calculation of $\boldsymbol{p}$ in the vertex shader and the evaluation of the visibility texture (with a potential discard of the fragment) in the fragment shader. In modern shading languages, these routines can be encapsulated into subroutines that can be easily added at the beginning of all relevant shaders in the application. Ideally, the engine already provides a mechanism to link subroutines conforming to a predefined interface dynamically, otherwise the subroutine call needs to be added manually. See Listing 5.1 for our

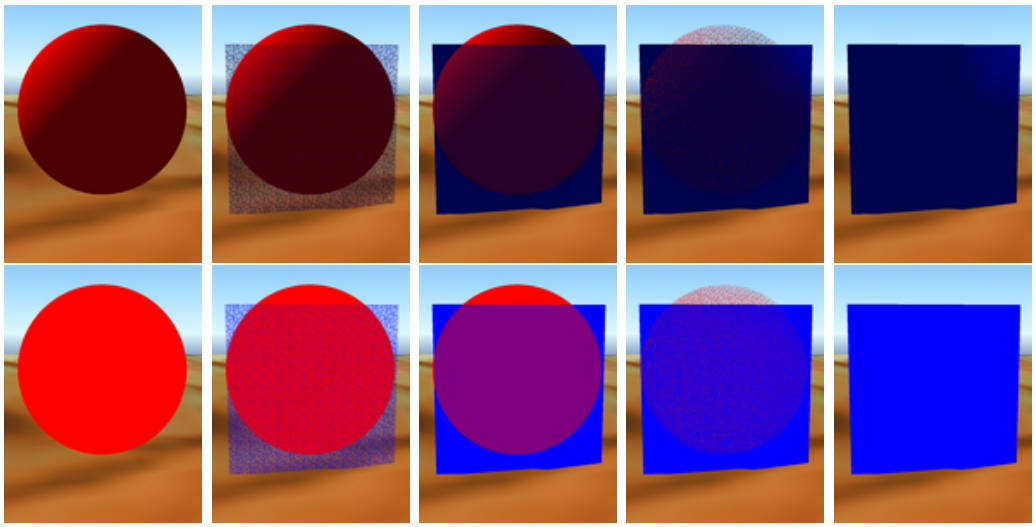


Figure 5.9: *The worst case scenario for our LOD transition algorithm: Two different models, a sphere model with 10k triangles and a box model using 12 triangles. Here the noise artifacts our algorithm introduces become clearly visible.*

```
1  varying vec4 coord_OS;
2  /*vertex*/ void saveObjectSpaceCoords() {
3    coord_OS = gl_Vertex;
4  }
5  uniform sampler2D visTexture;
6  uniform float tau;
7  /*fragment*/ void checkVisibility() {
8    vec2 coord_ts = coord_OS.xy+coord_OS.z;
9    float vis = texture2D(visTexture,coord_ts);
10   if(vis > tau) {
11     discard;
12   }
13 }
14 uniform int id;
15 gl_FragData[1].xy = vec2(gl_FragDepth,id);
```

Listing 5.1: *OpenGL Shading Language implementation of the shader subroutines for calculation of* $\mathbf{p}$ *(lines 1-4) and application of the visibility texture to a fragment (lines 5-13). In lines 14-15 we store the fragment depth and object id into the current* LOD-buffer.

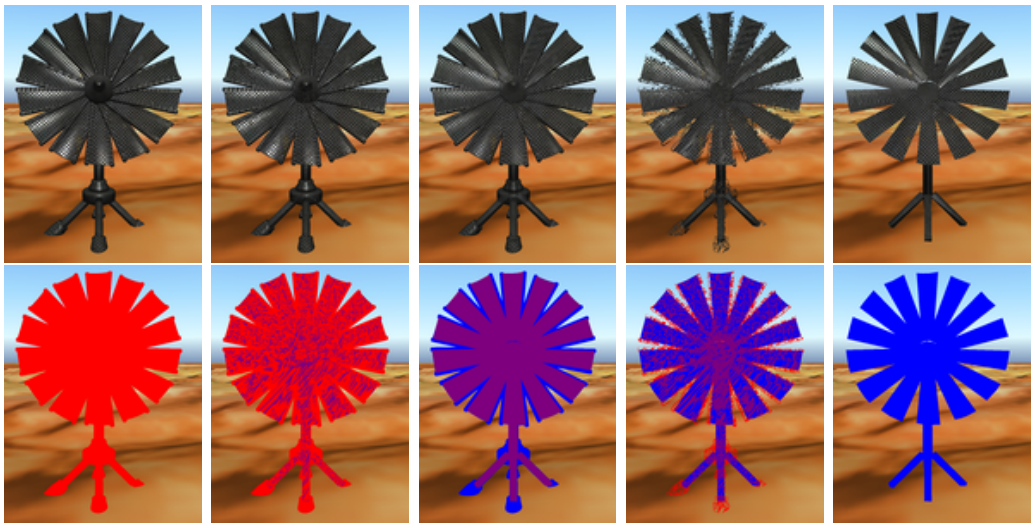


Figure 5.10: *LOD transition phase for the windmill model using an 18k triangle version for LOD_k and a 100 triangle version for LOD_{k+1} to emphasis the changes.*

implementation of these two functions in the OpenGL shading language.

Third, we need the ability to render the scene into off-screen buffers using multiple render targets (MRT). One render target holds the color, while the second holds id and depth. We need these two buffers twice: in each frame, only one color and id-depth buffer is used to write into, while the other is accessed by the shaders. We switch between the two buffers in a ping-pong fashion.

5.2.2 Limitations

The main artifact of our method is that the pattern of the visibility texture becomes visible if the silhouettes of the different LODs don't match well (see Figure 5.11 , 5.10 and 5.9 for examples). This happens either because the chosen LOD levels differ too strongly, especially in the silhouette, or if the silhouettes are transformed due to object movement (motion blur, see Section 5.1.2). In general, these artifacts are relatively unobtrusive as long as the mismatch is not too strong (e.g., fast object movement).

Another type of artifact occurs when previously unseen pixels appear and therefore no respective pixel in the old LOD buffer exists, for example at the viewport borders (see Section 5.1.4) or for disocclusions (if new scene parts appear behind occluders). These artifacts are typically very difficult to spot.

