Master's Thesis

Apelles
A Library for GPU-Based Real-Time Global Illumination

Alexander Moschig

Institute of ComputerGraphics and KnowledgeVisualisation, TU Graz

www.cgv.tugraz.at

Supervisor: Dr.-Ing. Sven Havemann

Deutsche Fassung:
Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008
Genehmigung des Senates am 1.12.2008

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Graz, am …… ……..
(Unterschrift)

Englische Fassung:

STATUTORY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

…… ……..
date (signature)

Kurzfassung

Fotorealistische Bildsynthese strebt danach ein digitales Bild einer künstlichen Szene so zu erzeugen, dass keine Unterschiede zu einem realen Bild feststellbar sind. Die komplexe Wechselwirkung zwischen Licht und Materie bringt unterschiedliche visuelle Effekte hervor. Diese Effekte tragen ganz wesentlich zur natürlichen Wahrnehmung von synthetischen Bildern bei. Realismus ist besonders wichtig, wenn Information adäquat veranschaulicht werden soll um in eine für Menschen natürlich erfassbare Form gebracht zu werden. Schattierungen und die perspektivisch korrekte Anordnung von Linien ermöglichen eine Wahrnehmung von Tiefe in einem zweidimensionalen Bild. Schatten liefern wichtige visuelle Hinweise zu Details von Oberflächen und zur Abschätzung der relativen Position von Objekten zueinander. Schärfentiefe lenkt die Wahrnehmung auf relevante Teile bestimmter Tiefe entlang der Blickrichtung. Bewegungsunschärfe drückt gerichtete Bewegung in einem statischen Bild aus. Reflexion und Brechung von Licht ermöglichen die Unterscheidung von blickdichten, spiegelnden und lichtdurchlässigen Stoffen. Kaustiken bilden sich aufgrund von reflektiertem oder gebrochenem Licht, das auf lichtundurchlässigen Oberflächen fokussiert wird. Partielle Reflexion oder Brechung verleiht der Oberfläche von glänzenden Stoffen ihre typische Rauheit. Durch die direkte Reflexion von Licht zwischen unterschiedlich gefärbten Objekten wird die wahre Farbe von benachbarten Objekten verfälscht. Lichtstrahlen und räumliche Kaustiken werden sichtbar durch räumliche Streuung von Licht, wenn kleine Staubpartikel in der Luft fein suspendiert sind. Eingedrungenes Licht, das innerhalb eines Stoffes gestreut wird und den Stoff an unterschiedlichen Stellen wieder verlässt, bringt die gefällige Erscheinung bestimmter Stoffe in der Natur erst zur Geltung.

Die computergestützte Erzeugung synthetischer Bilder erfordert eine besonders rechenintensive Simulation der komplexen Wechselwirkung zwischen Licht und Materie. Ein einzelner Punkt einer Oberfläche wird potentiell von seiner gesamten Umgebung beleuchtet, weil Lichtstrahlen während ihrer Ausbreitung durch die Umgebung sich an Oberflächen vermehren und abgelenkt werden, bis die Strahlen schließlich auf das Auge auftreffen oder absorbiert werden. Deshalb erfordern die eingesetzten Algorithmen besonders viele Strahlen oder müssen besonders lange ausgeführt werden um befriedigende Ergebnisse zu liefern. Ferner müssen die Algorithmen eine Auswertung von globalen Daten vornehmen, die von einer Vielzahl von anderen Oberflächenpunkten abhängen um die Effekte globaler Beleuchtung an einem Oberflächenpunkt berechnen zu können. Unter Vernachlässigung von extrem komplexen Effekten globaler Beleuchtung können heutzutage synthetische Bilder zur interaktiven Anzeige nur dann ausreichend schnell erzeugt werden, wenn die Algorithmen auf hoch leistungsfähiger Hardware entfernt ausgeführt werden und die Ergebnisbilder zur lokalen Anzeige zurück übertragen werden. Im Gegensatz dazu erreicht die Erzeugung von synthetischen Bildern auf Grafikhardware für Endverbraucher eine Näherung von Beleuchtungseffekten in Echtzeit. Dies wird durch den Einsatz von hardwarebeschleunigten Techniken zur effizienten Rasterisierung ermöglicht. Diese Vorgehensweise erfordert nur Daten, die an jenem Oberflächenpunkt zur Verfügung stehen, dessen Beleuchtung momentan berechnet wird.

Die Evolution der Grafikhardware mit programmierbaren Datenpfaden hat eine Vielzahl an Algorithmen hervorgebracht, die hardware-beschleunigte Techniken zur effizienten Rasterisierung einsetzen um einzelne Effekte globaler Beleuchtung in Echtzeit anzunähern. Diese Arbeit beschäftigt sich mit dem Entwurf der Bibliothek Apelles um zu ergründen, welche Algorithmen sich so kombinieren lassen, dass Effekte globaler Beleuchtung in Echtzeit überzeugend angenähert werden können. Aufgrund der enormen Vielfalt an verfügbaren Algorithmen beschränkt sich die vorliegende Arbeit auf die Erzeugung von harten und weichen geometrischen Schatten, die von lichtundurchlässigen Objekten geworfen werden. Eine beliebige Anzahl von Lichtquellen kann in der Szene auf einfache Weise aufgestellt werden. Die verfügbaren Lichtquellen umfassen gerichtete Lichter, Punktlichter, Spotlichter und Flächenlichtquellen. Zwei Algorithmen ermöglichen die Erzeugung weicher Schatten von Flächenlichtquellen. Optisch gefällige Schatten, deren Weichheit sich mit der Distanz zwischen Objekten ändert, lassen sich in Echtzeit erzeugen. Physikalisch korrekte Schatten mit weichen Kanten können erzeugt werden, wenn ein höherer Aufwand akzeptabel ist. Zusätzlich bietet Apelles einen hardwarebeschleunigten parallelen Algorithmus an um die Effizienz der Einstellung aller Lichtquellen zu steigern. Die Sicht des Lichtes kann auf jene Objekte beschränkt werden, die für die Bestimmung von Schatten im Schein des Lichts maßgeblich sind. Der prägnante und offene Entwurf der Softwarearchitektur von Apelles ermöglicht auf einfache Weise fehlende Effekte globaler Beleuchtung nachzurüsten, indem die implementierten Algorithmen mit vorhandenen Algorithmen kombiniert werden.

Abstract

Photorealistic image synthesis strives to create a digital image of an artificial scene indistinguishable from a real-world photograph taken with a digital camera. The complex interaction of light and matter causes different visual effects. These effects contribute considerably to the natural perception of synthetic images. Realism is particularly important for presenting information appropriately to support human understanding. Shadings and the perspective-correct composition of lines enable to perceive depth in a two-dimensional image. Shadows provide important visual cues for surface detail and for estimating the relative positions of objects in space. Depth of field focuses perception on relevant things only in everything being visible with a gradual blur along the depth of view. Motion blur expresses directional activity in a static image. Reflection and refraction of light allows to identify opaque, specular and transmissive materials. Caustics are formed by reflected or refracted light that is focused on opaque surfaces. Glossy reflection and refraction support to identify shiny materials exhibiting surface roughness. Color bleeding caused by inter reflection of light between differently coloured objects biases the true colour of nearby objects. Light shafts and volumetric caustics are introduced due to volume scattering of light if small particles are finely suspended in the air. Subsurface scattering of light exposes the pleasing appearance of distinctive natural materials.

Generating synthetic images requires to simulate the complex interaction of light and matter with a computer which is a computationally intense process. A single surface point is potentially illuminated by everything else in the scene. While travelling through the scene rays of light are spawned and redirected at multiple surfaces until, finally, impinging at the eye or being absorbed. Therefore, algorithms are required to trace huge numbers of rays or to run sufficiently long to yield convincing results. Furthermore, the algorithms need to evaluate data acquired globally from multiple other surface points for computing global illumination effects at a single surface point. While disregarding extremely complex effects of global illumination, today synthetic images can only be displayed at interactive rates by executing the algorithms remotely on extensive hardware resources off-site and transmitting the results to displays on-site. In contrast, real-time generation of synthetic images on consumer graphics hardware convincingly approximates visual effects with GPU-enabled rasterisation techniques which only require data locally available at the surface point currently being shaded.

The evolution of the programmable graphics pipeline has given birth to numerous algorithms which utilise GPU-enabled rasterisation techniques to approximate individual effects of global illumination in real-time. This thesis focuses on a library named Apelles in order to discover which of these algorithms can be combined to generate convincing global illumination effects in real-time. As a consequence of the huge variety of the available algorithms, this thesis concentrates on the generation of both geometric hard and soft shadows cast by opaque objects only. An arbitrary number light sources can be set-up in a scene naturally. The supported types of light sources include directional lights, point lights, spot lights and area lights. According to area lights, two algorithms are supplied for generating soft shadows. Visually pleasing soft shadows exhibiting contact hardening can be generated in real-time. Physically-correct soft shadows can be generated if a considerably higher cost is acceptable. Additionally, a GPU-enabled parallel algorithm is supplied to optimise the set-up of light sources with respect to those objects within the view of the light source that are relevant for resolving occlusion efficiently. The concise and open design of Apelles inherently supports to include missing global illumination effects by combining library algorithms with existing algorithms.

Acknowledgements

I am highly grateful for having the opportunity to graduate in the exciting field of computer graphics at the Institute of Computer Graphics and Knowledge Visualization at Graz University of Technology. Several people have contributed to the evolution of this thesis. I would like to thank my supervisors and reviewers for their invaluable assistance and constructive suggestions.
In particular, I would like to thank my family for making my studies possible and for their encouragement over the time as this thesis has evolved.

Table of Contents

Chapter 1

Introduction

Taking a picture with a camera generates a photorealistic image of the real world. Today, an image is acquired by measuring focused radiation of light with a digital sensor instead of a light-sensitive film. Although photorealistic image synthesis strives to achieve the same ambitious goal as photography, the used strategy is completely different and contrary. A synthetic camera is put up in a three-dimensional synthetic scene to generate an image without any perceivable differences compared to the real world. The scene specifies a spatial set-up of objects, light sources and a camera. Objects are described by shape, size, and material properties (e.g. colour, refractive index, etc.). Light sources are described by emissive properties (e.g. colour, attenuation, etc.). The camera is described by mapping properties (e.g. orientation, focal length, etc.). The image generation process (render process) requires to accurately simulate the complex nature of the interaction of light and matter using a computer. Digital photography and photorealistic image synthesis exhibit similarity in the final step of the image generation process. A colour must be assigned to the picture elements (pixels) comprising the resulting raster image.

The vital importance of photorealistic image synthesis becomes apparent in the context of visualisation to present information supporting human understanding. Considerable applications can be found in architecture, simulation, medicine and education. For example, in architecture a new office design can be evaluated under different lighting conditions (Figure 1.1). Furthermore, a virtual environment allows to evaluate the office design in different scenarios: reachability and visibility of emergency exits in case of crowded corridors and fire (different atmospheric conditions due to smoke).

Figure 1.1: Using iray® an office design is evaluated under different lighting conditions according to the time of day. Left: Day view. Middle: Evening view. Right: Night view (images courtesy of *mental images GmbH*, http://www.mentalimages.com, last access 2012-05-07).

Of course, entertainment industry has always been promoting significant advances in visual realism for movies and computer games. Pixar Animation Studios developed PhotoRealistic RenderMan®[1] for creating animated feature films (e.g. Toy Story). In particular, RenderMan® is used for visual effects in nearly all award-winning movies. Another example is the game Doom III[2] developed by Id Software that showed hard shadows for dynamic objects to the larger public for the first time.

[1] http://renderman.pixar.com/products/tools/rps.html (last access 2012-05-07).
[2] http://www.idsoftware.com/games/doom/doom3 (last access 2012-05-07).

Visual Effects in Paintings and Pictures For creating an image indistinguishable from a photograph, it is important to consider visual cues in paintings and pictures. In the antiquity painters used monochrome shadings (from bright to dark) and coloured shadings (from warm to cold) to enhance the depth effect in paintings. For example, filling a circle with a shade from bright to dark adds virtually a third dimension to a two-dimensional image as the circle turns into a sphere. In the renaissance DÜRER and DAVINCI introduced the perspective-correct composition of lines to support depth perception in flat paintings. Impressionist artists used warm and cold colours to increase the virtual distance between foreground and background in a painting.

Occlusion and shadows, cast by opaque objects in a picture, support to estimate relative positions of objects in space. In addition, if light shines on surfaces at a shallow angle, shadows unveil inconspicuous surface features (e.g. bulges and dents or unique landscape topography in the morning or evening). Soft shadows exhibiting hardening on contact convey further visually important information about the surface of objects, the spatial relation of objects and the size of light sources (Figure 1.2, top left). Environmental shadows additionally enrich the natural appearance of objects if light scatters uniformly in all directions due to diffuse inter reflection (indirect illumination) or diffuse light (e.g. overcast sky).

Depth of field aids in identifying only relevant things in everything we see in a picture. By altering aperture the visual apparatus of a camera can be setup to focus in a limited range at a specific distance only. Objects closer or farther than this range are out of focus and, therefore, appear gradually blurred in the taken picture.

Motion blur dramatically emphasises activity in a static picture and indicates the direction of movement. If an object is moving when the shutter is open, the image of the object moves across the sensor or film while being exposed. Hence, in the taken picture the object will have the appearance of being smeared in the direction of movement.

Reflection and refraction of light enables to distinguish between opaque, specular and transmissive objects. As a consequence of refraction, light is focused and, therefore, caustics are formed on diffuse surfaces (Figure 1.2, top right). For example, caustics appearing on the ground of a swimming pool contribute significantly to the natural perception of vivid water as a liquid. Of course, caustics can additionally originate from reflected light being focused by concave surfaces (e.g. light patterns inside a glazed mug).

Glossy reflections and refractions are a critical visual element for authentic reproduction of materials which exhibit surface roughness (Figure 1.2, bottom left). For example, in architectural visualisations a shiny parquet floor mirrors the blurry image of a white pillar due to glossy reflection. Glossy refraction causes objects appear to be translucent (e.g. slightly frosted glass of a bathroom window).

Indirect illumination introduces color bleeding (Figure 1.2, bottom right). Particularly, color bleeding can be irritating upon taking a picture of a person standing close to a coloured wall. Indirect illumination causes the wall to diffusely reflect coloured light instead of white light onto the person. Hence, such biased light creates an unnatural appearance of skin in the taken picture. As a consequence of color bleeding and colour adaption in the human brain, the colours in the taken picture differ from the colours seen in the viewfinder. The human brain constantly strives to instantly adapt colour perception such that objects appear always to be illuminated by white light regardless of color bleeding.

Different atmospheric conditions are observable as a result of volume scattering of light in participating media (e.g. smoke and fog). The TYNDALL effect causes light shafts and god rays. For example, god rays cast by ships into seawater are inevitable for realistic underwater scenes. Of course, transmissive materials introduce volumetric caustics depending on their surface curvature and light absorption properties. Furthermore, subsurface scattering of light causes particular translucent materials to uniquely expose their attractive texture (e.g. marble, jade or human skin).

Physically-Correct Photorealistic Image Synthesis Simulating the propagation of light rays as in geometrical optics for rendering a photorealistic image of a virtual scene is a computationally intense process. Indirect illumination causes a single point in a scene to be illuminated by everything else in the scene due to inter reflections. Light rays are spawned either by reflection or refraction and propagate repeatedly until finally an object absorbs the according photon depending on the wavelength of the photon. Physically-correct photorealistic image synthesis usually disregards physical optics (polarisation, interference, dispersion, diffraction), fluorescence and phosphorescence.

Instead of tracing rays of light, rays are cast for each pixel of the synthetic image to determine through intersection testing, first, what is visible and, then, which colour to assign. The colour is estimated by evaluating an illumination model (shader) together with occlusion testing which involves casting shadow rays. A recursive ray tracer successively traces a fan-out of rays at each intersection deeper into the scene in order to account for multiple specular reflections and refractions.

By oversampling a single pixel with a bundle of rays in a non-deterministic way, stochastic ray tracers are capable of rendering soft shadows, depth of field, motion blur, glossy reflections, glossy refractions and color bleeding due to diffuse reflections. Despite the capability of a stochastic ray tracer to implement a rather unsimplified model of light, the number of rays to be traced is severely increased by additional fan-outs of rays compared to recursive raytracing.

In contrast, a path tracer stems high numbers of rays by tracing only single rays for reflection or refraction. As a consequence of reduced complexity, images rendered with a path tracer are typically noisy due to high variance.

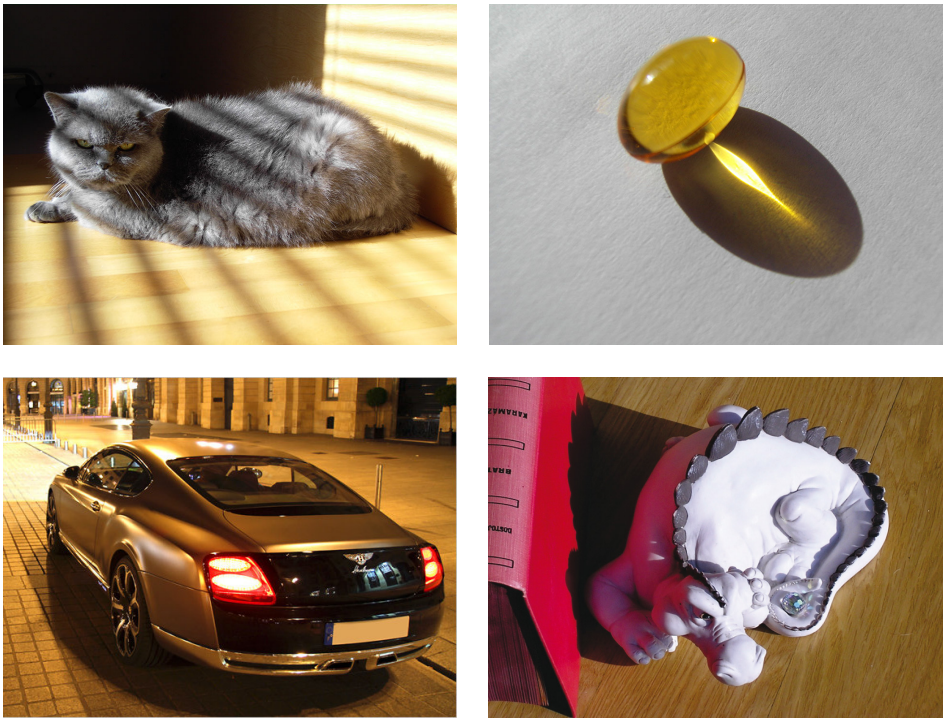


Figure 1.2: Shadows, specular effects and color bleeding.
Top left: Sunlight passes through a window with open blinds. The shadows caused by the blinds on the floor become softer from left to right as the distance between a blind (occluder) and the floor (receiver) increases. Small scale shadows are inevitable to convey the fluffiness of the fur of the cat.
Top right: Coloured caustics are formed by focusing sunlight through a transparent capsule filled with salmon oil. Note that a reflection of the caustic appears on the lower backside of the capsule due to *total reflection*.
Bottom left: Glossy and specular reflections on a car with matt paint (image courtesy of *carface*). Glossy highlights of street lighting appear on the roof of the car. Specular reflections of a passed by entrance appear on the black lid of the boot.
Bottom right: Color bleeding biases the colour of the left side of the dragon because the nearby book cover diffusely reflects red light onto the white dragon (image courtesy of *Jaroslav Křivánek*).

A photon tracer approximates indirect illumination on diffuse surfaces using a two pass approach. In the first pass, photons are emitted from light sources and traced through the scene until they are finally absorbed. When a photon hits a diffuse surface, it is registered in a data structure (photon map). In the second pass direct illumination, specular reflections and transmissions are computed using stochastic raytracing. Indirect illumination is computed by incorporating cached data in the photon map. A photon tracer is very efficient in rendering caustics which a stochastic ray tracer can only render at a very high cost. Remaining artefacts in the resulting image are removed by increasing the number of photons. On the contrary, a stochastic ray tracer will only yield a satisfying result if the algorithm runs long enough.

Real-Time Approximation of Global Illumination As a consequence of the high complexity in simulating light transport, real-time applications employ collections of convincing local shading tricks to efficiently approximate global illumination effects such as shadows, reflection, refraction, caustics and indirect illumination including color bleeding. The utilised approaches are optimised for particular scene setups, geometry and constrained viewports. Hence, in arbitrary scenes with deformable objects, free camera and light control, visually disturbing defects become objectionable easily.

Obviously, for increasing efficiency when processing a pixel it would be reasonable to exploit coherency of neighbouring pixels. Intermediate results could be cached in order to utilise them subsequently without requiring to recompute them repeatedly. Unfortunately, an algorithm utilising coherency for speedup is unsuitable to be implemented in cheap hardware efficiently. The implementation would require individual hardware elements that communicate with each other; the most expensive hardware design possible.

According to visibility testing, a graphics processing unit (GPU) achieves real-time performance by inefficiently discarding hidden pixels (z-buffer algorithm [Cat74]). Therefore, a GPU is designed to generate coloured pixels rapidly by evaluating simple local lighting models. A local lighting model only relies on data available at the surface point (e.g. surface normal, incident angle of light, texture colour, etc.). In contrast, global illumination algorithms need data from multiple other surface points in the scene than the surface point currently being shaded.

Although recent advances in both graphics hardware and software show interactive physically-correct simulation of light in complex scenes, real-time performance is still impossible to achieve on current systems. Global illumination algorithms based on ray tracing require to trace at least 20-40 rays per pixel to yield satisfying results [WDB*06]. Suppose that high definition content shall be rendered at 60 frames per second. Each frame is 1920 pixels wide and 1080 pixels high. Tracing a minimum of 20 rays per pixel in a single frame would require hardware being capable of tracing two and a half billion rays each second.

At the GPU Technology Conference in 2010 mental images iray® demonstrated push-button rendering of a physically-correct illuminated complex architectural scene with three million polygons on a workstation running two high-end GPUs [Hua10]. The path traced image converged on the final frame in about ten seconds. Despite the very accurate shadow detail, color bleeding computations were omitted. By moving computation into an off-site cloud running a cluster of 32 GPUs and streaming back the resulting images on a notebook showed almost interactivity. However, in movie production scenes are much more complex. In 2006 a typical scene in a Pixar movie consisted of hundreds of light sources, thousands of textures, ten-thousands of objects, hundred-thousands of polygons and shaders with ten-thousand lines of code [Chr06]. Hence, in such scenes shaders used to evaluate global illumination are much more expensive than visibility ray tracing.

Interactive photorealistic image synthesis was made possible at all by the GPGPU (general purpose GPU) revolution which started in 2008. GPGPU allows to harness the immense floating-point power of the massive parallel graphics processor for general purpose computations. Simulations in science and industry require an enormous amount of computational resources which usually super computers can provide only. Replacing numerous multi-core CPUs with a few many-core GPUs results in impressive speedups, more accuracy and less power consumption which makes computing more economical and affordable. More importantly, the performance boost allows new applications which were impossible before. For example, a polio virus infection can be interactively visualised by computing the movements of 100 million atoms which essentially turns a computer into a computational microscope. Of course, GPUs are incapable of performing all types of computations. Therefore, a hybrid combination of CPUs and GPUs is inevitable in practice.

Apelles – A Library for GPU-based Real-Time Global Illumination In the last decade, the evolution of the programmable graphics pipeline has given birth to numerous GPU-enabled algorithms which approximate individual aspects of global illumination in real-time utilising GPU-accelerated rasterisation techniques. This thesis focuses on Apelles, a library for global illumination computation running entirely on a GPU in order to discover which of these GPU-friendly algorithms can be combined to render convincing global illumination effects in real-time. As a consequence of the huge variety of GPU-based approaches to approximate individual aspects of global illumination, this thesis concentrates on shadow generation only. Accordingly, Apelles is capable of rendering both geometric hard and soft shadows caused by opaque occluders.

This thesis is structured as follows. Image-based approaches are investigated in the large field of hard and soft shadow generation to identify algorithms based upon *Shadow Mapping* [Wil78] which are capable of generating geometric shad-

ows for point lights, directional lights and area lights (see Chapter 2). Suitable algorithms avoid costly pre-computation and optimisation of acceleration data structures. Furthermore, cached intermediate results are reused intelligently in multiple passes while maintaining high frame rates.

After providing the theoretical background of global illumination the properties and consequences of the identified algorithms are discussed to detail how close the approximations can resemble the ground truth (see Chapter 3). Furthermore, a GPU-enabled algorithm is suggested to optimise light view volume culling (see Section 3.5).

The implementation demonstrates how a stable, open and efficient OpenGL-based implementation of the proposed concise architecture of Apelles is achieved (see Chapter 4). A local illumination model is extended with programmable shaders, render-to-texture and multi-texturing. Furthermore, the implementation of the GPU-enabled algorithm for optimising light view volume culling with an OpenCL-based *Parallel Reduction* is discussed (see Section 4.1.4).

In order to thoroughly expose the capabilities of the implementation of Apelles performance and visual results are evaluated with a complex model of the Kölner Dom (see Chapter 5). Particularly, the differences between convolution-based and sampling-based soft shadow generation are evaluated in more detail (see Section 5.2.2).

Several starting points are suggested for improving the limitations of Apelles concerning aliasing artefacts as well as convolution-based and sampling-based soft shadow generation (see Section 6.2). Additionally, since Apelles supplies shadow generation only, other capable approaches addressing global illumination effects are proposed to be integrated into Apelles for including specular and glossy reflections, refractions and diffuse indirect illumination (see Section 6.2.5).

Chapter 2

Related Work

Real-time shadowing algorithms can be categorised into two types: geometry-based and image-based. Both have had their practical application, however, image-based algorithms are the state of the art for real-time applications. These algorithms became so popular because they are easy to implement and perform well while generating shadows of acceptable quality. Furthermore, they are applicable to arbitrary geometry and scene configurations. Of course, all image-based algorithms are prone to aliasing artefacts.

Geometry-based algorithms are based on *Shadow Volumes* [Cro77] which are capable of generating shadows with precise boundaries. However, finding and forming of shadow volumes involves complex pre-processing steps (silhouette detection and edge extrusion). As a consequence, pre-processing requires a high fillrate. Additionally, to facilitate pre-processing, input is restricted to simple scenes with polygonal data only.

At the very heart of all image-based algorithms lies *Shadow Mapping* [Wil78]. The initially proposed algorithm suffers from aliasing artefacts and incorrect self-shadowing. Aliasing artefacts arise from undersampling, oversampling and reconstruction errors. These sampling errors manifest in visually disturbing jaggies at the boundaries of hard shadows. Furthermore, undersampling and numerical imprecision aggravate incorrect self-shadowing.

Soft shadows are introduced by light sources which exhibit spatial extend. The shadows caused by extended light sources contain partially and fully shadowed regions (penumbra and umbra). The different characteristics of soft shadows considerably contribute to the visual richness of an image. In particular, soft shadows which harden on contact are visually important to perceive spatial relations of objects in a two-dimensional image. Numerous approaches with different consequences on robustness, performance and accuracy have been proposed for image-based generation of soft shadows. Visually plausible soft shadows can be generated in real-time with convolution. Robust generation of physically plausible soft shadows is achieved with reconstructing and back projecting occluders. Environmental shadows are convincingly approximated with locally evaluating the visibility of nearby geometry to further enrich the final image.

As a consequence of the absence of a robust algorithm for generally computing shadows in real-time, decent overviews have been written which suggest when to use which algorithm addressing particular limitations of shadow mapping [EASW09, SWP10]. The following survey of approaches for hard and soft shadow generation is confined to the image-based computation of geometric shadows from opaque occluders only.

2.1 Hard Shadows

This section discusses the following attempts that have been made for overcoming the limitations of shadow mapping. Improved reconstruction filters lessen undersampling due to insufficient shadow map resolution. Fitting techniques maximise the effectiveness of shadow map resolution to hide undersampling. Warping and global partitioning techniques strive to reduce undersampling due to perspective aliasing. Adaptive partitioning analytically avoids undersampling due to projection aliasing. Temporal reprojection enables to incorporate previously taken samples for decreasing undersampling. Oversampling is addressed by recent contributions which enable to pre-filter the shadow map and make it applicable to standard texture minification techniques. Alias-free approaches avoid sampling errors completely through irregular sampling which leads to pixel-accurate results. Depth biasing simply resolves some causes of incorrect self-shadowing due to imprecisions with offsetting depth values slightly and adaptively. Omnidirectional shadows cast by point light sources require shadow maps covering a spherical view which can efficiently be acquired with geometry shaders and cube shadow maps.

2.1.1 Undersampling

Undersampling occurs if the sampling rate used to sample the depth into the shadow map is lower than the sampling rate at which the shadow map is sampled during shadow generation. Undersampling is caused by limited shadow map resolution, projection aliasing and perspective aliasing.

Projection aliasing occurs locally if surfaces are almost parallel to the direction of light. Therefore, adaptive algorithms analytically adjust the sampling rate locally depending on the orientation of surfaces. Perspective aliasing occurs if the shadow map is enlarged close to the viewer due to perspective projection. The global distribution of shadow map samples along the viewing direction is adjusted accordingly with a different parameterisation (warping). Therefore, the sampling rate close to the viewer increases as the sampling rate for distant objects decreases. The sampling rate can be raised further by splitting the view frustum into several sub-frusta and using a shadow map for each sub-frustum (partitioning).

2.1.1.1 Reconstruction

As a consequence of nearest neighbour reconstruction, undersampling causes reconstruction errors to appear as jagged shadow boundaries if the shadow map is magnified due to projection aliasing and perspective aliasing. Of course, if shadow maps were applicable to standard resampling techniques of texture mapping, undersampling could be significantly reduced by blurring jagged shadow boundaries in real-time. Several capable ideas were suggested to make shadow maps applicable to GPU-enabled filtering by linearising the occlusion test (Section 2.1.2).

Percentage-Closer Filtering (PCF) [RSC87] averages the results of several occlusion tests (samples) to introduce a refined reconstruction filter. The intensity of the blur depends on the number of samples under the support of the filter. Therefore, performance decreases dramatically with higher order filters and more blurriness, respectively. In addition, large filter kernels deteriorate incorrect self-shadowing and cause incorrect soft shadows which appear to be unrelated to the according shadow caster. It has to be noted, however, that current GPUs support bilinear PCF without any cost to hide minor undersampling. In the context of visual plausible soft shadow computation, significant performance improvements were achieved with stratified sampling (Section 2.2.1).

Shadow Silhouette Maps [SCH03] reconstruct refined shadow boundaries from a piecewise linear approximation instead from the piecewise constant approximation of naive shadow mapping. The algorithm can be interpreted to align orthogonal lines enclosing cells of a regular grid of depth samples in order to match the silhouette of shadow boundaries (Figure 2.1). In addition to a shadow map, the silhouette map stores the location of points on the geometric silhouette of shadow casters as seen from the light source. Connecting these points enables to reconstruct piecewise linear edges of shadow boundaries. Upon shading a fragment, first, querying the shadow map for four neighbouring texels identifies if a fragment lies on a shadow boundary. This is the case if at least one query result differs from the others because shadow boundaries separate shadowed and lit regions. Then, the area of the silhouette map texel containing the projected fragment is intersected with the shadow edge approximated from the silhouette map. The intersection divides the area of the silhouette map texel into four regions. In the corner of each region lies a shadow map texel because the shadow map is offset by half a texel with respect to the silhouette map. The fragment is shaded based on the result of the depth test of the shadow map texel which lies in the corner of the region containing the fragment.

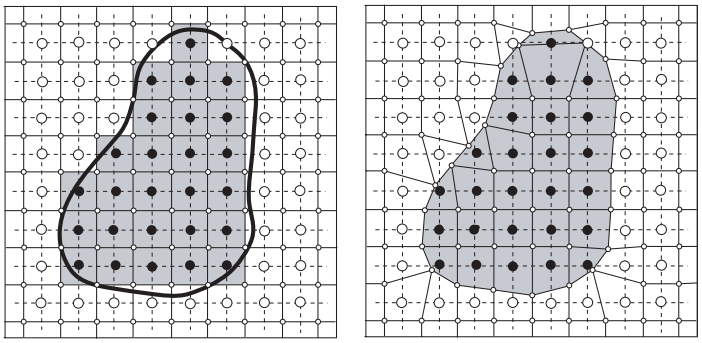


Figure 2.1: Shadow Silhouette Maps: Applying shadow silhouette maps corresponds to deforming a regular grid in order to yield the grid lines be aligned with boundaries of geometry projected to light-space. Left: The dotted lines indicate the silhouette map. The large circles indicate the depth values of the shadow map. The projected silhouette is superimposed to highlight its coarse representation in the shadow map. Right: The grid lines of the shadow map are aligned to finely match the projected silhouette (image courtesy of [SCH03]).

Of course, generating the silhouette map requires to efficiently detect silhouettes similar to shadow volumes. This considerably limits performance and hinders using the algorithm for complex geometry in real-time applications. Furthermore, artefacts are introduced if the resolution of the silhouette map is too low to represent silhouettes of surfaces with high curvature. As a consequence of storing the location of one silhouette point per silhouette map texel only, reconstruction fails if several silhouettes overlap. [Sen04] extends silhouette maps and proposes a simple filtering scheme for improved texture magnification.

Deep Shadow Maps (DSM) [LV00] enable to apply filtering for reconstruction and resampling which allows for anti-aliasing of undersampling artefacts. Each texel stores a piecewise linear representation of the visibility function at all depths along a ray of light passing through the texel. The representation is constructed by filtering the weighted transmittance functions of neighbouring texels at each depth which reduces aliasing. Subsequently, the representation is compacted to reduce the number of data points. Each piecewise linear transmittance function is a combination of the surface transmittance and the volume transmittance (Figure 2.2). Surface transmittance is approximated by tracing the light ray passing through a texel. Whenever the ray hits a surface at a certain depth, the light intensity is reduced according to the transparency of the intersected object. Therefore, semitransparent surfaces and partial coverage of opaque occluders are included. The volume transmittance is approximated by interpolating the integrated and exponentiated extinction function between vertices. The extinction function is approximated by uniformly sampling and linearly interpolating the atmospheric density along the light ray. In essence, DSM enable to query the attenuation of light at all depth levels along the ray through a given texel. Therefore, DSM are capable of generating geometric shadows for complex semitransparent geometry such as hair or fur which require extensive self shadowing. Additionally, volumetric shadows can be generated in participating media such as smoke or fog. In comparison to standard shadow mapping, generation of DSM is costly. In contrast, for scenes with complex geometry DSM consume less memory than standard shadow maps while producing shadows of superior quality. However, DSM were proposed as a software-based off-line rendering algorithm. This complicates an implementation on the GPU for two reasons. First, the required amount of storage per texel varies because each texel refers to a list of different length which is determined by the number of data points approximating the visibility function. Second, constructing the visibility function from a large number of samples per texel requires to efficiently process the input vertices in increasing depth order using an acceleration data structure which has to be amenable to a practical implementation on the GPU. However, DSM were implemented on the GPU to extend direct volume rendering with shadows at interactive rates [HKSB06]. As a consequence of restrictions in GPU memory access, the algorithm is incapable of rendering semitransparent geometry such as hair.

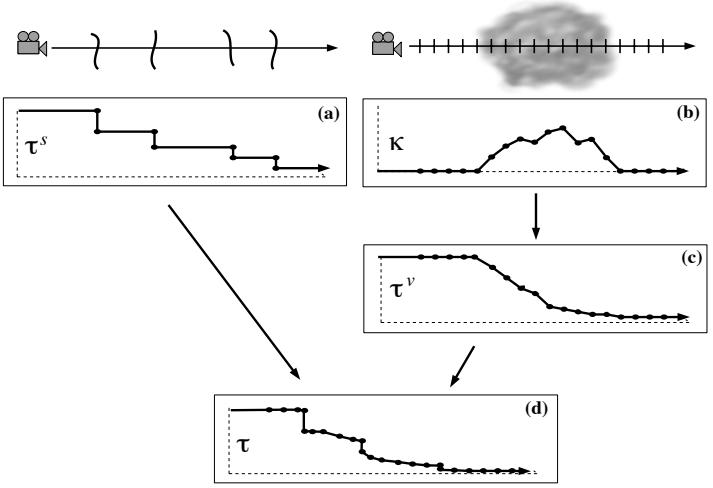


Figure 2.2: Deep Shadow Maps: Construction of a transmittance function. (a): First, the surface transmittance function τ_s is obtained by reducing the transmittance at each intersection point along a given ray according to the transparency of the hit object. (b): Second, sampling the atmospheric density at regular intervals along the ray yields the extinction function κ. (c): Third, the volume transmittance τ^v is obtained by integrating and exponentiating the extinction function. (d): Finally, the surface transmittance and volume transmittance are multiplied to yield the transmittance function τ (image courtesy of [LV00]).

2.1.1.2 Fitting

Fitting or *focusing* improves undersampling issues by utilising inactive regions of the shadow map to increase the effective resolution [BAS02a], [SD02]. If only a few objects are visible in large scenes, the view frustum of the light is large compared to the view frustum of the camera. Therefore, the majority of available shadow map resolution is wasted on invisible objects. Instead, the view frustum of the light should be fitted to only enclose visible objects and potential occluders which cast shadows onto visible objects (Figure 2.3).

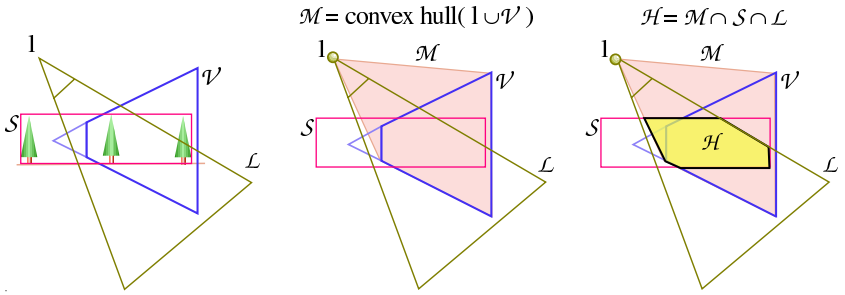


Figure 2.3: Fitting or focusing of the view frustum of the light determines the volume $\mathcal{H}$ enclosing all objects that should be rendered into the shadow map. Right: Exemplary setup of the view frustum of the light $\mathcal{L}$ at position l, the view frustum of the camera $\mathcal{V}$ and the bounding volume of the scene $\mathcal{S}$. Middle: First, the convex hull $\mathcal{M}$ is calculated from l and $\mathcal{V}$. Left: Then, $\mathcal{H}$ results from the intersection of $\mathcal{M}$, $\mathcal{S}$ and $\mathcal{L}$, respectively, (image courtesy of [SD02]).

The volume enclosing these objects is calculated by first determining the convex hull of the light position and the view frustum of the camera. Then, the convex hull is intersected with the view frustum of the light and the volume enclosing all objects in the scene. The resulting volume of objects which should be rendered into the shadow map, determines all sides of the improved view frustum of the light. If a coarse bounding volume representation of all objects in the scene has been used to accelerate intersection testing, a tighter fitting can be found by applying visibility algorithms subsequently. Initially, fitting suggested to determine the four sides of the view frustum of the light by focusing on the visible pixels in the unfitted view frustum of the light [BAS02a]. However, the approach is costly because it requires to read back the frame buffer and to efficiently find the two-dimensional convex hull of the visible pixels. Of course, fitting introduces temporal aliasing artefacts which manifest in shadow flickering, if the sampling rate of the shadow map changes abruptly between consecutive frames. Temporal aliasing will potentially occur if the light moves, if the camera moves or rotates, if objects move into or out of the view frustum of the camera and if objects are added or removed.

2.1.1.3 Warping

The shadow map of a point or spot light is magnified close to the view plane due to perspective projection. For some configurations of light and eye, magnification is reducible by employing a reparameterisation which globally improves undersampling due to perspective aliasing. In particular, the achievable improvement in shadow quality is limited to configurations in which perspective aliasing appears. Therefore, the improvement reaches its maximum when the light direction is orthogonal to the view direction and decreases as the light changes direction. Finally, the reparameterisation becomes entirely ineffective if the direction of light becomes parallel to the view direction where no perspective aliasing occurs. Accordingly, upon approaching this configuration the reparameterisation needs to successively turn into the uniform parameterisation (standard shadow mapping).

Perspective Shadow Maps (PSM) [SD02] initially presents warping to reduce perspective aliasing. First, fitting is used to determine the convex body of the scene that is relevant for shadow calculation. Hence, all potential shadow casters and receivers are included upon applying the reparameterisation. Then, the shadow map is generated in post-perspective space to reduce magnification of the shadow map due to perspective projection. Therefore, upon generating the shadow map first the view transformation is applied to the transformation of the light source. Then, the scene is rendered from the transformed view of the light to generate the shadow map. As a consequence of generating the shadow map in post-perspective space, on the image plane the size of the projected shadow map texels remains constant along the view direction (Figure 2.4).

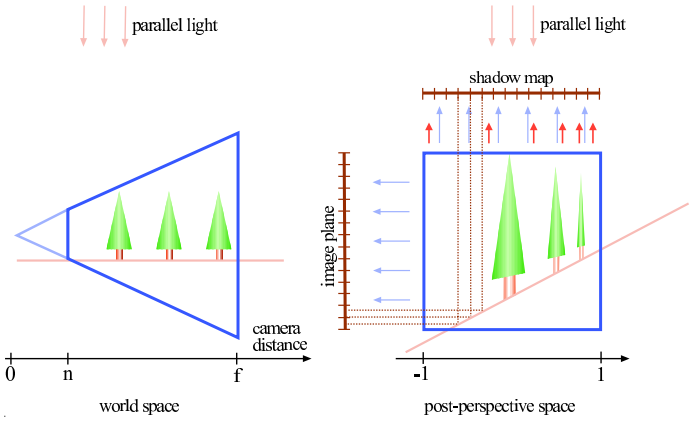


Figure 2.4: Perspective Shadow Maps. Generation of the shadow map in post-perspective space. Trees are illuminated by a directional light directly from above. Right: As a consequence of the parallel projection in post-perspective space, the samples of the shadow map are distributed evenly along the view direction. For comparison, the red arrows below the shadow map indicate how the shadow map would have been sampled if the parameterisation had been uniform (image courtesy of [SD02]).

However, the initially proposed approach of PSM infers several limiting consequences [WSP04]. The reparameterisation is most advantageous if the transformation yields a directional light in post-perspective space. This is only true for two configurations. First, if the light is directional and the direction of light is parallel to the view plane. Second, for point lights residing on the camera plane which is parallel to the view plane (miner's head lamp). In the latter case, the reparameterisation enables to move the point light on the camera plane further away from the camera. It should be noted, however, that lights possibly change their type and direction when being transformed into post-perspective space. Therefore, a practical implementation of PSM needs to handle special cases appropriately for many configurations of both directional and point lights. Furthermore, the effect of the reparameterisation highly depends on the viewing transformation. As a result, the near plane of the camera should be moved as close as possible to the scene. While objects close to the near plane of the viewer receive more samples, distant objects receive less which inevitably reintroduces undersampling. Moreover, a singularity of the reparameterisation causes objects behind the camera to appear upside-down on the opposite side of the infinity plane in post-perspective space of the light. Hence, these objects are missing in the shadow map. To circumvent this problem, the camera should be moved back to include all potential shadow casters upon generating the shadow map. However, this further reduces the achievable shadow quality considerably. Additionally, the reparameterisation aggravates incorrect self-shadowing due to scaling objects non-uniformly. In conclusion, numerous issues preclude both a practical implementation and an intuitive application of PSM.

Light-Space Perspective Shadow Maps (LiSPSM) [WSP04] avoid some of the limitations of PSM by separating the perspective transformation of the reparameterisation from the viewer. Additionally, the reparameterisation preserves the direction and type of lights, because the transformation into post-perspective space is setup to be always orthogonal to the direction of light (Figure 2.5).