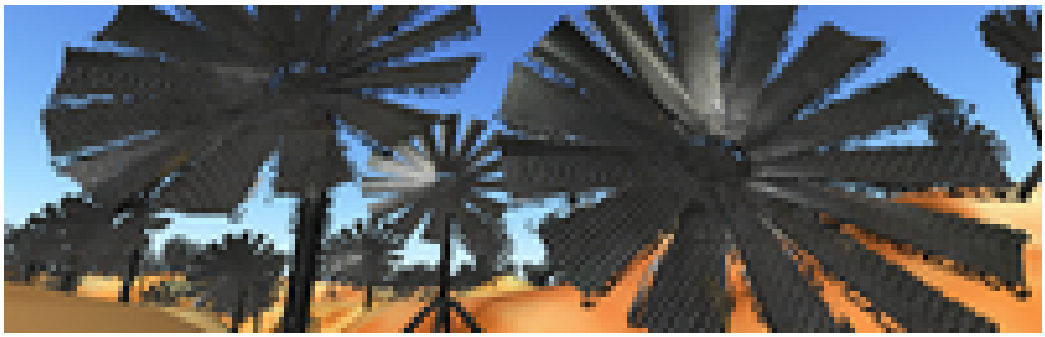


Figure 5.11: *Where the silhouettes of the LODs do not match, noise artifacts can become visible (over emphasized).*

5.3 Conclusions and Future Work

In this chapter we presented a method for creating smooth transitions between discrete levels of detail. Our method has several advantages over previous approaches: It is faster because it avoids rendering transition objects twice through frame-sequential rendering (see also drawbacks 1-3 mentioned in the introduction). It avoids alpha-blending artifacts because it requires no sorting, which is achieved through visibility textures (drawbacks 4-5). Even co-planar LOD levels pose no problem due to the use of object ids (drawback 6). The algorithm is also more robust, straightforward to implement and easier to integrate into common rendering engines.

A possible extension for future work is the reduction of the noise we introduced for our interpolation scheme, for instance with a history buffer as in Chapter 3 or by using multi-sampling. Another promising avenue for further research would be the topic of visibility textures, i.e., the automatic generation of an optimized visibility texture for a given object.

6 Summary

The goal of this work was to show that *temporal coherence* is a property so common in typical applications of real-time graphics, that by redesigning algorithms to account for this property, large benefits can be gained. To this end we presented three new algorithms that employ *temporal coherence*. This includes our pixel correct hard shadow algorithm from Chapter 3, our real-time approach to physically correct soft shadows from Chapter 4 and our discrete LOD blending algorithm from Chapter 5. We will now highlight the benefits that *temporal coherence* has brought to these algorithms.

Acceleration of algorithms: In scenarios with comparable complexity to ours most of the concurrent pixel correct hard shadow algorithms exhibit speeds that are interactive or just real-time. Frame rates are at most 60FPS for all of them, while our approach runs at hundreds of frames per second, due to the use of *temporal coherence* to incrementally compute its result.

A similar situation presents itself in the area of physically correct soft shadows. Our algorithm allows for the first time to perform light source area sampling with large numbers of samples in real-time. The algorithm is even faster than most of the soft shadow algorithms that are not physically correct.

The same is true for our discrete LOD blending algorithm, which is about twice as fast as previous work.

Increasing visual fidelity: The major point of our pixel correct hard shadow algorithm is the quality of the resulting shadows. Concurrent algorithms with a comparable speed produce inferior results containing at least projection aliasing artifacts. Additionally temporal aliasing, which plagues most hard shadow mapping algorithms, is greatly reduced.

For many applications plausible soft shadows are sufficient, because physical incorrectness is often hard to spot for the observer. Nevertheless, artifacts of physically incorrect methods become apparent for scenarios with overlapping occluders or with large light sources. Physically correct soft shadows

represent the ultimate visual quality in soft shadows and if this is the goal of an application, our approach is currently unique in its speed.

Our LOD blending approach, although not generally visually superior to concurrent work, can produce better results where blending creates disturbing object changes and noise does not.

The price of using temporal coherence: Most of our algorithms have restrictions concerning the changes that are allowed to happen from frame to frame: Our discrete LOD blending approach may show slight artifacts on silhouettes of fast moving objects currently in a LOD transition phase. More limited is our hard shadow approach, which creates strong artifacts for moving objects or light sources. Our soft shadow approach has trouble with moving light sources and will create lagging shadows in such cases. Also moving objects produce inferior shadows when compared to static objects.

6.1 Research Outlook

Although we were able to validate our main thesis, there is still much work to be done. Our algorithms produce the best results for static scenes (especially our pixel correct hard shadow algorithm). The main problem here is that the test we use for equivalence of fragments only takes camera movement into account. One way to solve this is to introduce a confidence measurement for the stored information (in the history respective shadow buffer). Up to now we only measure the confidence of the new information (shadow test result), but it is also possible to do something similar for values from the history buffer. We could base this confidence on movement, age or distance to an edge at this texel, thereby allowing for a more informed decision and probably better results in dynamic scenes.

Please note that our shadowing methods are so fast that in typical scenarios, we can also handle dynamic scenes by performing multiple passes of our techniques each frame, thereby increasing temporal coherence and increasing quality.

Secondly, the presented algorithms are only a small part of what is possible by including *temporal coherence* into rendering techniques. Since their publication, more and more papers take advantage of this basic property of animated scenes. We expect that almost all areas of rendering can profit from a temporal coherence-based approach.

Appendix: Shaders

Pixel Correct Hard Shadows

```
//vertex
uniform mat4 mtxNew2old; //new screen space to hbuffer
uniform mat4 mSM; //shadow map matrix
varying vec4 texelDiffuse;
varying vec4 texelPosHistoryBuffer;
varying vec4 texelPosLightSpace;
varying vec4 pos;

void main(void)
{
  gl_Position = ftransform();

  //need interpolated depth for discontinuity check
  pos = gl_Position;

  //calc old texture coordinates (2-dim)
  //input new pos_postpersp and transform back to old
  texelPosHistoryBuffer = mtxNew2old*gl_Position;

  //shadow texture coordinates
  texelPosLightSpace = mSM*gl_Position;

  //diffuse texture coordinates (2-dim)
  texelDiffuse = gl_MultiTexCoord0;
}

//Fragment
bool outsideOld(const vec2 texelPosOldSpaceH_)
{
  return any(lessThan(texelPosOldSpaceH_,vec2(0.0)))
    || any(greaterThan(texelPosOldSpaceH_,vec2(1.0)));
}

uniform sampler2D texDiffuse;
uniform sampler2D texHistoryBuffer;
uniform float u_fM; //parameter m from the paper
varying vec4 texelDiffuse;
```

```
38 varying vec4 texelPosHistoryBuffer;
39 varying vec4 texelPosLightSpace;
40 varying vec4 pos;
41
42 void main(void)
43 {
44   //do shadow mapping
45   const vec3 texCoordHomo =
46     texelPosLightSpace.xyz/texelPosLightSpace.w;
47   //calc sm lighting
48   float lighting = doShadowFilter(texCoordHomo);
49   //fM is stored in a color -> normalized
50   float fM = 0.0;
51   //get alpha from diffuse texture
52   const float alpha =
53     texture2D(texDiffuse,texelDiffuse.xy).a;
54
55   const vec3 texelPosOldSpaceH =
56     texelPosHistoryBuffer.xyz/texelPosHistoryBuffer.w;
57
58   if(!outsideOld(texelPosOldSpaceH.xy)) {
59     //inside of old data -> can check for depth delta
60     const vec4 dataOldSpace =
61       texture2D(texHistoryBuffer,texelPosOldSpaceH.xy);
62     const float oldDepth = dataOldSpace.g;
63     //check if depths are alike
64     //otherwise we have a very different sample
65     if(distance(oldDepth,texelPosOldSpaceH.z) < 0.001)
66     {
67       //distance to center in light space texel
68       const vec2 dist = abs(
69         fract(texCoordHomo.xy*texMapSize)-vec2(0.5,0.5)
70         )*2.0; //[0..1]x[0..1]
71       const float confidence =
72         abs(1.0-max(dist.x,dist.y)); //0..1
73       const float oldLighting = dataOldSpace.r;
74       const float weight = clamp(
75       pow(confidence,u_fM),0.0,1.0);
76       //exponential smoothing
77       lighting = oldLighting*(1.0-weight)
78         + lighting*weight;
```

```
79      }
80    }
81    //store lighting in r
82    gl_FragColor.r = lighting;
83    //store new depth in green channel of texture [0,1]
84    //so we can compare it to texelPosOldSpaceH
85    gl_FragColor.g = pos.z/pos.w*0.5+0.5;
86    //store info in blue
87    gl_FragColor.b = fM;
88    //store alpha in alpha
89    gl_FragColor.a = alpha;
90  }
```