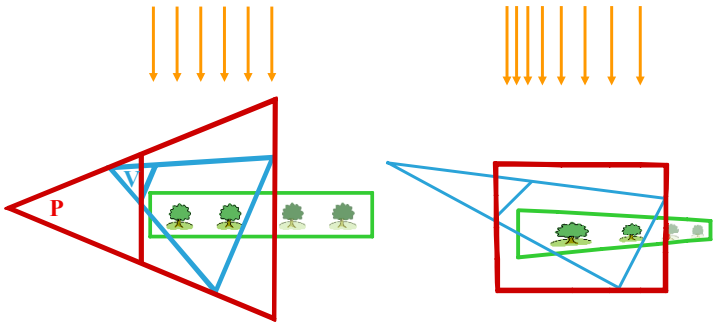


Figure 2.5: Light-Space Perspective Shadow Maps. Trees are illuminated by a directional light directly from above. *P* denotes the perspective transformation of the reparameterisation which is always orthogonal to the direction of light. Right: In post-perspective space, objects close to the near plane of *V* are magnified. Therefore, these objects are sampled finer than distant (image courtesy of [WSP04]).

As a consequence, the image plane of the transformation of LiSPSM is always parallel to the direction of light which prevents singularities in post-perspective space. Fitting is applied before determining the transformation to ensure that the frustum of the parameterisation always encloses the view frustum of the eye and all potential occluders symmetrically. The orientation of the transformation is determined by its three axes in light-space. The y-axis corresponds to the negative light vector l. As a consequence of the requirement to always align the transformation orthogonal to the direction of light, the z-axis is orthogonal to l and resides in the plane spanned by l and the view direction v. Finally, the x-axis is orthogonal to the yz-plane. If the light source is a point light, its transformation is applied first. As a result, the point light becomes directional in post-perspective space which allows to consistently apply the reparameterisation. In particular, the reparameterisation provides a free parameter to control the intensity of warping. The parameter corresponds to the distance between the centre of projection and the near plane of the LiSPSM transformation. Accordingly, a small distance results in applying the parameterisation of PSM (strong warping). On the contrary, a large distance is equivalent to standard shadow mapping (no warping). Therefore, warping is intuitively adjustable with respect to the spatial configuration of a scene. For the ideal case of PSM, where the directions of light and view are orthogonal, the optimal setting for the free parameter is $n_e + \sqrt{f_e n_e}$ where n_e and f_e are the near and far distances of the view transformation. The optimal setting keeps the error low and rising linearly but slightly while the depth of view increases. Therefore, changes in shadow quality are hardly appearing compared to PSM. If the angle between the directions of light and view changes, the free parameter has to be increased to compensate for extensive errors. In the limit, when the directions of light and view become parallel, the free parameter goes to infinity. In this situation, perspective warping is ineffective and standard shadow mapping is applied accordingly. Therefore, the free parameter should increase fast enough to diverge to infinity, otherwise the error exceeds the error of standard shadow mapping. Indeed, the proposed falloff function of LiSPSM is incapable of ensuring a sufficiently fast increase of the free parameter [LGQ*08]. However, this issue has profoundly been addressed by suggesting a more involved falloff function [LGQ*08]. The optimal setting of the free parameter evenly distributes the error between the two shadow map axes [LGQ*08]. In comparison, the error of PSM along the viewing direction becomes worse as the distance between objects and the near plane increases. In contrast, the error along the orthogonal shadow map axis remains constant.

Trapezoidal Shadow Maps (TSM) [MT04] is another attempt to moderate undersampling due to perspective aliasing. The proposed reparameterisation is based on a trapezoidal approximation of the view frustum of the eye in post-perspective space of the light. A trapezoid has the property of tightly approximating the view frustum of the eye in post-perspective space of the light. As a consequence of applying the transformation to trapezoidal space upon generation of the shadow map, the size of unused regions in the shadow map is minimised. Finding a suitable trapezoidal approximation is completely scene-independent because the two-dimensional convex hull of the view frustum of the eye in post-perspective space of the light needs to be computed only. The trapezoidal transformation is determined by the eight corners of the view frustum as well as the centres of the near and the far plane (Figure 2.6).

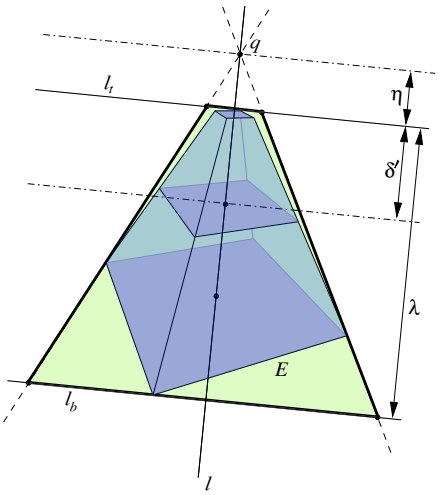


Figure 2.6: Trapezoidal Shadow Maps. Construction of the trapezoid (green) starts with the centre line l connecting the centres of the near and far plane of the view frustum (purple) in post-perspective space of the light. The top line l_t is orthogonal to l and touches the two-dimensional convex hull of the view frustum E close to the intersection of l and the near plane. The base line l_b is parallel to l_t and touches E close to the intersection of l and the far plane. Finally, the side lines touch E and intersect in the centre of the perspective transform q. The distance η between q and l_t determines the intensity of the warping. The proposed heuristic enables to setup η by intuitively adjusting the distance δ' which specifies the finely sampled region of the view frustum (image courtesy of [MT04]).

However, if the view frustum of the eye is not completely inside the view frustum of the light, the intersection of the two frusta needs to be considered instead. Additionally, this avoids situations in which vertices of the eye frustum are mapped to infinity due to a singularity of perspective transformation. Similar to LiSPSM, the reparameterisation allows to intuitively control the intensity of the trapezoidal warping. The proposed heuristic enables to dedicate the available shadow map resolution to a specified region along the viewing direction. The free parameter influences the slope and the position of the side edges of the trapezoid. Additionally, the reparameterisation addresses the continuity problem which causes shadows to flicker between consecutive frames if the viewer or the light moves. The disturbing artefacts arise from abrupt changes in size or orientation of the view frustum of the eye in shadow map space. In contrast, the approach to construct the trapezoidal transformation ensures that the projection of the view frustum of the eye gently changes between consecutive frames in trapezoidal space. This follows from aligning the top and base lines of the trapezoid orthogonal to the centre line of the view frustum in post-perspective space of the light. It has to be noted, however, that the continuity problem remains unsolved in several situations, especially if objects move into or out of the view frustum of the light. In particular, TSM provides significant improvements for large scenes with an overhead light (sun) when the viewer is close to the ground. In contrast, TSM is incapable of reducing perspective aliasing if objects cast stretched shadows towards the viewer (duelling frusta). In comparison to PSM, TSM generally exhibits the same error when considering both shadow map axes [LGQ*08]. According to fitting, TSM does not require to focus its reparameterisation because the approach to determine the reparameterisation implicitly ensures minimal wastage of shadow map space.

Logarithmic Perspective Shadow Maps (LogPSM) [LGQ*08] suggest a combination of a perspective projection and a logarithmic transformation to ensure a maximal reduction of undersampling due to perspective aliasing. The reparameterisation closely approaches the optimal distribution of error with respect to the accurate quantification of aliasing errors in shadow mapping (Equation 3.18). Other warping algorithms achieve less error in some regions of the view frustum but introduce higher error in other regions. The maximum error over all light directions of standard shadow mapping is $O((f/n)^2)$. A perspective reparameterisation (PSM, LiSPSM, TSM) reduces the error to $O(f/n)$. Indeed, a logarithmic parameterisation further reduces the error to $O(\log(f/n))$. For an overhead light, standard shadow mapping uniformly spaces samples along the view direction and the orthogonal direction of the view frustum of the eye. PSM improves the distribution of samples along the orthogonal direction at the expense of worsen sampling along the view direction. In contrast, LiSPSM improves sampling along the view direction at the expense of worsen sampling along the orthogonal direction. According to the simplified quantification (Equation 3.17), the optimal parameterisation keeps the error $\frac{dp}{ds}$ constant $= 1$ as the depth of view increases. Ideally, when the directions of light and view are orthogonal, the optimal parameterisation to make the perspective aliasing error $\frac{dz}{zds}$ constant is logarithmic [WSP04]

$$\frac{dz}{zds} = 1 \Leftrightarrow ds = \frac{dz}{z} \Rightarrow s = \int_0^s ds = \int_{z_n}^{z} \frac{dz}{z} = ln\frac{z}{z_n}. \tag{2.1}$$

Therefore, LogPSM combines a logarithmic transformation along the view direction and a perspective projection along the orthogonal direction to minimise the overall maximum error. The sampling rate decreases linearly along both the view direction and the orthogonal direction. This leads to a balanced distribution of the size of projected shadow map texels in both directions. As a consequence, errors between the two shadow map axes are distributed evenly. First, the perspective projection of LiSPSM is applied which affects both directions. Then, the logarithmic transformation is applied along the viewing direction only in post-perspective space of the perspective projection of LiSPSM. Therefore, the LogPSM parameterisation maps from the two-dimensional post-perspective space of LiSPSM to the two-dimensional space of the shadow map (Figure 3.12). Accordingly, the LogPSM parameterisation is defined by

$$\begin{bmatrix} s \\ t \end{bmatrix} = F_{lp}(u,v) = \begin{bmatrix} F_{p,s}(u,v) \\ c_0 \log(c_1 F_{p,t}(u,v) + 1) \end{bmatrix} \quad \text{with } c_0 = \frac{-1}{\log(f/n)} \text{ and } c_1 = \frac{n-f}{f}. \tag{2.2}$$

In equation 2.2,

- s,t are the shadow map coordinates
- u,v parameterise the light image plane
- $F_{p,s}$ is the perspective transform along the direction which is orthogonal to the view direction
- $F_{p,t}$ is the perspective transform along the view direction
- f,n are the far and near plane distances of the LiSPSM parameterisation F_p.

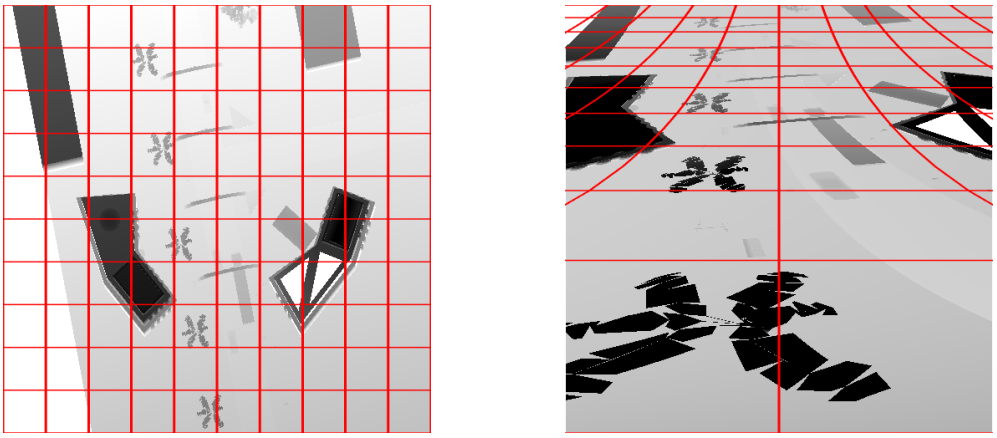


Figure 2.7: Comparison of standard and logarithmic perspective shadow maps. An alley of palm trees in a town is illuminated by a directional light directly from above. The eye is located at the bottom and looking along the alley to the top. Left: Standard shadow mapping uniformly spaces samples. Therefore, undersampling occurs close to the viewer (bottom). Right: As a consequence of the warping of LogPSM, the first palm tree of the alley (bottom) occupies a considerable amount of shadow map space. Therefore, nearby objects (bottom) receive more samples than distant. Note that due to logarithmic rasterisation planar primitives become curved (image courtesy of [LGQ*08]).

Analogous to LiSPSM, the proposed and more elaborated falloff function ensures to vary the free parameter of the perspective projection fast enough with the position of light. However, if the directions of view and light approach being parallel, the logarithmic transformation requires a different falloff function which diverges to infinity faster than the improved falloff function of LiSPSM. Otherwise, the error becomes worse than the error of uniform shadow mapping. Additionally, shearing artefacts are reduced if the magnitude of the free parameter is as large as possible. In the ideal case, when the view direction is orthogonal to the direction of light, the largest possible magnitude of the free parameter corresponds to the optimal parameter of LiSPSM $\sqrt{f_e n_e}$. However, it has to be noted that the logarithmic transformation depends upon logarithmic rasterisation (Figure 2.7). In order to support perspective mappings, current hardware hyperbolically interpolates inputs only. Of course, a GPU-enabled simulation of logarithmic rasterisation is possible but exhibits low performance. Hence, a logarithmic parameterisation is impractical, because hardwired logarithmic rasterisation would be required instead. Furthermore, logarithmic rasterisation necessitates a more general computation of polygon offset to reduce self-shadowing artefacts. On current hardware a constant polygon offset is applied to a single primitive. Concerning logarithmic rasterisation, polygon offset changes linearly in a definite direction over a primitive. Similar to all warping algorithms except TSM, LogPSM requires to apply fitting to ensure that the view frustum of the light encloses all objects relevant for shadow generation.

2.1.1.4 Global Partitioning

According to warping, the reparameterisation abruptly degenerates to standard shadow mapping if the directions of light and view become parallel. Therefore, warping algorithms are subject to show visually disturbing changes in shadow quality, despite the proposition of fast falloff functions. Global partitioning algorithms exhibit greater robustness in reducing perspective aliasing. Additionally, partitioning is capable of reducing perspective aliasing for cases in which warping is inapplicable. If the light comes from behind and the direction of view and light are parallel, warping is ineffective. Furthermore, global partitioning adds flexibility to reparameterisation because the algorithms are self-contained and combinable. Moreover, global partitioning is orthogonal to warping and fitting.

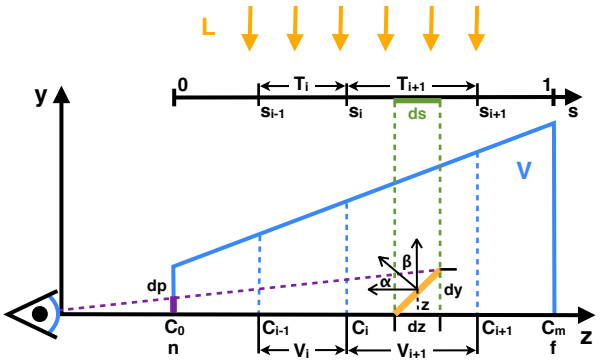


Figure 2.8: Parallel Split Shadow Maps partition the view frustum V into multiple sub-frusta V_i at position C_i. Each sub-frustum maintains a separate shadow map T_i to raise the sampling density along the viewing direction (image courtesy of [ZSXL06]).

Parallel Split Shadow Maps (PSSM) [ZSXL06] divide the view frustum into sub-frusta (z-partitioning) and maintain a single shadow map for each partition (Figure 2.8). As a result, objects in different partitions receive more samples which enables to reduce undersampling due to perspective aliasing. The distribution of the parallel split planes along the direction of view affects the distribution of aliasing error. An even distribution of error is achieved with a logarithmic parameterisation $s(z) \in [0,1]$ which theoretically ensures a constant perspective aliasing error $\rho = \frac{dz}{zds}$ throughout the view frustum (Equation 2.1)

$$s(z) = \frac{1}{\rho} \ln\left(\frac{z}{n}\right) = \frac{\ln(z/n)}{\ln(f/n)}. \tag{2.3}$$

The logarithmic split scheme discretely approximates the optimal logarithmic parameterisation. Accordingly, i split planes are distributed at the distances C_i^{log} to partition the view frustum logarithmically. The number of partitions m determines the accuracy of the approximation. The fraction of shadow map resolution allocated for each partition i is the same. Therefore, with $s(C_i^{log}) = i/m$ in equation 2.3, the projected texel area is distributed evenly between the near and far planes at the distances n and f, respectively.

$$s(C_i^{log}) = \frac{\ln(C_i^{log}/n)}{\ln(f/n)} \Leftrightarrow C_i^{log} = n\left(\frac{f}{n}\right)^{i/m} \tag{2.4}$$

$$\frac{C_{i+1}^{log}}{C_i^{log}} = \left(\frac{f}{n}\right)^{1/m} \tag{2.5}$$

As a consequence of distributing the split planes to equalise the ratio of near to far distance for each partition, the maximum error over all partitions is minimised [LGQ*08]. However, the optimal split scheme is often impractical. Close to the near plane the logarithmic split scheme provides most of the overall resolution which often remains unused. Therefore, a more practical split scheme combines the logarithmic split scheme and the uniform split scheme to reduce oversampling. Other z-partitioning algorithms such as *Cascaded Shadow Maps* (CSM) [Eng07] only differ from PSSM in how to distribute the split planes. CSM decrease the distance between the split planes along the direction of view. In particular, it has to be noted that in the duelling frusta case, PSSM is superior to a single large shadow map if fitting is applied to each partition individually. Of course, PSSM requires to render multiple shadow maps in several passes efficiently. Additionally, artefacts appear close to the adjacencies of split planes if the sampling density changes abruptly. [ZZB09] addresses several issues of PSSM such as effective split selection in the camera pass and reduction of flickering artefacts when the camera is moving or rotating. Additionally, advanced strategies are suggested for shadow map storage and filtering across splits is discussed. *Sample Distribution Shadow Maps* (SDSM) [LSL11] adaptively optimise the split scheme every frame in order to considerably improve upon the static distribution of split planes of PSSM. The location and size of a fixed number of splits is optimised with respect to the distribution of the shadow map samples that are required to render the current view. As a consequence, SDSM achieve predictable performance and constant memory usage while generating shadows of superior quality automatically compared to PSSM with hand-tuned splits. However, in order to apply frustum culling in light-space for large scenes, the CPU must wait for the GPU to finish generating the partition data. Furthermore, if the camera moves or rotates slowly temporal aliasing easily appears as a consequence of changing the shadow map projection for each frame.

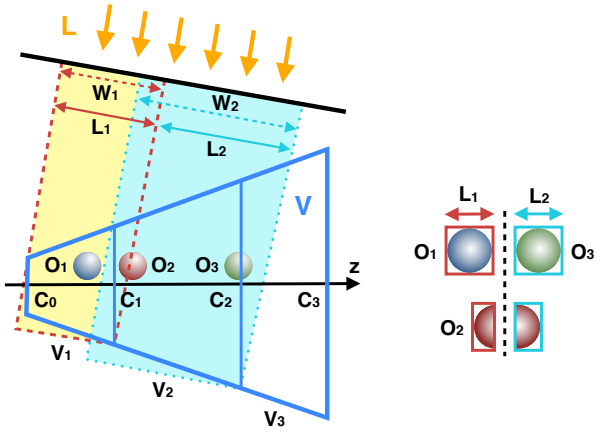


Figure 2.9: Light-Space Cascaded Shadow Maps. Intersecting view frustum splits in light-space. Left: According to CSM, if the direction of view and light are far from being orthogonal, two adjacent view frustum splits overlap in light-space. As a consequence of the intersection of W_1 and W_2, O_2 is rendered redundantly into two shadow maps. In contrast, LSCSM splits the scene based on light sub-frustums L_1 and L_2 which do not intersect. Right: O_2 is clipped against the split plane between L_1 and L_2. The resulting two parts of O_2 are rendered once into the different shadow maps T_1 and T_2, respectively, to avoid redundant rendering (image adapted from [LMCY11]).

Light-Space Cascaded Shadow Maps (LSCSM) [LMCY11] improve the performance of PSSM in more general configurations of light and view. If the directions of light and view are far from being orthogonal, repeated rendering of objects arises from intersecting view frustum splits in light-space (Figure 2.9). LSCSM circumvents redundant rendering by employing light-space scene splitting. The splitting in light-space ensures that the light frustum is partitioned into non-intersecting sub-volumes. Objects which reside in two adjacent sub-volumes are clipped against the separating split plane. The resulting sub-objects are rendered once into different shadow maps according to the unique partitions of the light frustum. Of course, clipping requires to efficiently determine which objects intersect split planes in light-space. Irregular frustum clipping creates multiple instances of objects and discards parts of clipped objects which were processed previously. Hence, the remaining parts are rendered into the currently processed shadow map only. However, objects which intersect several split planes remain to be rendered multiple times. In comparison to the suggested split selection of [ZZB09], LSCSM avoids costly conditional branches completely. However, the occlusion test always needs to access the shadow maps of all partitions. As a result, expensive conditional branching is relinquished in favour of increased texture bandwidth. Therefore, the applicability of texture filtering is limited considerably. Furthermore, shadow boundaries are susceptible to aliasing artefacts because the results of all depth tests are simply multiplied to determine occlusion without expensive conditional branching. According to shadow flickering artefacts, [ZZB09] suggests to use the minimal enclosing circle of a sub-frustum of the view of the camera to determine how to partition the view of the light. As a result, the projected area of a shadow map texel remains constant if the camera rotates between two consecutive frames. However, effective usage of shadow map resolution is decreased. Additionally, the probability of intersecting view frustum splits in light-space increases considerably which aggravates redundant rendering. Therefore, LSCSM proposes to use the minimum bounding sphere instead.

Frustum face partitioning [Koz04] assigns a separate shadow map to each face of the view frustum of the camera. This corresponds to wrapping a cube shadow map around the unit cube of the view in post-perspective space of the camera. However, more involved and, therefore, costly intersection and clipping routines are required to compute the face partitions robustly. Furthermore, frustum face partitioning is generally susceptible to shearing artefacts which need to be dealt with appropriately [LGQ*08]. In order to significantly increase the sampling density across frustum faces, z-partitioning can be applied to each cube map face additionally. A combination of frustum face partitioning and LogPSM further reduces perspective aliasing considerably [LGQ*08]. The top and bottom faces of the view frustum are parameterised uniformly. The side faces of the view frustum are parameterised logarithmically.

2.1.1.5 Adaptive Partitioning

Despite that global partitioning algorithms are fast, they are incapable of reducing undersampling due to projection aliasing. Adaptive algorithms analyse the scene to analytically identify surfaces being almost parallel to the direction of light and allocate more samples accordingly. Therefore, adaptive partitioning builds on a gradual refinement of the shadow map which is aggregated in a hierarchical data structure for improved access. The algorithms mainly differ at which level to stop the refinement. Of course, analysis and data structure management require additional resources. Furthermore, adaptive partitioning exhibits low performance because hardware acceleration is hardly utilisable.

Adaptive Shadow Maps (ASM) [FFBG01] are capable of robustly reducing both perspective and projection aliasing. The adaptive sampling progressively refines shadow boundaries only. Starting from a coarse shadow map, a hierarchical grid structure is iteratively updated to increase the sampling rate. In the limit, if the required resolution of the shadow map matches accordingly to the eye pixels of the camera image, shadows are completely free of aliasing errors. The hierarchical data structure is a quad-tree in which each node represents a shadow map of fixed size. Along shadow boundaries, the shadow map is subdivided into a fixed number of cells which contain further nodes. New nodes are added only if shadow quality is improved accordingly or a predefined memory limit is reached. The shadow quality is estimated by comparing the size of the projected area of an eye pixel and a shadow map texel. Of course, the cost of such an estimation is high. Therefore, *MIP Mapping* [Wil83] is used to approximate the projected area. However, the edge-finding algorithm potentially misses some shadow boundaries and remains expensive. Additionally, due to its iterative nature ASM require to efficiently manage the quad-tree without stalling the GPU. Therefore, ASM are impractical for real-time applications. In particular, alias-free shadows will only be generated if two conditions are satisfied. First, the edge-finding algorithm identifies all shadow map boundaries in the image of the current view of the camera. Second, the shadow map reaches the required resolution before the quad-tree exceeds its predefined maximum depth.

Tiled Shadow Maps [Arv04] adaptively increase the sampling rate for objects close to the camera and for shadow boundaries. First, a low resolution light view is analysed using a heuristic to subdivide the shadow map into equally-sized tiles. Each tile is assigned a weight which is proportional to the necessary sampling rate to avoid aliasing. The heuristic combines depth discontinuity, depth difference and surface distance to estimate a value for each texel of a tile. The values of the texels comprising a tile are added to yield the weight of a tile. Depth discontinuity identifies texels being part of shadow boundaries by applying an edge-detection filter to a low resolution light view. Depth difference weights discontinuity texels by taking the distance between occluder and receiver into account. Surface distance measures the distance between the eye and a surface point to increase the sampling rate close to the camera accordingly. Then, the shadow map is subdivided with respect to the tile weights using a recursive binary cut algorithm which ensures no wastage of available shadow map resolution. The subdivision scheme compares the weights of two adjacent tiles and splits the shadow map along the shadow map axes alternately to yield rectangular tiles of different size. Finally, each tile is rendered in a separate pass to generate the tile-based shadow map. However, the subdivision scheme causes aliasing if the height of a tile is considerably larger than the width. This can be circumvented if the binary cut algorithm balances the weights before comparing them. Furthermore, performance limitations mainly arise from the light view analysis and excessive multi-pass rendering of individual tiles. As a consequence of a read back to estimate the weights, the resolution for the light view analysis should be low in order to avoid further performance penalties. Moreover, a low resolution light analysis hinders edge detection to accurately identify depth discontinuities of complex geometry.

Resolution Matched Shadow Maps (RMSM) [LSO07] improve the efficiency of ASM by removing the costly edge-finding algorithm. Instead, the quad-tree hierarchically aggregates exactly those shadow map texels which are needed to perform the occlusion test for every eye pixel. Hence, the sampling rate of the shadow map is instantaneously adapted in accord with the resolution of the current camera view. The optimisation results from exploiting coherency between eye and shadow map image space. Neighbouring eye pixels of continuously visible surfaces map to contiguous regions in shadow map space. As a consequence, the acceleration data structure can be constructed instantaneously instead of iteratively. The adaptive shadow map maintains a mipmap hierarchy of shadow pages. Before generating the shadow map, page requests for several eye pixels can be coalesced due to coherency. First, the number of page requests is reduced with a GPU-enabled algorithm for connected-components analysis. The algorithm identifies continuously visible surfaces and eliminates redundant page requests due to coherency accordingly. Then, the resulting page requests are further refined by sieving out invalid page requests which were possibly missed in the previous step. The refined sets are sorted and compacted to create a small set of unique page requests. Finally, the unique page requests are transferred to the CPU which starts the shadow map generation pass. Of course, the optimisation is only applicable to scenes which exhibit coherency. For example, if the direction of light and view are far from being orthogonal or depths are concentrated locally. RMSM achieve higher performance than ASM while requiring significantly more memory. However, performance limitations arise from two major costs. First, sorting page requests for efficient compaction is costly for scenes with simple geometry. Originally, the sorting algorithm was implemented in OpenGL. Of course, an implementation in OpenCL will be more efficient today. Second, upon generating the shadow map the positive effect of the optimisation on performance is limited by coherency. Therefore, the number of unique shadow page requests depends on the complexity of shadow receivers.

Queried Virtual Shadow Maps (QVSM) [GW07b] simply partition a large shadow map into smaller equally-sized quadratic tiles to increase the effective resolution virtually. Each tile represents a self-contained shadow map which is queried during the occlusion test. Of course, brute-force generation of shadow maps for a huge amount of tiles infers excessive multi-pass rendering. Therefore, QVSM employ deferred shadowing. First, the scene is rendered once to capture lighting and eye-space depth values in offscreen buffers for repeated use. Then, for each tile the unprojection of fragments from screen-space to world-space enables to perform the occlusion test with the previously stored depth values efficiently. Finally, the according colour value is looked up from the lighting offscreen buffer and attenuated with respect to the result of the occlusion test. In this simplified form QVSM is easy to implement. However, adaptive methods refine the shadow map locally where aliasing occurs only. Therefore, QVSM stops repeated splitting of tiles if further refinement fails to improve shadow quality. Between subsequent refinements, improvements in shadow quality are measured by detecting how many texels have changed in a "shadow result texture" which contains the result of the occlusion test for every screen pixel. In order to avoid costly read-backs of the "shadow result texture" from the GPU QVSM diverts occlusion query from its intended use. The intent of occlusion query is to determine the number of fragments which passed the depth test in order to accelerate approximate visibility tests in image space. Accordingly, the depth test criterion is modified to let pass exactly those number of fragments which correspond to the number of texels which have changed in the "shadow result texture" between two subsequent refinements. As a consequence, QVSM completely avoids costly read-backs from the GPU to determine when to stop the refinement process. Furthermore, the number of fragments returned by the occlusion query can be used as a threshold. This allows to tune the refinement according to a quality/performance tradeoff being required for a particular application. In addition, it has to be noted that QVSM does not require a GPU-enabled hierarchical data structure because splitting is organised entirely by the CPU. After partitioning the shadow map with respect to the refinement measures, the CPU initiates shadow map generation for all tiles using deferred shadowing. Several optimisations are suggested for splitting tiles non-uniformly and improving refinement measures. However, results showed that applying these optimisations is too expensive and, therefore, the achievable benefit is easily undone.

Fitted Virtual Shadow Maps (FVSM) [GW07a] perform a separate analysis pass to determine analytically in advance which regions of the shadow map will be queried and how much resolution will be needed at each query location. In contrast, QVSM efficiently measure differences in shadow quality between two refinements and stop the refinement once a predefined threshold is reached. Compared to ASM the results of the analysis allow to derive the optimal structure of the quad-tree on the CPU to shadow the scene with sub-pixel accuracy avoiding both perspective and projection aliasing. With deferred shadowing the position of a query in the shadow map is calculated by, first, unprojecting a fragment from screen-space to eye-space. Then, the position of the fragment in eye-space is transformed to light-space to locate its position in the shadow map. The required resolution at the query location is calculated by first projecting the approximated area in eye-space of a screen pixel into shadow map space. Then, the resulting area is enclosed with an axis-aligned bounding box with respect to the two shadow map axes. The half length of each side of the box serves as the measure for the required resolution along the corresponding shadow map axis. Query position and required resolution are packed into the "Shadow Map Tile Mapping Map" (SMTMM) which inherently has the same resolution as the frame buffer (Figure 2.10).

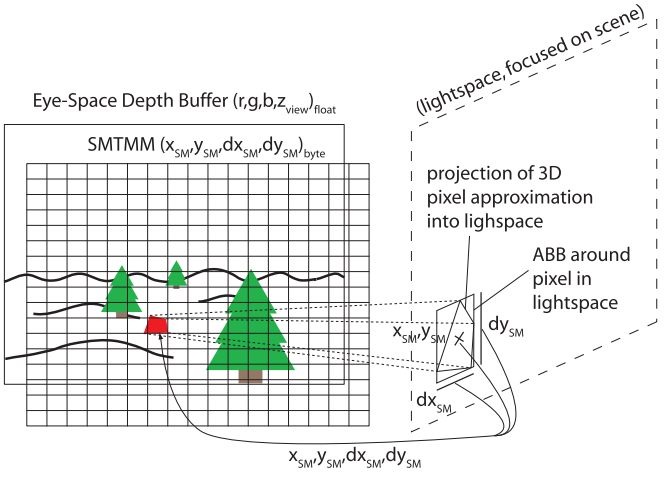


Figure 2.10: Fitted Virtual Shadow Maps. The Shadow Map Tile Mapping Map stores the query location and the required resolution needed to perform the occlusion test with sub-pixel accuracy. The query location (x_{SM}, y_{SM}) is determined by deriving the position of the fragment in eye-space from the Eye-Space Depth Buffer by unprojecting and transforming the eye-space position into light-space subsequently. The sides of the axis-aligned bounding box dx_{SM}, dx_{SM} provide a measure for the required resolution. The bounding box is calculated by projecting the approximated area a screen pixel covers in eye-space into light-space (image courtesy of [GW07a]).

Of course, to read back the SMTMM to the CPU is costly. Therefore, only a coarse analysis is performed to compensate for significant losses of performance. Once the SMTMM has been transferred to the CPU, a grid structure representing the virtual shadow suggested by QVSM is derived from the SMTMM by utilising the random memory access ability of the CPU. Then, a kd-tree is created to hierarchically organise the shadow map tiles to support efficient shadow generation while achieving the required quality. Upon traversing the kd-tree, tiles are symmetrically split if their resolution along a shadow map axis is below the required resolution. If a tile already matches the required resolution, the according shadow map is rendered with deferred shadowing immediately and the result of the occlusion test is stored in the shadow result texture. Once the kd-tree has been traversed, the shadow result texture is applied to the colour buffer of the lighting pass to combine the shadow term and lighting similar to QVSM. In contrast to RMSM which refine the shadow map on the GPU, FVSM determine the required partition of the shadow map from the SMTMM on the CPU. Similar to QVSM, FVSM provide a parameter to smoothly trade the overall shadow quality for performance by sacrificing shadow map tiles. However, results show that the complexity of FVSM hinders its usage with several light sources.

2.1.1.6 Temporal Reprojection

Aliasing due to perspective aliasing and projection aliasing can be reduced by increasing the sampling rate using samples which are reused from previous frames with *Temporal reprojection* [SJW07]. The approach jitters the viewport of the shadow map per frame and accumulates the occlusion test results of previous frames in a history buffer. Upon rendering the view of the camera, the current shadow test result is combined with previous results which are queried from the history buffer using reprojection. The combination is weighted according to more recent frames with exponential falloff in order to let the result quickly converge to the current frame. As a consequence, temporal aliasing which manifests in shadow flickering can be eliminated by increasing the number of accumulated frames sufficiently. Additionally, to improve the accuracy of shadow edges, the history buffer is updated based on a confidence criterion which measures the correctness of the shadow test according to the distance between the projected fragment and the closest shadow map sample:

$$conf_{x,y} = 1 - \max\Big(|x - center_x|, |y - center_y|\Big) * 2 \tag{2.6}$$

where $conf_{x,y}$ is the confidence at the projected fragment position (x,y) and $(center_x, center_y)$ is the centre of the nearest shadow map texel. Accordingly, confidence is high if a fragment is projected close to the centre of a shadow map texel. The probability to yield a more accurate result from the occlusion test increases. As a consequence of adjusting the weights upon updating the history buffer with respect to the confidence, the iteration will converge to pixel perfect shadows over time, if the rasterisation of the shadow map of the current frame differs from each consecutive frame. Therefore, the viewport of the shadow map is jittered per frame in order to increase the probability of yielding occlusion test results with high confidence. Accordingly, the weights are chosen to increase the influence of results with high confidence over results with low confidence. In particular, it has to be noted that if reprojection is combined with warping the iteration will converge faster to the exact shadow solution. However, the applicability of approaches based on reprojection is considerably limited to mildly dynamic scenes. Several frames will never converge to a satisfying result if objects or light sources move continuously and quickly. Additionally, shadow boundaries will be incorrectly blurred by temporal noise which can only be reduced by balancing how quickly the history buffer adapts with respect to the speed of a moving camera.

2.1.2 Oversampling

Oversampling occurs if the sampling rate used to sample the depth into the shadow map is higher than the sampling rate at which the shadow map is sampled during shadow generation. Utilising filtering to reduce oversampling artefacts is impossible due to the discrete nature of the shadow map. Instead, *Percentage-Closer Filtering* (PCF) [RSC87] applies filtering after performing the depth tests. With sufficiently large kernels, PCF is capable of reducing oversampling artefacts but exhibits poor performance due to sampling the shadow map excessively. *Deep Shadow Maps* (DSM) [LV00] store a distribution of depths encoded in piece-wise linear functions per texel which allow pre-filtering. As a consequence of requiring variable storage per-texel, a GPU-enabled implementation of DSM is non-trivial. In addition, DSM are inapplicable to texture filtering and antialiasing hardware units on the GPU. However, several linear representations for depth distributions have been suggested to allow GPU-enabled pre-filtering of shadow maps. In addition, these techniques further increase temporal coherence which results in less shadow flickering due to temporal aliasing.

Variance Shadow Maps (VSM) [DL06] efficiently estimate the result of PCF over a region with a single depth sample only. The approach approximates a distribution of depths with the first and second moments: the mean depth and the mean squared depth. The moments are stored in the shadow map and can be interpolated to make the shadow map applicable to filtering. From the moments M_1 and M_2, the mean μ and variance σ of the distribution of depths over a filter region are defined by

$$M_1 = E(x) = \int_{-\infty}^{\infty} x\,p(x)\,dx \qquad M_2 = E(x^2) = \int_{-\infty}^{\infty} x^2\,p(x)\,dx \tag{2.7}$$

$$\mu = E(x) = M_1 \qquad \sigma = E(x^2) - E(x)^2 = M_2 - M_1^2 \tag{2.8}$$

where $E(x)$ denotes the expected value of a continuous random variable x with probability density function $p(x)$. The variance σ expresses the width of a distribution. The inequality of CHEBYCHEV supplies a bound for the proportion of a distribution that is probably far away from the mean of the distribution. Therefore, VSM applies the one-sided inequality of CHEBYCHEV to estimate an upper bound on the probability that the surface at depth d is occluded. Accordingly, if x is a random variable drawn from a distribution of depths with mean μ and variance σ, then for $d > \mu$

$$P(x \geq d) \leq p_{max}(d) \equiv \frac{\sigma^2}{\sigma^2 + (d-\mu)^2} \tag{2.9}$$

In equation 2.9, $P(x \geq d)$ represents the fraction of pixels over a filter region that probably fail the depth test for a given depth d. In the case of a single planar occluder and a single planar receiver, the inequality of CHEBYCHEV is an equality. Therefore, the approximation yields the exact result of PCF. Upon generating the variance shadow map, first, the depth and the square of the depth are rendered into a two-channel depth texture. Then, the shadow map is optionally pre-filtered. In the context of oversampling this corresponds to band-limiting the signal before resampling. Finally, mipmaps are generated. As a result, the variance shadow map stores the two moments M_1 and M_2 in a linearised depth texture. Upon rendering the view of the camera the depth test is performed with $\mu = M_1$. If the current fragment is occluded ($d < \mu$), first, σ is computed from the two moments with $\sigma = M_2 - M_1^2$. Then, p_{max} is calculated from the inequality of CHEBYCHEV (Equation 2.9). Finally, p_{max} is used to scale the light intensity. In particular, it has to be noted that, if several occluders are intersected consecutively along a ray of light, light bleeding artefacts are introduced easily (Figure 2.11). As the depth complexity increases the variance over the filter region increases accordingly. This is a direct consequence of the inequality of CHEBYCHEV which provides only an upper bound on the result of PCF using a single depth sample per pixel. Therefore, the exact result could yield darker shadows. Increasing the number of samples reduces light bleeding but intrinsically reverts VSM to PCF in the limit.

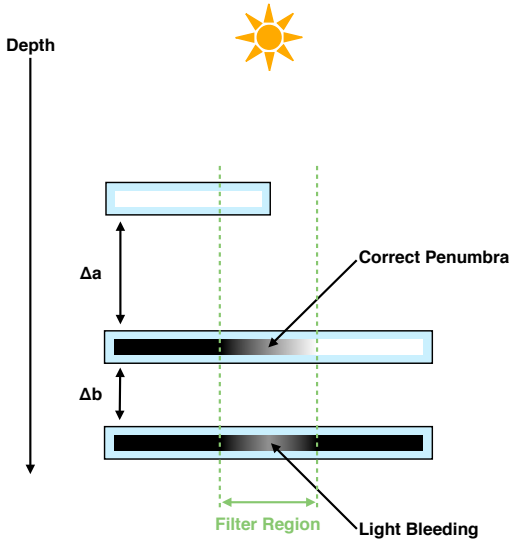


Figure 2.11: Variance shadow mapping inherently suffers from light bleeding artefacts, if variance is high over the filter region. The light shines from above onto three planar objects. The second object is partly occluded by the first object. The third object is fully occluded by the second object. Over the filter region, the penumbra on the second object incorrectly shows through onto the third object. In this setup, if the ratio $\Delta a/\Delta b$ increases, light bleeding will be aggravated accordingly (image courtesy of [Lau07]).

A simple solution to lessen light bleeding supplies a tuneable parameter to define a threshold below which all intensities are directly mapped to black [Lau07]. Of course, the softness of shadows decreases accordingly and shadows considerably loose detail for high settings. Moreover, in order to avoid numeric stability issues which introduce disturbing artefacts, the depth texture should have a 32 bit floating-point format which is applicable to both GPU-enabled mipmapping and anisotropic filtering. In comparison to standard shadow maps, VSM are inexpensive because mean and variance are obtained efficiently from a linearised depth texture which infers little memory overhead. Furthermore, efficiency is raised further if a separable filter is used in the optional pre-filtering step. In conclusion, VSM are considerably superior in performance to PCF for large filter kernels.

Layered Variance Shadow Maps (LVSM) [LM08] address the main disadvantage of VSM, light bleeding, by partitioning the depth range into multiple layers to efficiently apply a warping function to depth values. As a result, the accuracy of the approximation to estimate the result of PCF is improved considerably. If depths strongly differ over the filter region, variance is high. Therefore, the estimation yields incorrect results causing light bleeding. The approach is based on the observation, that the result of PCF is independent of the magnitude of the current depth value. It suffices to be able to determine if the current depth is smaller or larger. Therefore, a warping function can be applied to concentrate depth values in a small interval for increasing the accuracy of the approximation. Accordingly, the scene is split into multiple depth layers L_i covering the interval $[d_i, d_{i+1}]$ and the warping function ϕ_i is defined with

$$\phi_i(x) = \begin{cases} 0 & \text{if } x \leq d_i \\ (x - d_i)/(d_{i+1} - d_i) & \text{if } d_i < x < d_{i+1} \\ 1 & \text{if } d_{i+1} \leq x \end{cases} \tag{2.10}$$

As a result, the upper bound estimated with the inequality of CHEBYCHEV (Equation 2.9) becomes tighter, which results in darker shadows. Of course, the splitting scheme influences the reduction of light bleeding significantly. Several splits should be introduced near to surfaces suffering from light bleeding. In general, a considerable number of layers is needed to eliminate light bleeding completely. However, LVSM are very efficient for two reasons. First, the depth test needs to be performed only once for a given depth according to the interval covered by a single layer. Second, in comparison to VSM, the depth texture format can be of low precision because LVSM inherently provides high numerical precision. Therefore, high memory consumption caused by many layers can be reduced significantly. For best numeric precision over the entire depth range splits should be distributed uniformly.

Convolution Shadow Maps (CSM) [AMB*07] reformulate the shadow test as a weighted summation of basis terms in order to make the shadow map applicable to pre-filtering and hardware-accelerated anti-aliasing. Each texel stores a linear representation of a binary visibility function which is approximated with a basis function expansion using FOURIER series (Figure 2.12). The shadow test can be reformulated to linearise the shadow map by transforming z-values such that the terms are factored into independent functions:

$$f(d,z) \approx \sum_{i=1}^{M} a_i(d) B_i(z) \tag{2.11}$$

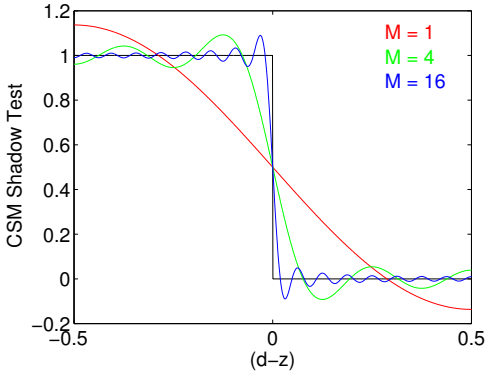


Figure 2.12: Convolution Shadow Maps approximate the shadow test using FOURIER expansion. The approximation becomes more accurate by increasing the number of FOURIER coefficients which reduces light bleeding. However, a large number of FOURIER coefficients introduce high frequencies which manifest in ringing artefacts. Therefore, a conflict arises upon trading a reduction in light bleeding for an increase of ringing artefacts (image courtesy of [AMB*07]).

In equation 2.11, B_i are the basis functions with respect to z. Each basis is weighted with coefficients a_i depending on the distance d to the light source. According to the FOURIER expansion order M, the basis functions are used to convert the depth map into a CSM comprising M basis images. As a result, the shadow function can be filtered by convolving the individual basis images. Additionally, the basis images can be convolved before performing the shadow test which allows for pre-filtering using separable filter kernels. After pre-filtering mipmapping can be applied to prevent aliasing due to oversampling. Upon rendering the view of the camera for each screen pixel a weighted sum of the filtered basis functions is evaluated to obtain a value for scaling the intensity of light. However, FOURIER representation introduces artefacts for two main reasons. First, every FOURIER representation suffers from ringing (GIBB's phenomenon), particularly when the expansion is truncated to a small number of terms (Figure 2.12). Ringing is suppressed by attenuating each term with an exponential. Second, FOURIER expansion smoothes the step function of reconstruction which results in incorrectly shadowing fully lit regions at contact shadows. As a consequence of the shift-invariance property of the FOURIER representation, a constant offset can be applied before reconstruction in order to avoid incorrect shadowing. In comparison to VSM, CSM exhibit less light bleeding artefacts and are free from artefacts introduced by high depth complexity. However, a high number of render passes is required to generate the basis textures because only a high FOURIER expansion order infers a reliable shadow test. As a consequence, the basis textures require a considerable amount of memory despite that the terms of the basis functions can be packed into four-channel colour textures with 8 bit precision. Moreover, a high number of basis textures increases the cost of pre-filtering.