Listing 6.1: *OpenGL Shading Language implementation of the shaders for pixel correct hard shadows*

Physically Correct Soft Shadows

```
1  PS_OUTPUT PSSoftShadowDani( PS_INPUT input) : SV_Target
2  {
3    //shadow sampling coordinates:
4    const float2 smCoord = texSpace(input.LightPos);
5    const float fragmentDepth = input.LightPos.z;
6    float ShadowAmount = 0.0f;
7    //negative value means no blocker found
8    float avgBlockerDepth = -1000000.0f;
9    float cnt = 1;
10   float softShadow = 0.0;
11   bool bCurrentShadowFromMovingObject = false;
12
13   float oldDepth = 0;
14   float oldSum = 0;
15   float oldCount = 0;
16   float oldAvgBlockerDepth = avgBlockerDepth;
17
18   //history buffer sampling coordinates:
19   const float2 HisBuffTexC = texSpace(input.BufferPos);
20
21   //check if the pixel is inside the history buffer:
22   if(!outsideOld(HisBuffTexC))
23   {
```

```
24      //inside of old data -> can check for depth delta
25      //and use new shadow map for sum
26      //sample depth in shadow map
27      const float Depth =
28        g_txSM.SampleLevel(samPoint,smCoord,0).x;
29      const float absDepth = abs(Depth);
30      ShadowAmount =
31        1.0-inShadowTest(absDepth,fragmentDepth);
32      if(ShadowAmount > 0.7)
33      {
34        //get new depth as estimate for avgBlockerDepth
35        avgBlockerDepth = Depth;
36        bCurrentShadowFromMovingObject = Depth < 0.0;
37      }
38
39      //get the history buffer info:
40      const float4 oldData =
41        g_txHistoryBuffer.Sample(samLinear,HisBuffTexC);
42      oldDepth = abs(oldData.x);
43      oldSum = oldData.y;
44      oldCount = oldData.z;
45      oldAvgBlockerDepth = oldData.w;
46
47      //check if depths are alike
48      //and current position not hit by moving shadow
49      //otherwise we have a very different sample
50      if(abs(1-input.BufferPos.z/oldDepth) < DEPTH_EPSIL
51        && !bCurrentShadowFromMovingObject)
52      {
53        //calculate the new shadow amount:
54        ShadowAmount = oldSum + ShadowAmount;
55        cnt = oldCount + 1;
56
57        //Average Blocker Depth
58        if(oldAvgBlockerDepth >= 0.0f) {
59          if(avgBlockerDepth >= 0.0f) {
60            float sum = oldAvgBlockerDepth*
61              (cnt-1+BLOCKER_SEARCH_LOOKUP);
62            sum += avgBlockerDepth;
63            avgBlockerDepth =
64              sum/(cnt+BLOCKER_SEARCH_LOOKUP);
```

```
65          }
66          else
67          {
68            avgBlockerDepth = oldAvgBlockerDepth;
69          }
70        }
71      }
72    }
73
74    if(cnt < 1.5)
75    {
76      //newly dissoccluded fragments - only pcss
77      const float fLightSize =
78        vLightDimensions[0] / LIGHT_FRUSTUM_WIDTH;
79
80      // STEP 1: blocker search
81      float avgBlockerDepth = -1000000;
82      float numBlockers = 0;
83      float penumbraSize = 0;
84      FindBlocker(avgBlockerDepth,numBlockers,fLightSize
85        ,smCoord, fragmentDepth - SHADOW_EPSILON);
86
87      if( numBlockers > 1 )
88      {
89        //only if occluders present
90        // STEP 2: penumbra size
91        penumbraSize =
92          (fragmentDepth-avgBlockerDepth)/avgBlockerDepth;
93        const float filterRadiusUV = penumbraSize
94          * fLightSize * NEAR_PLANE / fragmentDepth;
95        // STEP 3: filtering
96        softShadow = 1 - PCF_Filter(
97          smCoord,fragmentDepth,filterRadiusUV);
98      }
99      else
100     {
101       //no blockers found
102       softShadow = 0.0;
103     }
104
105     if(bCurrentShadowFromMovingObject)
```

```
106     {
107       const float2 kernelSize =
108         penumbraSize*vAspectRatio*0.1/input.Depth;
109       const float blurredShadow = neighborhoodFilter(
110         HisBuffTexC,kernelSize,input.Depth);
111       const float weightSurr = 16;
112     }
113
114     cnt = 1;
115     ShadowAmount = softShadow*cnt;
116
117   }
118   else
119   {
120     //not a new sample so check error
121     softShadow = (ShadowAmount/cnt);
122     const float errorThreshold = 1.0/50.0;
123     const float error =
124       sqrt(softShadow*(1-softShadow)/(cnt-1));
125
126     if( error >= errorThreshold && avgBlockerDepth > 0 )
127     {
128       //if above error threshold
129       //use neighborhood filtering
130       const float penumbraSize =
131         ((fragmentDepth - avgBlockerDepth)
132         / avgBlockerDepth);
133       const float2 kernelSize =
134         penumbraSize*vAspectRatio*0.1/input.Depth;
135       const float blurredShadow = neighborhoodFilter(
136         HisBuffTexC, kernelSize, input.Depth);
137       if(blurredShadow > 0.0)
138       {
139         softShadow = blurredShadow;
140       }
141     }
142   }
143
144   PS_OUTPUT output = (PS_OUTPUT)0;
145   output.Col1 = float4(
146     input.Depth, ShadowAmount, cnt, avgBlockerDepth);
```

```
147
148   // vLight is the unit vector from light to pixel
149   const float3 vLight =
150     normalize(float3(vLightPos.xyz - input.wPos.xyz ));
151   //per pixel lighting:
152   const float4 DiffuseColor =
153     DiffusePerPixelLighting(vLight, input.Norm);
154
155   //calc lighting (shadowing)
156   output.Col0.rgb = DiffuseColor.rgb * (1-softShadow)
157     + vLightColor * vMaterialAmbient;
158   output.Col0.rgb *= debugColor;
159   output.Col0.a = DiffuseColor.a;
160
161   //add texture information:
162   output.Col0 *=
163     g_txDiffuse.Sample(samLinear,input.Tex);
164
165   return output;
166 }
```

Listing 6.2: *HLSL implementation of the pixel shader for physically correct soft shadows*

Fast LOD Blending

```
1  uniform mat4 u_mtxNew2old; //new screen space to hbuffer
2  uniform sampler2D texNewColor; //LOD buffers
3  uniform sampler2D texOldColor; //LOD buffers
4  uniform sampler2D texNewData;  //LOD buffers
5  uniform sampler2D texOldData;  //LOD buffers
6  varying vec4 v_vTexelPosBuffer;
7  varying vec4 v_vPos_ps;
8
9  void set(const vec3 c_vColor_) {
10   gl_FragColor = vec4(c_vColor_,1.0);
11 }
12
13 struct Fragment {
14   vec3 color;
15   float id, depth;
```

```
16 };
17
18 void main(void) {
19   //load new values from LOD buffer
20   const Fragment cNew = decodeNew(decode(
21     texture2D(texNewColor,v_vTexelPosBuffer.st)
22     ,texture2D(texNewData,v_vTexelPosBuffer.st)));
23
24   const vec4 posNew =
25     vec4(v_vPos_ps.x,v_vPos_ps.y,cNew.depth,1.0);
26
27   //calculate texture coordinates for previous frame
28   const vec4 texelPosOldSpace = u_mtxNew2old*posNew;
29   const vec3 texelPosOldSpaceH =
30     texelPosOldSpace.xyz/texelPosOldSpace.w;
31
32   //load old values from LOD buffer
33   const Fragment cOld = decodeOld(decode(
34     texture2D(texOldColor,texelPosOldSpaceH.st)
35     ,texture2D(texOldData,texelPosOldSpaceH.st)));
36
37   //outside old LOD buffer
38   if(outside(texelPosOldSpaceH.xy)) {
39     set(cNew.color);
40     return;
41   }
42
43   // fragment from n-instances
44   if(0.0 == cNew.id && 0.0 == cOld.id) {
45     //both ids are zero
46     set(cNew.color);
47     return;
48   }
49
50   //fragment from same t-instance
51   if(cOld.id == cNew.id) {
52     set((cNew.color+cOld.color)*0.5);
53     return;
54   }
55
56   //depth compare
```

```
57    if(cNew.depth < cOld.depth) {
58       set(cNew.color);
59    }
60    else {
61       set(cOld.color);
62    }
63 }
```

Listing 6.3: *OpenGL Shading Language implementation of the pixel shader for combining new and old LOD buffer*

List of Figures

List of Tables

Bibliography

[ADM+08] Thomas Annen, Zhao Dong, Tom Mertens, Philippe Bekaert, Hans-Peter Seidel, and Jan Kautz. Real-time, all-frequency shadows in dynamic scenes. In *SIGGRAPH '08: ACM SIGGRAPH 2008 papers*, pages 1–8, New York, NY, USA, 2008. ACM.

[AH92] Stephen J. Adelson and Larry F. Hodges. Visible surface ray-tracing of stereoscopic images. In *ACM-SE 30: Proceedings of the 30th annual Southeast regional conference*, pages 148–156, New York, NY, USA, 1992. ACM.

[AH95] Stephen J. Adelson and Larry F. Hodges. Generating exact ray-traced animation frames by reprojection. *IEEE Comput. Graph. Appl.*, 15(3):43–52, 1995.

[AHL+06] Lionel Atty, Nicolas Holzschuch, Marc Lapierre, Jean-Marc Hasenfratz, Chuck Hansen, and François Sillion. Soft shadow maps: Efficient sampling of light source visibility. *Computer Graphics Forum*, 25(4), dec 2006.

[AL04] Timo Aila and Samuli Laine. Alias-free shadow maps. In *Proceedings of Eurographics Symposium on Rendering 2004*, pages 161–166. Eurographics Association, 2004.

[AMA02] Tomas Akenine-Möller and Ulf Assarssonk. Approximate soft shadows on arbitrary surfaces using penumbra wedges. In *Thirteenth Eurographics Workshop on Rendering*, pages 297–305, 2002.

[AMB+07] Thomas Annen, Tom Mertens, Philippe Bekaert, Hans-Peter Seidel, and Jan Kautz. Convolution shadow maps. In Jan Kautz and Sumanta Pattanaik, editors, *Rendering Techniques 2007: Eurographics Symposium on Rendering*, volume 18 of *Eurographics / ACM SIGGRAPH Symposium Proceedings*, pages 51–60, Grenoble, France, June 2007. Eurographics.

[AMS+08] Thomas Annen, Tom Mertens, Hans-Peter Seidel, Eddy Flerackers, and Jan Kautz. Exponential shadow maps. In *GI '08: Proceedings of graphics interface 2008*, pages 155–161, Toronto, Ont., Canada, Canada, 2008. Canadian Information Processing Society.

[ARHM00] Maneesh Agrawala, Ravi Ramamoorthi, Alan Heirich, and Laurent Moll. Efficient image-based methods for rendering soft shadows. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 375–384. ACM Press/Addison-Wesley Publishing Co., 2000.

[Arv04] Jukka Arvo. Tiled shadow maps. In *CGI '04: Proceedings of the Computer Graphics International (CGI'04)*, pages 240–247, Washington, DC, USA, 2004. IEEE Computer Society.

[Arv07] Jukka Arvo. Alias-free shadow maps using graphics hardware. *journal of graphics, gpu, and game tools*, 12(1):47–59, 2007.

[ASK06] Barnabas Aszodi and Laszlo Szirmay-Kalos. Real-time soft shadows with shadow accumulation. *EUROGRAPHICS 2006*, 2006.

[BAS02] Stefan Brabec, Thomas Annen, and Hans-Peter Seidel. Practical shadow mapping. *Journal of Graphics Tools: JGT*, 7(4):9–18, 2002.