Exponential Shadow Maps (ESM) [AMS*08] approximate the shadow test by using an exponential function. This allows to filter the shadow map for preventing aliasing due to oversampling with GPU-enabled filtering. In contrast to CSM, ESM use a single-term approximation which is based on the assumption that the domain of the shadow test is always positive. A texel in a shadow map always stores the depth z of the surface which is hit first along a ray of light passing through the texel. Therefore, in theory $z \leq d$ for any given distance d from the light source. The shadow function $f(d,z)$ can be linearised by factoring the terms into two functions, where the first depends on d and the second on z only. Therefore, if $z \leq d$ holds:

$$f(d,z) \approx e^{-c(d-z)} = e^{-cd}e^{cz} \tag{2.12}$$

In equation 2.12, c is a positive constant which influences the accuracy of the approximation (Figure 2.13). If c is too small, light bleeding artefacts are introduced. The upper bound of c is determined by the numerical precision of the number format of the depth texture (e.g. $c = 80$ is optimal for 32 bit floating-point numbers). However, the validity of the assumption $z \leq d$ is violated in two particular cases. First, limited numerical and spatial resolution cause imprecision which infers $f(d,z) > 1$ for unshadowed regions. This is circumvented by simply clamping the exponential to 1. Second, across slanted surfaces or at shadow boundaries assuming $z \leq d$ is uncertain as a consequence of pre-filtering. In particular, to reduce oversampling the filter kernel can be very large. This increases the probability that the z-values under the support of the filter dissatisfy $z \leq d$. Therefore, once a screen pixel has been determined to violate $z \leq d$, the ESM results of the four nearest neighbours are bilinearly interpolated. Hence, the approach essentially falls back to 2x2 PCF.

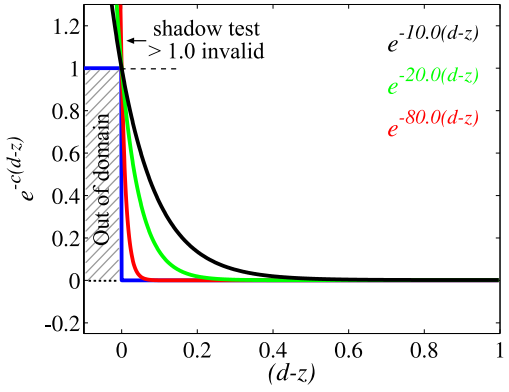


Figure 2.13: Exponential Shadow Maps approximate the shadow test using an exponential based on the assumption that the domain of the shadow test is always positive $[(d-z) \geq 0]$. As a consequence of increasing the magnitude of c, the falloff becomes steeper which makes the approximation more accurate. Note that the approximation yields invalid shadow test results if the assumption $z \leq d$ is violated because the new expansion increases exponentially above 1.0 (image courtesy of [AMS*08]).

Two approaches are suggested to determine if the assumption $z \leq d$ holds for a given screen pixel. First, *Z-Max Classification* uses an additional texture to estimate conservatively if the assumption holds. This texture stores the maximum z-values in a given neighbourhood. Second, *Threshold Classification* simply evaluates ESM and checks if the result of the shadow test is close to 1.0 according to a maximum distance defined by a specific threshold. While Z-Max Classification requires an additional texture which needs pre-processing, Threshold Classification is faster but prone to misclassification errors which introduce artefacts. In comparison to CSM, ESM require considerably less memory while achieving better results particularly at contact shadows. For example, ESM with a $c = 80$ is superior to CSM with a FOURIER expansion order $M = 16$ which requires 16 textures storing the basis images. Accordingly, the cost of pre-filtering is decreased with a low memory consumption. Furthermore, light bleeding is significantly reduced with ESM while achieving better performance compared to VSM and CSM. The exponential warping function of ESM and the estimation of VSM using the inequality of CHEBYCHEV can be combined to yield *Exponential Variance Shadow Maps* (EVSM) [LM08]. As a result, light bleeding is reduced considerably while decreasing artefacts for non-planar or multiple receivers if c is large.

2.1.3 Alias-Free Sampling

Shadow map aliasing occurs due to limited resolution which results in a sampling mismatch between a regular grid of screen pixels and another regular grid of shadow map texels. According to sampling theory, a shadow map with infinite resolution would be required to eliminate aliasing. However, a shadow map with infinite resolution is not required. Alias-free shadows can be generated for a camera image of fixed resolution if each screen pixel injectively maps to a shadow map texel. Irregular sampling ensures to provide a unique sample location for each shadow map query which enables to generate pixel-exact shadows (Figure 2.14). First, the pixel centres are projected into light-space to locate the necessary sample points in the shadow map. Then, these sample points are used to generate the shadow map. As a result, shadows remain exact upon zooming in because shadow detail is decoupled from resolution. As a consequence of distributing samples irregularly in the shadow map, low coherency requires efficient rasterisation. Accordingly, severe performance penalties are avoided with spatial acceleration data structures. In addition to the following algorithms, adaptive partitioning (Page 16) and temporal reprojection (Page 18) are capable of generating exact shadows in the limit.

Alias-free shadow maps [AL04], first, compute a depth buffer from the view of the camera to yield the visible samples. Second, the samples are transformed into light-space to obtain both the location of the sample points in the shadow map and the light-space depth values. Finally, upon rendering the view of the light source, if a sample point is covered by a geometric primitive, the depth value of the primitive is calculated at the sample location and compared to the light-space depth value of the sampling point. If the occlusion test is positive, the shadow term is used to mark the according screen-space pixel as shadowed which eliminates further shadow map queries. However, the irregular distribution of the sample points in light-space infers a considerable inefficiency. The determination of which sampling points are covered by a geometric primitive requires to test all sampling points for every primitive. In contrast, with standard rasterisation it suffices to test the edges of the primitive against a regular grid of sampling points which is a constant-time operation. Therefore, the sampling points are organised into an axis-aligned 2D BSP tree. The initial set of sampling points is alternatingly divided in half after each split with respect to the two axes of the image plane of the light source. The subdivision stops once the initial number of sampling points is reduced to a predetermined number.

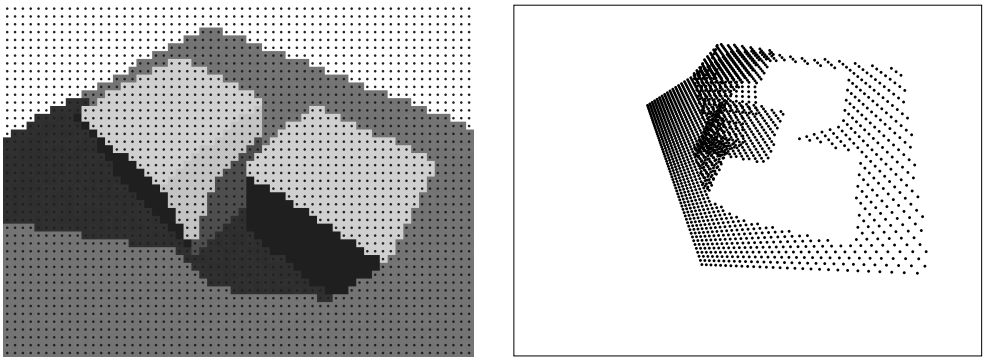


Figure 2.14: Alias-free Shadow Maps. Alias-free sampling obtains the query location in the shadow map for each screen-space pixel. The shadow map only needs to be sampled for these query locations in order to generate pixel-exact shadows. Right: Superimposed on the view of the camera are the pixel centres. Left: The pixel centres correspond to the sampling points when the scene is rasterised from the viewpoint of the light source. Accordingly, regions hidden in the view of the camera do not receive any sampling points. Note that the samples concentrate irregularly towards the eye. The sampling rate is varied with respect to visually important regions to eliminate aliasing (image courtesy of [AL04]).

During rasterisation the BSP tree is traversed from top to bottom according to the partial coverage of a geometric primitive until the sampling points of all visited leaf nodes have been tested. Additionally, hierarchical rasterisation is combined with occlusion culling. This prevents already occluded sampling points from being tested redundantly. As a consequence, irregular rasterisation becomes logarithmic in time complexity. However, the implementation is software-based and, therefore, impractical for real-time rendering.

Irregular Z-Buffering proposes a set of hardwired extensions to the standard z-buffer hardware in order to accelerate irregular rasterisation significantly [JMB04, JLBM05]. In essence, the irregular z-buffer is a standard z-buffer which stores a list of reprojected view samples per element. According to shadow generation, the shadow test is performed by, first, finding the elements which are covered by input primitives projected from the view of the light source. As a consequence of aligning the elements in a regular grid, *point-in-area testing* reduces the cost of coverage determination considerably. Then, the occlusion test is performed for each rasterised fragment of an input primitive. The depth of the view samples in the list of the covered elements is compared to the depth of the fragment. If a view sample is occluded it is marked to be in shadow. Finally, the scene is rendered from the view of the camera and fragments are shadowed accordingly if the corresponding sample in the list of view samples is marked to be occluded. However, an implementation of the irregular z-buffer is still missing on contemporary hardware platforms because the suggested extensions would infer a costly hardware design.

Without requiring any hardware changes, irregular sampling for shadow mapping was mapped to graphics hardware by using depth peeling [Arv07] and by using the NVIDIA CUDA framework [SEA08]. In the latter implementation, the construction of the list of reprojected view samples benefits from the scattering capabilities of CUDA. Furthermore, storage of the list exhibits constant memory consumption. As a consequence of representing the shadow state of a view sample with a bit-mask, the shadow state can be updated for each processed triangle with GPU-enabled bitwise blending. Shader Model 4 floating-point capabilities enable to update the shadow state of 1024 view samples per list. Hence, additional render passes are required if a list contains more samples. Furthermore, coverage determination is ameliorated with a GPU-enabled conservative rasterisation. This approach includes the view samples of light-space pixels which are only partly covered by the projection of a geometric primitive. Otherwise, no fragments would be generated for partly covered pixels whose centre is outside the projected area. In particular, a performance comparison reveals that GPU-enabled irregular shadow mapping is only three times slower than standard shadow mapping (8192×8192 shadow map, 512×512 viewport).

2.1.4 Incorrect Self-Shadowing

Incorrect self-shadowing is caused by two reasons. First, depth values have limited numerical precision. Second, upon projecting shadow map texels into the scene adjacent surface points at different depths on curved or steep planar surfaces, with respect to the direction of light, are compared to the same depth value. The first issue is addressed with *constant-biasing* [Wil78, RSC87]. The latter issue is addressed with *slope-based biasing* [Kil01].

According to PCF, a large filter kernel requires a large magnitude of depth bias. As a consequence, objectionable gaps are introduced at contact shadows. Several capable approaches were suggested to mitigate these artefacts. Instead of taking PCF samples in a plane, samples can be taken in a cone shaped region where the tip of the cone resides on the centre of the PCF filter kernel [VdB04, Bur06]. Another option is to apply a gradient-based bias along a tangent plane estimated from the PCF samples using GPU-enabled partial derivative operators [Sch05, Isi06, Sch07].

A simple and efficient biasing approach displaces the position of a surface along the surface normal [Hol11]. As a consequence, surfaces are moved out of regions where incorrect self-shadowing occurs. According to steep planar surfaces, with respect to the direction of light, the magnitude of bias necessary to eliminate incorrect self-shadowing is low compared to slope-based biasing. Accordingly, the size of shadow gaps is reduced significantly. However, displaced surfaces are possibly disconnected. Therefore, discontinuity artefacts are potentially introduced along shadow boundaries. Concerning soft shadows, the artefacts are effectively hidden in penumbrae and, therefore, are hardly objectionable. Of course, surface displacement may cause shadow boundaries to be slightly moved.

Biasing issues of VSM are resolved elegantly by parameterically approximating a locally planar distribution of depth to estimate the second moment more accurately [Lau07]. Furthermore, clamping the variance to a sufficiently small value and using a 32 bit floating-point number format for the variance shadow map increases numerical stability which improves biasing issues. In general, approaches which allow to filter the shadow map such as VSM, CSM, ESM, LVSM and EVSM inherently exhibit less incorrect self-shadowing because the outcome of PCF is estimated with a single depth sample. In particular, LVSM and EVSM weaken the requirement for a high precision number format while improving numerical stability issues.

Fitting reduces the magnitude of bias necessary to eliminate incorrect self-shadowing. If fitting yields a sufficiently smaller distance between the near and far planes of the view frustum of the light depth precision increases significantly. However, fitting generally aggravates temporal aliasing.

According to spot lights, point lights and warping techniques, another source of incorrect self-shadowing originates from perspective division. Depth samples are distributed non-uniformly along the direction of light. For point and spot lights this is avoidable by utilising a linear depth metric [Hei99, BAS02a]. According to TSM, occlusion testing simply compares world-space depth values because the perspective transformation is not applied to the z-coordinate [MT04]. According to PSM, slope-based depth biasing is applied, first, in world-space, then, the resulting world-space bias is transformed into post-perspective space and, finally, scaled with respect to the texel size in world-space [Koz04]. In contrast, slope-based biasing is applicable to LiSPSM directly. In general, it is preferable to use a linear depth metric because the additional cost is negligible with contemporary hardware.

Midpoint Shadow Maps [Woo92] store depth values of intermediate surfaces. The depth values are obtained by averaging the two depths of the nearest and second-nearest depth layers along a ray of light with respect to the position of the light source. The method requires two passes because the depths of the second-nearest surfaces are computed using depth peeling [Eve02]. Therefore, the depths of the nearest surfaces need to be acquired in advance. As a consequence of discrete midpoint depth values, incorrect self-shadowing problems persist if the depths of the nearest and second-nearest depth layers differ insufficiently (low depth disparity).

Second-Depth Shadow Maps [WM94] simply store the depth of the second-nearest surfaces. These surfaces remain after peeling off all front-facing surfaces with respect to the direction of light. As a consequence, the robustness of the occlusion test increases because the depth of surfaces is biased adaptively depending on the thickness of occluders. Of course, incorrect self-shadowing potentially appears on back-facing surfaces analogously which causes light bleeding. This is avoided by shading back-facing surfaces to be in shadow always. However, imprecisions remain where occluders are sufficiently thin or thick. Accordingly, imprecision caused by very thick occluders is avoided by limiting the magnitude of bias with a predefined threshold [WE03]. However, depth biasing needs to be reintroduced at occluder silhouettes where imprecisions remain because depth values differ insufficiently (low depth disparity).

A robust real-time solution which automatically resolves incorrect self-shadowing is still unavailable. All depth biasing approaches fail if the disparity of depth between samples is low. This is particularly true if surfaces exhibit concavities or high curvature in general. Furthermore, the magnitude of bias necessary to eliminate incorrect self-shadowing needs to be increased if the shadow map is magnified due to projection aliasing and perspective aliasing. Moreover, incorrect self-shadowing appears with adaptive partitioning and irregular sampling.

The need for depth biasing can be alleviated if each polygon is identified with a unique id [HN85]. Hence, visibility testing compares exact indices instead of depth values discretised defectively. However, the idea is impractical to be mapped to the GPU. If several triangles are projected to a single pixel maintaining several indices per pixel is non-trivial.

2.1.5 Omnidirectional Shadows

Shadow mapping is inherently limited to generating shadows cast by spot and directional light sources. These light sources require a single depth image only to cover their view. In contrast, a point light source emanates light equally in all directions. As a consequence, which a point light source has a spherical view. Accordingly, six depth images are acquired by rasterising six view frustums into a cube shadow map to cover the spherical view [Ger04]. Each frustum has a field of view of 90 degrees. Since hardware capable of Shader Model 4, a geometry shader allows to generate a cube shadow map in a single pass. First, the geometry is duplicated for six layers. Then, each layer is rasterised to the depth texture of the according cube map face using multiple render targets. However, generating a cube shadow map infers rasterising geometry repeatedly and redundantly because generally decomposing geometry into six disjoint subsets with respect to the cube map faces is non-trivial. Furthermore, artefacts possibly become objectionable along lines of junction formed by joining individual depth images to cover the spherical view.

Several non-trivial solutions have been proposed to reduce the number of depth images necessary to cover a spherical view adequately and efficiently. *Dual-Paraboloid Shadow Mapping* is capable of generating a shadow map for a point light source in two passes only [BAS02b]. The spherical view is decomposed into two hemispheres being parameterisable in 2D. Therefore, the shadow map lookup can be performed regularly. However, the technique is particularly impractical for two reasons. First, the sampling rate possibly varies by a factor of four over a hemisphere only. Second, implementing paraboloid mapping in a standard graphics pipeline is non-trivial. As a consequence of lines becoming curved, artefacts are introduced during rasterisation if the scene is not tessellated finely enough. Despite the proposal of a GPU-enabled approach, efficient rasterisation of non-linear projections remains expensive [GHFP08].

Another environmental mapping technique suggests to use a tetrahedron [Lia10]. Accordingly, only four passes are required to cover an omnidirectional view for shadow mapping. Upon generating the tetrahedron shadow map an equilaterally triangular region of the common rectangular view is captured four times using matrix transformations accordingly. However, a tetrahedron map wastes a considerable amount of memory. Until being supported natively by upcoming hardware a tetrahedron map must be simulated with a square texture. The depth images of four equilateral triangles are stored in the square texture while a considerable amount of texels remains uncovered.

2.1.6 Summary

Shadow mapping is a fast technique to generate hard shadows including self-shadowing in real-time because the algorithm can be implemented efficiently on contemporary hardware. However, the algorithm is image-based and, therefore, suffers from aliasing artefacts. The sampling locations upon rasterising the view of the camera and the light hardly coincide in practice. Therefore, undersampling (Section 2.1.1), oversampling (Section 2.1.2) and reconstruction errors (Section 2.1.1.1) cause aliasing artefacts which manifest in jagged shadow boundaries. Furthermore, if the sampling rate of the shadow map changes abruptly between consecutive frames, temporal aliasing introduces flickering artefacts at shadow boundaries due to insufficient reconstruction and undersampling. Additionally, incorrect self-shadowing (Section 2.1.4) arises from numerical imprecision and undersampling which necessitates depth biasing. An efficient algorithm, which provides enough robustness to address all of the aforementioned issues in every situation, is still missing. However, suggestions have been made which algorithms to apply for mitigating aliasing artefacts in particular situations [EASW09, SWP10].

Undersampling, oversampling and reconstruction errors can be addressed effectively and efficiently with algorithms which allow to filter the shadow map [DL06, AMB*07, AMS*08] (Section 2.1.2). These algorithms inherently support separable filtering and utilise hardware filtering units on contemporary GPUs to provide an improved reconstruction filter. Therefore, shadow maps can be of lower resolution because aliasing artefacts are hidden with filtering. In particular, it has to be noted that replacing PCF with VSM-based algorithms sacrifices precision in favour of significantly improved performance. This is a direct consequence of the fact that VSM estimates the outcome of PCF with a single depth sample. Light bleeding typically appearing with VSM is significantly reduced by applying exponential warping with EVSM. This approach is simple to implement, exhibits a low memory footprint and is less susceptible to incorrect self-shadowing. Remaining artefacts can be further reduced with z-partitioning (Section 2.1.1.4). However, VSM-based approaches are preferable for scenes which are free from multiple occluders and, therefore, exhibit low depth complexity (e.g. shadows from terrain).

In general, fitting considerably enhances shadow quality for outdoor scenes (Section 2.1.1.2). Additionally, fitting decreases the distance between the near and the far plane of the view frustum of the light. This is beneficial to warping (Section 2.1.1.3), global partitioning and mitigating incorrect self-shadowing. However, it has to be noted that fitting causes temporal aliasing.

LiSPSM [WSP04] have proven to be the most practical warping technique for all types of light sources. However, the improved falloff function suggested by [LGQ*08] must be used in order to ensure that the quality of LiSPSM remains at least as good as the quality of uniform shadow mapping. Both performance and memory consumption of LiSPSM are comparable to uniform shadow mapping. A tuneable parameter allows to adjust the strength of the warping with respect to the configuration of the scene. LiSPSM are mainly applicable in outdoor scenes where the light comes directly from above. Despite being applicable to directional lights, LiSPSM should be replaced with z-partitioning to yield better results if more than one shadow map per light source can be allocated [LGQ*08]. Of course, LogPSM are superior to LiSPSM. However, as long as logarithmic rasterisation is missing from GPUs, LogPSM are highly impractical to use.

In the limit z-partitioning approximates LogPSM which closely approaches the optimal distribution of error along the direction of light. However, due to multi-pass rendering of several shadow maps, memory consumption is increased and performance is decreased while shadow quality is increased. The partition scheme can be chosen flexibly with respect to the requirements of the application. For example, more resolution can be allocated for separate regions with high surface detail along the direction of light. According to covering the spherical view of a point light with a cube shadow map to generate omnidirectional shadows [Ger04] (Section 2.1.5), z-partitioning is impractical. As a consequence of several partitions for each face of the cube shadow map, memory consumption and memory bandwidth increase considerably. In this particular case face partitioning [Koz04] is preferable. Although z-partitioning can be combined with face partitioning, shearing artefacts require being handled appropriately [LGQ*08]. If the ratio far/near is large, z-partitioning with a few partitions is superior to face partitioning. Z-partitioning can be combined with warping. A uniquely chosen warping function is applied to each layer depending on the required accuracy for a given depth. However, combining z-partitioning and warping mainly results in increased temporal aliasing while quality improvements remain negligible.

Warping and z-partitioning improve undersampling due to perspective aliasing only. Projection aliasing can only be addressed with adaptive partitioning techniques (Section 2.1.1.5), alias-free sampling (Section 2.1.3) or temporal reprojection (Section 2.1.1.6). Despite providing shadows of high quality, adaptive partitioning infers both costly multi-pass rendering and read-backs. The read-backs are required to decide when to stop the iterative refinement of the shadow map. Alias-free sampling is capable of generating pixel-exact shadows with irregular sampling. However, the approaches are unsuitable for real-time applications. In the future alias-free sampling may become feasible with GPU-enabled irregular z-buffering or advanced capabilities of the parallel computing architecture exposed by upcoming GPUs. Temporal reprojection is completely unsuitable for generating shadows for dynamic scenes in real-time. Results of reasonable quality can only be achieved if objects and light sources are static.

Incorrect self-shadowing can be significantly reduced by storing the depth of back-facing geometry in the shadow map.

In essence, this corresponds to adaptively biasing depth values proportionally to the thickness of occluders. However, this is only applicable if geometry is closed (i.e. occluders exhibit thickness). Furthermore, additional depth biasing may be required where depth disparity is low (e.g. for exceptionally thin objects and close to object silhouettes). According to open geometry, constant- and slope-based biasing suffices for most configurations in practice. Although PCF is more precise in reducing reconstruction errors than VSM-based approaches, incorrect self-shadowing is seriously aggravated. This issue can only be addressed with more involved biasing methods. If depth complexity is low, LVSM or EVSM are an alternative. These approaches necessitate almost no depth biasing inherently.

According to point lights, standard shadow mapping needs to be extended to account for casting shadows into all directions of light. On contemporary hardware this is efficiently achievable with a cube shadow map. If geometry shaders are supported by the hardware a cube map can be created in a single pass. However, for geometry-intense scenes more involved culling techniques need to be employed in order to circumvent rasterising geometry redundantly.

2.2 Soft Shadows

Soft shadows are caused by light sources which exhibit spatial extend. The amount of light a surface point receives is proportional to the fraction of the area of the light source that is visible from that surface point. Partially occluded surface points introduce a smooth transition (penumbra) from fully lit points to fully occluded points (umbra). For accurate soft shadow generation visibility must be evaluated for an infinite number of samples on the light source which is a computationally intense process. Visibility calculation for real-time soft shadow generation builds on shadow mapping to represent occluder information. This is efficient and scales well but is prone to both aliasing and incorrect self-shadowing artefacts.

As a consequence of the numerous approaches to generate soft shadows, decent overviews have been written [HLHS03, EASW09]. Visually plausible algorithms are based on filtering the boundaries of hard shadows such that the generated shadows exhibit contact hardening. Of course, these algorithms are only applicable for small light sizes and are considerably limited. Physically plausible algorithms build on a more accurate representation of occluding geometry and evaluate light occlusion more precisely with back-projection. However, soft shadow mapping algorithms which utilise a single shadow map suffer from artefacts due to overlapping penumbrae, light bleeding and performance bottlenecks. Approaches utilising multiple shadow maps can only compute soft shadows for dynamic scenes at interactive rates because updating the extended shadow map is costly. However, real-time performance is reachable while preserving high shadow quality if occlusion is estimated more effectively with back-projection. These approaches only utilise a reduced number of shadow maps. Another fast variant sacrifices accuracy in favour of performance to yield visually plausible shadows with occlusion textures. Exploiting temporal coherence allows to sample the area of a light source efficiently over time. In the limit this results in accurate real-time soft shadows. In addition, soft shadows are caused by environmental lighting. At each surface point light impinges uniformly from all directions due to diffuse inter reflections. The light is partly blocked by nearby geometry. Ambient occlusion techniques effectively approximate environmental shadows in real-time based on the observation that a surface point appears darker the less it is exposed to the rest of the scene. Of course, geometry-based approaches generate soft shadows of superior quality but are inherently too costly for real-time applications.

An overview of image-based approaches identifies several techniques which are no longer of practical interest due to severe limitations, low shadow quality and clearly objectionable artefacts [EASW09]. Outer penumbra regions can be introduced by enlarging the boundaries of hard shadows which are generated from the centre of an area light source [PSS98]. Outer and inner penumbrae can be introduced by radially searching the neighbourhood of a depth sample to find an appropriate occluder [BS02]. The position of a pixel within the soft shadow of the found occluder is estimated with respect to the distance between occluder and receiver. A variant of the previous approach has been mapped to the GPU [KD03]. However, inner penumbrae can be introduced only. A different approach introduces both inner and outer penumbra [AHT04]. First, an edge detection filter is applied to find the boundaries of hard shadows. Then, the boundaries are successively enlarged by a single pixel on both the interior and the exterior, respectively, with a modified flood-fill algorithm. However, the approach remains expensive regardless of the optimisation of jump flooding [RT06].

2.2.1 Percentage-Closer Soft Shadows

Assuming a setup of an extended light source, a blocker and a receiver where all objects are planar and parallel, accurate soft shadows can be obtained with convolution [SS98]. General configurations require to vary the size of the filter in order to generate visually plausible soft shadows which harden on contact. *Percentage-Closer Soft Shadows* (PCSS) [Fer05] create visually plausible soft shadows exhibiting contact hardening by varying the size of a PCF filter kernel at each surface point of a receiver. Varying the filter size is based on a penumbra width estimation which involves the distances between light source, occluder and receiver (Figure 2.15).

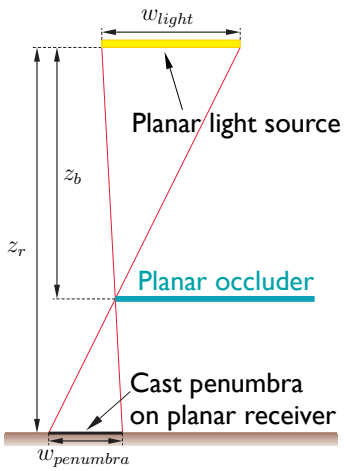


Figure 2.15: Percentage-Closer Soft Shadows, penumbra width estimation. Assuming a parallel and planar light source, occluder and receiver, the width of the penumbra $w_{penumbra}$ on a receiver at distance z_r from the light source is estimated based on the blocker distance z_b from the light source and the size of the light source w_{light} (image adapted from [EASW09]).

The blocker search step processes the neighbourhood of the corresponding depth sample of a receiver point to obtain the average blocker depth. Only depths being closer to the light source than the depth of the receiver point are averaged. Then, PCF is applied with a kernel size proportional to the estimated width of the penumbra. The penumbra width is estimated based on the size of the light source w_{light}, the average blocker depth z_b and the depth of the receiver z_r by

$$w_{penumbra} = \frac{z_r - z_b}{z_b} w_{light} \tag{2.13}$$

Despite the inapplicable assumption of a single parallel and planar occluder in general scene configurations, the results obtained with PCSS have proven to be sufficiently visually pleasing. Therefore, PCSS became widely used in real-time applications such as games. However, PCSS are inherently susceptible to yield incorrect results easily. This is a direct consequence of approximating an occluder representation, which generally consists of multiple occluders at different depths, by simply averaging depths being sampled from the centre of an extended light source. As a result, shadows generated with PCSS are considerably underestimated.

Furthermore, PCSS are only applicable to small lights for three reasons. First, achieving a smooth transition within large penumbrae infers a costly PCF stage and considerably limits performance. Second, a direct consequence of using PCF is that large filter widths dramatically aggravate incorrect self-shadowing which requires large biasing values. Therefore, objectionable gaps are introduced at contact shadows. Third, determining occlusion on the basis of a single average depth value across a large filter support results in incorrectly classifying umbrae fragments to be lit which causes light bleeding.

The performance of PCSS can simply be improved by adaptively limiting the number of shadow map accesses in both blocker search and PCF [Ura05]. Another option is to employ image processing techniques to identify visually important regions where PCF shall be applied [Isi06]. Confining costly PCF to penumbrae regions easily outweighs the additional overhead to identify those regions. However, reducing the number of samples introduces banding artefacts. If the number of samples in the PCF kernel does not increase accordingly while increasing the width of the filter window objectionable artefacts are introduced in penumbrae. These artefacts can efficiently be dealt with by utilising stratified sampling [VdB04, Ura05, Isi06]. The last approach randomly rotates a POISSON disk kernel in screen-space to hide banding artefacts in favour of less objectionable noise. As a consequence, stratified sampling improves quality and performance of PCSS. A considerably lower number of samples in the PCF window is necessary to achieve visually smooth penumbrae.

Without introducing banding or noise artefacts, PCSS has recently been accelerated considerably by utilising non-stationary and dynamic weight matrices to obtain unique per-sample filter weights [Gru10]. The dynamic weights are derived from evaluating a cubic BÉZIER function with four matrices. Blocker search and calculation of the filter weights are efficiently accelerated with the advanced capabilities of Shader Model 5. However, the approach is limited to rather small light sources to yield penumbrae of sufficient quality.

Screen-Space Percentage-Closer Soft Shadows (SSPCSS) [MKHS10] adopt edge-aware filtering to move the computational space for visibility computation of PCSS from light-space to screen-space which improves performance considerably. As a consequence of the fact that edge information is unavailable in screen-space, blocker search and convolution of the shadow map require to utilise edge-aware filtering. Accordingly, SSPCSS utilise cross-bilateral filters [TM98]. These

filters allow to smooth an image with respect to edges in a different image while preventing to convolve across edges. With a cross-bilateral POISSON filter SSPCSS is superior in performance and quality to PCSS with PCF utilising a randomly rotating POISSON disk for stratified sampling. With a separable cross-bilateral GAUSSIAN filter the performance becomes almost independent from the filter radius but the results are less accurate. In general, performance becomes almost independent of the shadow map resolution and only depends linearly on the number of screen pixels.

Utilising approaches which allow to pre-filter the shadow map (Page 19), the performance and robustness of the filtering stage of PCSS is considerably improved if costly PCF is replaced with VSM [Lau07]. Of course, PCSS requires filtering to support adaptively varying the size of the filter window for each screen pixel. Despite being easy to use and fast, mipmapping inherently needs to interpolate between several discrete levels with decreasing spatial resolution which are created with fixed-size filter kernels. Therefore, artefacts are easily objectionable in penumbrae. *Summed-Area Variance Shadow Maps* (SAVSM) [Lau07] utilise summed-area tables (SAT) for efficient and accurate adaptive box filtering which causes filtering of PCSS to become a constant-time operation. Despite being superior in quality to mipmapping, SAT are more expensive to create and consume significantly more memory. However, the cost of creation can considerably be improved [HSC*05]. Furthermore, SAT require high precision texture formats to avoid related artefacts. In particular, concerning SAVSM the creation of SAT infers extensively accumulating numbers which exhibit minor errors due to quantification. Therefore, the numerical stability of the CHEBYCHEV inequality (Equation 2.9) is considerably aggravated which needs to be compensated with higher precision [EASW09]. *N-Buffers* [Déc05] achieve a reasonable balance between cost of creation and quality if implemented efficiently on contemporary hardware utilising a hierarchical approach [ED06]. As a consequence of avoiding the resolution reduction of mipmapping, results remain inaccurate but exhibit higher quality. However, the performance of VSM-based approaches inherently suffers from a costly blocker search stage. Averaging of depths is inefficiently achieved with point sampling which results in numerous expensive depth texture accesses [Lau07]. Furthermore, a VSM-based filtering stage (SAVSM, LVSM, EVSM) easily leads to incorrect classification of occlusion with large penumbra widths and large lights [YDF*10].

Convolution Soft Shadow Maps (CSSM) [ADM*08] considerably accelerate the blocker search stage of PCSS. Averaging of depths can be expressed as a convolution by

$$z_b = \frac{\sum_{q \in \mathcal{R}_s} \omega_{avg}(q) \cdot \left(\bar{f}(z_r, z_q) \cdot z_q\right)}{\sum_{q \in \mathcal{R}_s} \omega_{avg}(q) \cdot \bar{f}(z_r, z_q)} \qquad \text{with } \bar{f}(z_r, z) = \begin{cases} 1 & \text{if } z < z_r \\ 0 & \text{if } z \geq z_r \end{cases} \tag{2.14}$$

In equation 2.14,

- z_b is the average blocker depth
- q denotes a sample within the region $\mathcal{R}_s$ of the shadow map that is being processed to average the depths of potential occluders
- ω_{avg} is the averaging kernel
- $\bar{f}(z_r, z_q)$ defines the "complementary" depth test which ensures selectively averaging the depths z_q within $\mathcal{R}_s$ which are closer to the light source only.

As a consequence, the blocker search becomes a constant-time operation. The convolution of a low and fixed number of basis images yields the average blocker depth within a region of the shadow map. The size of the region can be varied by pre-filtering the basis images at a constant cost (e.g. summed-area tables). The denominator is evaluated by reusing the basis images created for performing CSM-based filtering [AMB*07]. However, additional basis images are necessary to evaluate the numerator. This basis images need to be created and stored additionally which decreases performance and increases memory consumption. Another option to accelerate the averaging of depths is to exploit temporal coherence [SSMW09] (Section 2.2.4).

Variance Soft Shadow Maps (VSSM) [YDF*10] address two major limitations of VSM-based PCSS. First, instead of utilising point sampling, the average blocker depth is efficiently estimated with a VSM-enabled formula. Second, incorrect classification of occlusion due to large light sizes is improved with an efficient filter kernel subdivision scheme. For a given depth d, the depth values of all samples d_i within the blocker search region of the shadow map can be categorised into values $d_i \geq d$ and $d_i < d$ with the averages z_{unocc} and z_b, respectively. Then, the average blocker depth z_b can be estimated with

$$z_b = \frac{z_{avg} - P(x \geq d)}{1.0 - P(x \geq d)} z_{unocc} \tag{2.15}$$

In equation 2.15,

- z_{avg} is the average of all depth samples within the blocker search region of the shadow map and corresponds to the first moment of the depth distribution stored in the VSM
- $P(x \geq d)$ is the proportion of depth samples being unoccluded within the blocker search region and is obtained by evaluating the inequality of CHEBYCHEV (Equation 2.9)
- z_{unocc} is the average depth of samples being unoccluded within the blocker search region. As a consequence of yielding high quality soft shadows for general cases, $z_{unocc} = d$ is simply assumed.

The inequality of CHEBYCHEV is only applicable if $z_{avg} < d$. Therefore, VSM assumes that a fragment is lit if $z_{avg} \geq d$. However, the assumption is easily violated if the filter window is large. The probability of enclosing depth values smaller than z_{avg} under the support of the filter increases considerably. Therefore, VSSM subdivides large kernels into a set of sub-kernels. According to sub-kernels for which $z_{avg} < d$ holds, VSM-based average blocker depth estimation and PCF estimation can be performed straightforwardly. The precondition of the inequality of CHEBYCHEV holds. According to sub-kernels for which $z_{avg} \geq d$ holds, blocker depth samples are summed. The results of sub-kernels are accumulated to finally obtain the average blocker depth of the kernel. Then, soft shadow computation is performed by utilising the kernel subdivision scheme for sub-kernels for which $z_{avg} \geq d$ holds. PCF with a 2×2 kernel is applied to efficiently evaluate soft shadows. The subdivision scheme is considerably accelerated based upon the observation that fragments in a sub-kernel are entirely lit if $z_{avg} \geq d$ holds and σ^2 is smaller than a certain threshold. This allows to fine-tune the tradeoff between quality and performance depending on the application requirements. In contrast, adaptive kernel subdivision is preferable if the number of sub-kernels is large. The adaptive scheme constructs a quad-tree of varying-sized sub-kernels in the two-dimensional domain of the filter. The sub-kernels are efficiently traversed with a GPU-based approach which allows a linear forward traversal. Furthermore, efficient classification of lit and umbra regions in the scene is achieved by comparing the depth of a fragment to the depth range within a search region. The depth range of a search region is efficiently determined from a hierarchical shadow map. As a result, VSSM is effectively confined to penumbrae regions in the scene only. Contact noise arising from imprecisions due to pre-filtering the shadow map with summed-area tables (SAT) is mitigated with filtering. PCF with a 3×3 kernel is applied if the difference between z_{avg} and d is smaller than a certain threshold. In comparison to previous PCSS approaches, VSSM considerably improves overestimation of penumbrae while achieving real-time performance. However, maintaining a SAT for pre-filtering the shadow map significantly limits performance at high shadow map resolutions.

2.2.2 Soft Shadow Mapping

Physically plausible soft shadows can be generated with *soft shadow mapping* or *back-projection*. A more accurate occluder representation is reconstructed from a shadow map which is sampled from the centre of the extended light source. The visible fraction of the extended light source with respect to a single receiver point is estimated by back-projecting the approximated occluding geometry onto the plane of the extended light source. *Soft Shadow Maps* [AHL*06] unproject shadow map texels into world-space to represent occluding geometry with micro-occluders. The contribution of these micro-occluders to the overall occlusion is scattered into a soft shadow map which is projected onto the scene. As a consequence of performing visibility computation in light-space, occluders and receivers need to be dealt with separately (no self-shadowing). Furthermore only small shadow map resolutions are usable practically. Therefore, subsequent approaches operate directly in screen-space and iterate over all relevant micro-occluders to gather the blocking contribution for each receiving point using a single shadow map only [GBP06, BS06, ASK06]. If micropatches are used to represent micro-occluders, a micropatch is generated by unprojecting the texels of the shadow map into world-space (Figure 2.16). The shadow map is sampled from the centre of the extended light source. If a micropatch is determined to partially occlude the light source, the micropatch is back-projected from the receiving point onto the light plane to obtain the occluded proportion of the area of the light source. The total occlusion of the extended light source results from accumulating the individual occlusions of all back-projected micropatches. Finally, the ratio of the occluded light area and the total light area corresponds to the attenuation of light at a receiver point. However, individual occlusions of back-projected micropatches overlap in general. Therefore, incorrect occluder fusion easily occurs which results in overestimating light occlusion.

Bitmask Soft Shadows [SS07] avoid overlapping back-projections with a robust visibility determination utilising stratified sampling. The area of the light source is approximated with randomly distributed sample points representing binary visibilities. The visibilities are efficiently encoded in a bit field. While back-projecting micropatches the bitfields are uniquely updated with bitwise operations. As a consequence, overlapping of areas of back-projected micropatches is circumvented and, therefore, occluder fusion is correctly handled. Furthermore, discretely sampling the area of the light source enables to determine which parts of the area of the extended light source are occluded. However, artefacts are introduced if the discretisation of the area of the light source is too coarse.

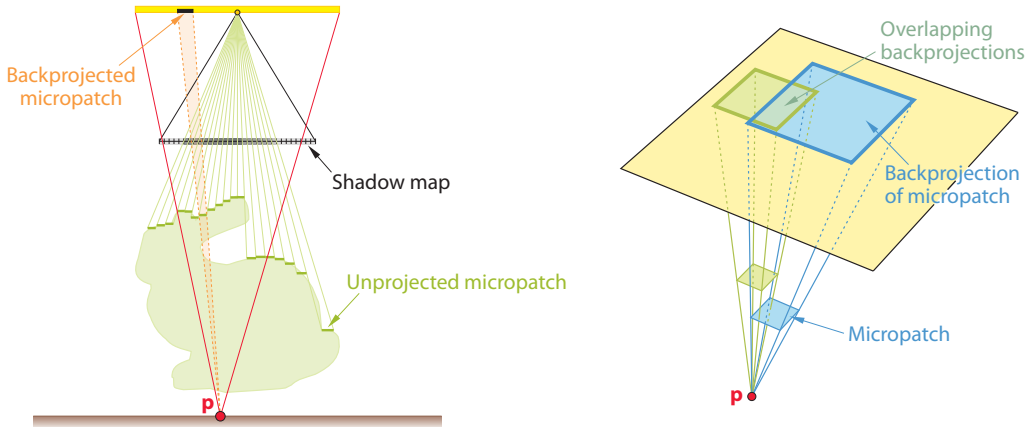


Figure 2.16: Soft Shadow Mapping. Left: Basically, soft shadow mapping, first, generates micro-occluders by unprojecting shadow map texels into world-space. Then, the resulting micropatches are projected onto the area of the light source. Finally, the resulting areas are accumulated to yield an estimate of the total occluded area of the extended light source. Right: The covered proportion of the area of the extended light source is determined by projecting micropatches onto the area of the extended light source individually. Of course, overlapping of the projected areas inevitably occurs. As a consequence, light occlusion is overestimated if the individual occluded areas are simply accumulated (image adapted from [EASW09]).

Occluding geometry is approximated with three different types of micro-occluders each having its particular advantages and disadvantages (Figure 2.17). *Micropatches* [AHL*06] are a simple piecewise-constant approximation which facilitates back-projection and coverage determination. However, occluders are easily overestimated and gaps potentially occur between neighbouring micropatches which introduces light bleeding. Of course, gaps can be roughly closed by accordingly extending micropatches with respect to neighbouring micropatches [GBP06, Bav08]. Instead of gap filling, which easily leads to overestimation, a multi-layer shadow map created with depth peeling [Eve02] can be used for improved but costly micropatch construction [BCS08, NJH10]. Furthermore, micropatches easily cause incorrect self-shadowing which can be costly alleviated with midpoint shadow maps [BCS08]. *Microquads* [SS07] are a piecewise-(bi)linear approximation which more accurately represents occluding geometry with a regular mesh of micro-occluders. Therefore, incorrect self-shadowing and light leaking due to gaps is inherently avoided. Furthermore, the occurrence of overlapping back-projections is considerably reduced. However, coverage determination becomes generally complicated and occluders are possibly underestimated. Therefore, the probability to ignore finely structured occluder geometry is increased considerably. Occluder underestimation can be improved with microtris [SS08a]. Visual quality is improved hardly but performance is increased significantly. *Occluder contours* [GBP07] are silhouettes enclosing aggregates of adjacent occluding samples within a region in the shadow map. The construction process of occluder contours ensures smoothness upon extracting contour edges connecting two adjacent sample borders which reduces aliasing. As a consequence of back-projecting contour edges only, efficiency is improved because the number of micropatches being enclosed by the contour is typically higher than the number of edges. Furthermore, the construction process of occluder contours inherently allows to accelerate visibility determination by exploiting coherence [YFGL09]. However, potential occluders are easily ignored because the edge extraction process is performed in two-dimensional space. Therefore, approaches using occluder contours are susceptible to popping artefacts if the light source smoothly moves relative to the occluder.

Multiscale representations of the shadow map accelerate soft shadow mapping considerably. The set of micro-occluders is refined to those which truly contribute to the result. An initial estimate for the region of the shadow map to search within for occluding geometry is obtained by intersecting the near plane of the shadow map with the pyramid formed by the receiving point and the extended light source. Starting from this conservative estimate, the region can be iteratively refined if the minimum and maximum depth values within the search region are known [GBP06]. Furthermore, with the known depth range entirely lit or umbrae fragments can be identified in advance. Therefore, processing of further irrelevant micro-occluders can be skipped [GBP06]. A conservative bound for the depth range within a shadow map region can be efficiently determined with a *Hierarchical Shadow Map* (HSM) [GBP06]. A significantly improved classification of fragments can be achieved with a *Multi-Scale Shadow Map* (MSSM) [SS07]. MSSM reduce cost of soft shadow mapping considerably. However, the high creational and storage costs of MSSM limit the practically usable shadow map resolution. In contrast, a HSM exhibits considerably lower costs due to decreasing spatial resolution across levels of hierarchy. Therefore, to unify the strengths of HSM and MSSM, both approaches are combined into a *Y Shadow Map* [SS08b]. Relevant occluding geometry is identified without iterating over samples within a search region. Micropatches are directly extracted from a HSM by hierarchically traversing a quad-tree derived from the HSM [DU07]. Occluder

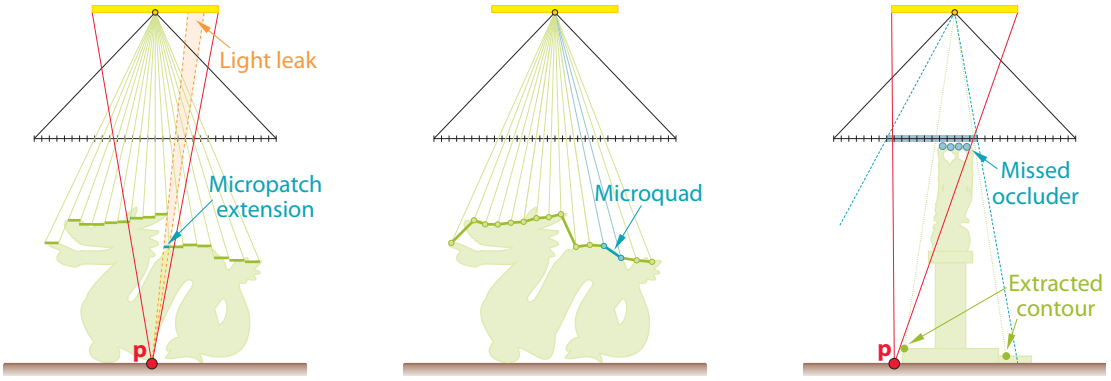


Figure 2.17: Soft Shadow Mapping. Three types of micro-occluders are commonly derived from shadow map samples. Left: Micropatches are individual rectangles parallel to the plane of the extended light source. The patches are constructed by unprojecting the extends of shadow map texels into world-space. Therefore, gaps easily occur which causes light bleeding. Middle: Microquads span a regular quad mesh. Each vertex of the mesh results from unprojecting the centre of a shadow map texel. As a consequence, gaps are implicitly avoided. Right: Occluder contours represent smoothly extracted silhouettes which enclose aggregates of adjacent occluding samples within the shadow map region. Despite avoiding gaps implicitly, occluders are easily missed while extracting occluder contours (image courtesy of [EASW09]).

contours are directly extracted from a MSSM in a similar way [YFGL09].

Performance of soft shadow mapping is further improved towards real-time by employing sampling strategies. These strategies enable to reduce the sampling rate while maintaining sufficiently accurate sampling of occluding geometry. Simply limiting the number of processed micro-occluders for a single receiver point causes inaccuracies. This is compensated by accordingly sub-sampling the relevant search area utilising a Gaussian Poisson distribution [BS06] or regular sub-sampling [BCS08]. Another efficient option to approximate occluding geometry coarsely is to construct a reduced number of micro-occluders from coarser levels of a HSM or a MSSM, respectively. In general, the consequences of choosing micropatches, microquads or occluder contours to represent micro-occluders remain largely valid. However, incorrect self-shadowing is easily aggravated while determining an appropriate depth bias for coarser levels is hardly achievable. Moreover, transition artefacts within penumbrae are introduced if levels differ for adjacent screen pixels. This is solved by combining the results of both levels utilising alpha blending [GBP07, SS08b]. The quality of coarser approximations significantly benefits from using a more flexible type of micro-occluder. However, *Microrects* [SS08b] are more costly to generate and represent. Finally, performance is considerably increased by applying a sparse sampling scheme in screen-space. This essentially confines costly accurate sampling to penumbrae regions identified previously in screen-space [GBP07]. However, regular sparse sampling patterns infer related artefacts which become objectionable in dynamic scenes particularly.

2.2.3 Multi-Layered Shadow Maps

More accurate soft shadows can be generated if several samples are taken along a ray which is cast from a surface point to an area light source. These samples are aggregated in an extended shadow map which is considered to be located at the centre of the light source. While rendering the view of the camera the extended shadow map is queried to determine the number of sampling points on the light source which are visible from a given surface point.

Layered Attenuation Maps [ARHM00] employ *Layered Depth Images* (LDI) [SGHS98] for creating the extended shadow map. The LDI is generated by rendering a shadow map for each sample point on the light source. The resulting depth samples are warped to the view of the LDI which resides on the centre of the light source. A new depth layer is added to the LDI for every warped depth sample which is missing from the LDI. If a warped depth sample maps to an existing depth sample in the LDI a counter is incremented for that depth sample. As a result, each pixel in the LDI stores a list of depth values associated with the number of visible sample points on the light source. From the LDI an attenuation map is derived by dividing the recorded number of light samples visible for a depth sample in the LDI by the total number of sample points on the light source. While rendering the attenuation map is queried to retrieve a measure for the fraction of light that is visible from a surface point. Accordingly, if no entry is found in the attenuation map the surface point is considered to be completely in shadow. Despite being capable of generating rather accurate soft shadows with a high number of samples, the generation of the LDI and the attenuation map particularly remains expensive. Therefore, the performance of this approach is rather limited.

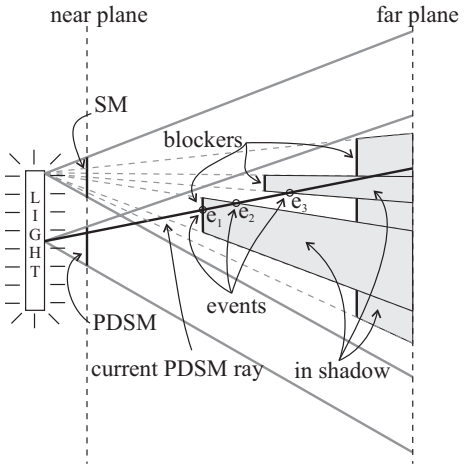


Figure 2.18: Penumbra Deep Shadow Maps. For each texel of the penumbra deep shadow map a three-dimensional ray is projected into a shadow map taken from a sample point on the light source. The ray gathers a fraction of the overall attenuation information from a single shadow map according to the changes in visibility (events) occurring along the ray (image courtesy of [SAPP05]).

Penumbra Deep Shadow Maps [SAPP05] base the extended shadow map on *Deep Shadow Maps* (DSM) [LV00] (Page 8). Each texel of a DSM stores the attenuation of light as a function of depth along a ray of light passing through the texel. The attenuation of light results from integrating visibility along the ray. Construction of the visibility function for each texel of a DSM requires to take a larger number of samples. First, a shadow map is generated for each sample on the light source. Then, for each shadow map a fraction of the overall attenuation information is gathered into the extended shadow map to capture the view of the light source from the centre of the area (Figure 2.18). For each texel of the extended shadow map a three-dimensional ray is projected into every shadow map to gather the attenuation in light-space. The attenuation contributed by one shadow map is computed by identifying changes in visibility along the projected ray for all covered shadow map texels. Accordingly, the resulting visibility function is integrated to yield the attenuation and compressed for optimised storage. While rendering the view of the camera the extended shadow map is queried to retrieve the according attenuation factor for a given surface point. Of course, if the extended shadow map needs to be updated repeatedly in dynamic scenes, performance considerably degrades below interactivity. Furthermore, memory consumption is high because a considerable amount of samples and, therefore, a large number of shadow maps are required to yield nearly accurate soft shadows.

Occlusion Textures [ED06, ED08] are based on the generation of approximate soft shadows with convolution [SS98]. First, the view frustum of the light is partitioned uniformly into several sub-frusta. Then, for each sub-frustum a binary occlusion texture is created by projecting geometry inside the sub-frustum onto the far plane of the sub-frustum. As a result, the occlusion texture represents the coverage of the light source inside a sub-frustum. Finally, shadows are generated by adaptively filtering the occlusion textures. In particular, if light source $\mathcal{L}$ and occluder $\mathcal{O}$ are planar and parallel, the visibility integral in the soft shadow equation (Equation 3.12) becomes a convolution which simply needs to be scaled proportionally to the size of the light source. According to equation 2.16, given a binary function $\delta_{\mathcal{O}}$ which describes the occlusion of the light source for a certain planar occluder in the plane $\Pi_{\mathcal{O}}$, the visible fraction of the light V at a surface point p can be calculated by integrating 1-$\delta_{\mathcal{O}}$ over the rectangular region $\mathcal{K}$ that results from intersecting the plane $\Pi_{\mathcal{O}}$ with the pyramid formed by the area of the light and the surface point p (Figure 2.19).

$$V = \frac{1}{|\mathcal{K}|} \int_{x \in \mathcal{K}} (1 - \delta_{\mathcal{O}}(x)) dx \tag{2.16}$$

Accordingly, the visible fraction of the light can be calculated by filtering the occlusion texture with a box filter whose size is obtained by scaling the size of the light source with respect to the ratio of the distances $\mathrm{d}(p, \Pi_{\mathcal{O}})/\mathrm{d}(p, \mathcal{L})$. Multiple planar occluders are dealt with by partitioning the view frustum of the light and generating an occlusion texture for each sub-frustum. Each occlusion texture is filtered accordingly with an appropriately sized box filter. The resulting visibility fractions are accumulated multiplicatively to yield the total visibility of the light source. Of course, shadowing within a sub-frustum is disregarded. Furthermore, the quality and the performance of the approach mainly depends on the chosen filtering algorithm for arbitrary filter sizes. Mipmapping is fast due to hardware support but introduces objectionable artefacts. Summed-area tables sacrifice performance in favour of more accurate results. N-buffers [Déc05] provide the best performance quality tradeoff because an efficient implementation is achievable on contemporary hardware [ED06]. In particular, it has to be noted that the performance of the algorithm is independent from the size of the light and the size

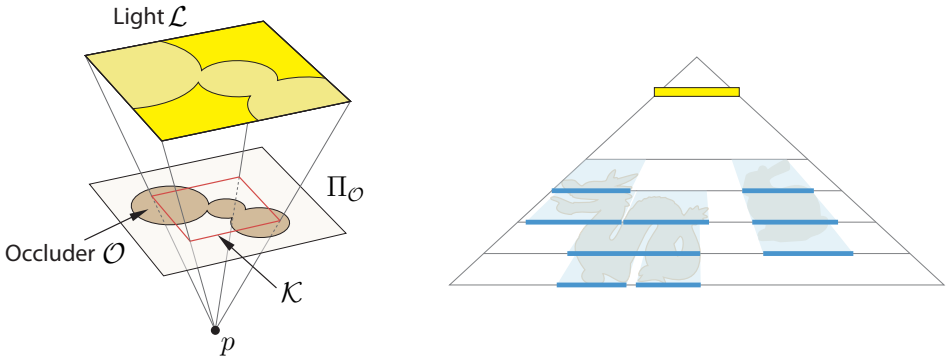


Figure 2.19: Occlusion Textures. Left: Occlusion textures represent occlusion of the light source with a binary function describing a planar occluder in the plane $\Pi_{\mathcal{O}}$. This representation is filtered with a box filter over the region $\mathcal{K}$ to yield an approximate measure for the visibility of the light source for a surface point p. Right: To account for multiple planar occluders the view frustum of the light is partitioned into several sub-frusta. The geometry inside each sub-frustum is projected onto the far plane of the sub-frustum to generate the according occlusion texture (image adapted from [EASW09]).

of the penumbra. However, occlusion textures scale poorly for large scenes and only generate visually plausible shadows. As a consequence of the discretisation of the view frustum of the light, occlusion textures are incapable of dealing with occluder fusion correctly. Furthermore, the approximation can cause light bleeding; particularly for fine geometry.

Layered Occlusion Maps (LOM) [NJH10] adopt LDI to overcome the limitations of soft shadow mapping [AHL*06]. First, an LDI is sampled from the centre of the extended light source. Then, a layered occlusion map is constructed by estimating an occlusion degree for each sample in the LDI using back-projection of micropatches. Finally, the attenuation of light for a receiver point is determined by filtering the LOM accordingly. The LOM is constructed from the LDI which is created by discretising the scene with depth peeling [Eve02]. For each sample in the LDI a micropatch is constructed. Potential occludees are identified based on the observation that the shadow volume of a micropatch is included in the shadow volume of another micropatch being closer to the light source. For each potential occludee the occlusion degree is estimated by back-projecting the micropatch being closest to the potential occludee. Upon filtering the LOM each fragment is transformed into light-space and four neighbouring samples are determined. The proportion of occlusion is obtained by trilinearly filtering the occlusion degrees of the samples with respect to the distances between the samples. Of course, LOM filtering is susceptible to incorrectly yield partial occlusion. Particularly, if an obviously fully lit fragment resides in a concavity and neighbouring samples are closer to the light source. This can be circumvented by increasing the resolution of the LDI. As a consequence of utilising depth peeling, the performance of LOM is mainly dependent on the number of layers in the LDI. The necessary number of layers depends on both scene configuration and surface detail. However, the contribution of deeper layers to partial occlusion is practically negligible. Therefore, creation and storage cost of the LDI remains reasonable. Light bleeding commonly occurring with micropatches is avoided by using back-faces for calculating occlusion degrees. For reducing the cost of occlusion degree computation, the set of potential occludees can be refined with a conservative estimate of the depth range within the kernel region that is obtained from a HSM. In summary, LOM are capable of rendering plausible soft shadows in real-time exhibiting noticeable underestimation of penumbrae.

2.2.4 Temporal Coherence

Temporal coherence can be exploited to calculate soft shadows in real-time by accumulating the single-sampled visibility of the light source randomly over time [SSMW09, SSM11]. An area light source is randomly sampled with several shadow maps. Instead of sampling all shadow maps within a frame, only a single shadow map is sampled per frame. The resulting hard shadows of previous frames are combined with a "shadow buffer" to obtain a soft shadow for each screen pixel. The convergence of the iterative refinement is considerably increased with exploiting temporal coherence. This approach is based on two observations. First, the majority of fragments remains unchanged between consecutive frames. Second, data of fragments of previous frames can be reused with temporal reprojection (Page 18). Accordingly, two fragments of adjacent frames are considered to be equal if their depth values hardly differ which allows to reuse previously computed data. As a consequence of sufficiently randomly sampling the area of the extended light source, an estimator for the fraction of the visibility of the light source can be derived from the variance of the Binomial distribution of the shadowed samples.

Therefore, the soft shadowing result ψ can be estimated with the proportion $\hat{\psi}_n(x,y)$ which is computed iteratively over time with the data reused from previous frames.

$$\hat{\psi}_{cur}(x,y) = \frac{\tau_{cur}(x,y) + \rho_n(x,y)}{n(x,y)+1} \tag{2.17}$$

In equation 2.17 for a fragment at position (x,y) in screen-space,

- $\hat{\psi}_{cur}(x,y)$ is the approximation of the soft shadowing result of the current frame
- $\tau_{cur}(x,y)$ is the result of the depth test of the current frame
- $\rho_n(x,y)$ is the sum of the results of the previous depth tests
- $n(x,y)$ is the number of depth tests which have been performed for this fragment. Note that n varies for each fragment because previous data is unavailable if a fragment was occluded in the previous frame.

The data required to efficiently evaluate $\hat{\psi}_{cur}(x,y)$ is aggregated in the "shadow buffer". For each fragment this buffer stores the depth, the results of all tests $\tau_{cur} + \rho_n$ and the number of all depth tests $n+1$. However, issues arise for two reasons. First, using temporal coherence to improve the convergence of the estimation of ψ with $\hat{\psi}_n$ requires fragments to stay visible for many frames. Second, the estimation exhibits large and abruptly changing errors if new fragments suddenly disocclude. Initially, the data for exploiting temporal coherence is unavailable and, therefore, needs to be recorded over subsequent frames. Accordingly, a fast variant of PCSS is used for initialisation which is subsequently refined with $\hat{\psi}_n$. However, additional smoothing with a special neighbourhood filter is necessary to reduce noise and flickering artefacts. Furthermore, to improve the performance of the blocker search step of PCSS, temporal coherence is exploited by storing the average blocker depth over several frames in the fourth channel of the shadow buffer. Of course, moving objects aggravate temporal coherence because disocclusions become very frequent. As a consequence of applying PCSS repeatedly to shadow maps sampled from different points on the light source between consecutive frames, shadows irritatingly fidget. This is circumvented by applying the smoothing neighbourhood filter which removes objectionable differences between moving and static shadows. In general, generating soft shadows with temporal coherence is superior to PCSS in both quality and performance. However, objectionable artefacts remain in highly dynamic scenes due to low temporal coherence.

2.2.5 Environmental Shadows

In general, a surface point non-uniformly receives light from all directions as a consequence of indirect illumination. Therefore, additional soft shadows appear which considerably contribute to the visual richness of an image. Real-time generation of environmental shadows is achieved with convincingly approximate solutions because evaluating global illumination is still too costly. Finally, environmental shadows and shadows due to direct illumination are combined to yield convincing results.

Similar to covering the spherical view of a point light with several shadow maps, environmental lighting can be simulated with several point lights distributed non-uniformly on a surrounding hemisphere [Pha04]. Another approach uses several area lights to decompose an environment map [ADM*08]. Despite the availability of fast shadowing techniques, a separate pass is required for each light which considerably limits the performance of these approaches.

Multi-pass rendering is avoidable if large low-frequency light sources are assumed. Accordingly, incident radiance and light visibility become expressible as weighted basis functions using spherical harmonics. Utilising a hierarchical sphere approximation for each object, visibility can be efficiently determined with generic pre-tabulated spherical harmonics projection in real-time [RWS*06]. Despite being capable of handling occluder fusion correctly, the approach yields inaccurate results as a consequence of coarse sphere approximations and the low-frequency spherical harmonics representation. In particular, inaccuracies are noticeable at contact shadows which hinders plausible appearance.

Environmental shadows can be convincingly and efficiently approximated with ambient occlusion. The approximation provides important visual cues of depth, curvature and spatial proximity. Based on the assumption that a surface point receives light from all directions uniformly, the amount of incident light is attenuated with respect to an accessibility value [Mil94]. Accordingly, a surface point appears darker the more light is blocked by nearby geometry. A decent overview on ambient occlusion techniques has identified only a few techniques being capable of computing GPU-based ambient occlusion in real-time for dynamic scenes [MFS09].

Pre-computed ambient occlusion gathers occlusion information in advance. An approximation of occlusion is stored in an environment map for efficient storage and access. *Ambient Occlusion Fields* [KL05] use a quadratic rational function of distance to represent occlusion information. The components of the function are pre-computed and stored in a cube map

for every object. This allows to compute a spherical cap approximation of ambient occlusion in real-time. However, the approach is inapplicable to self-shadowing and deformable objects. Ambient occlusion information can be interpreted as the solid angle projected to every texel in a three-dimensional texture surrounding an object [MMAH07]. This technique supports self-shadowing and solid moving objects. Pre-computed ambient occlusion approaches are fast and exhibit high temporal coherence. However, shadows possibly appear artificial due to high smoothness.

Real-time ambient occlusion for deformable objects can be computed by constructing surface elements at each vertex and determining accessibility between the surface elements [Bun05, HJ07]. Each surface element is projected to all other surface elements to obtain the proportion of occlusion. Therefore, the complexity of this approach is $O(n^2)$. This limitation can be improved to $O(n \log n)$ by hierarchically grouping surface elements. Hence, groups of surface elements irrelevant to local occlusion can be efficiently skipped. These surface elements insignificantly contribute to the occlusion of the currently processed surface element due to large distances. However, artefacts arise from computing ambient occlusion on a per-vertex basis and per-fragment interpolation if objects are insufficiently tessellated. Furthermore, temporal incoherences easily occur which results in abruptly changing shadows.

Screen-Space Ambient Occlusion (SSAO) [SA07, Mit07] is based on the observation that ambient occlusion is caused in large part by nearby blockers if geometry exhibits high surface detail. The amount of locally blocked light can be effectively derived from an offscreen buffer storing the depth and normal at each screen pixel. This is possible because neighbouring points in world-space remain neighbouring in screen-space. The occlusion at each pixel is obtained by averaging partial occlusion [BSD08]. The approach estimates partial occlusion with the horizon and tangent angles being sampled in multiple two-dimensional directions within the neighbourhood of the pixel. The horizon angle is obtained by marching along the nearby height field in a certain direction to find the steepest angle. The tangent angle is obtained from the normal. The number of samples necessary to yield plausible results is significantly reduced by randomly sampling in varying directions per pixel. However, noise is introduced and needs to be suppressed with edge-aware filtering to yield visually pleasing results. The performance of SSAO is mainly dependent on the number of samples taken per pixel and independent of the scene complexity. As a consequence of the low frequency nature of ambient occlusion, the resolution of the offscreen buffer can be half the resolution of the frame buffer. Therefore, SSAO is capable of computing ambient occlusion in real-time for deforming objects and became widely used in games. However, artefacts are introduced if occluders are nearly parallel to the direction of view. In particular, false occlusion halos appear around thin geometry (e.g. fences, grids). Furthermore, distant occluders are easily undersampled in screen-space. Moreover, SSAO requires sufficiently tessellated and high detail geometry in order to yield convincing results.

2.2.6 Summary

Generating soft shadows is a computationally intense process. The fraction of the visible area of an extended light source must be determined for each surface point. Image-based approaches provide real-time performance while hardly exhibiting aliasing artefacts. Visually plausible soft shadows can be generated at acceptable performance with convolution-based approaches. Physically plausible soft shadows are derived with evaluating the contribution of back-projected micro-occluders to light occlusion. In general, approaches utilising a single shadow map deliver convincing soft shadows for small light sources only. In contrast, larger lights require additional depth layers whose creation and storage cost decreases performance considerably.

PCSS [Fer05] and occlusion textures [ED06] are convolution-based approaches. Performance is independent of the size of the light and the size of the penumbra. As a consequence of its wide application in games, PCSS received considerable attention and has undergone several improvements to finally become a constant-time algorithm. The average blocker depth can be determined with a CSM-based convolution [ADM*08] or with a VSM-based formula [YDF*10]. Adaptively varying the size of the filter window for each screen pixel is efficiently and accurately achieved with SAT and a linear representation of the shadow map [Lau07]. Incorrect classification of occlusion of VSM-based PCSS with large lights has been addressed with an efficient filter kernel subdivision scheme [YDF*10]. However, PCSS generally underestimate shadows and are only applicable if lights are small. In particular, it has to be noted that PCSS deliver convincing shadows caused by the sun in large scenes. According to smaller indoor scenes illuminated by large lights, occlusion textures are more practical and exhibit higher performance [ED08]. Both PCSS and occlusion textures assume parallel and planar occluders, receivers and light sources. This assumption is violated in general scene configurations which, therefore, leads to incorrect occluder fusion.

Soft shadow mapping (Section 2.2.2) and multi-layered shadow maps (Section 2.2.3) provide higher precision and more robustness for a considerably higher cost. Back-projection of microquads and determining light source coverage with occlusion bitmasks achieve the most accurate results if a single shadow map shall be used only [SS07]. As a consequence of utilising occlusion bitmasks, overlapping of the projections of the micro-occluders on the area of the extended light source is avoided and, therefore, occluder fusion is correctly evaluated. Representing micro-occluders with occluder contours improves shadow quality of large lights [GBP07]. However, performance is aggravated considerably

and shadow flickering is easily introduced in dynamic scenes due to missing occluders. In contrast, maintaining shadow quality while improving performance can be achieved with hierarchical extraction of occluder approximations [YFGL09]. However, the practically usable shadow map resolution is limited considerably. If more than one shadow map is allowed, the highest reachable shadow quality at reasonable cost is delivered by replacing the single shadow map with a shallow LDI [NJH10]. As a consequence the accuracy of representing and back-projecting occluders is improved significantly.

Exploiting temporal coherence and reusing data of previous frames with temporal reprojection enables to compute accurate soft shadows iteratively [SSMW09] (Section 2.2.4). The extended light source is randomly sampled with a single shadow map each frame. The data necessary to estimate the visible fraction of the light source is stored in a four channel screen-sized offscreen buffer. Despite exhibiting superior shadow quality and performance compared to PCSS, objectionable artefacts appear in highly dynamic scenes due to low temporal coherence. As a consequence of converging to the accurate solution in the limit, the algorithm deals with occluder fusion correctly.

Environmental shadows (Section 2.2.5) can be approximated efficiently and convincingly with ambient occlusion. The light uniformly reaching a surface point is attenuated accordingly with respect to the occlusion of nearby geometry. Pre-computed ambient occlusion techniques store occlusion information in environment maps for efficient storage and fast access [RWS*06, MMAH07]. The resulting shadows are easily perceived as being artificially attached due to high smoothness. Abruptly changing shadows are hardly appearing with moving objects. However, only moving objects are supported. SSAO techniques obtain occlusion information by sampling the heightfield within the neighbourhood of a pixel in multiple directions [SA07, BSD08]. According to geometry exhibiting high surface detail, SSAO provides convincing results while being particularly efficient. Furthermore, SSAO remains applicable if objects are continuously deformed due to animation. The performance of SSAO is independent from the scene complexity. However, shadow flickering is easily introduced with dynamic objects. Furthermore, shadows caused by fine geometry are generally overestimated. In conclusion, SSAO is only a quality-improving supplement to explicitly generated soft shadows. Important visual cues are missing due to coarse approximations in screen-space.

Chapter 3

Theory

3.1 Global Illumination

Rendering a photorealistic image of a virtual scene requires to evaluate light transport. The simulation of the complex interaction of light and matter obtains the proportion of radiance reaching each pixel in the resulting image. While propagating in a scene, rays of light are reflected, refracted and scattered until they reach the eye or are absorbed. Therefore, a surface point possibly receives light coming from all other surface points. Global illumination computation incorporates all various paths the light travels from the light source to the eye for estimating the radiance reaching each surface point. The contents and notation of this section mainly follows [DBB06].

Despite being computationally intense, global illumination computation delivers important visual clues that would be missing with local illumination (Figure 3.1). *Indirect illumination* causes surface points, which are not directly lit, to receive light impinging from other surface points. As a consequence of indirect illumination, the colour of a surface point appears to be biased with the colour of light impinging from other surface points which manifests in *color bleeding*. Depending on the curvature of specular or transmissive surfaces, reflected or refracted light is focused and forms *caustics* on diffuse surfaces. At the surface of objects made of translucent materials, light enters the object, scatters below the surface and leaves the object at a different surface point. As a consequence of *subsurface scattering*, objects exhibit a distinct natural appearance according to a certain material (e.g. marble, human skin, etc.). The light directly reaching a surface point is attenuated accordingly to the fraction of the area of extended light sources that is visible from the surface point which introduces *soft shadows*. Of course, additional effects appear with the presence of participating media (e.g. light shafts, volumetric caustics).

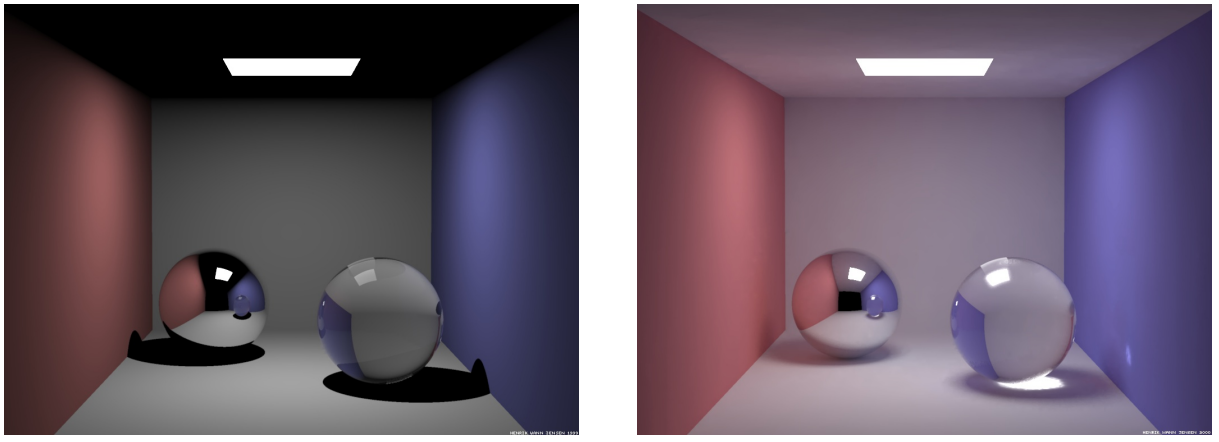


Figure 3.1: Comparison of local illumination and global illumination. A specular (left) and a transmissive sphere (right) are illuminated by an area light source directly from above. Left: Local illumination only evaluates illumination based on information available locally at a surface point. Note that the ceiling receives no light and important visual clues are missing (soft shadows, caustics). Right: Global illumination computation simulates light transport to incorporate all possible interactions of light and matter. Note that shadows are soft and the presence of caustics under the right sphere and on the blue wall. Furthermore, note that the ceiling and the shadowed regions receive additional light due to indirect illumination. Therefore, the neutrally coloured ceiling and the shadows appear to be slightly tinted with the colour of nearby coloured walls (images courtesy of *Henrik Wann Jensen*, http://graphics.ucsd.edu/~henrik/images/global.html, last access 2012-05-07).

3.1.1 The Rendering Equation

Global illumination computation has been generalised as solving the *rendering equation* [Kaj86] (Figure 3.2). The rendering equation expresses the equilibrium distribution of light energy in a scene (hemispherical formulation). At a surface point x, the exitant radiance $L(x \to \Theta)$ into direction Θ is the sum of the self emitted radiance $L_e(x \to \Theta)$ and the reflected radiance $L_r(x \to \Theta)$ into direction Θ. The reflected radiance $L_r(x \to \Theta)$ is obtained by accumulating the incident radiance $L(x \leftarrow \Psi)$ over all incident directions Ψ on the hemisphere Ω_x. The proportion of reflected radiance of each radiance incident through a differential solid angle $d\omega_\Psi$ is obtained with respect to the irradiance distribution over the hemisphere Ω_x defined by the *bidirectional reflectance distribution function* (BRDF) $f_r(x, \Psi \to \Theta)$ and the angle θ_Ψ between the incoming direction Ψ and the surface normal N_x. Accordingly, the rendering equation is expressed as

$$\begin{aligned} L(x \to \Theta) &= L_e(x \to \Theta) + L_r(x \to \Theta) \\ &= L_e(x \to \Theta) + \int_{\Omega_x} f_r(x, \Psi \to \Theta)\, L(x \leftarrow \Psi) \cos\theta_\Psi \, d\omega_\Psi. \end{aligned} \tag{3.1}$$

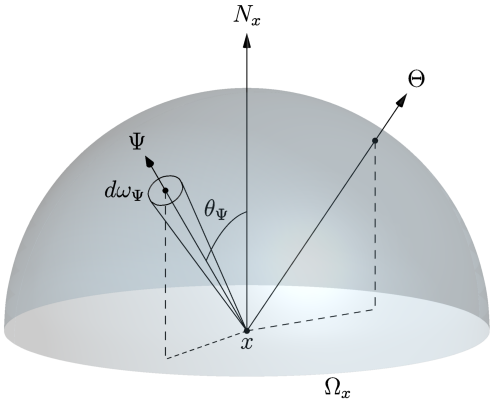


Figure 3.2: The rendering equation, hemispherical formulation. The total amount of light reflected from point x into direction Θ is obtained by summing the radiances arriving at point x over all incoming directions Ψ on the hemisphere Ω_x. For each incoming direction the radiance arriving at x is multiplied by the BRDF and the cosine term, respectively. Finally, the self emitted light at x is added only if x resides on a light source.

Of course, the unknown radiance L appears on both sides of the rendering equation. Therefore, it is particularly challenging to numerically evaluate the rendering equation for global illumination computation. In order to approximate a solution to the rendering equation for global illumination computation, the integration over directions can be reformulated as an integration over a sphere onto which all surfaces being visible at the surface point x are projected (Figure 3.3). Accordingly, integration over all surfaces A in the scene instead of integration over directions yields the area formulation of the rendering equation defined by

$$L(x \to \Theta) = L_e(x \to \Theta) + \int_A f_r(x, \Psi \to \Theta)\, L(y \to -\Psi)\, V(x,y)\, G(x,y)\, dA_y. \tag{3.2}$$

In equation 3.2,

- $L(y \to -\Psi)$ is the exitant radiance at surface point y which is equal to the radiance incident through a differential solid angle $d\omega_\Psi$ at surface point x if ignoring participating media and absorption. Hence, $L(x \leftarrow \Psi) = L(y \to -\Psi)$.
- $V(x,y)$ is the visibility function that specifies the visibility between two points x and y as

$$\forall x, y \in A : V(x,y) = \begin{cases} 0 & \text{if } x \text{ and } y \text{ are mutually visible} \\ 1 & \text{if } x \text{ and } y \text{ are not mutually visible} \end{cases}$$

- $G(x,y)$ is the geometry term which depends on the relative orientation and distance r_{xy} of the surfaces at points x and y

$$G(x,y) = \frac{\cos(N_x, \Psi)\cos(N_y, -\Psi)}{r_{xy}^2} \tag{3.3}$$

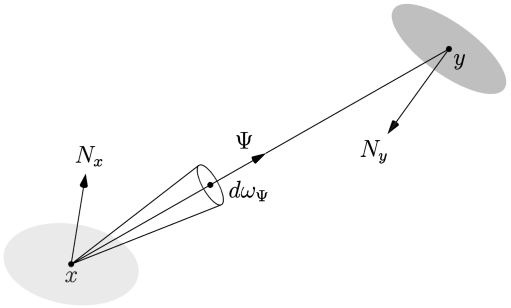


Figure 3.3: The rendering equation, area formulation. If participating media and absorption are ignored the radiance incident through a differential solid angle $d\omega_\Psi$ at surface point x corresponds to the exitant radiance at surface point y. Hence, $L(x \leftarrow \Psi) = L(y \rightarrow -\Psi) = L(y \rightarrow \overrightarrow{yx})$.

The illumination a surface point in the scene receives can be separated into direct and indirect illumination. This allows to split the rendering equation into two components. The reflected radiance $L_r(x \rightarrow \Theta)$ is the sum of reflected radiance due to direct illumination L_{direct} and indirect illumination $L_{indirect}$. The contribution of direct illumination is sampled using the area formulation of the rendering equation (Equation 3.2). The direct term includes the exitant radiance $L_e(y \rightarrow \overrightarrow{yx})$ at surface point y that is visible at surface point x along direction $\overrightarrow{yx} = -\Psi$ for all surfaces A in the scene. The contribution of indirect illumination is sampled using the hemispherical formulation of the rendering equation (Equation 3.1). The indirect term includes the reflected radiance $L_i(x \leftarrow \Psi)$ visible from all points over the hemisphere Ω_x, leading to

$$
\begin{aligned}
L(x \rightarrow \Theta) &= L_e(x \rightarrow \Theta) + L_r(x \rightarrow \Theta) \\
L_r(x \rightarrow \Theta) &= L_{direct} + L_{indirect} \\
L_{direct} &= \int_A f_r(x, \overrightarrow{xy} \rightarrow \Theta)\, L_e(y \rightarrow \overrightarrow{yx})\, V(x,y)\, G(x,y)\, dA_y \qquad (3.4) \\
L_{indirect} &= \int_{\Omega_x} f_r(x, \Psi \rightarrow \Theta)\, L_i(x \leftarrow \Psi)\, \cos\theta_\Psi\, d\omega_\Psi \qquad (3.5)
\end{aligned}
$$

3.1.2 The Bidirectional Reflectance Distribution Function

In the rendering equation the appearance of an object is modelled with the *Bidirectional Reflectance Distribution Function* (BRDF) [Nic70]. The BRDF describes the irradiance distribution over the entire sphere of directions around a surface point. Refraction at transmissive objects is described with the *Bidirectional Transmittance Distribution Function* (BTDF). In general, the description of reflection and refraction is unified with the *Bidirectional Scattering Distribution Function* (BSDF). Subsurface scattering is described with the *Bidirectional Surface Scattering Reflectance Distribution Function* (BSSRDF) where light enters surfaces at a point and leaves surfaces at another point.

The BRDF is defined at a surface point x by the ratio of the differential radiance L reflected in an exitant direction Θ and the differential irradiance E incident through a differential solid angle $d\omega_\Psi$. With the angle θ_Ψ between the surface normal N_x and the incident direction Ψ, the BRDF $f_r(x, \Psi \rightarrow \Theta)$ is defined over the entire sphere of directions around surface point x by

$$
\begin{aligned}
f_r(x, \Psi \rightarrow \Theta) &= \frac{dL(x \rightarrow \Theta)}{dE(x \leftarrow \Psi)} \\
&= \frac{dL(x \rightarrow \Theta)}{L(x \leftarrow \Psi)\cos\theta_\Psi\, d\omega_\Psi} \qquad (3.6)
\end{aligned}
$$

The BRDF exhibits several important properties. First, the BRDF is a four-dimensional positive function whose value can vary with wavelength. Second, the BRDF is anisotropic such that the value of the BRDF changes if the surface is rotated about the surface normal. Third, the BRDF exhibits HELMHOLTZ reciprocity such that the value of the BRDF remains unchanged upon swapping the incident and exitant directions. Fourth, the value of the BRDF for a specific incident direction is independent from irradiance along other incident directions. Finally, as a consequence of energy conservation the total amount of power reflected over all directions is always less than or equal to the total amount of power incident at a surface point.

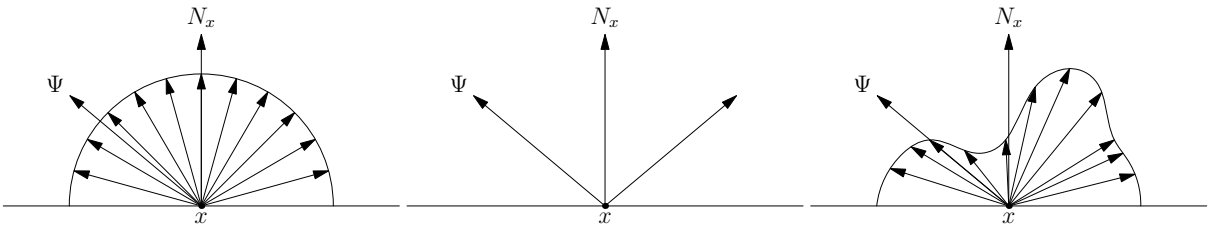


Figure 3.4: BRDF Examples. Left: Diffuse surfaces reflect light uniformly over the entire reflecting surface. Middle: Pure specular surfaces reflect light in one specific direction only. Right: Glossy surfaces exhibit a non-uniform combination of diffuse and specular reflection.

Three types of surfaces are commonly modelled with BRDFs (Figure 3.4). At diffuse surfaces the reflected radiance is independent of the exitant direction for a given irradiance distribution. Therefore, the value of the BRDF remains constant for all directions and only depends on the fraction of incident energy being reflected. At specular surfaces the BRDF is positive for one exitant direction only and zero otherwise. At transmissive surfaces the direction of specular refraction is calculated using the law of SNELL. Rays of light bend towards the surface normal upon entering a dense medium from a less dense medium. Therefore, reciprocity of the BRDF does not hold for the BSDF in general. Of course, *total internal reflection* occurs at the critical angle if light travels from a dense into a less dense medium. The equations of FRESNEL estimate the proportion of light that is being reflected and refracted at a purely specular and smooth surfaces. These equations are particularly costly to evaluate because of including the wavelength of the light, the geometry at the surface and the incident direction of light. According to glossy surfaces, modelling the nature of BRDFs with analytical formulae is hardly achievable because these surface exhibit a non-uniform combination of diffuse and specular reflection.

Several local lighting models have been suggested to simplify the evaluation of complex BRDFs. Local lighting models ignore secondary reflections and only deliver effects of direct light from direction Ψ being reflected to the eye into direction Θ at a surface point x (Figure 3.5). Additionally, the BRDF of local lighting models is isotropic in general. The models can be categorised into physically-based and empirically-based.

Physically-based models strive to model realistic BRDFs, despite the complex nature of physics of reflection. The COOK-TORRANCE model [CT81] proposes a microfacet model which incorporates shadowing effects at micro surface detail and FRESNEL terms. Additionally, the model is capable of delivering the visual effects of anisotropic surfaces (e.g. brushed metals). A more comprehensive and more expensive physically-based model has been developed [HTSG91]. The model is applicable to metallic, nonmetallic and plastic materials with smooth and rough surfaces. Furthermore, the model incorporates the wavelength, the incident angle, two surface roughness parameters and the surface refractive index to provide a smooth transition from specular to diffuse effects.

Empirically-based models attempt to visually plausibly capture a general class of surfaces efficiently for real-time rendering. Purely diffuse surfaces are modelled with the LAMBERT model [Lam60]. The BRDF of the LAMBERT model is a constant whose value only depends on the fraction of incident energy being reflected ρ_d (diffuse reflectance),

$$f_r(x, \Psi \leftrightarrow \Theta) = k_d = \frac{\rho_d}{\pi} \tag{3.7}$$

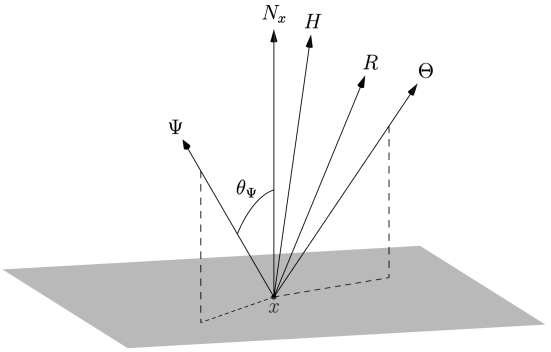


Figure 3.5: Geometry of local lighting models. Vector Ψ denotes the direction of incident light coming directly from a point light source. The angle θ_Ψ is enclosed by Ψ and the normal N_x at surface point x. Vector Θ denotes the viewing direction. Vector R denotes the reflection direction which lies in the plane spanned by Ψ and N_x. Vector H denotes the halfway vector between the light direction Ψ and the viewing direction Θ. Note that H lies in the plane spanned by Ψ and Θ.

The PHONG model [Pho75] allows for diffuse and specular reflection. As a result, *specular highlights* are introduced on curved surfaces where intense specular reflection occurs. Specularly reflected light leaves a surface into direction R with an angle of reflection between R and the normal N_x at the surface point x. This angle is approximately equal to the angle θ_Ψ between N_x and the direction Ψ of incident light. The size of specular highlights decreases as the specular exponent n increases which expresses the shininess of surfaces. Accordingly, the BRDF of the PHONG model is

$$f_r(x, \Psi \leftrightarrow \Theta) = k_s \frac{(R \cdot \Theta)^n}{N_x \cdot \Psi} + k_d \qquad \text{with } R = 2(N_x \cdot \Psi)N_x - \Psi \tag{3.8}$$

The BLINN-PHONG model [Bli77] uses the *half vector* H between the direction of incident light Ψ and the viewing direction Θ,

$$f_r(x, \Psi \leftrightarrow \Theta) = k_s \frac{(N_x \cdot H)^n}{N_x \cdot \Psi} + k_d \qquad \text{with } H = \frac{\Psi + \Theta}{\|\Psi + \Theta\|} \tag{3.9}$$

The BLINN-PHONG model has several advantages due to using the half vector H instead of the reflection vector R. First, the half vector H is cheaper to compute than the reflection vector R. Second, surfaces exhibiting narrow specular highlights if viewed at glancing angles are more appropriately and convincingly captured. Third, real materials are more closely resembled with half-vector-based BRDFs. However, the BRDF yields too bright intensities at glancing angles. As the angle θ_Ψ between Ψ and N_x approaches 90° the reflection term goes to infinity. Furthermore, the BRDF violates reciprocity and introduces visually disturbing artefacts due to an objectionable cutoff if $\theta_\Psi = 90$ [AMHH08].