[BCT08] Louis Bavoil, Steven P. Callahan, and Cláudio T.Silva. Robust soft shadow mapping with backprojection and depth peeling. *journal of graphics tools*, 13(1):19–30, 2008.

[BFMZ94] Gary Bishop, Henry Fuchs, Leonard McMillan, and Ellen J. Scher Zagier. Frameless rendering: double buffering considered harmful. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 175–176, New York, NY, USA, 1994. ACM.

[BJ88] S. Badt Jr. Two algorithms for taking advantage of temporal coherence in ray tracing. *VC*, 4:123–132, 1988.

[BS02] Stefan Brabec and Hans-Peter Seidel. Single Sample Soft Shadows Using Depth Maps. In *Proceedings of Graphics Interface*, pages 219–228, May 2002.

[CD03] Eric Chan and Frédo Durand. Rendering fake soft shadows with smoothies. In *Proceedings of the Eurographics Symposium on Rendering*, pages 208–218. Eurographics Association, 2003.

[CG04] H. Chong and S. J. Gortler. A lixel for every pixel. In *Proceedings of Eurographics Symposium on Rendering 2004*, 2004.

[CG07] Hamilton. Y. Chong and Steven J. Gortler. Scene optimized shadow mapping. harvard computer science technical report: Tr-07-07. Technical report, Harvard University, Cambridge, MA, 2007.

[CGG+03] Paolo Cignoni, Fabio Ganovelli, Enrico Gobbetti, Fabio Marton, Federico Ponchio, and Roberto Scopigno. Planet-sized batched dynamic adaptive meshes (p-bdam). In *VIS '03: Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, page 20, Washington, DC, USA, 2003. IEEE Computer Society.

[Cha04] Bryan Chan. Soft shadow mapping the inner penumbra. In *?* University of Waterloo, 2004.

[Cho03] Hamilton Chong. Real-time perspective optimal shadow maps. Senior thesis, Harvard College, Cambridge, Massachusetts, April 2003.

[CS06] Ioan Cleju and Dietmar Saupe. Evaluation of supra-threshold perceptual metrics for 3d models. In *APGV '06: Proceedings of the 3rd symposium on Applied perception in graphics and visualization*, pages 41–44, New York, NY, USA, 2006. ACM.

[CT96] Satyan Coorg and Seth Teller. Temporally coherent conservative visibility (extended abstract). In *SCG '96: Proceedings of the twelfth annual symposium on Computational geometry*, pages 78–87, New York, NY, USA, 1996. ACM.

[CW93] Shenchang Eric Chen and Lance Williams. View interpolation for image synthesis. In *SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 279–288, New York, NY, USA, 1993. ACM.

[Dav98] Timothy A. Davis. Generating computer animations with frame coherence in a distributed computing environment. In *ACM-SE 36: Proceedings of the 36th annual Southeast regional conference*, pages 1–7, New York, NY, USA, 1998. ACM.

[dB06] Willem H. de Boer. Smooth penumbra transitions with shadow maps. In *journal of graphics, gpu, and game tools*, volume 11, pages 59–71, 2006.

[DD99] Timothy A. Davis and Edward W. Davis. Exploiting frame coherence with the temporal depth buffer in a distributed computing environment. In *PVGS '99: Proceedings of the 1999 IEEE symposium on Parallel visualization and graphics*, pages 29–38, Washington, DC, USA, 1999. IEEE Computer Society.

[DL06] William Donnelly and Andrew Lauritzen. Variance shadow maps. In *I3D '06: Proceedings of the 2006 symposium on Interactive 3D graphics and games*, pages 161–165, New York, NY, USA, 2006. ACM.

[DS04] Carsten Dachsbacher and Marc Stamminger. Rendering procedural terrain by geometry image warping. *Eurographics Symposium on Rendering (2004)*, 2004.

[ED06a] Elmar Eisemann and Xavier Décoret. Fast scene voxelization and applications. In *I3D '06: Proceedings of the 2006 symposium on Interactive 3D graphics and games*, pages 71–78, New York, NY, USA, 2006. ACM.

[ED06b] Elmar Eisemann and Xavier Décoret. Plausible image based soft shadows using occlusion textures. In Rodrigo Lima Oliveira Neto, Manuel Menezes deCarceroni, editor, *Proceedings of the Brazilian Symposium on Computer Graphics and Image Processing, 19 (SIBGRAPI)*, Conference Series. IEEE, IEEE Computer Society, 2006.

[ED07] Elmar Eisemann and Xavier Décoret. Visibility sampling on gpu and applications. *Computer Graphics Forum (Proceedings of Eurographics 2007)*, 26(3), 2007.

[ED08] Elmar Eisemann and Xavier Décoret. Occlusion textures for plausible soft shadows. *Computer Graphics Forum*, 27(1):13–23, July 2008.

[Eng07] Wolfgang Engel. *Cascaded Shadow Maps*. Thomson Learning, 2007.

[FBP08] Vincent Forest, Loïc Barthe, and Mathias Paulin. Accurate Shadows by Depth Complexity Sampling. *Computer Graphics Forum, Eurographics 2008 Proceedings*, 27(2):663–674, 2008.

[Fer05] Randima Fernando. Percentage-closer soft shadows. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Sketches*, page 35, New York, NY, USA, 2005. ACM.

[FFBG01] 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.

[For06] Tim Forsyth. *Making shadow buffers robust using multiple dynamic Shadow Maps*. Charles River Media, 2006.

[FS93] Thomas A. Funkhouser and Carlo H. Séquin. Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. In *SIGGRAPH 93 Conference Proceedings*, pages 247–254, 1993.

[Gau08] Pascal Gautron. Temporal radiance caching. In *SIGGRAPH '08: ACM SIGGRAPH 2008 classes*, pages 1–49, New York, NY, USA, 2008. ACM.

[GBP06] Gael Guennebaud, Loïc Barthe, and Mathias Paulin. Real-time soft shadow mapping by backprojection. In *Eurographics Symposium on Rendering (EGSR), Nicosia, Cyprus*, pages 227–234. Eurographics, June 2006.

[GBP07] Gael Guennebaud, Loïc Barthe, and Mathias Paulin. High-Quality Adaptive Soft Shadow Mapping. *Proceedings of Computer Graphics Forum, Eurographics 2007*, 26(3):525–534, Sept 2007.

[GH98] Michael Garland and Paul S. Heckbert. Simplifying surfaces with color and texture using quadric error metrics. In David Ebert, Hans Hagen, and Holly Rushmeier, editors, *Proceedings of the conference on Visualization '98*, pages 263–270. IEEE, October 1998. ISBN 1-58113-106-2.

[GM07] Enrico Gobbetti and Fabio Marton. Gpu-friendly accelerated mesh-based and mesh-less techniques for the output-sensitive rendering of huge complex 3d models. In *SIGGRAPH '07: ACM SIGGRAPH 2007 courses*, page 8, New York, NY, USA, 2007. ACM.

[Grö92] Meister Eduard Gröller. *Coherence in Computer Graphics*. PhD thesis, Institute of Computer Graphics and Algorithms, Vienna University of Technology, Favoritenstrasse 9-11/186, A-1040 Vienna, Austria, 1992.

[GW06] Markus Giegl and Michael Wimmer. Unpopping: Solving the image-space blend problem for smooth discrete lod transitions. *Computer Graphics Forum*, 26(1):46–49, March 2006.

[GW07a] Markus Giegl and Michael Wimmer. Fitted virtual shadow maps. In *GI '07: Proceedings of Graphics Interface 2007*, pages 159–168, New York, NY, USA, 2007. ACM.

[GW07b] Markus Giegl and Michael Wimmer. Queried virtual shadow maps. In *I3D '07: Proceedings of the 2007 symposium on Interactive 3D graphics and games*, pages 65–72, New York, NY, USA, 2007. ACM.

[HBS00] Wolfgang Heidrich, Stefan Brabec, and Hans-Peter Seidel. Soft shadow maps for linear lights. In *Proceedings of the Eurographics Workshop on Rendering Techniques 2000*, pages 269–280. Springer-Verlag, 2000.

[HBS03] Vlastimil Havran, Jiří Bittner, and Hans-Peter Seidel. Exploiting temporal coherence in ray casted walkthroughs. In *SCCG '03: Proceedings of the 19th spring conference on Computer graphics*, pages 149–155, New York, NY, USA, 2003. ACM Press.

[HH97] Paul S. Heckbert and Michael Herf. Simulating soft shadows with graphics hardware. Technical Report CMU-CS-97-104, CS Dept., Carnegie Mellon U., Jan. 1997. CMU-CS-97-104, http://www.cs.cmu.edu/ ph.

[HKSB06] Markus Hadwiger, Andrea Kratz, Christian Sigg, and Katja Bühler. Gpu-accelerated deep shadow maps for direct volume rendering. In *GH '06: Proceedings of the 21st ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware*, pages 49–52, New York, NY, USA, 2006. ACM.

[HLHS03] Jean-Marc Hasenfratz, Marc Lapierre, Nicolas Holzschuch, and François Sillion. A survey of real-time soft shadows algorithms. *Computer Graphics Forum*, 22(4):753–774, dec 2003.

[HN85] J. C. Hourcade and A. Nicolas. Algorithms for antialiased cast shadows. *Computers and Graphics*, 9(3):259–265, 1985.

[Hop96] Hugues Hoppe. Progressive meshes. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 99–108. ACM SIGGRAPH, Addison Wesley, August 1996. held in New Orleans, Louisiana, 04-09 August 1996.

[Hop98] Hop98. Smooth view-dependent level-of-detail control and its application to terrain rendering. *In Proceedings of the conference on Visualization*, 98:35–422, 1998.

[HZ81] Harold Hubschman and Steven W. Zucker. Frame-to-frame coherence and the hidden surface computation: Constraints for a convex world. In *SIGGRAPH '81: Proceedings of the 8th annual conference on Computer graphics and interactive techniques*, pages 45–54, New York, NY, USA, 1981. ACM.

[Jev92] David A. Jevans. Object space temporal coherence for ray tracing. In *Graphics Interface 92*, pages 176–183, 1992.

[JHH+09] Gregory S. Johnson, Warren A. Hunt, Allen Hux, William R. Mark, Christopher A. Burns, and Stephen Junkins. Soft irregular shadow mapping: fast, high-quality, and robust soft shadows. In *I3D '09: Proceedings of the 2009 symposium on Interactive 3D graphics and games*, pages 57–66, New York, NY, USA, 2009. ACM.

[JLBM05] Gregory S. Johnson, Juhyun Lee, Christopher A. Burns, and William R. Mark. The irregular z-buffer: Hardware acceleration for irregular data structures. *ACM Trans. Graph.*, 24(4):1462–1482, 2005.

[JMB04] 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.

[JWLL05] Junfeng Ji, Enhua Wu, Sheng Li, and Xuehui Liu. Dynamic lod on gpu. In *CGI '05: Proceedings of the Computer Graphics International 2005*, pages 108–114, Washington, DC, USA, 2005. IEEE Computer Society.

[KD03] Florian Kirsch and Juergen Doellner. Real-time soft shadows using a single light sample. In *Journal of WSCG (Winter School on Computer Graphics 2003)*, page 11(1), 2003.

[Koz04] S. Kozlov. Perspective shadow maps - care and feeding. *GPU Gems*, pages 217–244, 2004.

[Lau07] Andrew Lauritzen. *GPU Gems 3*, chapter Summed-Area Variance Shadow Maps. Addison-Wesley, 2007.

[LGMM07] D. Brandon Lloyd, Naga K. Govindaraju, Steven E. Molnar, and Dinesh Manocha. Practical logarithmic rasterization for low-error shadow maps. In *GH '07: Proceedings of the 22nd ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware*, pages 17–24, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.

[LGQ+08] D. Brandon Lloyd, Naga K. Govindaraju, Cory Quammen, Steven E. Molnar, and Dinesh Manocha. Logarithmic perspective shadow maps. *ACM Trans. Graph.*, 27(4):1–32, 2008.

[LH04] Frank Losasso and Hugues Hoppe. Geometry clipmaps: terrain rendering using nested regular grids. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, pages 769–776, New York, NY, USA, 2004. ACM.

[LKR+96] Peter Lindstrom, David Koller, William Ribarsky, Larry F. Hughes, Nick Faust, and Gregory Turner. Real-Time, continuous level of detail rendering of height fields. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 109–118. ACM SIGGRAPH, Addison Wesley, August 1996. held in New Orleans, Louisiana, 04-09 August 1996.

[LM08] Andrew Lauritzen and Michael McCool. Layered variance shadow maps. In *GI '08: Proceedings of graphics interface 2008*, pages 139–146, Toronto, Ont., Canada, Canada, 2008. Canadian Information Processing Society.

[LRC+02] David Luebke, Martin Reddy, Jonathan D. Cohen, Amitabh Varshney, Benjamin Watson, and Robert Huebner. *Level of Detail for 3D Graphics (The Morgan Kaufmann Series in Computer Graphics)*. Morgan Kaufmann, July 2002.

[LS97] Jed Lengyel and John Snyder. Rendering with coherent layers. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 233–242, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[LSO07] Aaron E. Lefohn, Shubhabrata Sengupta, and John D. Owens. Resolution-matched shadow maps. *ACM Trans. Graph.*, 26(4):20, 2007.