Several limitations of the BLINN-PHONG model are alleviated with the *modified* BLINN-PHONG model,

$$f_r(x, \Psi \leftrightarrow \Theta) = k_s (N_x \cdot H)^n + k_d \tag{3.10}$$

In comparison to the BRDF of the BLINN-PHONG model (Equation 3.9), the reflectance term of the BRDF of the modified BLINN-PHONG model remains bounded. Furthermore, the BRDF satisfies reciprocity. Finally, the artefacts at glancing angles between Ψ and N_x are inherently avoided. However, the BRDF is not energy conserving [AMHH08].

3.1.3 The Soft Shadow Equation

Soft shadow computation determines for each surface point the fraction of the area of an extended light source that is visible from a surface point (Figure 3.6). Depending on the visibility of the light source a surface point is lit or shadowed. In shadowed regions surface points are considered to be in the *umbra* or *penumbra* according to the degree of occlusion of the light source. If the light source is fully occluded the surface point receives no light and is in the umbra. If the light source is partially occluded the surface point is in the penumbra. A reduced amount of light proportionally to the visible fraction of the area of the light source arrives at the surface point. Objects blocking the light are denoted as *occluders* (or *blockers* or *shadow casters*). Objects onto which shadows are cast are denoted as *receivers*. If an object is both occluder and receiver *self-shadowing* occurs. Although the contents of this section follows [EASW09], the notation of [DBB06] is used to maintain consistency with previous sections.

For evaluating partial illumination soft shadow computation involves direct illumination only. With the direct term of the rendering equation (Equation 3.4), the dependency of the rendering equation on itself is removed while integrating over the surface of the light source $\mathcal{L}$. Given a radiance distribution over the surface of the light source, $L_e(y \to \overrightarrow{yx})$ corresponds to the radiance emitted into the direction of a surface point x on a receiver from a point y on the surface of the light source. For simplification the self emitted radiance $L_e(x \to \Theta)$ is omitted which causes the light source to be missing in the final image. If several light sources are involved their contributions can be computed individually and simply accumulated due to the additivity of the integral. Accordingly, the *soft shadow equation* is obtained from the rendering equation with

$$L(x \to \Theta) = \int_{\mathcal{L}} f_r(x, \overrightarrow{xy} \to \Theta)\, L_e(y \to \overrightarrow{yx})\, V(x,y)\, G(x,y)\, dA_y \tag{3.11}$$

Solving the soft shadow equation is particularly challenging because in general sampling must be used if complex BRDFs or complex visibility configurations are involved.

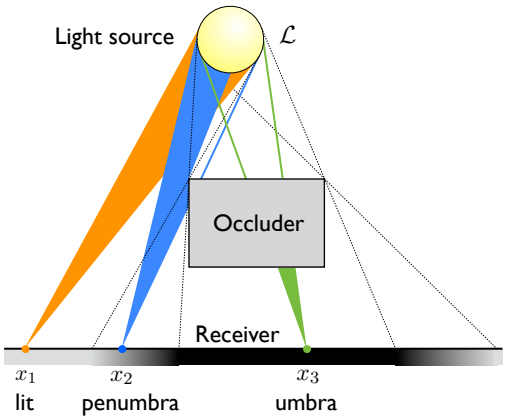


Figure 3.6: The soft shadow equation. Given a light source, an occluder and a receiver, a surface point receives direct light proportionally to the fraction of the area of an extended light source $\mathcal{L}$ that is visible from the surface point. Depending on the visibility of the light source, a surface point is lit or shadowed. Accordingly, surface point x_1 is totally lit because it is unoccluded. Surface point x_2 is partially occluded and, therefore, resides in the penumbra region. Surface point x_3 is fully occluded and, therefore, resides in the umbra region.

However, the soft shadow equation can be simplified while achieving reasonable results close to the reference. First, if all surfaces are perfectly diffuse, the BRDF becomes independent of directions such that $f_r(x, \overrightarrow{xy} \rightarrow \Theta) = \rho_d(x)/\pi$ (Equation 3.7). Second, if the distance between the light source and the receiver is relatively large with respect to the solid angle of the light source and the surface of the light source is sufficiently well-behaved, then, the geometric term $G(x,y)$ varies little. Based on the aforementioned assumptions, the soft shadow equation becomes independent of directions and can be decomposed into a shading and a shadow term, leading to

$$L(x) = \underbrace{\frac{\rho_d(x)}{\pi} \int_{\mathcal{L}} G(x,y)\, dA_y}_{\text{shading}} * \underbrace{\int_{\mathcal{L}} L_e(y \rightarrow \overrightarrow{yx})\, V(x,y)\, dA_y}_{\text{shadow}} \tag{3.12}$$

The error of the approximation (Equation 3.12) depends on the correlation between the functions of the shading and shadow term which is implicitly assumed to be low [SS98]. Furthermore, the approximation ignores that due to $G(x,y)$ the influence of the light source on the surface point x is non-uniform and, therefore, falls off with distance and orientation.

In practice, the shading term is efficiently evaluated with a local shading model to achieve real-time performance. Over the receiver the shading term is attenuated by the shadow term which is represented by the *visibility integral*. For real-time evaluation the visibility integral can be further simplified, if the light source radiates homogeneously and directionally over its surface. As a result, $L_e(y \rightarrow \overrightarrow{yx})$ simplifies to a function of position $L_c(y)$ which reduces to a constant L_c if the light source emits uniformly coloured light. Accordingly, the visibility integral simplifies to

$$L_c \int_{\mathcal{L}} V(x,y)\, dA_y \tag{3.13}$$

Of course, computing partial illumination in real-time by efficiently estimating the visibility integral (Equation 3.13) only leads to visually plausible soft shadows. Only the proportion of visibility is evaluated and not which regions on the surface of the light source are occluded. In contrast, physically plausible shadows require to correctly sample visibility for a considerable number of points on the light source which is slow. In conclusion, the results obtained with the visibility integral (Equation 3.13) are convincing, while exhibiting objectionable differences to the physically-based results obtained with the soft shadow equation (Equation 3.11).

In addition to being computationally-intense, estimating the visibility integral for soft shadow computation is non-trivial for two further reasons. First, a simple and general solution to combine the contribution of individual occluders is unavailable. This results in inaccurate *occluder fusion* (Figure 3.7). As a consequence, objectionable artefacts are introduced which manifest in overestimation of umbrae. Second, only considering objects being visible from a single point on the light source leads to incomplete shadows (Figure 3.8). Furthermore, temporal artefacts are possibly introduced due to temporal incoherence if shadows change abruptly between incomplete and complete.

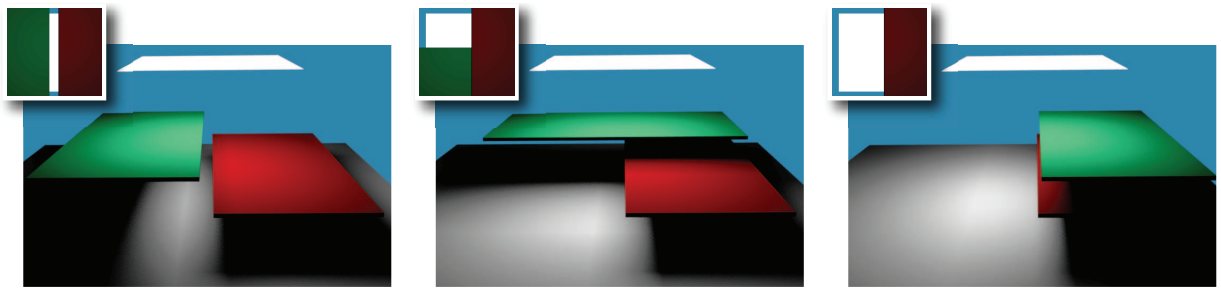


Figure 3.7: Occluder fusion. The figure shows three different cases of the fusion of two planar and parallel occluders. The scene is illuminated by a planar and parallel area light source from above. The small images indicate the fraction of the unoccluded area of the light source that is visible from a surface point under the centre of the light source. Left: The individual contribution of the occluders should be summed because the occluders do not overlap. Middle: The individual contributions of the occluders should be multiplied because the occluders partially overlap. Right: The maximum of both shadow contributions should be taken because one occluder is completely obscured by the other (image courtesy of [EASW09]).

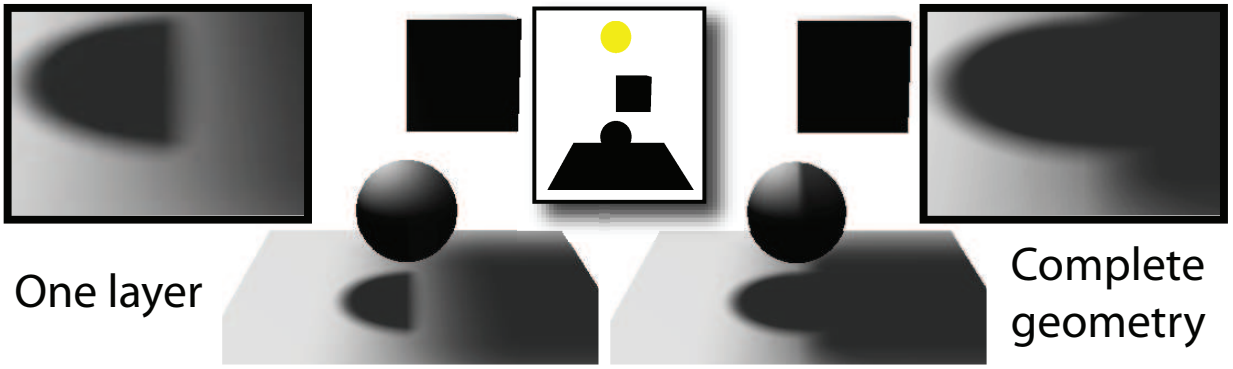


Figure 3.8: Single-sample soft shadows. The small image in the middle illustrates the scene setup consisting of a light source, a cubical occluder, a spherical occluder and a large planar receiver. Left: Incomplete shadows are computed upon using the depth of front-facing geometry only that is visible from a single point in the centre of the light source. The shadow cast by the cube incorrectly exhibits large penumbrae because the top face of the cube is incorporated only. As a result, the shadow cast by the cube onto the right hemisphere of the sphere is hardly objectionable. The shadow cast by the cube onto the plane incorrectly appears too soft. Furthermore, only the left hemisphere of the sphere is incorporated. The right hemisphere is invisible from the centre of light source. Therefore, the shadow cast by the sphere artificially appears to be cut off. This is amplified by the large penumbrae of the shadow cast by the cube. Right: The accurate shadow solution samples more layers of depth to completely capture the occluding geometry. The right hemisphere of the sphere is correctly occluded. The shadows on the plane correctly unify (image courtesy of [EASW09]).

An accurate but costly method to obtain an estimate of the visibility integral is sampling. The surface of an area or volumetric light source is covered with a distribution of numerous point lights at position y_i and the resulting hard shadows are accumulated accordingly [HH97]. With a sufficiently large number of point lights k the visibility integral is well approximated by

$$\int_{\mathcal{L}} V(x,y)\,dA_y \approx \frac{1}{k+1}\sum_{i=0}^{k} V(x,y_i) \tag{3.14}$$

An efficient but generally only approximate approach to estimate the visibility integral is based on convolution. If the area light source, occluder and receiver are planar and parallel, the visibility integral can be exactly calculated by projecting the occluder onto the surface of the light source and measuring the remaining unoccluded area of the light [SS98]. As a consequence of the fact that the projection of the occluder only translates as the surface point x moves on the receiver, the unoccluded area of the light source can be expressed as a convolution between the light source and the occluder images. With the distance d_1 between the light source and the occluder, the distance d_2 between the occluder and the receiver and the characteristic function $P(x)$, the visibility integral is expressible as a convolution by

$$\int_{\mathcal{L}} V(x,y)\,dA_y = \int_{\mathcal{L}} P\left(\frac{d_1 x + d_2 y}{d_1 + d_2}\right) dA_y \quad \text{with } P(x) = \begin{cases} 0 & \text{if } x \text{ is on the occluder} \\ 1 & \text{elsewhere} \end{cases} \tag{3.15}$$

3.2 Shadow Mapping

An efficient technique to generate hard shadows cast by spot lights is *Shadow Mapping* [Wil78]. Shadowed geometry is invisible to the light source. A light visibility test over the view volume of the light source can be efficiently generated and stored in a depth texture. This texture is used to evaluate light visibility upon rendering the view of the eye to generate shadows. Therefore, the complexity of shadow mapping is comparable to standard scene rendering. As a consequence of its ease of use and low complexity, shadow mapping scales well with scene complexity. Furthermore, shadow mapping is orthogonal to many existing image-based techniques. Moreover, shadow mapping is widely supported on different hardware platforms. This facilitates to implement the algorithm highly efficiently. In particular, shadow mapping does not have to treat occluders and receivers separately. As a consequence, arbitrary configurations of occluders and receivers are supported; including self-shadowing. In addition, arbitrary input data can be processed as long as depth values can be derived from the data.

Shadow mapping requires two passes to perform the shadow computation (Figure 3.9). First, the *shadow map* is acquired from the view of the light source $\mathcal{L}$. The shadow map is a depth image essentially. Each pixel stores the distance between the light source and the closest surface point that is hit along a ray of light passing through the centre of the pixel. Second, upon rendering the view of the eye $\mathcal{E}$ the shadow map is queried with GPU-enabled texturing techniques to efficiently resolve visibility. For each rasterised fragment, its *eye-space* coordinates (x^e, y^e, z^e) are reprojected into *light-space* (x^s, y^s, z^s). The position (x^s, y^s) on the image plane of the light source identifies the corresponding depth entry in the shadow map. After performing the *occlusion test*, a fragment is in shadow if the light-space depth of the fragment z^s is larger than the depth stored in the shadow map. Otherwise the fragment is lit. The linear transformation M required to reproject eye-space fragments to light-space is defined by

$$M : (x^e, y^e, z^e) \mapsto (x^s, y^s, z^s) \text{ with } M = B P_{\mathcal{L}} V_{\mathcal{L}} V_{\mathcal{E}}^{-1} \tag{3.16}$$

In equation 3.16,

- $V_{\mathcal{E}}^{-1}$ is the inverse viewing transformation matrix of the eye which transforms from eye-space into world-space
- $V_{\mathcal{L}}$ is the viewing transformation matrix of the light source which transforms from world-space into light-view-space
- $P_{\mathcal{L}}$ is the projection transformation matrix of the light source which transforms from light-view space into light-clip-space
- B is a bias matrix to remap from normalised device coordinates $[-1,+1]$ after perspective division to texture coordinates $[0,1]$ in light-space.

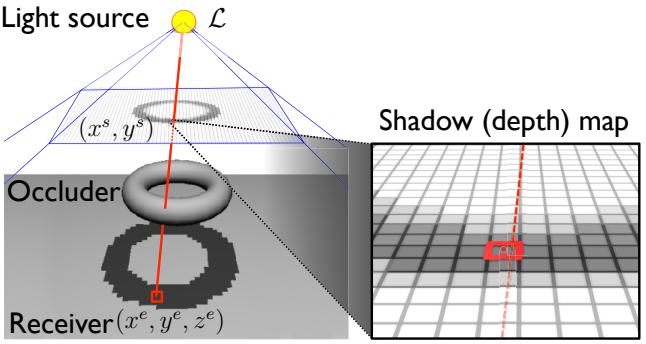


Figure 3.9: Shadow mapping. Shadows cast by a spot light source can be computed by considering geometry invisible to the light source in shadow. First, a depth image is generated from the view of the light source $\mathcal{L}$. Second, upon rendering the view of the eye each eye-space fragment (x^e, y^e, z^e) is reprojected into light-space to locate the corresponding depth entry in the shadow map at position (x^s, y^s) on the image plane of the light source. Accordingly, the fragment is determined to be in shadow if the light-space depth of the fragment z^s is greater than the depth in the shadow map. Otherwise the fragment is lit (image adapted from [EASW09]).

Several issues are inherent to shadow mapping. First, imprecisions arise from the image-based nature of shadow mapping. In general, a sampling mismatch occurs between the sampling of the shadow map and the sampling of fragments projected into the shadow map which introduces aliasing artefacts (Section 3.2.1). Accordingly, if the view volumes of the light and the eye have similar location and orientation the sampling rates almost match. In contrast, in the case of *duelling frusta*, where the light and the eye are looking toward each other from opposite locations, the sampling rates mostly differ close to the image plane of the eye. Second, incorrect self-shadowing is caused as a consequence of imprecisions (Section 3.2.2). Third, point lights require to cover a spherical view with several shadow maps to generate omnidirectional shadows (Section 3.3). Fourth, the reprojection from eye-space to light-space with matrix M causes a dual projection (Equation 3.16). Therefore, secondary shadow images are introduced along the negative view direction of the light (Section 3.3).

3.2.1 Aliasing

Aliasing appearing with shadow mapping can be categorised into spatial and temporal aliasing. Spatial aliasing manifests in blocky shadows. These artefacts result from jagged shadow boundaries due to undersampling, oversampling and reconstruction errors. Temporal aliasing manifests in flickering shadows if the rasterisation of shadows changes abruptly between consecutive frames.

Undersampling occurs if depth has been sampled coarsely with respect to the screen resolution. This occurs when the sampling rate used to create the shadow map is lower than the rate at which the shadow map is sampled upon generating the shadows. In this context, a single shadow map texel is projected to several screen pixels. According to signal processing, alias-free reconstruction of a signal requires to sample the initial signal with a rate sufficiently high (NYQUIST rate). If the sampling rate is lower than the NYQUIST rate undersampling leads to inadequate reconstruction of the initial signal. Shadow mapping inherently applies a box filter to reconstruct a function from previously sampled depth values to resolve visibility (nearest neighbour reconstruction). As a result, shadows have jagged boundaries because the reconstructed function is a noncontinuous stair case. However, improved reconstruction filters are inapplicable. The shadow map stores a discrete representation of depth values which is inapplicable to filtering. Accordingly, the occlusion test returns invalid results if neighbouring depth values are simply averaged. Additionally, undersampling is caused if the shadow map is magnified due to two types of aliasing: projection aliasing and perspective aliasing (Figure 3.10).

Projection aliasing is a local artefact. As a consequence of oblique light, shadows are strongly expanded along surfaces. Therefore, a surface which is almost parallel to the direction of light requires very fine sampling, otherwise undersampling occurs. In order to reduce projection aliasing the sampling rate must be varied depending on local surface orientations. Adaptive algorithms employ hierarchical acceleration data structures to analytically adjust the sampling rate (Section 2.1.1.5).

Perspective aliasing is a global artefact. As a consequence of perspective projection, the shadow map is enlarged close to the viewer which causes undersampling. However, setting up a different parameterisation for the shadow map allows to adjust the distribution of samples with warping (Section 2.1.1.3). Hence, geometry close to the viewer receives more samples than distant. Additionally, subdividing the view frustum and maintaining a dedicated shadow map for each subdivision enables to further increase the number of samples with global partitioning (Section 2.1.1.4).

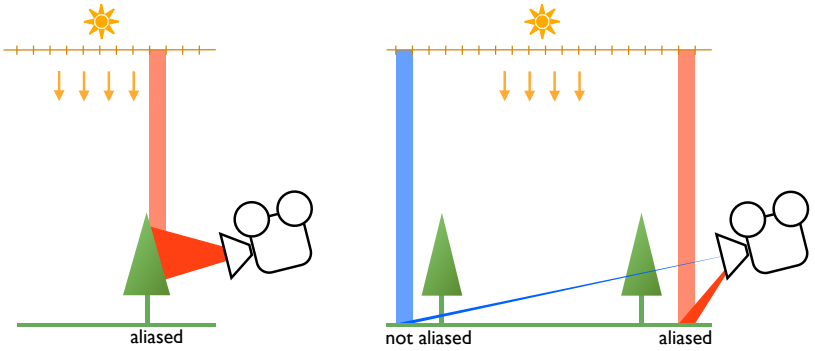


Figure 3.10: Shadow map aliasing due to undersampling. A directional light illuminates trees directly from above. Left: The steep slope of the surface of the tree causes projection aliasing. Therefore, a small region of the shadow map is projected to a large region on the view plane. Right: The shadow map has a constant resolution along the viewing direction. As a consequence, perspective aliasing occurs if the shadow map is enlarged while being projected on the view plane (image courtesy of [SWP10]).

A simplified quantification of aliasing errors in shadow mapping for a directional light source [WSP04] (Figure 3.11) is expressible as

$$\frac{dp}{ds} = \frac{z_n}{z}\frac{dz}{ds}\frac{\cos\alpha}{\cos\beta} \tag{3.17}$$

where aliasing occurs if dp/ds is larger than the ratio of shadow map and screen resolutions or if dp is larger than the width of a screen pixel. In equation 3.17,

- dp is the width of a shadow map texel projected to the view plane
- ds is the length of a side of a square representing a shadow map texel
- dp/ds is the shadow map aliasing error
- z_n is the distance from the eye to the view plane
- z is the distance from the eye to the surface
- dz is the length of a bundle of parallel light rays being comprised by a shadow map texel
- α is the angle between an eye ray and the surface normal
- β is the angle between a light ray and the surface normal.

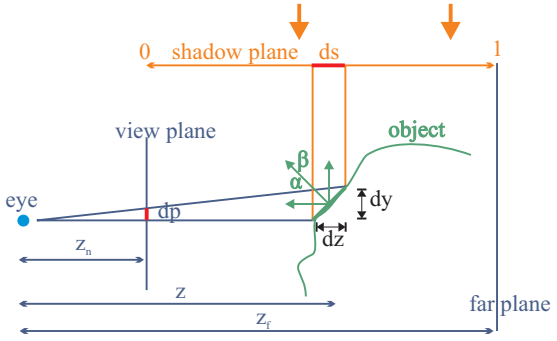


Figure 3.11: Simplified analysis of shadow map aliasing errors for a directional light source. A shadow map texel of size $ds \times ds$ comprises a bundle of parallel rays of light. The bundle has the length $dz = (z_f - z_n)\,ds$ in world-space. The rays impinge on the object on a small edge along a length of $dz/\cos\beta$. In eye-space, $dy = dz(\cos\alpha/\cos\beta)$ projects to $dp = z_n/z\,dy$ on the view plane (image courtesy of [WSP04]).

The resolution of the shadow map is limited locally and globally. Projection aliasing occurs if the term $\cos\alpha/\cos\beta$ is large. This occurs at surfaces whose normal is almost orthogonal to the direction of light. Therefore, adaptive methods increase the sampling rate for surfaces at which $\cos\alpha/\cos\beta$ is large to reduce the shadow map aliasing error dp/ds locally. Perspective aliasing occurs if the term $dz/(zds)$ is large. Upon approaching the view plane from afar $1/z$ increases and, finally, becomes very large. As a consequence, the shadow map aliasing error dp/ds becomes large close to the near plane because dz/ds is constant throughout the entire view frustum if the shadow map has a uniform parameterisation. Therefore, warping algorithms select a non-uniform shadow map parameterisation. The ratio dz/ds is varied such that more samples are taken where $1/z$ is large (i.e. near the view plane) to reduce the aliasing error dp/ds globally.

A more accurate and more general quantification [LGQ*08] (Figure 3.12) expresses the aliasing errors in shadow mapping as

$$m = \frac{r_j}{r_t}\frac{dG}{dt}\frac{W_l}{W_e}\frac{n_e}{n_l}\frac{d_l}{d_e}\frac{\cos\phi_l}{\cos\phi_e}\frac{\cos\psi_e}{\cos\psi_l} \tag{3.18}$$

where aliasing occurs if $m > 1$. In equation 3.18,

- r_j/r_t is the ratio of the screen and shadow map resolutions
- dG/dt is the derivative of the shadow map parameterisation; corresponds to dz/ds in the simplified quantification
- W_l/W_e is the ratio of the widths of the light and eye viewports in world-space
- n_e/n_l is the ratio of the distances from the light to the view plane of the light and from the eye to the view plane of the eye
- d_l/d_e is the ratio of the distances from the light to the surface and from the eye to the surface
- ϕ_l is the angle between the normal of the light view plane and the light direction
- ϕ_e is the angle between the normal of the eye view plane and the viewing direction
- ψ_l is the angle between the light direction and the normal of the surface; corresponds to α in the simplified quantification
- ψ_e is the angle between the viewing direction and the normal of the surface; corresponds to β in the simplified quantification.

The accurate quantification describes aliasing errors with respect to more general configurations of eye and light. The addition of the terms W_l/W_e, n_e/n_l and d_l/d_e allows to treat lights with arbitrary positions and directions. In comparison, the simplified quantification assumes that the surface element resides in the centre of the view frustums of the light and the eye. This requirement is dispensable with the introduction of the angles ϕ_l and ϕ_e.

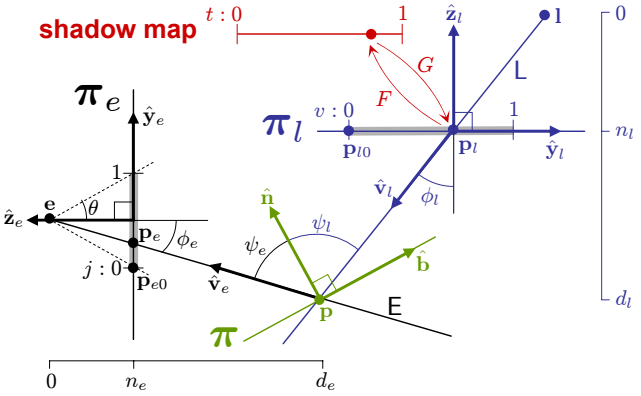


Figure 3.12: Accurate analysis of shadow map aliasing errors. The function $j(t)$ maps a point $t \in [0,1]$ in the shadow map to a point $\mathbf{p}_e$ on the image plane of the eye π_e. F and G are the two-dimensional shadow map parameterisation and its inverse. The point $\mathbf{p}_l$ on the image plane of the light π_l is projected through the light to $\mathbf{p}$ on a planar surface π in the scene. The point $\mathbf{p}$ is projected through the eye to a point $\mathbf{p}_e$ on the image plane of the eye π_e. Accordingly, the spacing between the light and eye sample locations is related to the derivatives of the function $j(t)$. This enables to accurately quantify the aliasing error (image courtesy of [LGQ*08]).

Therefore, the accurate quantification is applicable to directional and point light sources. Furthermore, varying widths of light and eye rays become addressable by using a function of ϕ_l and ϕ_e. It is important to emphasise that both aliasing error quantifications are based on two-dimensional formulations. Hence, they treat sampling errors along a single shadow map axis only. The sampling error along the orthogonal shadow map axis needs to be included for completeness.

Oversampling occurs when the shadow map has a higher resolution than the screen. In this case several shadow map texels map to a single screen pixel. According to (Equation 3.18), shadow map oversampling occurs if $m < 1$. Artefacts arise from aliasing which appears in a similar context upon sampling high frequency colour textures to improve the appearance of distant objects (e.g. tiled floor patterns). According to signal processing, aliasing due to oversampling is avoided by, first, band-limiting the signal and, then, resampling the resulting signal at larger intervals. This corresponds to blurring a colour texture before sampling the texture at lower resolution for minification. Band-limiting before resampling is impossible because filtering is inapplicable to shadow maps due to their discrete nature. However, shadow maps have been made applicable to filtering with linear functions for resolving visibility (Section 2.1.2).

Temporal aliasing occurs if the sampling rate of the shadow map changes abruptly between consecutive frames. This causes the discrete shadow boundaries of hard shadows to flicker. The change of the sampling rate is proportionally to the change of the transformation matrix M which reprojects from eye-space to light-space (Equation 3.16). Accordingly, the change of the reprojection matrix M depends on the change of

- the position of the light
- the orientation of the view frustum of the light
- the size of the view frustum of the light
- the position of the eye
- the orientation of the view frustum of the eye.

In addition, the sampling rate of the shadow map changes if the reparameterisation to mitigate perspective aliasing changes. In conclusion, temporal aliasing is generally aggravated in dynamic scenes with fitting (Section 2.1.1.2), warping (Section 2.1.1.3) and global partitioning (Section 2.1.1.4).

3.2.2 Incorrect Self-Shadowing

Incorrect self-shadowing arises from a resampling problem whose origin is twofold. First, limited numerical precision causes quantified numbers to exhibit minor deviations from their exact value. Second, limited shadow map resolution infers capturing a single depth value for the entire region a shadow map texel covers in the scene. Furthermore, sample locations between the view of the eye and the view of the light hardly coincide. As a consequence, the occlusion test is susceptible to incorrectly fail upon comparing depth values if the depth value of a lit view sample is lower than the depth value of the corresponding shadow map sample. This introduces self-shadowing artefacts in lit regions which manifest in moiré patterns (shadow acne, surface acne and z-fighting).

Incorrect self-shadowing is addressed with depth biasing which offsets depth values before comparison. Numerical precision issues are circumvented by uniformly adding a small value to all depth values in the shadow map (constant biasing). Geometrically dependent resampling issues are circumvented by slightly increasing depth values proportionally to the slope of the sampled polygon (slope-based biasing) (Figure 3.13). The depth slope of a sampled polygon is estimated efficiently with GPU-enabled partial derivative operators. However, depth biasing is heavily scene dependent and, therefore, needs to be adjusted manually and thoroughly. In general, depth biasing is incapable of alleviating incorrect-self shadowing if surfaces exhibit concavities or high curvature.

Incorrect self-shadowing seriously aggravates if the reprojection matrix M involves a perspective transformation $P_{\mathcal{L}}$ (Equation 3.16). As a consequence of perspective division, the majority of depth samples concentrates close to the near plane of the view frustum of the light source [BAS02a]. This results in a seriously non-uniform sampling distribution along the direction of light. In general, the majority of the scene maps to the first half of the range of depth. Therefore, oversampling occurs close to the near plane of the light while distant geometry is sampled coarsely. Accordingly, undersampling reaches a maximum in the case of *duelling frusta* if light and the camera are looking toward each other. In contrast, a uniform sampling distribution is achieved with a linear depth metric. Instead of the post-perspective depth, the distance to the light source is sampled. Furthermore, light clip-space depth values do not vary across surfaces being planar and parallel to the near plane of the view frustum of the light. In contrast, the distance to the light source increases upon moving over surfaces without depth slope towards the edge of the view frustum of the light source. As a result, the depth disparity between neighbouring surface points is considerably increased which, therefore, reduces incorrect self-shadowing.

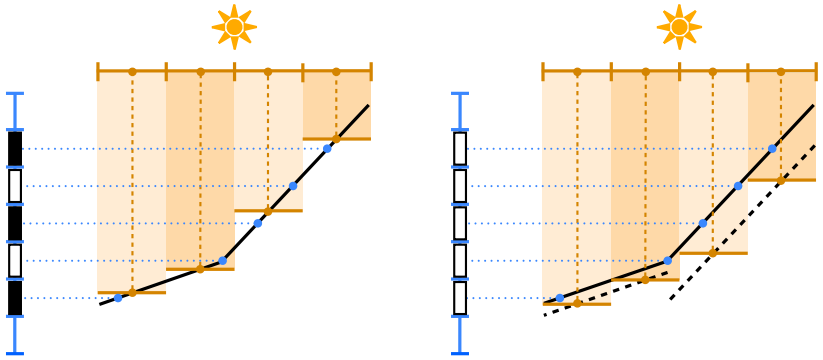


Figure 3.13: Incorrect self-shadowing arises from sampling geometry at different locations into the frame buffer and the shadow map. This necessitates to vary the magnitude of depth bias locally. Left: Alternatingly the depth test incorrectly fails for the totally lit polygon which results in disturbing moiré patterns. Right: Geometry rasterised into the shadow map is pushed away from the light source to slightly increase depth values which solves incorrect self-shadowing. Note that the depth offset of the long polygon is larger than the depth offset of the short polygon due to slope-based depth biasing (image adapted from [SWP10]).

Incorrect self-shadowing is considerably improved with *Second-Depth Shadow Maps* (SDSM) [WM94] which store the depth of back-facing surfaces with respect to the direction of light. Therefore, the method is only applicable to closed occluders which exhibit a manifold face topology where each triangle edge has one and only one neighbouring triangle. During shadow map generation the faces on the frontside of objects are culled to capture the depth of the second-nearest surfaces. Accordingly, on front-facing surfaces the reliability of the occlusion test increases if depth values of front- and back-facing surfaces differ sufficiently upon being compared. Of course, incorrect self-shadowing reappears on back-facing surfaces. This is circumvented by selectively applying the occlusion test to front-facing polygons only. Back-facing polygons are simply shaded to be shadowed accordingly. Additionally, SDSM significantly reduce the creational cost of the shadow map if sufficiently less geometry is rasterised into the depth image. However, the reliability of the depth test depends on the thickness of occluders. Depth values of front-facing surfaces are essentially biased proportionally to the thickness of occluders. This can cause imprecisions if occluders are exceptionally thin or thick.

Incorrect self-shadowing is considerably aggravated locally and globally if the shadow map is magnified due to projection aliasing, perspective aliasing or *Percentage-Closer Filtering* (PCF) [RSC87] with large filter kernels. Therefore, a large magnitude of depth bias may be necessary to eliminate incorrect self-shadowing on polygons spanning large depth ranges. As a result, contact shadows exhibit large gaps between occluders and receivers. This causes occluders to incorrectly appear to be floating and detached from receivers (peter panning). If occluders are closed, the necessary magnitude of depth bias and, therefore, peter panning is effectively reduced with SDSM. In particular, it has to be noted that, PCF requires the magnitude of depth bias to be proportional to the depth range over the entire filter window. However, concerning PCSS, if the size of the PCF window varies for each screen pixel, choosing an appropriate magnitude of depth bias for all window sizes is impossible (Section 2.2.1).

In order to enable effective reduction of incorrect self-shadowing with depth biasing, objects should exhibit a sufficiently high depth disparity. Furthermore, objects should be correctly oriented with respect to the direction of light. Hard edges easily introduce artefacts because the depth disparity is exceptionally small nearby the tip of the edge. As a consequence, even small magnitudes of depth bias cause objectionable shadow gaps. Accordingly, thin objects should be closed even if the back-facing geometry is invisible to the viewer. According to the direction of light, the outside of an object should be turned away from the light and the inside of an object should be turned towards the light.

3.3 Hard Shadows

Hard shadows appear with three types of light sources: directional lights, point lights and spot lights. The visibility integral simplifies to a binary visibility query $V(x,y)$ between a surface point x and the light source at position y (Equation 3.13). The visibility query is efficiently performed with shadow mapping to allow for real-time performance. The type of the light source determines which type of projection transformation is being applied upon reprojecting from eye-space to light-space (Equation 3.16). A directional light requires an orthographic projection. The rays of light are parallel because a directional light source is considered to be infinitely far away from a surface point. A point light requires a perspective transformation because the rays of light converge at a single point (Figure 3.14). A spot light is essentially a point light whose rays of light are confined to emanate within a cone only. Of course, point lights turn into directional lights in the limit if point lights are moved sufficiently far away from a surface.

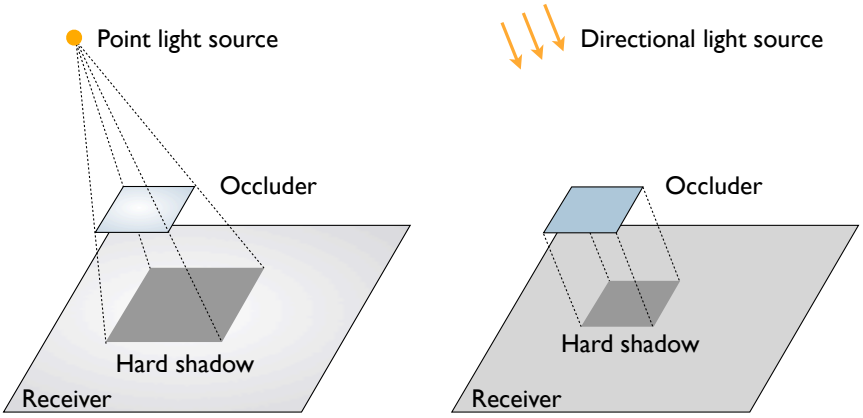


Figure 3.14: Hard shadows. Left: A hard shadow caused by a point light source. The rays of light converge at a single point being located at the position of the light source. Accordingly, as the distances between light source, occluder and receiver change, the size of the hard shadow on the receiver changes. Furthermore, planar surfaces exhibit a non-uniform distribution of light intensity across the surface because the direction of incident light varies across the surface. Right: A hard shadow caused by a directional light source being infinitely far away which causes rays of light to be parallel. Accordingly, as the distances between occluder and receiver change, the size of the hard shadow on the receiver remains unchanged. Additionally, the direction of incident light remains unchanged between different surface points. Therefore, planar surfaces exhibit a uniform distribution of light intensity across the surface.

A point light emanates light equally in all directions and, therefore, casts omnidirectional shadows. However, shadow mapping is inherently applicable to directional lights and spot lights only whose view can be captured with a single depth image. Therefore, the spherical view of a point light is covered by using six depth images to form a cube shadow map [Ger04]. This is efficiently supported by contemporary GPUs. Each depth image is acquired with a perspective projection having a field of view of $90°$. The viewing transformation is setup such that the apex of each view frustum resides at the position of the light source. The viewing direction of each view frustum is oriented towards the centre of the according cube map face. Of course, multiple passes are required to generate a cube shadow map. This infers to rasterise geometry repeatedly and redundantly. However, decomposing geometry into six unique subsets with respect to the six cube map faces is non-trivial and, therefore, impractical for real-time rendering.

Hard shadows are introduced as sharp transitions between shadowed and lit regions while the shadow term $V(x,y)$ is binarily modulating the shading term over receivers (Equation 3.12). The shading term is efficiently and visually plausibly evaluated in real-time with the modified BLINN-PHONG illumination model (Equation 3.10). The PHONG illumination model applies to point lights and involves diffuse and specular reflection. Diffusely reflected light is reflected uniformly in all directions and models the appearance of non-shiny surfaces. Specularly reflected light is reflected into a single direction only and models the appearance of polished or glossy surfaces with specular highlights. Accordingly, the PHONG illumination model distinguishes light incident from a point light source as being diffusely reflected I_d and specularly reflected I_s. In addition, light impinging uniformly from all directions due to indirect illumination is treated as ambient light I_a. The PHONG illumination model describes the interaction of light and matter with material properties for each surface point x. The diffuse reflectivity coefficient ρ_d determines the proportion of diffusely reflected light. The specular reflectivity coefficient ρ_s determines the proportion of specularly reflected light. The specular exponent n determines the narrowness of the spread of specularly reflected light which conveys the shininess of the surface with highlights of according size and smoothness. The ambient reflectivity coefficient ρ_a determines the proportion of reflected ambient light. Finally, the emissivity of a surface determines the amount of light I_e that is emitted at a surface point upon disregarding any other illumination. Of course, the PHONG illumination model is a local illumination model and captures indirect illumination with an ambient term only. Therefore, the emitted light I_e only biases the illumination locally at a surface point without affecting the illumination at other surface points.

With the superposition principle, the PHONG illumination model becomes applicable to multiple light sources and multiple colours of light, respectively. As a consequence of the linear nature of light transport, the light intensities are calculated independently and accumulated for each light source and for each of the three colour components of light (red, green, blue). Therefore, light intensities $\mathbf{I}$ and material properties ρ have three components with respect to the three wavelengths of light. Accordingly, with $\otimes$ denoting component-wise vector multiplication, the total illumination $\mathbf{I}$ including occlusion at a surface point x with respect to k point lights at positions y_i is defined by

$$\mathbf{I} = \rho_a \otimes \mathbf{I}_a + \rho_d \otimes \sum_{i=0}^{k} V(x,y_i)\,\mathbf{I}_{d,i}\,(N_x \cdot \Psi_i) + \rho_s \otimes \sum_{i=0}^{k} V(x,y_i)\,\mathbf{I}_{s,i}\,(N_x \cdot H_i)^n + \mathbf{I}_e \tag{3.19}$$

where $\Psi_i = \overrightarrow{xy_i}/\|\overrightarrow{xy_i}\|$ and $H_i = (\Psi_i + \Theta)/\|\Psi_i + \Theta\|$ for a given viewing direction Θ (Figure 3.5). In equation 3.19, $\mathbf{I}_d$ is the incident intensity of light being diffusely reflected and measured as energy flux through the unit area being perpendicular to the direction of light Ψ. Therefore, the energy flux incident along direction Ψ through the unit area being perpendicular to the surface normal N_x is $\mathbf{I}_d\,(N_x \cdot \Psi)$. Furthermore, the incident light intensities $\mathbf{I}_d$ and $\mathbf{I}_s$ are taken to be zero if the light is not shining from above the surface, that is if $N_x \cdot \Psi \leq 0$.

According to a spot light, the incident light intensities $\mathbf{I}_d$ and $\mathbf{I}_s$ are taken to be zero if the surface point is outside the cone of illumination, that is if $\Psi_{spot} \cdot \Psi < \cos(\alpha_{cutoff})$. The cone is defined by the spot direction Ψ_{spot} and the spot cutoff angle α_{cutoff} between Ψ_{spot} and an edge of the lateral surface of the cone. In addition, the light intensities $\mathbf{I}_a$, $\mathbf{I}_d$ and $\mathbf{I}_s$ at a surface point x inside the cone of illumination are moderated with the spot attenuation factor $(\Psi_{spot} \cdot \Psi)^c$ where c is the spot light exponent. Therefore, the spot attenuation increases towards the perimeter of the base of the cone of illumination.

According to a positional light (point light and spot light), the light intensities $\mathbf{I}_a$, $\mathbf{I}_d$ and $\mathbf{I}_s$ are moderated with the distance attenuation factor $1/(k_c + k_l d + k_q d^2)$ where $d = \|\overrightarrow{xy}\|$ is the distance to the light and k_c, k_l and k_q are the constant, the linear and the quadratic attenuation factors of the light, respectively. The geometric term $G(x,y)$ involves the squared distance to the surface point x to the light at position y (Equation 3.3). At sufficiently large distances of the light source from the surface, the attenuation changes in a quadratic way. In contrast, upon decreasing the distance the attenuation decreases in a linear way. This changing behaviour is more generally reflected by the polynomial in the distance attenuation factor.

According to a directional light, the computation of $\mathbf{I}$ considerably simplifies for two reasons. First, the halfway vector H can be pre-computed because the rays of light are parallel. Therefore, the direction of light Ψ does not vary between different surface points. As a consequence, planar surfaces exhibit a uniform distribution of intensity because the diffuse and specular terms do not vary across the surface. Furthermore, the attenuation of light with respect to the distance between the light source and the surface point is disregarded. A directional light is assumed to be infinitely far away.

According to projection aliasing, the PHONG illumination model implicitly reduces related artefacts. The intensity of diffusely reflected light $\mathbf{I}_d\,(N_x \cdot \Psi)$ is very small if the surface normal N_x is almost parallel to the direction of light Ψ. The intensity of specularly reflected light $\mathbf{I}_s\,(N_x \cdot H)^n$ is only large if projection aliasing is hardly appearing. Therefore, regions which are susceptible to projection aliasing are sufficiently darkened to hide related artefacts.

Shadow mapping is confined to the view volume of the light for two reasons. First, secondary shadow images are eliminated. As a consequence of a dual projection upon reprojecting from eye-space to light-space, inverted shadow images appear along the negative view direction of the light (Equation 3.16). Second, performance improves considerably, if the view volume is setup tightly (Section 3.5). Upon generating the shadow map less geometry needs to be rasterised. Irrelevant shadow map queries for testing occlusion of fragments outside the view volume of the light are avoided.

3.4 Soft Shadows

Soft shadows appear with light sources which exhibit spatial extent. A surface point only receives a proportion of illumination according to the partial view of the surface of the extended light source. The light source is totally occluded in umbra regions. The light source is partially occluded in penumbra regions which introduce smooth transitions of partial illumination between totally lit and umbra regions (Figure 3.15). As the size of a light source increases the size of the penumbra of a shadow increases. Accordingly the size of the umbra decreases. If the light is sufficiently large the umbra disappears. Furthermore, decreasing the distance between occluder and receiver while keeping the light fixed decreases the size of the penumbra. If the distance between occluder and receiver is sufficiently small the penumbra almost disappears. As a consequence, soft shadows exhibit *contact hardening* if the occluder touches the receiver. The size of the penumbra decreases towards the junction of occluder and receiver. Accordingly, the size of the umbra increases. The penumbra finally disappears in the limit upon approaching the junction. Contact hardening is indispensable for realistic appearance of soft shadows. Important information is expressed about the spatial relation of objects, the size of lights, the curvature of the surface of receivers and the silhouette of occluders.