[LTYM06] Brandon Lloyd, David Tuft, Sung-eui Yoon, and Dinesh Manocha. Warping and partitioning for low error shadow maps. In *Proceedings of the Eurographics Symposium on Rendering 2006*, pages 215–226. Eurographics Association, 2006.

[LV00] Tom Lokovic and Eric Veach. Deep shadow maps. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 385–392. ACM Press/Addison-Wesley Publishing Co., 2000.

[MB95] Leonard McMillan and Gary Bishop. Head-tracked stereoscopic display using image warping. In *Proceedings SPIE, volume 2409*, pages 21–30, 1995.

[McC00] Michael D. McCool. Shadow volume reconstruction from depth maps. *ACM Transactions on Graphics (TOG)*, 19(1):1–26, 2000.

[MMB97] William R. Mark, Leonard McMillan, and Gary Bishop. Post-rendering 3d warping. In *SI3D '97: Proceedings of the 1997 symposium on Interactive 3D graphics*, pages 7–ff., New York, NY, USA, 1997. ACM.

[MT04] 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.

[NSI06] Diego Nehab, Pedro V. Sander, and John R. Isidoro. The real-time reprojection cache. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Sketches*, page 185, New York, NY, USA, 2006. ACM Press.

[NSL+07] Diego Nehab, Pedro V. Sander, Jason Lawrence, Natalya Tatarchuk, and John R. Isidoro. Accelerating real-time shading with reverse reprojection caching. In *GH '07: Proceedings of the 22nd ACM SIGGRAPH/EUROGRAPHICS symposium*

on Graphics hardware, pages 25–35, Aire-la-Ville, Switzerland, Switzerland, August 2007. Eurographics Association.

[PSS98] Steve Parker, Peter Shirley, and Brian Smits. Single sample soft shadows. Technical Report UUCS-98-019, Computer Science Department, University of Utah, october 1998.

[RP94] Matthew Regan and Ronald Pose. Priority rendering with a virtual reality address recalculation pipeline. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 155–162, New York, NY, USA, 1994. ACM.

[RSC87] William T. Reeves, David H. Salesin, and Robert L. Cook. Rendering antialiased shadows with depth maps. In Maureen C. Stone, editor, *Computer Graphics (SIGGRAPH '87 Proceedings)*, volume 21, pages 283–291, July 1987.

[RWS+06] Zhong Ren, Rui Wang, John Snyder, Kun Zhou, Xinguo Liu, Bo Sun, Peter-Pike Sloan, Hujun Bao, Qunsheng Peng, and Baining Guo. Real-time soft shadows in dynamic scenes using spherical harmonic exponentiation. *ACM Trans. Graph.*, 25(3):977–986, 2006.

[SaLY+08a] Pitchaya Sitthi-amorn, Jason Lawrence, Lei Yang, Pedro V. Sander, and Diego Nehab. An improved shading cache for modern gpus. In *GH '08: Proceedings of the 23rd ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware*, pages 95–101, Aire-la-Ville, Switzerland, Switzerland, 2008. Eurographics Association.

[SaLY+08b] Pitchaya Sitthi-amorn, Jason Lawrence, Lei Yang, Pedro V. Sander, Diego Nehab, and Jiahe Xi. Automated reprojection-based pixel shader optimization. In *SIGGRAPH Asia '08: ACM SIGGRAPH Asia 2008 papers*, pages 1–11, New York, NY, USA, 2008. ACM.

[SAPP05] Jean-François St-Amour, Eric Paquette, and Pierre Poulin. Soft shadows from extended light sources with penumbra deep shadow maps. In *Graphics Interface 2005 Conference Proceedings*, pages 105–112, 2005.

[Sch96] Gernot Schaufler. Exploiting frame to frame coherence in a virtual reality system. In *VRAIS '96: Proceedings of the 1996*

Virtual Reality Annual International Symposium (VRAIS 96), page 95, Washington, DC, USA, 1996. IEEE Computer Society.

[SD02] Marc Stamminger and George Drettakis. Perspective shadow maps. In *Siggraph 2002 Conference Proceedings*, volume 21, 3, pages 557–562, July 2002.

[SDC04] Veronica Sundstedt, Kurt Debattista, and Alan Chalmers. Selective rendering using task-importance maps. In *APGV 2004 - Symposium on Applied Perception in Graphics and Visualization*, pages 175–175. ACM SIGGRAPH, August 2004.

[SEA08] Erik Sintorn, Elmar Eisemann, and Ulf Assarsson. Sample-based visibility for soft shadows using alias-free shadow maps. *Computer Graphics Forum (Proceedings of the Eurographics Symposium on Rendering 2008)*, 27(4):1285–1292, June 2008.

[SHSS00] Marc Stamminger, Jörg Haber, Hartmut Schirmacher, and Hans-Peter Seidel. Walkthroughs with corrective texturing. In *Proceedings of the Eurographics Workshop on Rendering Techniques 2000*, pages 377–388, London, UK, 2000. Springer-Verlag.

[SJW07] Daniel Scherzer, Stefan Jeschke, and Michael Wimmer. Pixel-correct shadow maps with temporal reprojection and shadow test confidence. In Jan Kautz and Sumanta Pattanaik, editors, *Rendering Techniques 2007 (Proceedings Eurographics Symposium on Rendering)*, pages 45–50. Eurographics, Eurographics Association, June 2007.

[SKDM05] Miloslaw Smky, Shin-ichi Kinuwaki, Roman Durikovic, and Karol Myszkowski. Temporally coherent irradiance caching for high quality animation rendering. In *The European Association for Computer Graphics 26th Annual Conference EUROGRAPHICS 2005*, volume 24 of *Computer Graphics Forum*, pages xx–xx, Dublin, Ireland, 2005. Blackwell.

[SS98] Cyril Soler and François X. Sillion. Fast calculation of soft shadow textures using convolution. In *Computer Graphics Proceedings, Annual Conference Series:* SIGGRAPH '98 (Orlando, FL), pages 321–332. ACM SIGGRAPH, New York, ACM Press, July 1998.

[SS00] Maryann Simmons and Carlo H. Sequin. Tapestry: A dynamic mesh-based display representation for interactive rendering. In

Proceedings of the 11th Eurographics Workshop on Rendering, pages 329–340, 2000.

[SS07] Michael Schwarz and Marc Stamminger. Bitmask soft shadows. *Computer Graphics Forum*, 26(3):515–524, September 2007.

[SS08a] Michael Schwarz and Marc Stamminger. Microquad soft shadow mapping revisited. In *Eurographics 2008 Annex to the Conference Proceedings: Short Papers*, pages 295–298, April 2008.