Soft shadow computation obtains an estimate of the visibility integral by determining the fraction of the surface of an extended light source that is visible from a surface point. The obtained visibility factor is used to modulate the shading term accordingly (Equation 3.13). Visually plausible soft shadows including contact hardening are efficiently generated in real-time by estimating the visibility integral based upon convolution (Equation 3.15). *Percentage-Closer Soft Shadows* (PCSS) [Fer05] vary the size of a PCF kernel per screen pixel based on the distance between occluder and receiver. Visually plausible soft shadows exhibiting contact hardening are generated. Only a single shadow map sampled from the centre of the extended light source is required. Physically plausible soft shadows are generated by sampling the visibility integral which is computationally intense (Equation 3.14). Several spot lights are placed non-uniformly on the surface of the extended light source. At each surface point the visibility of each spot light is combined to obtain an estimate of the partial view of the light source. In conclusion, offering two approaches for soft shadow computation enables to choose the appropriate approach depending on different requirements and constraints (quality, performance, geometry detail, etc.).

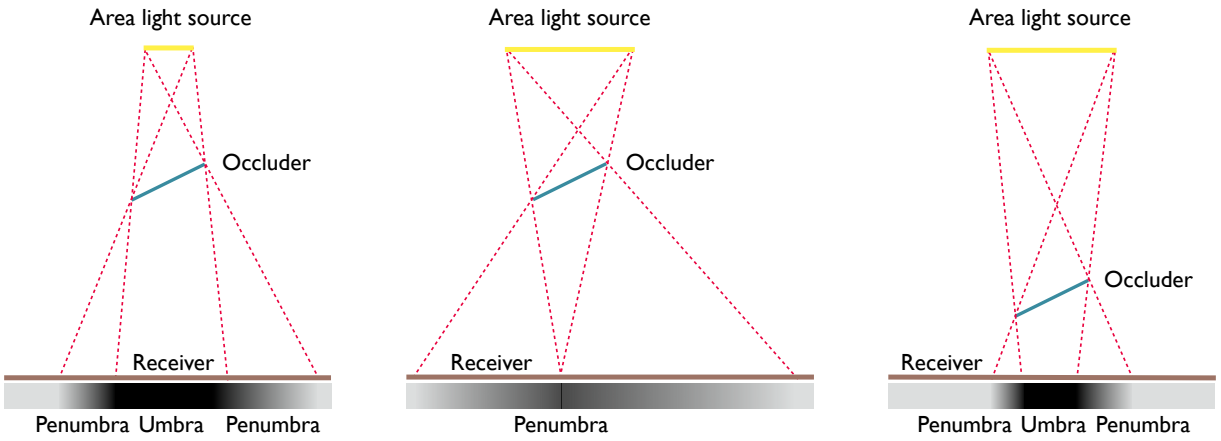


Figure 3.15: Soft shadows. Soft shadows are introduced due to the partial view of an extended light source. The size of the penumbra depends on the size of the light source and on the distance between the light source, the occluder and the receiver. Left: The area light source introduces regions of penumbra which are smooth transitions between lit and umbra regions (fully occluded). Middle: As the size of the light increases, the size of the penumbra increases and the umbra disappears. Right: Moving the occluder closer to the receiver while keeping the light source fixed decreases the size of the penumbra and reintroduces the umbra.

3.4.1 Percentage-Closer Soft Shadows

Percentage-Closer Soft Shadows (PCSS) [Fer05] provide an estimate of the proportion of shadowing at a surface point which is used to modulate the shading term. The visibility factor is obtained by filtering a shadow map being sampled from the centre of an extended light source. The filter size is varied with respect to an estimate of the penumbra width involving the distances between light source, occluder and receiver. The approach is based on the observation that accurate soft shadows are obtained with convolution if light source, occluder and receiver are assumed to be planar and parallel (Equation 3.15). In general, upon deviating from this assumption, acceptable results are often achieved if the size of the filter is varied accordingly to generate penumbrae of varying widths and, therefore, shadows which exhibit contact hardening. However, occluder fusion is handled incorrectly and shadows are considerably underestimated due to utilising a single shadow map for sampling the occluder representation from the centre of an extended light source. In addition, the performance of PCSS is limited by the cost of the filter stage necessary to achieve smooth transitions of penumbrae. Therefore, PCSS is only applicable to small lights or scene configurations which avoid excessively stretched penumbrae along steep receivers with respect to the direction of light.

PCSS estimate the width of a PCF window for each screen sample using a blocker search to mimic varying penumbra widths in real-time. Accordingly, PCSS involves three steps: blocker search, penumbra width estimation and PCF (Figure 3.16). First, a blocker search obtains the average depth of potential blockers. The region of the shadow map to search within $\mathcal{R}_s$ results from intersecting the shadow map near plane with the pyramid formed by the receiver point p (apex) and the area light source (bottom). Accordingly, with the depth z_n of the near plane of the view volume of the light source and the assumption of a single planar receiver at the depth z_r being parallel to a planar light source of width w_{light}, the width of the shadow map region $\mathcal{R}_s$ to search for potential occluders is obtained using the intercept theorem by

$$\mathcal{R}_{s,width} = \frac{z_r - z_n}{z_r} w_{light} \tag{3.20}$$

Within the search region, the depths of all samples which are closer to the light source than the receiver point p are averaged to obtain the blocker depth z_b. Accordingly, if all depths within the search region are larger than or equal to the depth z_r of the receiver point p, the receiver point p is assumed to be totally lit and the subsequent stages of PCSS are skipped (early out). Otherwise, with the assumption of a single planar occluder at the average depth z_b and a single planar receiver at depth z_r both being parallel to a planar light source of width w_{light}, the width of the penumbra $w_{penumbra}$ is estimated using the intercept theorem by

$$w_{penumbra} = \frac{z_r - z_b}{z_b} w_{light} \tag{3.21}$$

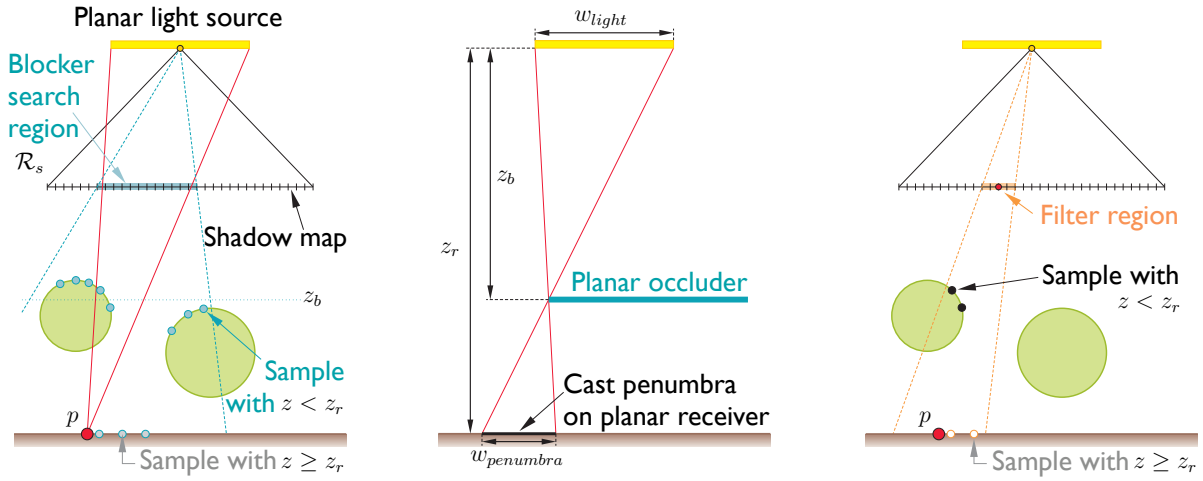


Figure 3.16: Percentage-Closer Soft Shadows. The three stages of PCSS: blocker search, penumbra width estimation and filtering the shadow map. Left: First, the shadow map is processed within the region $\mathcal{R}_s$ to average depths which are closer to the light than the receiver point p. Middle: Then, assuming a single planar occluder at the average depth z_b and a single planar receiver at depth z_r both being parallel to a planar light source of width w_{light}, the penumbra width $w_{penumbra}$ is estimated using the intercept theorem. Right: Finally, PCF is applied over a filter region whose width is derived from the penumbra width (image adapted from [EASW09]).

The penumbra width estimation ensures that the size of the penumbra changes accordingly as the distance between light source, occluder and receiver changes. Therefore, shadows exhibit contact hardening. Finally, the shadow map is filtered with PCF using a window width w_{filter} which is determined from the estimated penumbra width $w_{penumbra}$, the depth z_r of the receiver point and the depth z_n of the near plane of the view volume of the light source by

$$w_{filter} = \frac{w_{penumbra}}{z_r} z_n \tag{3.22}$$

Wrong shadows easily appear with PCSS because shadows obtained with PCF are physically incorrect. In order to correctly calculate shadows, rays are cast from the surface to different locations on the extended light source to determine the fraction of the area of the light source that is visible. In contrast, PCF evaluates visibility sampled with rays which are cast from the light source through the texels of the shadow map. Therefore, PCF averages shadow contributions which are physically dissociated. Furthermore, comparing a single reference depth value to depths across a totally lit and steep surface easily leads to incorrect self-shadowing if some depth samples on the surface are considered to be closer to the light than the reference depth. As a result, a large magnitude of depth bias is required. This introduces objectionable peter panning at contact shadows. According to completely avoiding incorrect self-shadowing, the magnitude of bias must be proportional to the depth range over the entire filter window. As a consequence of dynamically estimating the filter window size at each receiver point, choosing a single magnitude of depth bias which applies for all filter sizes is impossible [Lau07]. The magnitude of depth bias necessary is reduced with a *Second-Depth Shadow Map* [WM94] if the geometry is closed. However, depth biasing is still required at surfaces which exhibit insufficient depth disparity.

Two noticeable consequences stem from using the distance to the light source as a linear depth metric in order to reduce incorrect self-shadowing. First, on planar surfaces parallel to the light source the blocker search region increases towards the edge of the view frustum of the light. In the limit, the search region is bounded by the size of the light source. Second, assuming a single planar blocker and a single planar receiver both being parallel to the light source, the penumbra width remains constant towards the edge of the view frustum of the light due to similar triangles.

As a consequence of approximating an occluder representation, which generally comprises multiple occluders at different depths, by simply averaging depths, PCSS easily includes depth samples which do not contribute to occlusion (Figure 3.17). Occluders are identified based on if their depth is closer to the light source than a single reference depth while ignoring their true blocking contribution of the area of the light source. Therefore, occluders are easily identified incorrectly if shadow map samples are included which are outside the light-point pyramid. This possibly leads to incorrect estimates of the average blocker depth. As a result, penumbrae are too large due to overestimated PCF window sizes.

The applicability of PCSS is considerably limited to lights of small size for three reasons. First, for generating smooth penumbrae the number of samples of the PCF kernel must be increased accordingly while increasing the size of the filter window to avoid banding artefacts caused by abruptly varying degrees of shadowing. Furthermore, in the blocker search stage taking too few samples within a large search region yields largely varying average blocker depths at neighbouring receiver points. This introduces similar artefacts due to largely differing PCF window sizes. As a consequence, many

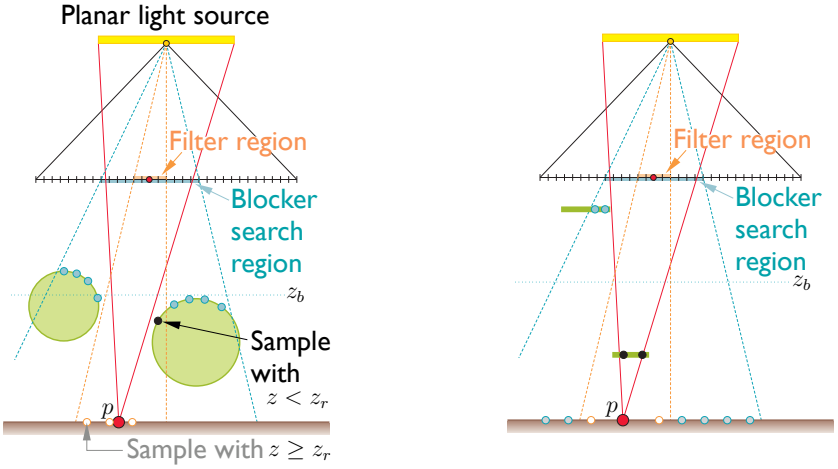


Figure 3.17: Errors of Percentage-Closer Soft Shadows. Left: In the blocker search step, the depths of the samples on the spheres are averaged to incorrectly identify an occluder at depth z_b. In the filtering step, PCF of the three non-blocking red samples on the receiver and one blocking black sample on the right sphere incorrectly yields partial illumination at the totally lit receiver point p. Right: PCF of the two blocking black samples on the occluder and the two non-blocking red samples on the receiver incorrectly yields partial illumination at the fully occluded receiver point p (image adapted from [EASW09]).

shadow map accesses may be required which considerably limits performance. Second, to avoid incorrect self-shadowing with large PCF window sizes, a large magnitude of depth bias is necessary which considerably exacerbates peter panning. Third, light bleeding is introduced and penumbrae are overestimated. Errors that inherently appear with PCSS are aggravated with large blocker search regions and large filter window sizes, respectively (Figure 3.17).

The performance and shadow quality of PCSS is considerably improved with stratified sampling [Ura05, Isi06]. The performance of PCSS depends on the number of shadow map accesses in both the blocker search stage and the PCF stage. Of course, sub-sampling introduces banding artefacts. Monte Carlo methods place samples randomly to obtain an estimate of the expected result of a function over a complicated domain. Accordingly, the shadow map is sampled randomly. The sample locations are varied between neighbouring surface points in order to vary the error over subsequent computations of the visibility factor. This is achieved by randomly rotating a POISSON disk kernel to generate randomised PCF offsets for each computation of the visibility factor. According to different kernel sizes, the sample locations for each kernel size are pre-computed [DH06] and transformed from the unit square to the unit disk [SC97] for preserving distances between the samples. As a result, banding is replaced with less objectionable noise. The error is still present but unstructured and visually pleasingly hidden. The beneficial unstructuring effect is considerably amplified if receivers are covered with high-frequency textures. In summary, with stratified sampling, the number of samples can be reduced to increase performance in favour of slowly degrading shadow quality. Furthermore, aliasing is considerably improved if penumbrae are excessively stretched along receivers due to oblique light.

Of course, in comparison to a combination of PCSS with a linearised shadow map (Section 2.1.2), PCF-based PCSS is considerably limited due to incorrect-self shadowing and costly filtering which infers poor performance. However, if multiple occluders overlap with respect to the direction of light, a high depth complexity introduces objectionable light bleeding in general with linearised shadow maps. This is a direct consequence of estimating the outcome of PCF with a single depth sample using the CHEBYCHEV inequality (Equation 2.9). Increasing the number of samples reduces light bleeding but reverts performance to the performance of PCF in the limit. Therefore, PCSS with pre-filterable shadow maps are mainly applicable for large scenes which exhibit low depth complexity (i.e. sunlight shadows on terrain). In summary, PCF-based PCSS are the generally more precise solution for complex and dynamic scenes which exhibit a small range of depth (i.e. indoor scenes).

3.4.2 Progressive Sampling of Area Light Sources

Accurate soft shadows are obtained by estimating the visibility integral with sampling the area of the extended light source (Equation 3.14). On the area of the extended light source a spot light is setup at every sample location. The individual hard shadows are accumulated to yield a visibility factor for modulating the shading term at each surface point. The spot lights exhibit the same properties concerning direction of light and view volume, respectively. Using a POISSON disk sampling pattern, the spot lights are distributed randomly on the area to achieve a smooth transition of penumbrae. Of course, a sufficiently large number of spot lights is required to yield smooth penumbrae. Given k spot lights, penumbrae exhibit

$k-1$ levels of attenuation. Visually pleasing penumbrae require 256 or more levels if the light is large or penumbrae are stretched across steep surfaces with respect to the direction of light. Accordingly, the area must be sampled densely enough to yield accurate soft shadows. As a consequence, real-time performance is hardly achievable upon generating and evaluating a large number of shadow maps. Furthermore, all shadow maps must be accessible during evaluation which infers high memory consumption and high memory bandwidth.

In order to achieve real-time performance, the shadow maps are generated and evaluated progressively and the individual results are accumulated over time. If the frame rate falls below interactivity (15 frames per second), a sampled area light is replaced with a single spot light. Hence, the frame rate increases and allows for instantaneous interaction upon changing the camera, the light source or geometry. If the scene remains unchanged between two frames, the soft shadow of the area light source is progressively refined until all samples have been evaluated. Of course, if the camera, the light source or geometry is changed between two upcoming frames, the refinement needs to be repeated.

3.5 Optimisation of Light View Volume Culling

The set-up of a light source involves to specify the extent of the view volume of the light with near and far. As a consequence, light view volume culling confines costly shadow map accesses to fragments which are potentially lit (Figure 3.18). Furthermore, PCSS requires to move the near plane as close as possible to the geometry in order to maximise efficiency of early out (Figure 5.11). The optimal distances are derived from a depth image storing light clip-space depth values which were acquired with a very small near and a very large far. The minimum and maximum among these depth values corresponds to the optimal near and far distances, respectively. According to a point light, the minimum and maximum is derived from six depth images comprising the spherical view.

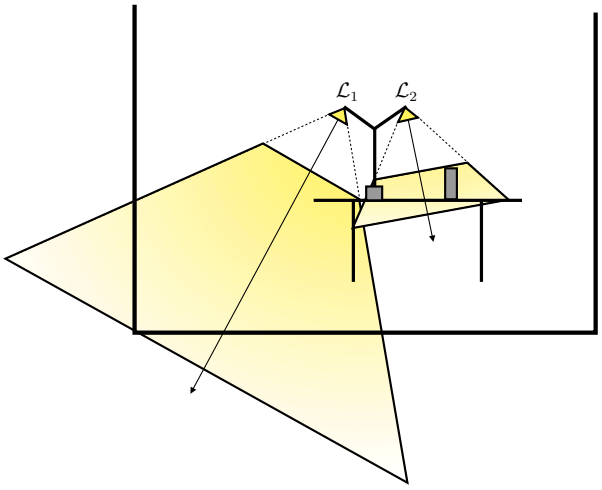


Figure 3.18: Optimisation of light view frustum culling. A desk lamp with two spot lights illuminates a small office in a large building. The optimal view volumes of the lights mostly confine illumination to potentially lit regions if the camera resides inside the room. Light $\mathcal{L}_1$ emanates light beyond the lid of the desk into a corner of the room. Light $\mathcal{L}_2$ only emanates light onto the lid of the desk. Hence, upon rendering the view of the camera, performing occlusion tests for fragments beyond the lid is highly inefficient.

Of course, the optimisation is beneficial only if the performance gain outweighs the additional overhead. That is, if a local light only illuminates a small part of a complex scene (e.g. desk lamp in an office of a power plant). Of course, the optimisation must be repeated in two cases. First, if a dynamic view volume changes position, orientation or size. Second, if dynamic objects enter or leave the view volume. Therefore, the optimisation is mostly beneficial to static lights if dynamic objects are predetermined to never partly leave or enter the view volume which requires to reapply the optimisation. As a consequence of using the distance to the light source as a linear depth metric, exactly deriving the optimal distances directly from the shadow map is unfeasible. Given a spot light with a cutoff angle φ and a maximum distance value d derived from its shadow map, the optimal far plane distance f would be in the interval $d\cos(\varphi) \leq f \leq d$ which depends on the unknown angle between the direction of the spot light and the ray of light passing through the texel storing d. Therefore, the exact determination is achieved with a different depth metric using light clip-space depth. Of course, the shadow map needs to be recreated with the optimal near and far distances using the linear depth metric before subsequent occlusion testing. In summary, the optimisation is a convenience tool to optimally set-up a light source without the need for explicitly specifying near and far at the cost of additional overhead.

Chapter 4

Implementation

4.1 Apelles - A Library for GPU-Based Global Illumination

Apelles is an open library to serve as a base for real-time approximation of a solution of *The Rendering Equation* (Section 3.1.1). Light sources are simply placed in a scene to compute convincing images exhibiting approximated global illumination effects. These effects are generated by extending the common local illumination model with a combination of GPU-friendly rasterisation techniques (programmable shaders, render-to-texture and multi-texturing). Rendering an image involves a scatter pass, if necessary, and a gather pass. Apelles is easily usable in real-time applications which run on consumer graphics hardware.

According to direct illumination, Apelles generates geometric shadows from opaque occluders by utilising *Shadow Mapping* [Wil78]. The algorithm is well supported on contemporary hardware platforms. Therefore, shadow mapping is state of the art for real-time applications. Furthermore, shadow mapping is applicable to arbitrary geometry and scene configurations. However, the generated shadows are susceptible to aliasing artefacts.

Despite extensive research, a universal and robust algorithm for real-time shadow generation is still unavailable (Section 2). As a consequence, Apelles supplies two different algorithms for soft shadow generation of area light sources. First, *Percentage-Closer Soft Shadows* (PCSS) [Fer05] with randomly rotated POISSON disk kernels is utilised for real-time generation of visually pleasing soft shadows (Section 3.4.1). Of course, considerably faster approaches exist which allow to pre-filter the shadow map such as *Variance Shadow Mapping* (VSM) [DL06] (Section 2.1.2). However, in small complex scenes PCSS provide higher precision and more robustness for a considerably higher cost. Second, physically-correct soft shadows are generated by randomly sampling the surface of an area light with several spot lights (Section 3.4.2). Despite yielding the exact solution in the limit, sampling is computationally intense. Therefore, Apelles supplies a progressive render mode which successively refines the resulting soft shadows for real-time applications.

Apelles is implemented in C++ and is built on the open standard and multi-platform OpenGL ecosystem. No additional third-party libraries are required. All necessary auxiliary libraries are developed closely in accord with Apelles. The auxiliary libraries supply side-effect free functionality without introducing any unnecessary ballast. Of course, other well developed libraries exist which provide similar functionality. However, the auxiliary libraries can be independently adapted, extended and, even more importantly, instantaneously be corrected in case of a recently identified defect. Hence, an efficient and independent implementation for a wide range of different platforms is achievable.

At least Apelles requires graphics hardware being capable of OpenGL version 3.0 and OpenCL version 1.1. No vendor specific OpenGL extensions are utilised. The guts of Apelles are mainly implemented against the OpenGL version 2.1 and the OpenGL Shading Language (GLSL) version 1.20. However, despite deprecating old and outdated features, upcoming specifications and recent implementations of the OpenGL ecosystem strive to support existing applications via compatibility profiles. Therefore, the risk of having to port Apelles to upcoming software and hardware platforms is minimal.

The interface of Apelles is simple, concise and stable. As a consequence of applying abstraction precisely, the architecture of Apelles exposes small sub-interfaces only. Each sub-interface aims to accomplish a single specific purpose only. For example, given a light source and some geometry, images can be rendered instantly without requiring any further customisation. Scatter data is implicitly updated if necessary only. In contrast, the update of specific scatter data is controllable finely if a known part of the scene changed only. Several options are exposed to allow for fine-tuning the tradeoff between performance and image quality. Light sources are specifiable using either a reduced set or the full set of available parameters. A simple render callback mechanism ensures not to force the application to render the scene in a particular way. Of course, providing universally applicable rendering semantics which optimally suit all applications is non-trivial.

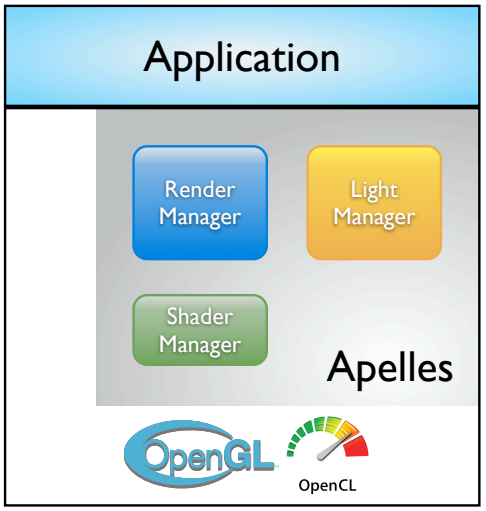


Figure 4.1: Apelles Overview. Apelles is situated in between the application and the OpenGL. The LightManager creates, destroys and maintains the light sources. The RenderManager exposes the main interface of Apelles and houses the rendering logic. Additionally, the RenderManager allocates and updates scatter data if necessary (shadow maps). The ShaderManager provides all library shaders and offers to combine existing shaders with the library shaders.

Creating new applications with Apelles and integrating Apelles into existing applications is uncomplicated. The modular architecture of Apelles strives for uniformity with OpenGL to minimise extra overhead upon learning and using Apelles. Furthermore, Apelles is open for supplementary extension. The algorithms utilised by Apelles are easily combinable with existing algorithms (e.g. *Displacement Mapping* [SKU08], *Caustics Mapping* [SKP07], etc.). No optimisations are applied which constrain the utilised algorithms to a reduced set of input data. Apelles inherently supports to combine the utilised algorithms with existing algorithms. The semantics of the supplied light sources are closely related to the semantics of the standard light sources of OpenGL. Several different render modes are supported: single image rendering, offscreen rendering, progressive rendering and adaptive rendering. According to sampled area light sources, with progressive rendering the result image is progressively refined over consecutive frames to maintain real-time performance. Adaptive rendering continuously monitors performance and reduces image quality once the performance dropped below real-time.

Apelles comprises three main modules (Figure 4.1). First, the LightManager creates, destroys and maintains the different types of light sources: directional lights, point lights, spot lights and area lights (Section 4.1.1). Second, the RenderManager exposes the main interface and houses the rendering logic (Section 4.1.2). Additionally, the RenderManager allocates and updates scatter data (shadow maps) if necessary. Third, the ShaderManager maintains both library and user shaders (Section 4.1.3). User shaders can be created by combining the library shaders with existing shaders (Section 4.1.3.3).

4.1.1 Light Manager

The OpenGL inherently supports a maximum of eight light sources only which are of type directional, point or spot light. No shadows are generated for any of these lights. Apelles supports an unlimited number of lights and generates both hard and soft shadows depending on the type of the light source. The types of light sources supported by the LightManager include

Global directional light Rays of light are parallel because the light is assumed to be infinitely far away. Therefore, the amount of light reaching a surface point varies with respect to the direction of the surface normal only. Shadows are hard.

Directional light Similar to global directional light but the light is confined to emanate within a specified cuboidal volume. The cuboid is defined by a position, a square field of view and near and far plane distances, respectively. The orientation of the cuboid is defined by an up vector. As a consequence, only a part of the scene is illuminated. For example, if parallel light shines through a window into a room.

Point light Rays of light converge at the position of the light source. Light emanates equally in all directions and is attenuated with increasing distance. Shadows are hard.

Spot light While being similar to point light, the light is confined to emanate within a conical volume. The cutoff angle specifies the angle between the direction of light and the lateral surface of the cone of illumination. The exponent specifies the attenuation of light towards the perimeter of the base of the cone.

Area light While being similar to spot light, shadows are soft depending on the size of the light. PCSS is utilised to quickly generate soft shadows which harden on contact convincingly. A POISSON disk kernel is randomly rotated to generate randomised PCF offsets. The spot attenuation factor (spot exponent) is varied implicitly depending on the distance to the light source to simulate a soft spot iris which exhibits a varying penumbra depending on the distance to the light source.

Sampled area light The planar surface of the light source is randomly sampled with several spot lights. While being computationally intense, dense sampling yields accurate soft shadows in the limit. The spot lights are distributed randomly on the surface of the light source utilising a POISSON disk sampling pattern. As a consequence of a sufficiently large number of spot lights, accurate soft shadows are generated by accumulating the individual shadows of the spot lights. The softness of the shadows depends on the size of the planar surface of the light which is spanned by the samples and orthogonal to the direction of light.

The light sources are implemented by applying the *Interface Segregation Principle* (ISP) [Mar96] (Figure 4.2). Despite sharing several properties, light sources are considerably different. All light sources can be enabled and emit coloured light (ambient, diffuse, specular). SpotLight, PointLight, AreaLight and SampledAreaLight have attenuation (constant, linear, quadratic). PointLight lacks a direction because light is emanated in all directions equally. AreaLight and SampledAreaLight lack an exponent (spot attenuation factor) because the soft spot iris is generated implicitly. GlobaDirectionalLight has a direction only because light shines throughout the entire scene. DirectionalLight confines emanation of light to a cuboid with a square cross section. The cuboid is specified by position, field of view, and near and far plane distances. The orientation of the the cuboid with respect to the direction of light can be specified explicitly with an up vector. In contrast, the up vector of other light sources is computed implicitly upon creation (SpotLight, AreaLight, SampledAreaLight and GlobalDirectionalLight). SampledAreaLight has samples which specify the positions of spot lights. The spot lights are randomly distributed on a planar surface which is centred at the position of SampledAreaLight. The surface is orthogonal to the direction of light. The samples have equal direction, cutoff, near and far plane distances, respectively. The size specifies the distance between the samples and the position of SampledAreaLight. As a consequence of applying the ISP, the implementation is centralised in one single class LightImpl which eases reusing common code. Clutter of code over several classes and duplication of code is avoided. Furthermore, supplementary changes need to be done to a single class only.

Similar to the implementation of light sources, the ISP is applied to unify the implementation of the shadow maps in ShadowMapImpl. A ShadowMap is essentially a DepthTexture of squared size with an internal format of `GL_R32F`. The DepthTexture stores the distance to the light source along a ray of light passing through a texel (linear depth metric). The 32 bit floating-point format of the DepthTexture considerably improves the accuracy of the occlusion test and numerical stability; particularly for PCSS. Each light source is associated with the according shadow map. The associated shadow map is invalidated and implicitly updated afterwards, once the properties of a light change (except colour, Attenuation, exponent and size of AreaLight). For occlusion testing during gathering, the ShadowMap needs to be bound to a texture addressing unit (TAU). Therefore, the number of available TAUs constrains the number of light sources that can be rendered in a single pass. In contrast, the depth update of a ShadowMap does not require the ShadowMap to be bound to a TAU.

Shadow generation of multi-sample lights such as SampledAreaLight require a shadow map for each sample. The SampledAreaLight is associated with a ShadowMapArray which aggregates several equally sized depth textures (layers) in a texture array. As a consequence, all types of light sources only require a single TAU to address the associated shadow map for occlusion testing during gathering. If the number of layers is smaller than the number of samples, multi-pass rendering is necessary. In contrast, single-sample lights only require a single shadow map. Although the omnidirectional CubeShadowMap of a PointLight is formed by six depth textures, occlusion testing requires a single depth sample only.

According to AreaLight, the PCSS quality setting determines the visual quality of the penumbrae. Large penumbrae require higher order filters with more samples to effectively hide banding artefacts. As a consequence of excessively accessing the shadow map, performance decreases dramatically with higher quality settings.

Creation of light sources is simple. All light sources are creatable by specifying either the full or a reduced parameter set. As a consequence of closely relating the design and creation of light sources to OpenGL, the usability and integratability of Apelles is considerably enhanced. For example, parameters, which are omitted at the creation of a light source, are implicitly initialised with OpenGL defaults.

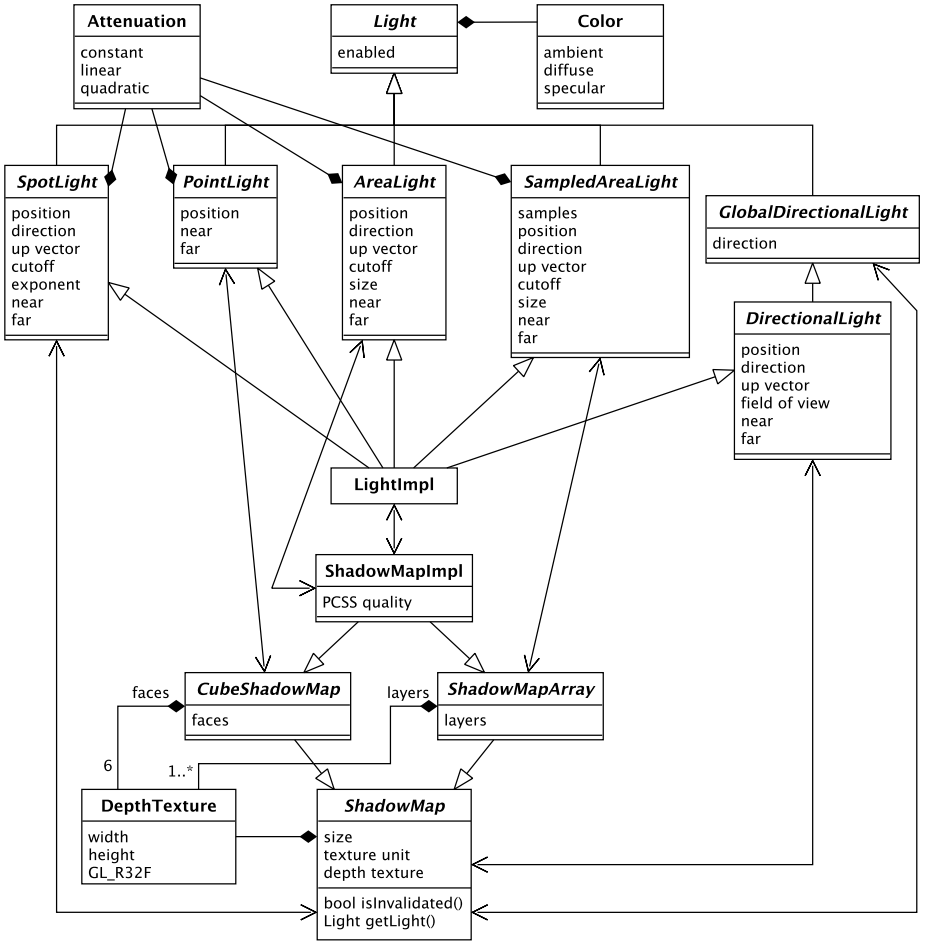


Figure 4.2: Apelles architecture. Light sources. Despite sharing several properties, light sources are considerably different. The *Interface Segregation Principle* (ISP) [Mar96] is applied to unify the implementation of different light sources in LightImpl. Following the same intent, the ISP is applied to implement the associated shadow maps of the light sources in ShadowMapImpl. As a consequence, clutter of code over several classes is avoided which considerably facilitates reusing common code.

The LampLighter aids in creating and modifying light sources instantly. The creation of GlobalDirectionalLight requires the specification of the bounding sphere of the scene. The LampLighter provides methods to setup GlobalDirectionalLight by specifying the bounding sphere, the bounding box or a list of vertices. According to light sources having direction, a new direction of light is settable by specifying *pitch* and *yaw*. According to SampledAreaLight, the LampLighter allows to generate the positions of the randomly distributed spot lights on both circularly or rectangularly shaped and planar surfaces. The positions can be generated from two-dimensional POISSON disk distributions of different sizes. The distributions are pre-computed [DH06] and are statically added to the code base of Apelles for avoiding costly on demand generation while initialising or executing Apelles. As a consequence, the individual hard shadows of the spot lights unify naturally to generate accurate soft shadows with smooth transitions of umbrae and penumbrae. According to progressive rendering, the positions of the spot lights are sorted with respect to the increasing distance to the centre of the surface of the extended light source. As a consequence, penumbrae grow naturally from inner to outer penumbra while progressively refining soft shadows over consecutive frames. Finally the LampLighter enables to efficiently optimise the view volume culling of all types of light sources using an OpenCL-enabled algorithm (Section 4.1.4).

4.1.2 Render Manager

The RenderManager is at the heart of Apelles (Figure 4.3). Each time a light source is created or destroyed the RenderManager is notified by the LightManager. The according ShadowMap of the Light is allocated or deleted, respectively. For decoupling subject and observer, all notifications are implemented utilising the *Observer Pattern* [GHJV94]. After creating light sources, rendering is initiated by calling render(DrawCallback). The callback function is called multiple times, if necessary, during both scattering and gathering.

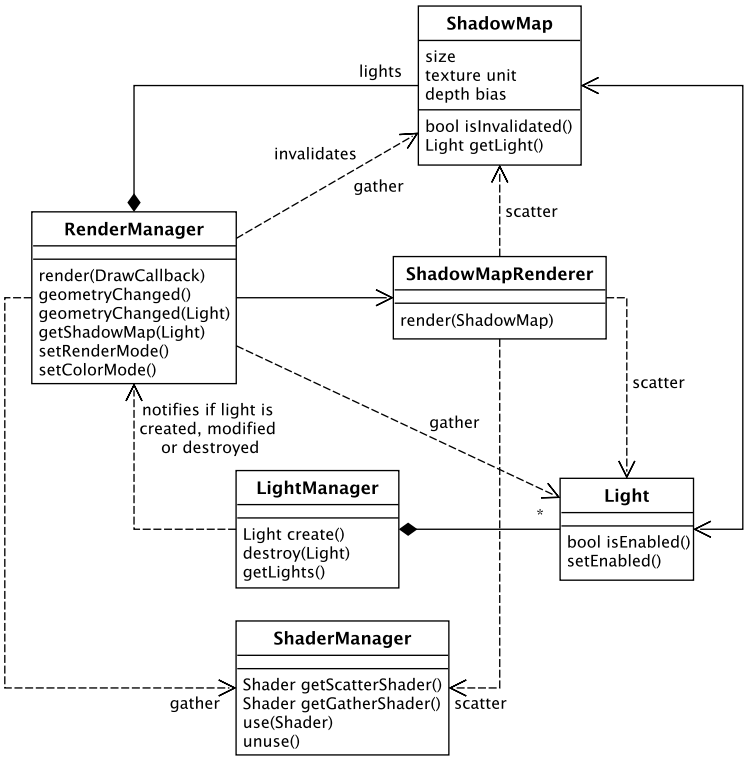


Figure 4.3: Apelles Architecture. Overview. The LightManager notifies to the RenderManager if a Light is created or destroyed. The RenderManager allocates or deletes the ShadowMap being associated with the Light. Calling render(DrawCallback) initiates rendering which is divided into scattering and gathering. During scattering invalidated ShadowMaps are updated by the ShadowMapRenderer. ShadowMap is implicitly invalidated if the according state of ShadowMap or Light changes. In contrast, ShadowMap is explicitly invalidated if the geometry has changed completely or partly. During gathering the result of the occlusion test modulates the result of direct illumination computations. The ShaderManager provides the shaders for both scattering and gathering.

The callback function must not compromise the state of the GL set by Apelles. First, the TAUs used to address scatter data must be unchanged. Therefore, Apelles provides a TextureUnitAcquirer to mark TAUs as occupied by textures of the application before rendering is initiated. Of course, at least one unoccupied TAU must be available for initiating rendering. Second, positional invariance must be ensured for the vertex shader output over both several scatter and gather passes. As a consequence of multi-pass rendering, viewing, projection and viewport transformations must be computational invariant. The current frame buffer object must be unchanged for both scattering and gathering. Third, culling must be disabled in scattering, otherwise depth information with respect to the direction of light for subsequent occlusion testing in gathering is lost. In particular, if *Second-Depth Shadow Mapping* (SDSM) [WM94] is enabled, front-faces of closed surfaces are culled in scattering. Accordingly, culling of front-faces must always be disabled in gathering for both depth biasing and SDSM. Finally, Apelles only saves and restores the necessary states of the GL before and after of scattering to minimise possible losses in performance due to obsolete state savings.

Rendering is divided into scattering and gathering. During scattering the invalidated ShadowMap of each Light is updated using the ShadowMapRenderer. The ShadowMap is implicitly invalidated if an according state change of Light or ShadowMap requires to update the ShadowMap. Additionally, all shadow maps can be explicitly invalidated by calling geometryChanged() if geometry has changed. If the geometry within the view volume of a specific Light has changed only, the affected ShadowMap is explicitly invalidated by specifying the according Light with geometryChanged(Light). During gathering the ShadowMap of each Light is queried to perform the occlusion test for modulating the result of the direct lighting computations. Of course, rendering ignores lights which are disabled. The ShaderManager provides the necessary shaders for both scattering and gathering.

For broadening the range of applications in which Apelles can be used, the RenderManager provides different render and colour modes. With non-progressive rendering for each call to render() a single frame is rendered which shows shadows for all enabled light sources. In progressive rendering mode, soft shadows are successively refined over consecutive frames. The first frame only shows hard shadows for all light sources. Subsequent frames add soft shadows iteratively until the final frame which equals the frame being rendered with non-progressive rendering. Therefore, non-progressive rendering requires the application to call render() repeatedly until rendering has finished. Accordingly, RenderManger allows to query the render progress and to register an observer being notified once rendering finished. Progressive rendering is combinable with adaptive rendering to maintain real-time performance. The frame rate is monitored and image quality is reduced accordingly. On demand adaptive rendering reduces the size of shadow maps, the size of PCF kernels and the number of samples of a SampledAreaLight being rendered in a single pass. The size of PCF kernels is changed in three steps which define the quality of the soft shadows of AreaLight generated with PCSS (high, medium and low quality).

The colour mode of Apelles enables to render separate specular colour. Single colour mode computes colour as a combination of ambient, diffuse, specular and emissive terms. In separate specular colour mode, first, a primary colour is rendered which combines the ambient, diffuse and emissive terms. Then, the secondary colour is rendered which contains the specular term only. As a consequence of the increased flexibility in separately outputting specular colour, specular highlights are applyable after texturing. Hence, specular highlights exhibit the colour of the light source instead of the colour of the texture. Of course, separate specular colour mode infers to render additional passes. Therefore, the RenderManager additionally supports to generate a light map using offscreen rendering.

In scattering the RenderManager updates invalidated ShadowMaps by using the ShadowMapRenderer (Figure 4.4). ShadowMap is invalidated if ShadowMap, Light or geometry inside the view volume of Light has changed. The ShadowMapRenderer uses a frame buffer object to efficiently write the distance to the light source into DepthTexture which has the internal format `GL_R32F` (linear depth metric). As a consequence of using `GL_R32F`, the accuracy of the occlusion test and numerical stability is considerably improved; particularly for PCSS. Therefore, Apelles requires at least OpenGL version 3.0. The necessary viewing and projection transformation is established from the associated Light. Therefore, ShadowMap is invalidated if properties of Light have changed which affect the viewing and projection transformation (position, direction, up vector, cutoff and near and far plane distances). The viewport transformation is established from the ShadowMap. Therefore, ShadowMap is invalidated if the dimensions of the DepthTexture have changed (size).

If depth biasing is enabled, depth values are biased with respect to the local depth slope and a constant bias. Therefore, ShadowMap is invalidated if depth biasing values have changed (slope and const bias). If second-depth shadow mapping is enabled the depth of back-facing surfaces with respect to the direction of light is written to the ShadowMap only. Therefore, ShadowMap is invalidated upon enabling or disabling second-depth shadow mapping. As a consequence of back-face culling, the workload of the geometry stage during scattering is considerably decreased. However, it has to be emphasised that second-depth shadow mapping is applicable if and only if geometry is closed. Otherwise, occluders, which do not exhibit thickness, do not cast any shadows.