[SS08b] Michael Schwarz and Marc Stamminger. Quality scalability of soft shadow mapping. In *GI '08: Proceedings of graphics interface 2008*, pages 147–154, Toronto, Ont., Canada, Canada, 2008. Canadian Information Processing Society.

[SSMW09] Daniel Scherzer, Michael Schwärzler, Oliver Mattausch, and Michael Wimmer. Real-time soft shadows using temporal coherence. *Lecture Notes in Computer Science (LNCS)*, November 2009.

[SSS74] Ivan E. Sutherland, Robert F. Sproull, and Robert A. Schumacker. A characterization of ten hidden-surface algorithms. *ACM Comput. Surv.*, 6(1):1–55, 1974.

[Ste97] A. James Stewart. Hierarchical visibility in terrains. In *Eurographics Rendering Workshop*, June 1997.

[Ste98] A. James Stewart. Fast horizon computation at all points of a terrain with visibility and shading applications. *IEEE Transactions on Visualization and Computer Graphics*, 4(1):82–93, March 1998.

[SW06] Jens Schneider and Rüdiger Westermann. Gpu-friendly high-quality terrain rendering. *Journal of WSCG*, 14(1-3):49–56, 2006.

[SW08] Daniel Scherzer and Michael Wimmer. Frame sequential interpolation for discrete level-of-detail rendering. *Computer Graphics Forum (Proceedings EGSR 2008)*, 27(4):1175–1181, June 2008.

[TK96] Jay Torborg and James T. Kajiya. Talisman: commodity real-time 3d graphics for the pc. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 353–363, New York, NY, USA, 1996. ACM.

[TQJN99] Katsumi Tadamura, Xueying Qin, Guofang Jiao, and Eihachiro Nakamae. Rendering optimal solar shadows using plural sunlight depth buffers. In *CGI '99: Proceedings of the International Conference on Computer Graphics*, page 166, Washington, DC, USA, 1999. IEEE Computer Society.

[VALBW06] Edgar Velázquez-Armendáriz, Eugene Lee, Kavita Bala, and Bruce Walter. Implementing the render cache and the edge-and-point image on graphics hardware. In *GI '06: Proceedings of Graphics Interface 2006*, pages 211–217, Toronto, Ont., Canada, Canada, 2006. Canadian Information Processing Society.

[WDG02] Bruce Walter, George Drettakis, and Donald P. Greenberg. Enhancing and optimizing the render cache. In *EGRW '02: Proceedings of the 13th Eurographics workshop on Rendering*, pages 37–42, Aire-la-Ville, Switzerland, Switzerland, 2002. Eurographics Association.

[WDP99] Bruce Walter, George Drettakis, and Steven Parker. Interactive rendering using the render cache. In D. Lischinski and G.W. Larson, editors, *Rendering techniques '99 (Proceedings of the 10th Eurographics Workshop on Rendering)*, volume 10, pages 235–246, New York, NY, Jun 1999. Springer-Verlag/Wien.

[WGS99] Michael Wimmer, Markus Giegl, and Dieter Schmalstieg. Fast walkthroughs with image caches and ray casting. In Michael Gervautz, Dieter Schmalstieg, and Axel Hildebrand, editors, *Virtual Environments '99. Proceedings of the 5th Eurographics Workshop on Virtual Environments*, pages 73–84. Eurographics, Springer-Verlag Wien, June 1999. ISBN 3-211-83347-1.

[WH03] Chris Wyman and Charles Hansen. Penumbra maps: approximate soft shadows in real-time. In *Proceedings of the 14th Eurographics workshop on Rendering*, pages 202–207. Eurographics Association, 2003.

[Wil78] Lance Williams. Casting curved shadows on curved surfaces. *Computer Graphics (SIGGRAPH '78 Proceedings)*, 12(3):270–274, Aug. 1978.

[WSP04] Michael Wimmer, Daniel Scherzer, and Werner Purgathofer. Light space perspective shadow maps. In *Proceedings of Eurographics Symposium on Rendering 2004*, 2004.

[YNS+09] Lei Yang, Diego Nehab, Pedro V. Sander, Pitchaya Sitthi-amorn, Jason Lawrence, and Hugues Hoppe. Amortized supersampling. *ACM Transactions on Graphics (Special issue of SIGGRAPH Asia 2009)*, 28(5):To appear, 2009.

[ZSXL06] Fan Zhang, Hanqiu Sun, Leilei Xu, and Lee Kit Lun. Parallel-split shadow maps for large-scale virtual environments. In *VRCIA '06: Proceedings of the 2006 ACM international conference on Virtual reality continuum and its applications*, pages 311–318, New York, NY, USA, 2006. ACM.

[ZWL05] Tenghui Zhu, Rui Wang, and David Luebke. A gpu-accelerated render cache. *Pacific Graphics, (Short Paper Session)*, October 2005.

Curriclulum vitae

Name:	Daniel Scherzer
Date of birth:	4. September 1978
Nationality:	Austrian
Email:	scherzer@cg.tuwien.ac.at
Languages:	German, English, French

Education

Abbreviations: VUT = Vienna University of Technology

1985–1989:	Primary school in Kufstein
1989–1997:	Secondary school (Gymnasium) in Kufstein
June 1997:	Graduation (Abitur) (with distinction) from BRG Kufstein
1998–2005:	Studies in Computer Science at the VUT with special emphasis on Computer Graphics
October 2005:	Graduation in Computer Science (with distinguished honours) as “Diplom-Ingenieur der Informatik” from the VUT (thesis: “Shadow Mapping of Large Environments”)
Since Dec 2005:	Phd student at the Institute of Computergraphics and Algorithms
2006-2008:	Studies in “Informatikmanagement” at the VUT
Since 2007:	Studies in Technical Mathematics at the VUT with special emphasis on Computer Mathematics
May 2008:	Graduation (with distinguished honours) in ”Informatikmanagement” as “Mag. rer. soc. oec.” from the VUT

Employment

Abbreviations: WS= winter semester; SS = summer semester;

Summer 1994:	Practica at the computer company SciCon working at the GS-Manager
July 1997–February 1998:	Army Service
WS99:	Tutor in introduction to programming (VUT)

2000:	Summer practica at the Institute for Technical Informatics, project “DSOS” (VUT)
WS00-SS02:	Organisation of the course system programming (400 students) (VUT)
WS00-WS02:	Tutor in system programming (VUT)
04/05WS,05/06WS:	Tutor in real-time graphics (VUT)
05SS:	Tutor in computer graphics 2+3 (VUT)
05SS,05/06WS:	Tutor in advanced computer graphics (FH Hagenberg)
Dec 2005–Oct 2006:	Research assistant at the EU project CROSSMOD (VUT)
Since November 2006:	Assistant Professor at the Institute of Computergraphics and Algorithms (VUT)