CubeShadowMap of PointLight is rendered in six scatter passes. One pass is rendered for each face of the cube forming the omnidirectional shadow map. Of course, if geometry shaders are utilised, a cube shadow map can be rendered more efficiently in a single pass only. However, significant performance boosts are inhibited on most hardware platforms due to feeble geometry processing units; particularly on Shader Model 4 hardware. ShadowMapArray is rendered as many times

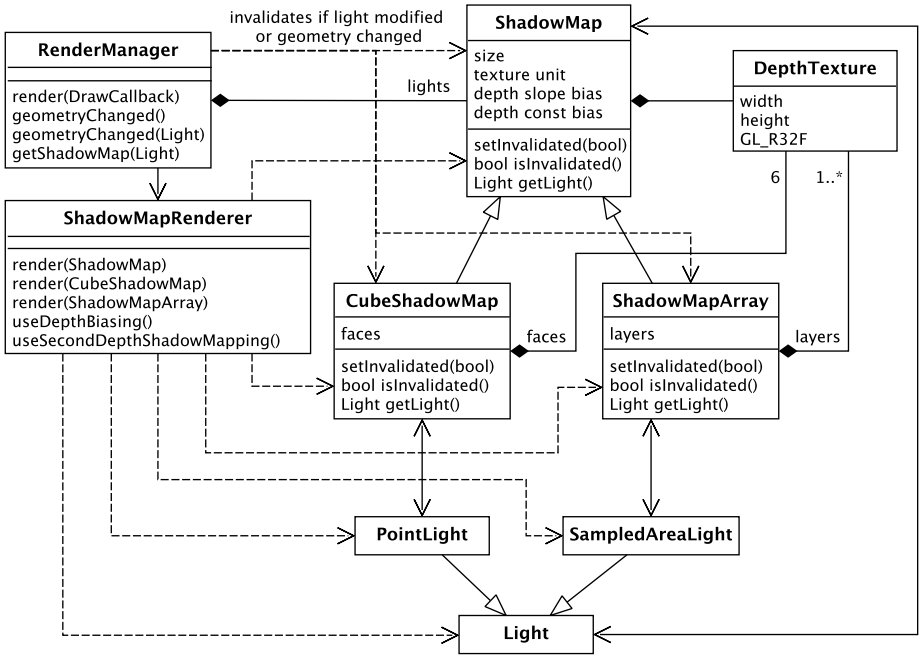


Figure 4.4: Apelles Architecture. Scatter. ShadowMapRenderer updates ShadowMaps being invalidated due to a change of geometry, a ShadowMap or the associated Light. For rendering eye-space depth values to ShadowMap, ShadowMapRenderer establishes the viewing and projection transformation from the associated Light. The viewport transformation is established from ShadowMap. With depth biasing enabled, depth is biased depending on local depth slope and constant bias. With second-depth shadow mapping enabled, depth of back-facing surfaces is written to ShadowMap only. The CubeShadowMap of a PointLight is rendered six times. The ShadowMapArray of a SampledAreaLight is rendered as many times required to render all layers of the texture array.

required to render all layers of the texture array of DepthTexture. Therefore, ShadowMapArray is invalidated if the number of layers is changed. If the number of layers is smaller than the number of samples of the associated SampledAreaLight, in the subsequent gather pass the occlusion test can be performed for a subset of samples only. As a consequence, the number of layers constrains the number of passes of both scattering and gathering required to completely render SampledAreaLight. Furthermore, if the number of layers is unequal to the number of samples, the ShadowMapArray remains invalidated always.

Using the distance to the light source as a linear depth metric for all light sources infers considerable improvements for shadow mapping. First, depth disparity is introduced across polygons without depth slope which supports to lessen incorrect self-shadowing. Second, shadow mapping remains applicable if the view volume of the camera is skewed. Third, distribution of depth along the depth range becomes uniform if the reprojection from eye-space to light-space involves a perspective projection (spot light, point light, area light). Finally, depth is not mapped to $[0, 1]$ with respect to the near plane and the far plane. Therefore, both depth biasing and the accuracy of the occlusion test become independent of the near and far plane distances. Hence, the depth biasing setting no loner must be readjusted if the distance between the near plane and the far plane has changed. Furthermore, the near and far planes no longer have to be moved as close as possible to the geometry. However, PCSS still requires to move the near plane as close as possible in order to enable an efficient early out. Of course, computing the distance to the light source per fragment is more expensive. However, the loss in performance is negligible; particularly on Shader Model 4 hardware platforms and beyond. Furthermore, the evaluation of the attenuation of light already requires to compute the distance to the light source in gathering (spot light, point light and area light). According to AreaLight, PCSS requires to compute the distance to the light source as well (penumbra width estimation). As a consequence of using `GL_R32F` as internal format of DepthTexture, a noticeable but negligible inconsistency arises concerning depth testing during scattering. The z-buffer has the internal format `GL_DEPTH_COMPONENT32F`. Therefore, depth values of the z-buffer are not converted to fixed-point but are clamped to the range $[0, 1]$. As a consequence, the accuracy of depth in the z-buffer differs from the accuracy of depth in the DepthTexture. The inconsistency is only avoidable if the DepthTexture uses the internal format `GL_DEPTH_COMPONENT32F_NV`. However, this would confine Apelles to run on NVIDIA hardware only.

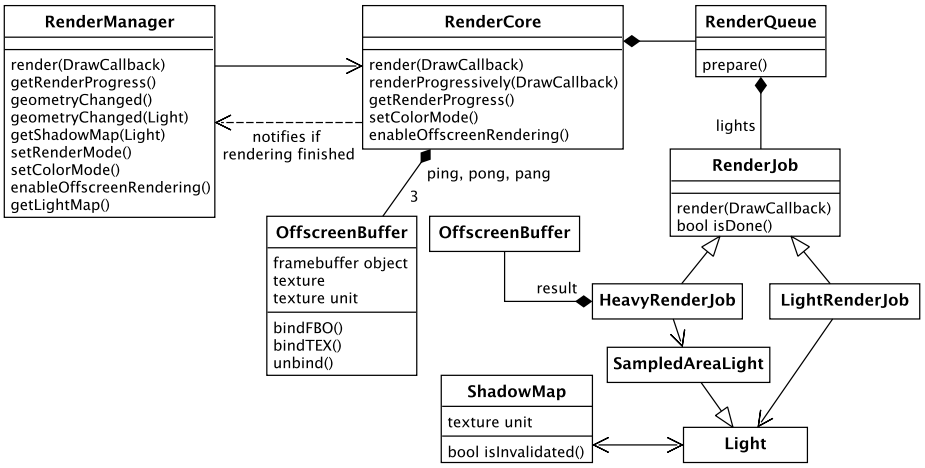


Figure 4.5: Apelles Architecture. Gather. RenderCore prepares and processes a queue of RenderJobs. A LightRenderJob comprises single-sample Lights and, therefore, is done after one call to render(). A HeavyRenderJob comprises a single SampledAreaLight which requires several calls to render() if the number of samples is larger than the number of layers of the associated ShadowMapArray. The offscreen buffers ping and pong are used to accumulate the result of done RenderJobs in ping to the final frame. According to progressive rendering, OffscreenBuffer pang is necessary to accumulate intermediate results of SampledAreaLights in pong.

In gathering RenderCore processes RenderJobs on a RenderQueue to render a frame exhibiting shadows for all enabled Lights (Figure 4.5). The concept of aggregating RenderJobs in a RenderQueue is a direct consequence of applying the *Command Pattern* [GHJV94]. Upon the preparation of the RenderQueue, a RenderJob is created for each enabled Light. After calling render(DrawCallback), a RenderJob is removed from the RenderQueue if the job is done. A LightRenderJob comprises several Lights whose shadow generation only requires a single pass (single-sample lights). Hence, LightRenderJob is a lightweight job in the sense that the job is done after a one call to render(). The number of single-sample lights for which shadows can be generated in a one pass is constrained by the number of free TAUs to address the according shadow maps. All types of shadow maps only require a single TAU for being bound. As a consequence, each Light requires a TAU to address the associated ShadowMap. If more single-sample lights shall be rendered than free TAUs are available, several render jobs are created (multi-pass rendering). In contrast, a HeavyRenderJob comprises only a single SampledAreaLight (multi-sample light). Heavyweight render jobs require several calls to render() until the job is done. The number of necessary calls depends on the number of layers of the ShadowMapArray. Therefore, HeavyRenderJob accumulates the result of each pass in an OffscreenBuffer. Accordingly, the results of done RenderJobs are accumulated to the final frame by utilising *ping-pong* offscreen rendering.

Suppose, for example, that a SpotLight SL, a DirectionalLight DL, a PointLight PL and an AreaLight AL with PCSS shall be rendered. Additionally, a SampledAreaLight SAL_{32}^{32} with 32 samples and a ShadowMapArray with 32 layers shall be rendered. Only two free TAUs are available which implies only two single-sample Lights are renderable in a pass (scatter and gather). Therefore, the RenderQueue aggregates two LightRenderJobs (LRJ) and one HeavyRenderJob (HRJ). Processing of the RenderQueue LRJ(SL,DL) LRJ(PL,AL) HRJ(SAL_{32}^{32}) with ping-pong offscreen rendering runs as follows

RENDERNONPROGRESSIVELY($SL, DL, PL, AL, SAL_{32}^{32}$, 2 TAU)
1 SCATTER(SL, DL)
2 $ping \leftarrow$ GATHER(SL, DL)
3 SCATTER(PL, AL)
4 $pong \leftarrow ping +$ GATHER(PL, AL)
5 SWAP($ping, pong$)
6 SCATTER(SAL_{32}^{32})
7 $pong \leftarrow$ GATHER(SAL_{32}^{32})
8 SWAP($ping, pong$)
9 $\mathbf{frame_0} \leftarrow ping$

Line 2, the result of the gather pass is rendered into the offscreen buffer ping. Line 5 and 8, swapping ping and pong ensures that the final result accumulates in ping always. Of course, a redraw is initiated if the viewing, projection or viewport transformation of gather has changed between two frames. If ShadowMaps are not invalidated, scattering is skipped and only gather passes are rendered. Note that ShadowMapArray remains invalidated always if the number of layers is smaller than the number of samples of the associated SampledAreaLight.

Progressive rendering distributes the high computational cost of sampling of a SampledAreaLight over consecutive frames evenly. The first frame exhibits hard shadows for all Lights only. For AreaLights PCSS is skipped and a hard shadow is rendered as if the AreaLight would have been a SpotLight. For SampledAreaLight the closest sample with respect to the position of the light source is rendered. Then, only subsets of samples are being rendered to successively refine soft shadows until all samples were rendered. The samples are sorted with respect to the increasing distance to the position of SampledAreaLight. Hence, penumbrae grow naturally from inner to outer penumbra while rendering progressively.

With progressive rendering, an AreaLight requires two passes because PCSS is rendered in the second pass. Hence, as a consequence of utilising the command pattern for RenderJobs, AreaLight is wrapped in a HeavyRenderJob to implement the progressive rendering behaviour. In contrast, AreaLight is wrapped in a LightRenderJob to implement non-progressive rendering behaviour. Similar to non-progressive rendering, a redraw is initiated if the viewing, projection or viewport transformation of gathering has changed between two frames. Unfinished HeavyRenderJobs are discarded and a new RenderQueue is prepared.

Suppose, for example, that a SpotLight *SL*, a DirectionalLight *DL* and a PointLight *PL* shall be rendered. Additionally, a SampledAreaLight SAL_8^{32} with 32 samples and a ShadowMapArray with 8 layers shall be rendered. Therefore, to be completely rendered the SAL_8^{32} requires four passes (scatter and gather). Only two free TAUs are available which implies only two single-sample Lights are renderable in a pass (scatter and gather). Therefore, the RenderQueue aggregates two LightRenderJobs (LRJ) and one HeavyRenderJob (HRJ). Processing of the RenderQueue LRJ(*SL*,*DL*) LRJ(*PL*) HRJ(SAL_8^{32}) with ping-pong offscreen rendering runs as follows

RENDERPROGRESSIVELY(SL, DL, PL, SAL_{32}^8, 2 TAU)

1 SCATTER(SL, DL)
2 $ping \leftarrow$ GATHER(SL, DL)
3 SCATTER(PL)
4 $pong \leftarrow ping +$ GATHER(PL)
5 SWAP($ping, pong$)
6 SCATTER(SAL_{32}^1)
7 $\mathbf{frame_0} \leftarrow ping +$ GATHER(SAL_{32}^1)
8 SCATTER(SAL_{32}^7)
9 $pong \leftarrow$ GATHER(SAL_{32}^8)
10 $\mathbf{frame_1} \leftarrow ping + pong$
11 SCATTER(SAL_{32}^8)
12 $pang \leftarrow \frac{1}{2} pong + \frac{1}{2}$GATHER($SAL_{32}^8$)
13 SWAP($pong, pang$)
14 $\mathbf{frame_2} \leftarrow ping + pong$
15 SCATTER(SAL_{32}^8)
16 $pang \leftarrow \frac{2}{3} pong + \frac{1}{3}$GATHER($SAL_{32}^8$)
17 SWAP($pong, pang$)
18 $\mathbf{frame_3} \leftarrow ping + pong$
19 SCATTER(SAL_{32}^8)
20 $pang \leftarrow \frac{3}{4} pong + \frac{1}{4}$GATHER($SAL_{32}^8$)
21 SWAP($pong, pang$)
22 $pang \leftarrow ping + pong$
23 SWAP($ping, pang$)
24 $\mathbf{frame_4} \leftarrow ping$

Line 6 and 7, as a consequence of rendering hard shadows to the first frame, one scatter pass and one gather pass of the first sample suffices. Line 8, if the viewing, projection and viewport transformation of gathering remain unchanged, progressive rendering continues by scattering the seven remaining samples. Line 9, eight samples are gathered to pong. Line 10, all hard shadows (ping) and a soft shadow sampled with eight samples (pong) are rendered to the second frame. Line 12, 16 and 20, previous intermediate results and the current result of sampling are accumulated in equal parts (pang) to be displayed in intermediate frames. Line 7, 10, 14, 18 and 24, if the viewing, projection or viewport transformation of gathering has changed, a redraw is initiated.

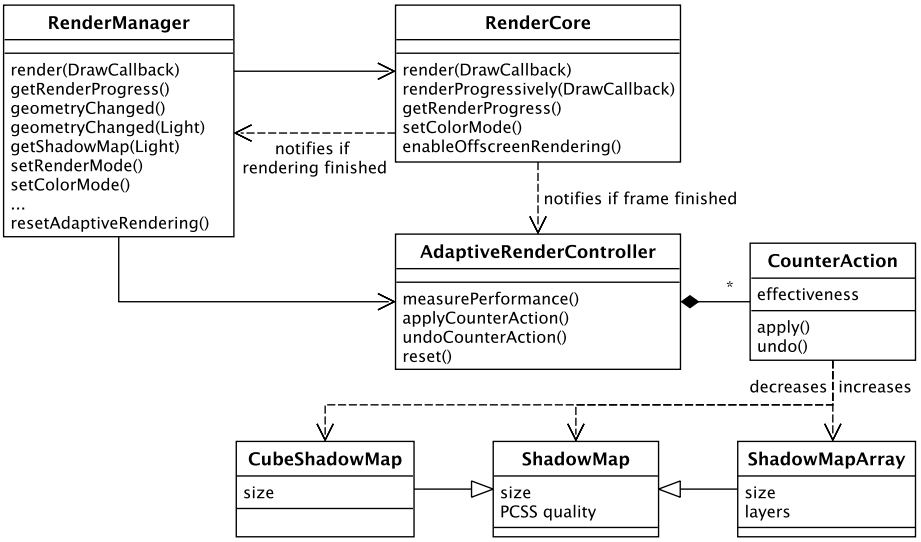


Figure 4.6: Apelles Architecture. Adaptive rendering. The AdaptiveRenderController is notified by the RenderCore if rendering a frame finished to measure the frame rate. If the frame rate drops below real-time, CounterActions are applied. The size of ShadowMaps, the PCSS quality and the number of layers of ShadowMapArray are decreased. If the frame rate raises sufficiently high due to a significant reduction of workload between two calls of render(), CounterActions are undone. Resetting the AdaptiveRenderController undos all previously applied CounterActions.

Adaptive rendering measures the frame rate and applies CounterActions once the frame rate dropped below real-time (Figure 4.6). The concept of CounterAction stems from the requirement of undoing the CounterAction if the frame rate again raises sufficiently high. Therefore, the implementation of CounterAction utilises the *Command Pattern* [GHJV94]. The AdaptiveRenderController is notified by the RenderCore every time a frame finished to measure the time between two calls of render(). Another option is to only measure the time the RenderCore needs to render a frame. However, the latter option ignores time consumed by user processes between two calls of render(). Therefore, the first option is implemented.

As a consequence, adaptive rendering is additionally sensitive to workload in-between calls of render() instead of workload inside render() only. Measuring the frame rate is considerably improved by reducing spikes in the signal with a *Weighted Moving Average* (convolution with a lowpass filter). A greater weight is assigned to more recent frames with linearly decreasing weights. As a consequence, the filtered signal exhibits an improved balance between sufficient signal response sensitivity and reduction of spikes which are introduced due to GPU command queue caching.

Undoing of CounterActions carries the risk of a feedback loop where the same CounterAction is applied and undone repeatedly. As a consequence, the CounterAction remains applied after the sequence *apply, undo, apply* has been detected. Furthermore, if the last three CounterActions were ineffective, no further CounterActions are applied. The effectiveness of CounterAction records the difference in frame rate after the CounterAction has been applied. Therefore, after applying a CounterAction the frame rate is given a setting time to reflect the effect of the CounterAction before continuing to measure the frame rate.

In essence, applying CounterActions reduces image quality. The workload of scattering is considerably reduced by decreasing the size of ShadowMaps. The workload of gathering of AreaLights is considerably reduced by decreasing the size of PCF kernels (PCSS quality). Hence, image quality is traded for less texture accesses. The workload of both scattering and gathering of SampledAreaLight is reduced by decreasing the number of layers of the associated ShadowMapArray. Of course, after CounterActions have been applied, the application is responsible for keeping freed VRAM free to allow for undoing the CounterActions once the frame rate raised sufficiently high again. Other options to raise the frame rate were examined but not implemented due to considerable visual defects. First, postponing scattering for every second frame already causes shadows to fidget which visually disconnects occluders and receivers. Second, replacing a SampledAreaLight with an AreaLight requires to setup an appropriate magnitude of depth bias. The magnitude depends on the area size and the current scene configuration. Finally, frequent optimisation of view volume culling with an OpenCL-enabled algorithm easily outweighs achievable gains in performance.

4.1.3 Shader Manager

The ShaderManager creates and compiles the shaders necessary for scattering and gathering. The source of all shaders is written against the OpenGL Shading Language version 1.20. If a shader is modified after creation, notifications are fired utilising the *Observer Pattern* [GHJV94]. Additionally, the ShaderManager enables to combine the shaders of Apelles with existing shaders to yield customised user shaders. For facilitating the management of user shaders, the ShaderManager aggregates user shaders.

In scattering, the distance to the light source is computed and optionally biased. Using a frame buffer object the distance is directly written to the according texture with the internal format `GL_R32F`.

Listing 4.1: Code of the scattering shader program for writing depth into ShadowMap.

```
1  //-----------------------------------------------------------------------
2  // Vertex Shader
3  varying vec4 vpos_ec_;
4  void main()
5  {
6     vpos_ec_ = gl_ModelViewMatrix * gl_Vertex;
7     gl_Position = ftransform();
8  }
9  //-----------------------------------------------------------------------
10 // Fragment Shader
11 uniform float DEPTH_SLOPE_BIAS;
12 uniform float DEPTH_CONST_BIAS;
13 varying vec4 vpos_ec_;
14 void main()
15 {
16    float depth = length(vpos_ec_.xyz);
17    depth += DEPTH_SLOPE_BIAS * (abs(dFdx(depth)) + abs(dFdy(depth)));
18    gl_FragColor = vec4(depth + DEPTH_CONST_BIAS);
19 }
```

Line 17 and 18, if depth biasing is disabled, `DEPTH_SLOPE_BIAS` and `DEPTH_CONST_BIAS` are set to zero. Accordingly, on specific hardware platforms line 17 is potentially optimised away due to constant propagation and constant folding (e.g. NVIDIA).

In gathering lighting computations are performed depending on the type of RenderJob. According to LightRenderJobs, lighting computations are performed for directional lights, point lights, spot lights and area lights. According to HeavyRenderJobs, lighting computations are performed for sampled area lights. Hence, the main program of the gathering shader declares a function computeLighting(). Different implementations of the function are achieved flexibly by attaching the according fragment shader. Each fragment shader defines a particular implementation of the function.

Listing 4.2: Code of the main gathering shader program.

```
1  //-----------------------------------------------------------------------
2  // Vertex Shader
3  varying vec4 vpos_ec_;
4  varying vec3 N_;
5  void main(void)
6  {
7    vpos_ec_ = gl_ModelViewMatrix * gl_Vertex;
8    N_ = normalize(gl_NormalMatrix * gl_Normal);
9    gl_Position = ftransform();
10 };
11 //-----------------------------------------------------------------------
12 // Fragment Shader
13 const int FIRST_PASS = 1;
14 const int INTERMEDIATE_PASS = 2;
15 const int LAST_PASS = 3;
16 uniform int WHICH_PASS;
17 varying vec4 vpos_ec_;
18 varying vec3 N_;
19 //-----------------------------------------------------------------
20 void computeLighting( in vec3 N, in vec4 vpos_ec,
21                       in vec4 material_ambient,
```

```
22                      in vec4 material_diffuse,
23                      in vec4 material_specular,
24                      in float material_shininess,
25                      out vec4 color );
26 //------------------------------------------------------------------
27 void main(void)
28 {
29   vec4 color = vec4(0.0);
30   computeLighting( N_, vpos_ec_,
31                    gl_FrontMaterial.ambient,
32                    gl_FrontMaterial.diffuse,
33                    gl_FrontMaterial.specular,
34                    gl_FrontMaterial.shininess,
35                    color );
36   if (WHICH_PASS == LAST_PASS)
37    color += gl_FrontLightModelProduct.sceneColor;
38   gl_FragColor = color;
39 }
```

Line 20, computeLighting() is defined in an additionally attached fragment shader to allow for a flexible exchange of the implementation of computeLighting() between the two implementations for LightRenderJob and HeavyRenderJob. Line 36 and 37, `gl_FrontLightModelProduct.sceneColor` shall only be added to the surface colour in the last pass. This is a direct consequence of rendering multiple passes due to ping-pong offscreen rendering of several RenderJobs. With progressive rendering upon displaying the final frame a sudden change in intensity will be noticeable. However, this at first undesirable effect helps in identifying visually when progressive rendering finished.

4.1.3.1 Lighting Computation of Single-Sample Lights

LightRenderJob computes lighting for several single-sample Lights. In order to declare an array of Lights, attributes of Light are aggregated in one structure Light for all types of light sources. Depending on the type of Light, some attributes are *active* and some of the attributes are *inactive*. For example, if the type of Light is `DIRECTIONAL_LIGHT`, then `attenuation` is unused and, therefore, inactive. Of course, excessive wastage of unused uniform variables needs to be considered. The number of available uniform variables varies between different hardware platforms. In particular, note that the GLSL specification allows for hardware-specific optimisations that remove inactive uniforms implicitly (e.g. NVIDIA).

Listing 4.3: Code of the structure Light aggregating the attributes of Light.

```
1  //------------------------------------------------------------------
2  const int SPOT_LIGHT = 1;
3  const int POINT_LIGHT = 2;
4  const int DIRECTIONAL_LIGHT = 3;
5  const int AREA_LIGHT = 4;
6  //------------------------------------------------------------------
7  struct Light
8  {
9    int type; // SPOT_LIGHT, POINT_LIGHT, DIRECTIONAL_LIGHT, AREA_LIGHT
10 // common attributes
11   vec4 ambient, diffuse, specular; // color
12   vec3 attenuation;                // constant, linear, quadratic
13 // transformed by the modelview matrix
14   vec3 position_ec3;
15 // specific attributes
16 // transformed by the upper 3x3 of the modelview matrix
17 //   and normalised
18   vec3 direction_ec3;     // SPOT_LIGHT, DIRECTIONAL_LIGHT, AREA_LIGHT
19   float spot_cos_cutoff; // SPOT_LIGHT, AREA_LIGHT
20   float spot_exponent;    // SPOT_LIGHT
21   float area_size;        // AREA_LIGHT
22   float near;             // AREA_LIGHT, POINT_LIGHT
23   float far;              // POINT_LIGHT
24 };
```

LightRenderJob computes lighting for different types of several single-sample Lights. A single shader is implemented to be used for all single-sample Lights. Hence, lighting computations of several different types of Lights are performable in a single gather pass. The overhead introduced if one type of Light is rendered only is negligible if the hardware-specific implementation of the GLSL optimises away most of the unused code. However, the GLSL allows to declare arrays of Light of fixed size only. Therefore, the ShaderManager creates and compiles several copies of the gathering shader. In each copy the array of Lights has a different size. Of course, modifying and compiling shaders repeatedly is costly and, therefore, easily stalls rendering. Therefore, the ShaderManager allows to either compile the shaders in advance or on demand while keeping already compiled shaders cached. Hence, several LightRenderJobs, whose number of Lights varies, are renderable efficiently by quickly switching between already compiled shaders. The RenderManager allows to specify the maximum number of single-sample Lights comprised by a LightRenderJob. Additionally, the maximum number of Lights being comprised by a LightRenderJob is constrained by the number of free TAUs. Both bounds infer the partition of single-sample Lights into LightRenderJobs. Experiments showed that computing the lighting in one pass for eight Lights is optimal (Section 5.2.5).

Listing 4.4: Code of the gathering shader for surface colour computation of LightRenderJob.

```
1  //------------------------------------------------------------------
2  const int LIGHT_COUNT = 1;
3  uniform mat4 CAMERA_VIEW_INVERSE_MAT4;
4  uniform Light LIGHTS[LIGHT_COUNT];
5  uniform sampler2D SHADOW_MAPS[LIGHT_COUNT];
6  uniform mat4 SHADOW_MATRICES[LIGHT_COUNT];
7  uniform samplerCube CUBE_SHADOW_MAPS[LIGHT_COUNT];
8  //------------------------------------------------------------------
9  void computeLighting( in vec3 N, in vec4 vpos_ec,
10                       in vec4 material_ambient,
11                       in vec4 material_diffuse,
12                       in vec4 material_specular,
13                       in float material_shininess,
14                       out vec4 color )
15 {
16   vec3 shadow_coords[LIGHT_COUNT];
17   for (int i = 0; i < LIGHT_COUNT; i++)
18   {
19     vec4 shadow_coords_ = SHADOW_MATRICES[i] * CAMERA_VIEW_INVERSE_MAT4 * vpos_ec;
20     shadow_coords[i] = vec3(shadow_coords_) / shadow_coords_.w;
21   }
22   vec3 vpos_ec3 = vec3(vpos_ec) / vpos_ec.w;
23   vec4 amb, diff, spec = vec4(0.0);
24   for (int i = 0; i < LIGHT_COUNT; i++)
25   {
26     if (LIGHTS[i].type == SPOT_LIGHT)
27       SpotLight(i, N, vpos_ec3, shadow_coords[i], material_shininess, amb, diff, spec);
28     else if (LIGHTS[i].type == POINT_LIGHT)
29       PointLight(i, N, vpos_ec3, material_shininess, amb, diff, spec);
30     else if (LIGHTS[i].type == DIRECTIONAL_LIGHT)
31       DirectionalLight(i, N, vpos_ec3, shadow_coords[i], material_shininess, amb, diff, spec);
32     else if (LIGHTS[i].type == AREA_LIGHT)
33       AreaLight(i, N, vpos_ec3, shadow_coords[i], material_shininess, amb, diff, spec);
34   }
35   color = amb * material_ambient + diff * material_diffuse + spec * material_specular;
36 }
```

Line 2, the ShaderManager increases `LIGHT_COUNT` for each copy of the gathering shader to enable rendering LightRenderJobs with a varying number of single-sample Lights. Line 5–7, `SHADOW_MAPS` and `SHADOW_MATRICES` are unnecessary to perform the occlusion test for PointLight. The direction for querying the cube shadow map is computed implicitly. Line 17–21, concerning SpotLight, DirectionalLight and AreaLight, shadow coordinates are computed to reproject from eye-space to light-space (Equation 3.16). In essence, the shadow matrix transforms from three-dimensional world-space to the two-dimensional texture-coordinate-space of a shadow map. Note that depending on hardware-specific optimisations the computation of shadow coordinates is optimised away by the GLSL if PointLights are rendered only (Line 29). Line 24–34, loop over all Lights of the LightRenderJob and compute the ambient, diffuse and specular terms with respect to the type of Light.

GlobalDirectionalLight and DirectionalLight are rendered using the same shader code. Lighting computation and occlusion testing are confined to potentially lit fragments only to efficiently skip costly and unnecessary parts of the shader for three reasons. First, as a consequence of using projective texturing for occlusion testing, the projection transformation causes a dual shadow projection along the negative direction of light. Second, fragments, which are on the backside of surfaces with respect to the direction of light, are unlit. Third, if second-depth shadow mapping is used, occlusion testing shall be confined to front-facing fragments only. Otherwise incorrect self-shadowing artefacts reappear on back-facing surfaces which are illuminated by another light source.

Listing 4.5: Code of the gathering shader for rendering GlobalDirectionalLight and DirectionalLight.

```
1  //-----------------------------------------------------------------
2  bool isOccluded( const in sampler2D shadow_map,
3                   in vec3 shadow_coords )
4  {
5    float blocker_depth = texture2D(shadow_map, vec2(shadow_coords)).r;
6    float receiver_depth = shadow_coords.z;
7    if (blocker_depth < receiver_depth)
8      return true;
9    return false;
10 }
11 //-----------------------------------------------------------------
12 void DirectionalLight( in int i, in vec3 N, in vec3 vpos_ec3,
13                        in vec3 shadow_coords,
14                        in float material_shininess,
15                        inout vec4 ambient, inout vec4 diffuse,
16                        inout vec4 specular )
17 {
18   if (shadow_coords.z < 0.0 || shadow_coords.z > 1.0)
19     return;
20   vec3 L = LIGHTS[i].position_ec3 - vpos_ec3;
21   vec3 L_wc = vec3(CAMERA_VIEW_INVERSE_MAT4 * vec4(L, 0));
22   L = -LIGHTS[i].direction_ec3;
23   ambient += LIGHTS[i].ambient;
24   float lambert_term = max(dot(N, L), 0.0);
25   if (lambert_term == 0.0)
26     return;
27   float phong_term =
28     pow(clamp(dot(N, normalize(L - normalize(vpos_ec3))), 0.0, 1.0), material_shininess);
29   float d = length(L_wc);
30   shadow_coords.z = d;
31   float shadow_attenuation = 1.0 - float(isOccluded(SHADOW_MAPS[i], shadow_coords));
32   diffuse += LIGHTS[i].diffuse * lambert_term * shadow_attenuation;
33   specular += LIGHTS[i].specular * phong_term * shadow_attenuation;
34 }
```

Line 5 and 6, the depth being closest to the light source is stored in the red component because the depth texture has the internal format `GL_R32F`. Line 18 and 19, early out which confines the shader input to fragments inside the view volume of the light source. Line 20–22, in order to compute the distance to the light source correctly, the vector L is transformed into world-space because the length of vector L depends on the viewing and projection transformation of the camera (eye-space). Line 24, computation of the LAMBERT term implicitly hides projection aliasing. The intensity of light decreases dramatically close to where the surface normal N and the direction of light L become orthogonal. Line 25 and 26, early out which further confines the shader input to fragments which are on the frontside of surfaces with respect to the direction of light. Line 30, use the distance to the light source as a linear depth metric. Line 31–33, perform the occlusion test and modulate the diffuse and specular term.

The shader for rendering SpotLight differs from DirectionalLight in three ways. First, the direction of light varies over fragments if the SpotLight is sufficiently close to the surface point. Second, the input of the shader is confined to a cone of illumination. Finally, the ambient, diffuse and specular terms are modulated with the attenuation of light which involves spot, constant, linear and quadratic attenuation factors.

Listing 4.6: Code of the gathering shader for rendering SpotLight.

```
1  void SpotLight( in int i, in vec3 N, in vec3 vpos_ec3,
2                  in vec3 shadow_coords,
3                  in float material_shininess,
4                  inout vec4 ambient, inout vec4 diffuse,
5                  inout vec4 specular )
6  {
7    if (shadow_coords.z < 0.0 || shadow_coords.z > 1.0)
8      return;
9    vec3 L = LIGHTS[i].position_ec3 - vpos_ec3;
10   vec3 L_wc = vec3(CAMERA_VIEW_INVERSE_MAT4 * vec4(L, 0));
11   L = normalize(L);
12   float spot_dot = dot(-L, LIGHTS[i].direction_ec3);
13   if (spot_dot <= LIGHTS[i].spot_cos_cutoff)
14     return;
15   float spot_attenuation = pow(spot_dot, LIGHTS[i].spot_exponent);
16   float d = length(L_wc);
17   float attenuation = spot_attenuation / ( LIGHTS[i].attenuation[0] +
18                                            LIGHTS[i].attenuation[1] * d +
19                                            LIGHTS[i].attenuation[2] * d * d );
20   ambient += LIGHTS[i].ambient * attenuation;
21   float lambert_term = max(dot(N, L), 0.0);
22   if (lambert_term == 0.0)
23     return;
24   float phong_term =
25     pow(clamp(dot(N, normalize(L - normalize(vpos_ec3))), 0.0, 1.0), material_shininess);
26   shadow_coords.z = d;
27   float shadow_attenuation = 1.0 - float(isOccluded(SHADOW_MAPS[i], shadow_coords));
28   diffuse += LIGHTS[i].diffuse * lambert_term * attenuation * shadow_attenuation;
29   specular += LIGHTS[i].specular * phong_term * attenuation * shadow_attenuation;
30 }
```

Line 9–11, direction of light varies for all fragments if the light source is sufficiently close to the surface point. Line 13 and 14, early out which confines the shader input to fragments inside the cone of illumination defined by the cosine of the cutoff angle. Line 15–19, computation of the attenuation of light with respect to the distance to the light source. Line 26, use the distance to the light source as a linear depth metric instead of the non-linear post-perspective depth. Line 20, 28 and 29, the ambient, diffuse and specular terms are modulated with the spatial attenuation of light.

In contrast to SpotLight, PointLight emanates light equally in all directions. Using a cube map for omnidirectional occlusion testing infers two consequences. First, shadow coordinates which transform from eye-space to clip-space of the light are unnecessary. The cube map is simply queried with the negative direction of light for a given surface point. Second, as a consequence of missing shadow coordinates, confining the shader input to fragments only inside the view volume requires additional computations and two additional uniform variables (near and far distances).

Listing 4.7: Code of the gathering shader for rendering PointLight.

```
1  //-------------------------------------------------------------------
2  bool isOccluded( const in samplerCube shadow_map,
3                   in vec4 shadow_coords )
4  {
5    float blocker_depth = textureCube(shadow_map, vec3(shadow_coords)).r;
6    float receiver_depth = shadow_coords.w;
7    if (blocker_depth < receiver_depth)
8      return true;
9    return false;
10 }
11 //-------------------------------------------------------------------
12 void PointLight( in int i, in vec3 N, in vec3 vpos_ec3,
13                  in float material_shininess,
14                  inout vec4 ambient,  inout vec4 diffuse,
15                  inout vec4 specular )
16 {
17   vec3 L = LIGHTS[i].position_ec3 - vpos_ec3;
18   vec3 L_wc = vec3(CAMERA_VIEW_INVERSE_MAT4 * vec4(L, 0));
```

```
19   if (any(bvec3( all(greaterThan(abs(L_wc).xxx, vec3(LIGHTS[i].far, abs(L_wc).yz))),
20                  all(greaterThan(abs(L_wc).yyy, vec3(LIGHTS[i].far, abs(L_wc).xz))),
21                  all(greaterThan(abs(L_wc).zzz, vec3(LIGHTS[i].far, abs(L_wc).xy))) )))
22     return;
23   float d = length(L_wc);
24   if (d < LIGHTS[i].near)
25     return;
26   L = normalize(L);
27   float attenuation = 1.0 / ( LIGHTS[i].attenuation[0] +
28                               LIGHTS[i].attenuation[1] * d +
29                               LIGHTS[i].attenuation[2] * d * d );
30   ambient += LIGHTS[i].ambient * attenuation;
31   float lambert_term = max(dot(N, L), 0.0);
32   if (lambert_term == 0.0)
33     return;
34   float phong_term =
35     pow(clamp(dot(N, normalize(L - normalize(vpos_ec3))), 0.0, 1.0), material_shininess);
36   vec4 shadow_coords = vec4(-L_wc, d);
37   float shadow_attenuation = 1.0 - float(isOccluded(CUBE_SHADOW_MAPS[i], shadow_coords));
38   diffuse += LIGHTS[i].diffuse * lambert_term * attenuation * shadow_attenuation;
39   specular += LIGHTS[i].specular * phong_term * attenuation * shadow_attenuation;
40 }
```

Line 19–25, early out which confines the shader input to fragments inside the view volume determined by the near and far planes. Due to optimising far plane testing to efficiently utilise the vector units of stream processors, performance degradations are minimised. Line 36 and 37, the cube map is queried with the inverted direction of light in world-space.

Despite being mainly equivalent, the shader for rendering AreaLight differs from SpotLight in three considerable ways in order to generate visually pleasing soft shadows with contact hardening. First, the spot light iris exhibits a penumbra whose size is varied with respect to the distance to the light source. Second, harsh transitions between specular and diffuse intensities are avoided where the direction of light L becomes orthogonal to the surface normal N. Third, soft shadows are generated by computing a shadow attenuation factor utilising PCSS.

Listing 4.8: Code of the gathering shader for rendering AreaLight.

```
1  //-------------------------------------------------------------------
2  float calcAreaLightShadowAttenuation( in float area_size,
3                                        in float light_near,
4                                        const in sampler2D shadow_map,
5                                        in vec3 shadow_coords );
6  //-------------------------------------------------------------------
7  void AreaLight( in int i, in vec3 N, in vec3 vpos_ec3,
8                  in vec3 shadow_coords, in float material_shininess,
9                  inout vec4 ambient, inout vec4 diffuse, inout vec4 specular )
10 {
11   if ((shadow_coords.z < 0.0) || (shadow_coords.z > 1.0))
12     return;
13   vec3 L = LIGHTS[i].position_ec3 - vpos_ec3;
14   vec3 L_wc = vec3(CAMERA_VIEW_INVERSE_MAT4 * vec4(L, 0));
15   L = normalize(L);
16   float spot_dot = dot(-L, LIGHTS[i].direction_ec3);
17   if (spot_dot <= LIGHTS[i].spot_cos_cutoff)
18     return;
19   float d = length(L_wc);
20   float spot_attenuation = clamp( (spot_dot - LIGHTS[i].spot_cos_cutoff) /
21                                   (LIGHTS[i].area_size * d * 0.2), 0.0, 1.0 );
22   float attenuation = spot_attenuation / ( LIGHTS[i].attenuation[0] +
23                                            LIGHTS[i].attenuation[1] * d +
24                                            LIGHTS[i].attenuation[2] * d * d );
25   ambient += LIGHTS[i].ambient * attenuation;
26   float lambert_term = max(dot(N, L), 0.0);
27   if (material_shininess == 0.0 && lambert_term == 0.0)
28     return;
29   float phong_term_attenuation = 1.0;
```

```
30   if (material_shininess > 0.0)
31   {
32     float lambert_cutoff = min(LIGHTS[i].area_size * d * 0.2, 0.15);
33     if (dot(N, L) < -lambert_cutoff)
34       return;
35     phong_term_attenuation = smoothstep(-lambert_cutoff, lambert_cutoff, dot(N, L));
36   }
37   float phong_term =
38     pow(clamp(dot(N, normalize(L - normalize(vpos_ec3))), 0.0, 1.0 ), material_shininess);
39   phong_term *= phong_term_attenuation;
40   shadow_coords.z = d;
41   float shadow_attenuation =
42     calcAreaLightShadowAttenuation( LIGHTS[i].area_size, LIGHTS[i].near,
43                                     SHADOW_MAPS[i], shadow_coords );
44   diffuse += LIGHTS[i].diffuse * lambert_term * attenuation * shadow_attenuation;
45   specular += LIGHTS[i].specular * phong_term * attenuation * shadow_attenuation;
46 }
```

Line 20 and 21, a penumbra of the spot light iris is simulated by scaling the spot attenuation with respect to the area size and the distance to the light source. Therefore, the AreaLight has two cones of illumination (inner and outer). The intensity of light decreases from the inner cone towards the outer cone to create an area of penumbra. Line 27–39, the PHONG term is smoothly attenuated to simulate symmetric penumbrae along lines where the LAMBERT term abruptly falls off to zero. Accordingly, penumbrae are scaled with respect to the area size and the distance to the light source. Line 27 and 28, the early out ensures that the according computation is only applied for surfaces which exhibit shininess. Line 41–43, compute a smooth shadow attenuation from querying a single shadow map multiple times.

Visually pleasing and perceptually-correct soft shadows are rendered for AreaLight utilising PCSS which samples a single shadow map multiple times. For efficiently hiding banding artefacts in penumbrae with noise, PCF is extended with non-uniform POISSON disk sampling (Section 3.4.1). A POISSON disk sampling pattern is randomly rotated per fragment to implement stratified sampling [Isi06]. Several differently sized sampling patterns are pre-computed [DH06]. The generated sampling patterns are mapped to the unit circle [SC97] for preserving distances between the samples while rotating the kernel. Finally, the patterns are added statically to the code base of Apelles. Therefore, costly generation of sampling patterns during initialisation or execution of Apelles is circumvented.

The PCSS shader computes the shadow attenuation factor in three steps. First, a blocker search averages neighbouring depths which are closer to the light (blocker depth) than the depth of the point currently being shaded (receiver depth). Second, using a planar planes approximation the penumbra width is estimated based on the area size and the distance between occluder and receiver. Finally, the penumbra width is used to scale the filter width of a PCF kernel per fragment. For both blocker search and PCF non-uniform POISSON disk sampling is applied. The shader is parameterised concerning different PCSS quality settings which specify the number of samples for both blocker search and PCF. Accordingly, the ShaderManager creates three different PCSS shaders to provide three PCSS qualities (low, medium and high, Table 4.1). The filter kernel sizes of the settings were chosen in accord with experiments. These experiments evaluated if practical results are achieved concerning both performance and quality (Table 5.1).

As a consequence of non-uniform POISSON sampling, low quality settings easily introduce noise. However, noise is less objectionable and, therefore, effectively hides banding artefacts which are caused by a low number of samples. Furthermore, with higher quality settings banding artefacts are effectively hidden if penumbrae are stretched along surfaces due to oblique light.

For computing the shadow attenuation factor, the PCSS shader requires the specification of the POISSON disk kernel offsets for both blocker search and PCF, the shadow coordinates of the fragment, the area size and the distance of the near plane.

PCSS quality	Blocker search	PCF kernel
Low	4×4	4×4
Medium	6×6	8×8
High	8×8	16×16

Table 4.1: AreaLight PCSS quality settings. AreaLight supports to define three different PCSS quality settings. While soft shadow quality is increasing from low to high, performance decreases dramatically due to the increased number of shadow map accesses (PCF samples).

Listing 4.9: Code of the PCSS shader for rendering AreaLight.

```
1  //-----------------------------------------------------------------
2  const int BS_KERNEL_SIZE = 6;
3  const int PCF_KERNEL_SIZE = 8;
4  const int BS_POISSON_DISK_SIZE = BS_KERNEL_SIZE * BS_KERNEL_SIZE;
5  const int PCF_POISSON_DISK_SIZE = PCF_KERNEL_SIZE * PCF_KERNEL_SIZE;
6  uniform vec2 BS_POISSON_DISK[BS_POISSON_DISK_SIZE];
7  uniform vec2 PCF_POISSON_DISK[PCF_POISSON_DISK_SIZE];
8  //-----------------------------------------------------------------
9  float PCSS( in sampler2D shadow_map, in vec3 shadow_coords,
10             in int blocker_search_kernel_size, in int pcf_kernel_size,
11             in float light_size, in float light_near )
12 {
13   // Step 1: blocker search
14   float avg_blocker_depth = 0.0; int num_blockers = 0;
15   findBlockerRandomly( shadow_map, shadow_coords, blocker_search_kernel_size,
16                        light_size, light_near, BS_POISSON_DISK,
17                        avg_blocker_depth, num_blockers );
18   if (num_blockers == 0)
19     return 0.0;
20   // Step 2: penumbra width estimation
21   float receiver_depth = shadow_coords.z;
22   float penumbra_width = ((receiver_depth - avg_blocker_depth) / avg_blocker_depth) * light_size;
23   // Step 3: PCF
24   return pcfShadowPoissonRotated( shadow_map, shadow_coords, penumbra_width,
25                                   pcf_kernel_size, PCF_POISSON_DISK );
26 }
27 //-----------------------------------------------------------------
28 float calcAreaLightShadowAttenuation( in float area_size, in float light_near,
29                                       in sampler2D shadow_map, in vec3 shadow_coords )
30 {
31   return 1.0 - PCSS( shadow_map, shadow_coords, BS_KERNEL_SIZE, PCF_KERNEL_SIZE,
32                      area_size, light_near );
33 }
```

Line 2 and 3, the number of samples of both blocker search and PCF is modified by the ShaderManager to create three shaders with respect to three PCSS quality settings (low, medium and high). Line 4–7, specifies the two-dimensional positions of the samples being distributed with respect to a POISSON disk sampling pattern across the filter window. The samples reside on the unit circle. Line 18 and 19, early out to skip penumbra width estimation and costly PCF if no blocker has been found. Line 22, in order to ensure numerical stability, the receiver depth must be the distance to the light.

The blocker search averages depths which are closer to the light than the depth of the fragment currently being shaded (receiver). The average depth is assumed to be the depth of a planar occluder along the line from the receiver to the centre of the planar and parallel light source. For each fragment a POISSON disk kernel is randomly rotated upon sampling several depths from the shadow map. Despite being specified since version 1.0 of the GLSL, noise functions to instantly generate per-fragment random numbers are still missing on current hardware platforms. As a workaround to establish a random rotation transformation to rotate the POISSON kernel, random numbers are pre-computed and stored in an offscreen buffer for efficient access during shader execution [Isi06]. As a consequence, two TAUs are required to render the gather pass of an AreaLight. This is an unattractive property for LightRenderJobs where each Light shall occupy a single TAU only. However, pseudorandom numbers are generated instantly in the PCSS shader without losing performance compared to fetching pre-computed random numbers from a texture. Therefore, rendering a LightRenderJob only requires as many free TAUs as Lights are rendered. The number of available TAUs constrains the partition of several single-sample Lights into LightRenderJobs.

Listing 4.10: Code of the non-uniform blocker search of the PCSS shader.

```
1  //-----------------------------------------------------------------
2  // Generates pseudorandom numbers
3  float rand(vec2 v)
4  {
5    return fract(sin(dot(v, vec2(12.9898, 78.233))) * 43758.5453);
6  }
7  //-----------------------------------------------------------------
```

```
8  // Blocker search
9  void findBlockerRandomly( in sampler2D shadow_map, in vec3 shadow_coords,
10                           in int kernel_size, in float light_size, in float light_near,
11                           in vec2 poisson_disk[BS_POISSON_DISK_SIZE],
12                           out float avg_blocker_depth, out int num_blockers )
13 {
14   avg_blocker_depth = 0.0;
15   num_blockers = 0;
16   float receiver_depth = shadow_coords.z;
17   float search_width = light_size * ((receiver_depth - light_near) / receiver_depth);
18   float step_size = 2.0 * search_width / float(kernel_size);
19   float phi = 2.0 * 3.14159265 * rand(vec2(gl_FragCoord));
20   mat2 rotation_mat2 = mat2(cos(phi), -sin(phi), sin(phi),  cos(phi));
21   float blocker_sum = 0.0;
22   for (int i = 0; i < kernel_size; i++)
23     for (int j = 0; j < kernel_size; j++)
24   {
25     vec2 offset = (search_width + step_size / 2.0) *
26                   vec2(rotation_mat2 * poisson_disk[i * kernel_size + j]);
27     float blocker_depth = texture2D(shadow_map, vec2(shadow_coords) + offset).r;
28     if (blocker_depth < receiver_depth)
29     {
30       blocker_sum += blocker_depth;
31       num_blockers++;
32     }
33   }
34   avg_blocker_depth = blocker_sum / float(num_blockers);
35 }
```

Line 5, compute pseudorandom numbers in the interval $[0,1]$[1]. Line 17, the size of the area of the shadow map around the current fragment being searched for blockers is computed using similar triangles. Line 19 and 20, compute a rotation matrix for randomly rotating samples on the unit circle. Line 21–33, sample depths from the shadow map using a randomly rotating POISSON disk sampling pattern. Line 34, compute the blocker depth by only averaging depths which are closer to the light than the depth of the receiver.

Similar to the non-uniform blocker search, a randomly rotating POISSON disk sampling pattern is used upon sampling the shadow map to perform PCF. The filter width is varied with respect to the previous penumbra width estimation to compute a varying shadow attenuation factor for rendering soft shadows which exhibit contact hardening.

Listing 4.11: Code of non-uniform PCF of the PCSS shader.

```
1  float pcfShadowPoissonRotated( in sampler2D shadow_map, in vec3 shadow_coords,
2                                 in float filter_width, in int kernel_size,
3                                 in vec2 poisson_disk[PCF_POISSON_DISK_SIZE] )
4  {
5    float step_size = 2.0 * filter_width / float(kernel_size);
6    float phi = 2.0 * 3.14159265 * rand(vec2(gl_FragCoord));
7    mat2 rotation_mat2 = mat2(cos(phi), -sin(phi), sin(phi),  cos(phi));
8    float shadow_att = 0.0;
9    for (int i = 0; i < kernel_size; i++)
10     for (int j = 0; j < kernel_size; j++)
11   {
12     vec3 offset =  (filter_width + step_size / 2.0) *
13                    vec3(rotation_mat2 * poisson_disk[i * kernel_size + j], 0.0);
14     shadow_att += float(isOccluded(shadow_map, shadow_coords + offset));
15   }
16   return shadow_att / float(kernel_size * kernel_size);
17 }
```

Line 6 and 7, establish the rotation transformation for randomly rotating the POISSON disk kernel upon sampling the shadow map (Line 13). Line 8–16, compute the shadow attenuation factor by averaging the result of multiple occlusion tests (PCF) instead of incorrectly averaging depths.

[1] Source: unknown. For further information on texture-free noise in GLSL, refer to https://github.com/ashima/webgl-noise (last access 2012-05-07).

4.1.3.2 Lighting Computation of Multi-Sample Lights

HeavyRenderJob computes lighting for one multi-sample Light being sampled with several spot lights (SampledAreaLight). The ShaderManager creates several copies of the gathering shader with respect to the number of samples for efficiently switching between shaders for HeavyRenderJobs with a varying number of samples. The number of samples being renderable in a single pass is constrained by three bounds: the number of unoccupied TAUs, the size of free texture memory and the capabilities of the programmable fragment processor. As a consequence of using a texture array to aggregate the shadow maps of the spot lights, the gather pass of a HeavyRenderJob is rendered while occupying a single TAU only. Hence, the number of occlusion tests performable in a single gather pass is independent from the number of unoccupied TAUs. Of course, generating accurate soft shadows demands a high number of samples. The shadow map array easily consumes more video memory than available. Furthermore, the compiler of the GLSL potentially fails upon assembling the fragment program for a large number of samples. Therefore, *ping-pong* offscreen rendering is utilised to accumulate the result of several gather passes each rendering a subset of samples only. In order to keep the memory footprint of the shadow map array at a reasonable size, scattering is divided into several passes too. Experiments showed that rendering eight samples per pass achieves peak performance for both scattering and gathering (Section 5.2.5).

Listing 4.12: Code of the gathering shader for surface colour computation of HeavyRenderJob (SampledAreaLight).

```
//------------------------------------------------------------------
#extension GL_EXT_texture_array : enable
const int AREA_LIGHT_SAMPLE_COUNT = 1;
uniform mat4 CAMERA_VIEW_INVERSE_MAT4;
//------------------------------------------------------------------
struct SampledAreaLight
{
  vec4 ambient, diffuse, specular;  // color
  vec3 attenuation;                 // constant, linear, quadratic
// transformed by the upper 3x3 of the modelview matrix
//   and normalised
  vec3 direction_ec3;
  float spot_cos_cutoff;
};
uniform SampledAreaLight AREA_LIGHT;
uniform vec3 AREA_LIGHT_SAMPLES_POS_EC3[AREA_LIGHT_SAMPLE_COUNT];
uniform mat4 AREA_LIGHT_SHADOW_MATRICES[AREA_LIGHT_SAMPLE_COUNT];
uniform sampler2DArray AREA_LIGHT_SHADOW_MAP;
uniform float PASS;
uniform sampler2D PREVIOUS_PASS_TEX;
uniform int SCREEN_WIDTH;
uniform int SCREEN_HEIGHT;
uniform bool RENDER_MODE_IS_PROGRESSIVE;
//------------------------------------------------------------------
void combineResults(sampler2D previous_pass, inout vec4 this_pass)
{
  vec2 frag_coord = vec2( gl_FragCoord.x / float(SCREEN_WIDTH),
                          gl_FragCoord.y / float(SCREEN_HEIGHT) );
  if (PASS == 1.0 && RENDER_MODE_IS_PROGRESSIVE)
    this_pass = float(AREA_LIGHT_SAMPLE_COUNT) / float(AREA_LIGHT_SAMPLE_COUNT+1) * this_pass +
                (1.0 / float(AREA_LIGHT_SAMPLE_COUNT+1)) * texture2D(previous_pass, frag_coord);
  else
    this_pass = (1.0 / (PASS + 1.0)) * this_pass +
                (PASS / (PASS + 1.0)) * texture2D(previous_pass, frag_coord);
}
//------------------------------------------------------------------
void computeLighting( in vec3 N, in vec4 vpos_ec, in vec4 material_ambient,
                      in vec4 material_diffuse, in vec4 material_specular,
                      in float material_shininess, out vec4 color )
{
  vec3 shadow_coords[AREA_LIGHT_SAMPLE_COUNT];
  for (int sample = 0; sample < AREA_LIGHT_SAMPLE_COUNT; sample++)
  {
    vec4 shadow_coords_ = AREA_LIGHT_SHADOW_MATRICES[sample] * CAMERA_VIEW_INVERSE_MAT4 * vpos_ec;
    shadow_coords[sample] = vec3(shadow_coords_ /shadow_coords_.w);
```

```
46   }
47   vec3 vpos_ec3 = vec3(vpos_ec) / vpos_ec.w;
48   vec4 amb, diff, spec = vec4(0.0);
49   for (int sample = 0; sample < AREA_LIGHT_SAMPLE_COUNT; sample++)
50     AreaLightSample( sample, N, vpos_ec3, shadow_coords[sample],
51                      material_shininess, amb, diff, spec );
52   color = amb * material_ambient + diff * material_diffuse + spec * material_specular;
53   combineResults(PREVIOUS_PASS_TEX, color);
54 }
```

Line 2, enable to use texture arrays in a shader written against the GLSL version 1.20. This is supported on all Shader Model 4 hardware platforms and beyond. Line 3, the ShaderManager increases `AREA_LIGHT_SAMPLE_COUNT` for each copy of the gathering shader to enable rendering HeavyRenderJobs with a varying number of samples. Line 6–15, declare a struct and a uniform, respectively, for aggregating properties of SampledAreaLight which are equal for all samples. Line 16, declare the uniform array storing the positions of the spot lights on the planar surface of the SampledAreaLight. Line 17 and 18, declare the array aggregating the shadow matrices of the samples and the array of shadow maps. Line 19–35, combine the result of the previous gather pass and the current gather pass using ping-pong offscreen rendering. The combination ensures that previously rendered and current results sum up to one always. This is important for displaying intermediate results if progressive rendering is used. Line 29, branch is necessary because pass 0 renders a single hard shadow only (progressive rendering). For example, if a maximum of eight samples are rendered per pass, pass 0 renders and displays a single sample. Pass 1 renders seven samples but displays the result of eight samples. Pass 2 renders eight samples but displays the result of sixteen samples, etc. Line 41–46 compute the shadow coordinates for performing the occlusion tests of the samples (Equation 3.16). Line 49–51, loop over all samples of the HeavyRenderJob and compute the ambient, diffuse and specular terms. Line 53, finally combine the previous and current surface colour such that the contributions sum up to one. This allows to display intermediate results if multiple passes are rendered (progressive rendering).

The shader for rendering a sample of a SampledAreaLight differs from the shader for rendering a SpotLight in two ways only. First, the computed shadow attenuation factor is only a fraction of the overall attenuation. Second, spot attenuation is not evaluated because SampledAreaLight lacks an exponent property.

Listing 4.13: Code of rendering a sample of SampledAreaLight.

```
1  void AreaLightSample( in int sample, in vec3 N, in vec3 vpos_ec3,
2                        in vec3 shadow_coords, float material_shininess,
3                        inout vec4 ambient, inout vec4 diffuse,
4                        inout vec4 specular )
5  {
6    [...]
7    vec4 shadow_coords_;
8    shadow_coords_.x = shadow_coords.x;
9    shadow_coords_.y = shadow_coords.y;
10   shadow_coords_.z = float(sample);
11   shadow_coords_.w = d;
12   float shadow_attenuation = (1.0 - float(isOccluded(AREA_LIGHT_SHADOW_MAP, shadow_coords_))) /
13                              float(AREA_LIGHT_SAMPLE_COUNT);
14   diffuse += AREA_LIGHT.diffuse * lambert_term * attenuation * shadow_attenuation;
15   specular += AREA_LIGHT.specular * phong_term * attenuation * shadow_attenuation;
16 }
```

Line 7–11, the z coordinate of the shadow coordinates corresponds to the layer index in the shadow map array. Line 12 and 13, occlusion testing uses texture2DArray() accordingly to query the shadow map array. The result of the occlusion test only contributes a fraction to the overall occlusion with respect to the number of samples evaluated in a gather pass.

4.1.3.3 Combining Existing Shaders

Combining the shaders of Apelles with existing shaders is based on the observation that two shader objects of the same type are combined by merging the source strings. The source of the shaders used by Apelles is merged with the source of user shaders to yield a new shader program with a single main() routine (extended shader). The merging process essentially consists of two steps. First, code of the user shader before and after main() is extracted and inserted before main() of the library shader. Second, the body of main() of the user shader is extracted and appended to the body of main() of the library shader. In particular, the merging process is considerably facilitated by having only a single implementation of the main shaders for all Lights (Listing 4.2). In particular, it has to be noted that user shaders must not declare any varying variables which conflict with variable declarations of the library shaders. Of course, several user shaders are combinable in lockstep with a single library shader. However, only shaders which are written against the GLSL version 1.20 are combinable with library shaders. For an example of how to combine existing shaders with library shaders see *Examples and Tutorial* (Section 4.2.2).

According to vertex shaders, appending the body of main() of a user shader to the body of main() of the library vertex shader always yields valid vertex shader source. This is a direct consequence of the fact that `gl_Position` must be assigned to at least once but can be assigned to multiple times; the last assignment statement is effective. Of course, the user shader must ensure to output modified vertex positions and normalised normals to the varying variables of Apelles: `vpos_ec_` and `N_`, respectively.

According to fragment shaders, the colour computed by the library fragment shader and the colour computed by the user fragment shader shall be combined accordingly. The library shader essentially computes a surface colour for light mapping. Therefore, the colour computed by a user shader is modulated with the surface colour computed by the library shader. If several user shaders are combined in lockstep with a library shader, the merging process modulates the colour contributions in lockstep as well. Of course, for increased flexibility several user fragment shaders shall sometimes be combined differently before modulating the resulting colour with the colour computed by Apelles. Therefore, the auxiliary library libGLSL exposes a SourceMerger. Upon merging fragment shaders the SourceMerger allows to specify four different combination functions: *replace*, *modulate*, *decal* and *add* (Section 4.1.5).

Once created, the extended shaders are enabled explicitly by the user at the appropriate points in the code of the render callback. Therefore, extended shaders must be created for gather as well as scatter passes, if necessary. Furthermore, extended shaders must be created for a different number of single-sample Lights and for a different number of samples for multi-sample Lights. Additionally, the values of the uniform variables of the library shaders need to be transferred to the extended shaders before using them. Therefore, the auxiliary library libGLSL exposes a UniformSnapshot. For transferring values of uniform variables the UniformSnapshot allows to pull values from a library shader and push the values to an extended shader. In order to initiate value transfer of uniform variables, the ShaderManager notifies an observer if the uniforms of a library shader changed by utilising the *Observer Pattern* [GHJV94].

4.1.4 Optimisation of Light View Volume Culling

The setup of the view volume of a light source requires to specify the distances of the near plane and the far plane. If the associated shadow map contains light-clip-space depth values, the minimum and maximum depth values correspond to the optimal near and far plane distances. As a consequence, the view volume of the light tightly encloses geometry as seen from the viewpoint of the light (Section 3.5). Accordingly, the LampLighter allows to optimise the near and far plane distances for all types of light sources. After creation of a light source, the distance of the near plane equals 0.1 and the distance of the far plane equals 10000. A subsequent optimisation of the near and far plane distances with simplified geometry considerably improves the efficiency of upcoming light view volume culling. However, the optimisation involves to render light-clip-space depth into the shadow map. Therefore, after tightening the near and far plane distances, the shadow map is updated implicitly with depth being equal to the distance of the light source as required by occlusion testing in gathering. Of course, the optimisation is mostly beneficial if the light is static and dynamic objects are ensured to entirely remain within the view volume of light always.

The minimum and maximum is efficiently computed utilising an OpenCL-enabled *Parallel Reduction* (Section 5.2.6). First, using the current near and far plane distances of the light source, light-clip-space depth is rendered to the shadow map. The implementation of shadow mapping in Apelles utilises the distance to the light source. This restrains deriving the optimal near and far plane distances directly from the minimum and maximum depth values in the shadow map. Second, the shadow map is directly served as input to the OpenCL kernel to avoid costly data transfers between CPU and GPU. Accordingly, the extension `CL_KHR_gl_sharing` is utilised to establish the necessary association of an OpenGL context and an OpenCL context as specified by OpenCL version 1.1 [OCL10]. The two-dimensional problem domain (square 2D texture) is partitioned into numerous square blocks of equal size. Accordingly, for each of the square blocks the minimum and maximum is determined in parallel. The maximum size of a square block is constrained by the limits of the OpenCL implementation (underlying hardware platform). As a consequence of a limited block size, the

parallel reduction requires several passes depending on the size of the shadow map. Third, subsequent passes reduce an array. After the first pass the problem domain becomes one-dimensional. Finally, once the parallel reduction has terminated after several passes, only the minimum and maximum is transferred to host memory.

For example, assuming a maximum block size of 512, the reduction of a texture 2048×2048 texels large, which are partitioned into square blocks of 16×16 elements, yields 16384 minima and maxima, respectively. In particular, it has to be noted that the maximum block size is unusable in the first pass because mapping 512 elements to a square block is impossible. However, after the first pass of the reduction (texture) the problem domain becomes one-dimensional (array). Therefore, the maximum block size of 512 is fully usable in subsequent passes. Hence, the reduction requires three passes to terminate: $4194304 \rightarrow 16384 \rightarrow 32 \rightarrow 1$. In the last pass, a single multiprocessor reduces the remaining elements to yield the minimum and maximum.

The computation of minimum and maximum using an OpenCL-enabled parallel reduction turns down complexity from $O(n)$ to $O(log\, n)$ (Figure 4.7). The algorithm reduces numerous values to a single value in parallel to yield the minimum and the maximum, respectively. The values are partitioned into multiple work-groups which are assigned to the available multiprocessors of the underlying hardware platform. The large number of work-groups ensures a high occupancy of the multiprocessors which effectively hides memory latency. The minimum and maximum of a work-group is computed in a tree-based approach iteratively. Accordingly, in each work-group work-items are executed in parallel in groups of warps. Each warp executes work-items in lockstep (SIMD). Synchronisation barriers are utilised to avoid data anti-dependencies between warps (read-before-write). For improved performance the input values of a work-group are loaded into local memory. In order to take advantage of coalescing, adjacent work-items read and write adjacent memory addresses.

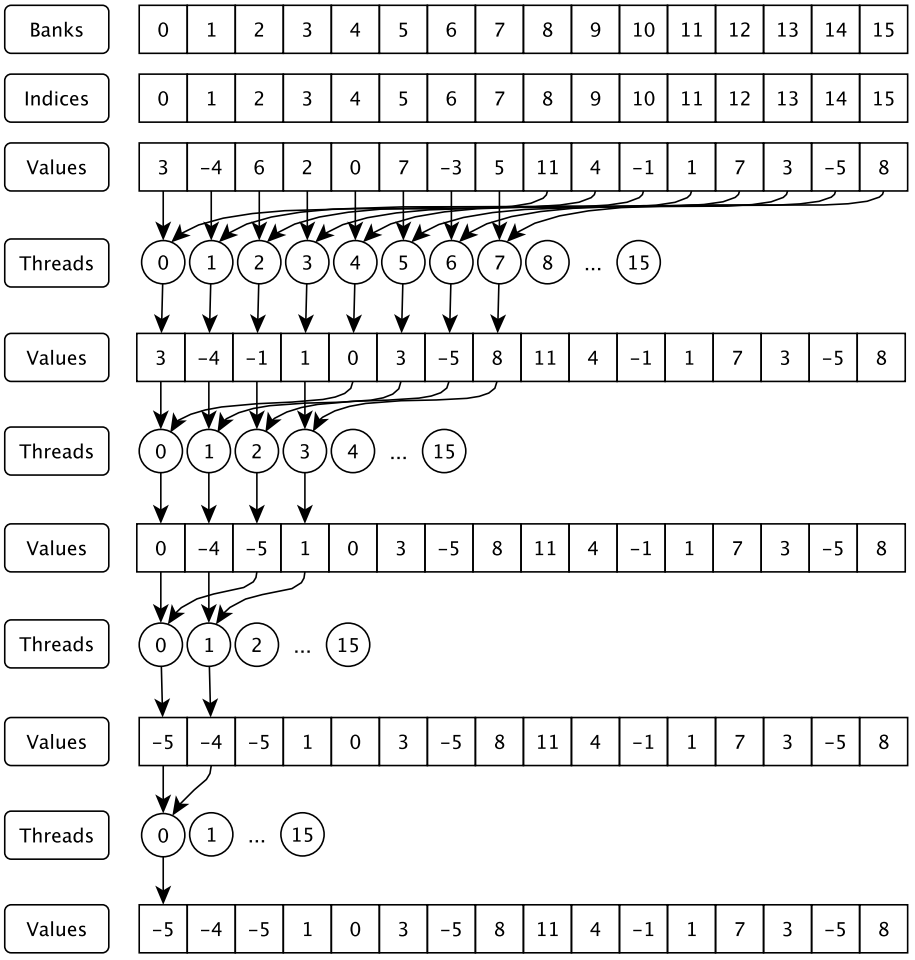


Figure 4.7: Parallel reduction. An example of a tree-based reduction to yield the minimum within a work-group of 16 elements in four iterations. In each iteration the number of elements is reduced by a factor of two to finally yield the minimum. Within the work-group, 16 work-items are executed in lockstep (threads). In iteration one, adjacent work-items 0...7 use sequential addressing of 16 elements to take advantage of coalescing while avoiding divergence. The work-items 8...15 are idle to avoid memory bank conflicts if several work-items access the same bank.

However, accesses to local memory are serialised if several work-items within a warp access the same memory bank (bank conflict). A warp consists of two half-warps. A half-warp executes as many work-items in lockstep as memory banks are available. However, a single work-item needs to access two memory banks which infers conflicts. As a consequence of sequential addressing, memory access of work-items within a half-warp is ensured to be conflict-free.

Listing 4.14: Code of the OpenCL kernel which determines the minimum and maximum from a two-dimensional float texture with a parallel reduction.

```
1  //------------------------------------------------------------------
2  const sampler_t sampler = CLK_NORMALIZED_COORDS_FALSE | CLK_ADDRESS_CLAMP_TO_EDGE |
3                            CLK_FILTER_NEAREST;
4  //------------------------------------------------------------------
5  __kernel void minmaxTexture2D( read_only image2d_t texture,
6                                 __global float *od_min, __global float *od_max,
7                                 __local float *lmin, __local float *lmax )
8  {
9    int gid_x = get_global_id(0);
10   int gid_y = get_global_id(1);
11   int lid_x = get_local_id(0);
12   int lid_y = get_local_id(1);
13   int lsize = get_local_size(0) * get_local_size(1);
14   int grpid = get_group_id(0) + get_num_groups(1) * get_group_id(1);
15   int2 tex_coords = (int2)(gid_x, gid_y);
16   int lid = lid_x + lid_y * get_local_size(1);
17   float4 texel = read_imagef(texture, sampler, tex_coords);
18   lmin[lid] = texel.x;
19   lmax[lid] = texel.x;
20   barrier(CLK_LOCAL_MEM_FENCE);
21   for (int stride = lsize/2; stride > 0; stride >>= 1)
22   {
23     if (lid < stride)
24     {
25       lmin[lid] = min(lmin[lid], lmin[lid + stride]);
26       lmax[lid] = max(lmax[lid], lmax[lid + stride]);
27     }
28     barrier(CLK_LOCAL_MEM_FENCE);
29   }
30   if (lid == 0)
31   {
32     od_min[grpid] = lmin[0];
33     od_max[grpid] = lmax[0];
34   }
35 }
```

Line 2 and 3, specify the state of the TAU for fetching floating-point values (depths) from the DepthTexture. Line 15–19, load depth from texture memory into local memory. The hardware detects that adjacent device memory is read through an image object (texture) and, therefore, automatically coalesces the reads for optimal performance. Line 20, wait until all work-items of this work-group have finished writing to local memory for circumventing read-before-write data hazards. Line 21–29, do reduction in local memory using sequential addressing which coalesces consecutive reads and writes of adjacent work-items for optimal performance. Hence, divergence of work-items and bank conflicts are prevented. Line 21, according to sequential addressing, stride computation utilising the bitwise operator avoids costly operations (integer division or modulo). Line 28, wait until all work-items finished the current iteration. As a consequence, work-items which are already done in the current iteration are prevented to read from local memory that has not been written to by other work-items being in the current iteration (read-before-write). Line 30–34, the first work-item of this workgroup writes the result of this workgroup to global memory.

4.1.5 Auxiliary Libraries

The auxiliary libraries necessary to use Apelles are closely implemented in accord with Apelles. No additional third-party libraries are required for both developing new applications or integrating Apelles into existing applications. As a consequence, applications using Apelles are independent from additional libraries. Of course, other well developed libraries exist which provide similar functionality. However, at the time Apelles was developed, some libraries were unavailable (libGLMath). Furthermore, the auxiliary libraries can independently be changed. Newly required features can be added instantly. Existing features can be corrected instantly in case of recently identified defects.

libGLSL An object-oriented framework for handling shader and program objects of the GLSL. Shader objects can be created from strings or files, respectively. UniformBroker supports in conveniently querying and setting active uniform variables of a shader. UniformSnapshot enables to save and restore the values of all active uniform variables of a shader program. Furthermore, UniformSnapshot facilitates managing the values of active uniform variables of different shader programs which share sets of active uniforms. Of course, OpenGL version 3.1 introduced *Uniform Buffers* which follow the same intent. The SourceModifier enables to modify the source of a shader. For example, change the size of arrays or replace the right hand side of an assignment statement. The SourceMerger allows to merge the sources of two shaders of the same type. According to increasing the flexibility upon combining two fragment shaders, how to combine the colours of the two shaders is explicitly specifyable: *replace*, *add*, *decal* and *modulate* (Listing 4.15).

Listing 4.15: Example code of the options upon combining the colours of two fragment shaders.

```
1 vec4 color1, color2;
2 vec4 color_replace = color2;
3 vec4 color_add.rgb = color1.rgb + color2.rgb;
4      color_add.a *= color2.a;
5 vec4 color_decal = vec4(mix(color1.rgb, color2.rgb, color2.a), color1.a);
6 vec4 color_modulate = color1 * color2;
```

libGLMath A header only collection of mathematical classes and operations whose semantics are closely tied to the GLSL version 1.20 which specifies vector and matrix mathematics concisely. As a consequence, uniformity of code in Apelles and in applications integrating Apelles is considerably improved. The library encompasses vector and matrix types, operators and built-in functions except swizzling. Additionally, existing API calls of the GL are extended to accept vector and matrix types naturally. Of course, other well developed alternatives exist[2]. However, an alternative which is closely tied to the GLSL was not available at the time of the implementation of Apelles.

libGLUtil An object-oriented C++ framework that provides facilities for addressing common issues in developing OpenGL applications. TextureFactory aids in the creation of both colour and depth textures. FramebufferObject enables to efficiently render to different types of textures including cube map textures and texture arrays. TextureUnitAcquirer facilitates the management of free TAUs by tracking which TAUs are occupied. FPSCounter captures the frame rate and allows to attenuate undesirable spikes in the acquired data using a *Weighted Moving Average*.

libUtil A framework which supports in developing applications in C++. One of its invaluable components facilitates the memory management of dynamically-allocated shared objects in C++ utilising the *Counted Pointer Idiom* [BMR*96]. Of course, the new C++ standard [C++11] inherently supports reference counting via `unique_ptr`, `shared_ptr` and `weak_ptr`.

[2] OpenGL Mathematics (GLM), http://glm.g-truc.net. Imath which is part of Ilmbase of OpenEXR, http://www.openexr.com/downloads.html (last access 2012-05-07).

4.2 Examples and Tutorial

4.2.1 Simple Example

This example demonstrates how easily Apelles is setup for rendering an image exhibiting shadows given some geometry and some light sources (Figure 4.8). The scene consists of a plane which serves as a shadow receiver. An object hovers above the plane to serve as a shadow caster (occluder). Note how important shadows are to generate the correct visual impression of a hovering object. In the scene are three light sources: a global directional light, a point light and an area light sampled with 512 spot lights.

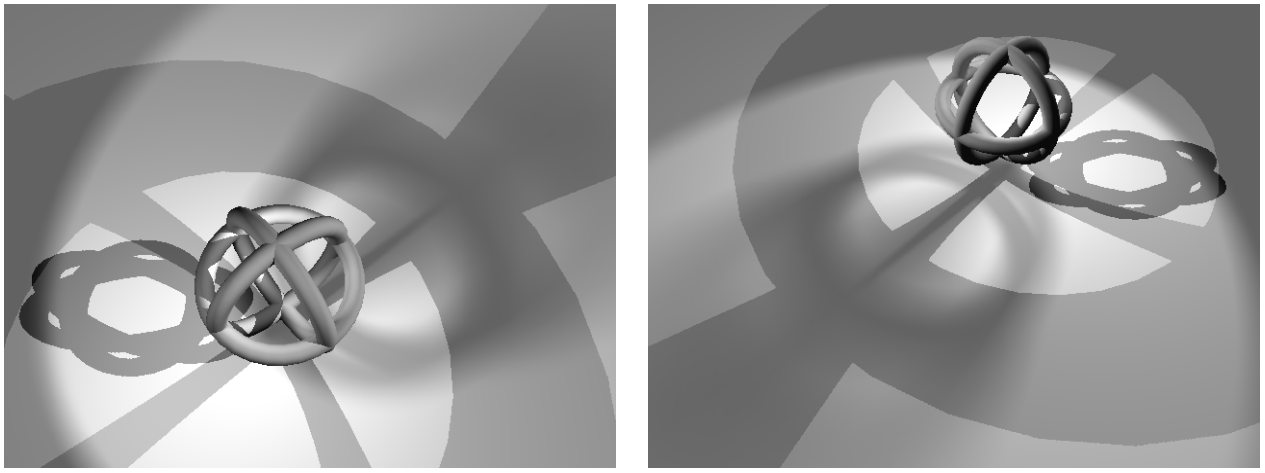


Figure 4.8: Apelles example simple. An object consisting of three tori is illuminated by three light sources: a global directional light, a point light and an area light. The point light casts an omnidirectional shadow from inside the object which introduces a circular-shaped hard shadow with a cross in the middle. The area light is sampled with 512 spot lights being randomly distributed on the area of the light source with respect to a POISSON disk distribution. Note that the object exhibits self-shadowing.

The first step is to initialise Apelles. Of course, Apelles requires a render context with a depth buffer to allow depth testing. Then, the light sources are created. Finally, rendering is initiated by calling the function display() which is assumed to render a single frame. The function draw() is assumed to issue the drawing commands (rendering callback). Of course, since Apelles uses shadow mapping, shadows are rendered for all geometric primitives for which depth values are generated by the rasteriser in both scattering and gathering (triangles, quads, lines and points). As a consequence of using progressive rendering, display() needs to be called repeatedly until the final frame has been rendered. For example, if GLUT is used, glutDisplayFunc() is only called if the window requires to be painted. Therefore, GLUT needs additionally to be setup for calling display() repeatedly using glutIdleFunc(). Once ping-pong rendering finished, only a screen-sized quad is rendered to display the contents of the ping offscreen buffer.

Listing 4.16: Pseudocode of the simple example using Apelles.

```
1  //-----------------------------------------------------------------
2  #include "Apelles/render_manager.h"
3  #include "Apelles/internal/lamp_lighter.h"
4  using namespace Apelles_;
5  using namespace Apelles_::Internal_;
6  RenderManager *render_manager;
7  LightManager *light_manager;
8  LampLighter *lamp_lighter;
9  //-----------------------------------------------------------------
10 void init()
11 {
12     glClearDepth(1.0);
13     glEnable(GL_DEPTH_TEST);
14     render_manager = &RenderManager::getInstance();
15     render_manager->setRenderMode(RENDER_MODE_PROGRESSIVE);
16     render_manager->setDepthBiasingEnabled(GL_FALSE);
17     render_manager->setSecondDepthShadowMappingEnabled(GL_TRUE);
18     light_manager = &render_manager->getLightManager();
```

```
19    lamp_lighter = &LampLighter::getInstance();
20    createGlobalLight();
21    createPointLight();
22    createSampledLight();
23 }
24 //------------------------------------------------------------------
25 void draw() {...}
26 //------------------------------------------------------------------
27 void display()
28 {
29    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
30    establishViewingTransformation();
31    establishProjectionTransformation();
32    establishViewportTransformation();
33    render_manager->render(&draw);
34 }
```

Line 2–8, the public interface of Apelles is exposed by the RenderManager and the LightManager. The namespace `Internal_` exposes those guts of Apelles which might be useful to the user additionally. However, while the guts are susceptible to change over time, the public interface is assumed to be a published contract between the user and Apelles (stable interface). Line 8, the LampLighter aids in setting up light sources (Listing 4.17). Line 12 and 13, enable depth testing because it is inevitable for Apelles. Line 14–17 enable progressive rendering and second-depth shadow mapping (closed geometry). As a consequence, front-facing geometry is culled with respect to the direction of light during scattering which speeds up rendering additionally. Note that depth biasing should remain enabled to mitigate incorrect self-shadowing where the geometry exhibits insufficient thickness (low depth disparity). Line 18–22 create the light sources. Line 25, draw() issues drawing commands without compromising the state of the GL setup by Apelles in both scattering and gathering. Line 30–32 essentially puts-up a camera in the scene. Apelles computes the inverse viewing matrix implicitly for reprojecting from eye-space to light-space (Equation 3.16). Line 33, initiate rendering which calls the function draw() multiple times for both scattering and gathering. In scattering draw() is called 519 times; once for the global directional light, six times for the point light and 512 times for the area light. In gathering draw() is called 65 times; once for the global directional light and the point light as well as 64 times for the area light (eight samples per pass).

Listing 4.17: Pseudocode of creating the global directional light of the simple example of Apelles.

```
1  void createGlobalLight()
2  {
3    GlobalDirectionalLight *global_light;
4    GLvec3d global_light_dir = normalize(GLvec3d(...));
5    GLvec4d scene_bounding_sphere = GLvec4d(GLvec3d(center), radius);
6    GLvec4f global_light_diffuse = GLvec4f(GLvec3f(...), 1.0);
7    light_manager->createGlobalDirectionalLight(global_light_dir);
8    lamp_lighter->setupLight(global_light, scene_bounding_sphere);
9    global_light->setDiffuse(global_light_diffuse);
10   render_manager->getShadowMap(global_light)->setSize(8192);
11 }
```

Line 7–9, create the global directional light with the specified direction. However, a global light shines throughout the entire scene. Therefore, the LampLighter is used to setup the view volume of the global light (cube) to encompass the bounding sphere of the scene (Line 8). As a consequence, changing the direction of light essentially rotates the view volume around the scene. Line 10, increase the size of the shadow map to 8192×8192 for reducing aliasing artefacts.

Listing 4.18: Pseudocode of creating the point light of the simple example of Apelles.

```
1  void createPointLight()
2  {
3    PointLight *point_light;
4    GLvec3d point_light_pos = GLvec3d(...);
5    GLfloat point_light_near = ...;
6    GLfloat point_light_far = ...;
7    GLvec4f point_light_diffuse = GLvec4f(GLvec3f(...), 1.0));
8    point_light = light_manager->createPointLight(point_light_pos);
9    lamp_lighter->tightenNearAndFar(point_light, &draw);
10   point_light->setDiffuse(point_light_diffuse);
11 }
```

Line 3–8, create the point light with the specified position. Line 9, in order to optimise light view volume culling, the LampLighter is used to make the near and far planes of the view volume tightly enclose the geometry seen from the viewpoint of the light source. First, light-clip-space depth is rendered into the cube shadow map of the point light. Then, for each face of the cube map an OpenCL-enabled algorithm determines the minimum and maximum depth. Finally, the optimal near and far plane distances are determined from the six per face minima and maxima, respectively. The optimal near and far plane distances are assigned to the point light. Of course, the shadow map remains invalidated. For occlusion testing in gathering the shadow map is assumed to store the distance to the light (linear depth metric). Hence, the shadow map is implicitly updated with a different depth metric before occlusion testing is applicable.

Listing 4.19: Pseudocode of creating the sampled area light of the simple example of Apelles.

```
1  void createSampledLight()
2  {
3    SampledAreaLight *sampled_light;
4    GLvec3d sampled_light_pos = GLvec3d(...);
5    GLvec3d sampled_light_lookat = GLvec3d(...)
6    GLvec3d sampled_light_dir = normalize(sampled_light_lookat - sampled_light_pos);
7    GLfloat sampled_light_cutoff = ...;
8    GLfloat sampled_light_near = ...;
9    GLfloat sampled_light_far = ...;
10   GLfloat sampled_light_size = ...;
11   GLvec4f sampled_light_diffuse = GLvec4f(GLvec3f(...), 1.0));
12   GLint sample_count = 512;
13   GLvec3d *samples_pos[sample_count];
14   lamp_lighter->createSquareAreaLightSamples( sampled_light_pos, sampled_light_dir,
15                                               samples_pos, sample_count );
16   sampled_light =
17     light_manager->createSampledAreaLight( sampled_light_dir, samples_pos, sample_count,
18                                            sampled_light_cutoff, sampled_light_near,
19                                            sampled_light_far, sampled_light_size );
20   sampled_light->setDiffuse(sampled_light_diffuse);
21 }
```

Line 4–6, derive the normalised direction of light from the position of the light and the look-at position. Line 12–15, using the LampLighter, create the positions of the 512 samples on the planar and square-shaped area of the light source. The samples are randomly distributed with respect to a POISSON disk distribution. As a consequence, the individual hard shadows of the samples unify accordingly to create smooth transitions of penumbrae. Furthermore, the samples are sorted with respect to the increasing distance to the centre of the light source. As a consequence, penumbrae grow naturally from inner to outer penumbra if progressive rendering is enabled. Line 16–19, create the area light being sampled with 512 spot lights. The spot lights have equal direction of light, spot cutoff, near and far plane distances. The area size essentially scales the distance of the samples to the centre of the area of the light source.

4.2.2 Combining Existing Shaders

This example demonstrates how easily Apelles is setup for combining existing shaders (Figure 4.9). The scene consists of an object and a plane (shadow receiver). The object hovers above the plane to serve as a shadow caster (occluder). The vertices of the plane are displaced along the y axis using an existing vertex shader which applies a sinusoidal surface deformation. Additionally, the plane is textured with a grid to enhance the visual impression of surface deformation. In the scene are two light sources: a spot light and an area light. Soft shadows are approximated utilising PCSS to allow for real-time performance. Note how important shadows are to create the correct visual impression of a hovering object.

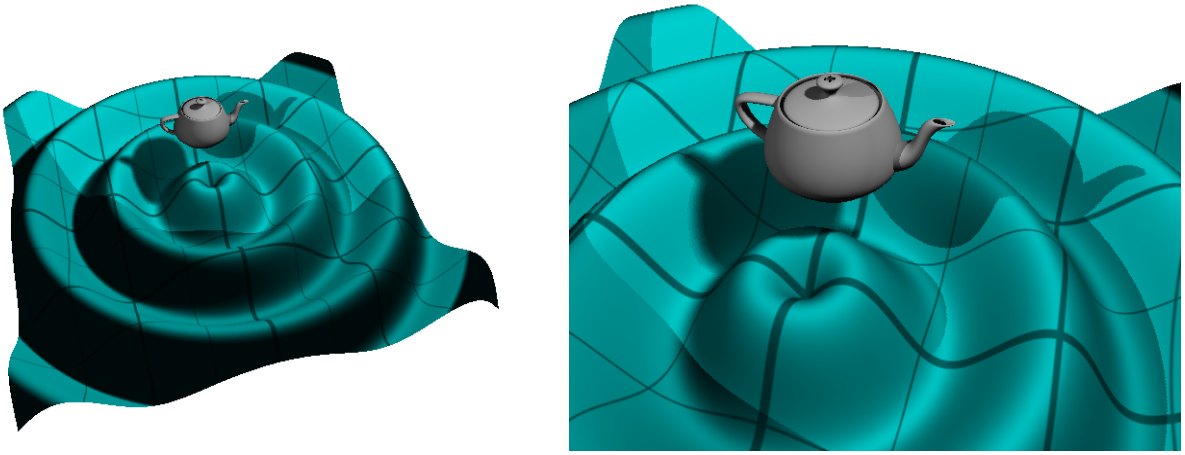


Figure 4.9: Apelles example combining existing shaders. A teapot hovering above a deformed surface is illuminated by two light sources: a spot light and an area light. The deformed surface is generated by displacing the vertices of a plane along the y axis with a sinusoidal deformation function. In order to enhance the visual impression of surface deformation, the deformed surface is covered with a grid texture. The soft shadow of the area light is generated with PCSS utilising a randomly rotating POISSON disk kernel for PCF (quality setting low).

The existing displacement vertex shader is combined with both scattering and gathering vertex shaders of Apelles to create user shaders. First, however, in order to yield valid shader source, the existing shader source needs to be adapted to ensure compatibility with the shaders of Apelles. The existing shader must ensure to output modified vertex positions and normalised normals to the varying variables of Apelles. Second, the library shaders are copied in order to allow for selectively rendering objects with or without vertex displacement. The according gathering vertex shader is copied with respect to the number of light sources and the utilised PCSS quality setting. Finally, using the ShaderManager the adapted vertex shader source is combined with vertex shaders of both scattering and gathering to yield user shaders. Of course, in order to apply the grid texture a simple texturing shader is combined additionally.

Listing 4.20: Pseudocode of creating user shaders by combining existing shaders with the shaders of Apelles.

```
1  void createVDShaders( int light_count, PCSSQuality pcss_quality,
2                        ShaderProgramPtr &vd_depth_prog, ShaderProgramPtr &vd_main_prog )
3  {
4    string vd_vert_src = existing_vertex_displacement_vert->getSource();
5    SourceModifier &sm = SourceModifier::getInstance();
6    findReplace(vd_vert_src, "vec3 vpos_ec3_", "vec4 vpos_ec_");
7    findReplace( vd_vert_src,
8                 "vpos_ec3_ = vec3(gl_ModelViewMatrix * displaced_vertex)",
9                 "vpos_ec_ = gl_ModelViewMatrix * displaced_vertex");
10   sm.deleteVaryingDecl(vd_vert_src, "vpos_ec_");
11   findReplace(vd_vert_src, "n_", "N_");
12   findReplace( vd_vert_src,
13                "N_ = gl_NormalMatrix * displaced_normal",
14                "N_ = normalize(gl_NormalMatrix * displaced_normal)" );
15   sm.deleteVaryingDecl(vd_vert_src, "N_");
16   vd_depth_prog = shader_manager->getDepthProg()->copy();
17   vd_main_prog = shader_manager->getMainProg(light_count, pcss_quality)->copy();
18   vd_depth_prog->link();
19   vd_main_prog->link();
```

```
20   string tex_vert_src = "void main() {"
21                         "gl_TexCoord[0] = gl_MultiTexCoord0; gl_Position = ftransform; }";
22   string tex_frag_src = "uniform sampler2D TEX;"
23                         "void main() { gl_FragColor = texture2D(TEX, gl_TexCoord[0].st); }";
24    shader_manager→use(vd_depth_prog);
25    shader_manager→combine(GL_VERTEX_SHADER, vd_vert_src);
26    shader_manager→use(vd_main_prog);
27    shader_manager→combine(GL_VERTEX_SHADER, tex_vert_src);
28    shader_mamager→combine(GL_FRAGMENT_SHADER, tex_frag_src);
29    shader_manager→combine(GL_VERTEX_SHADER, vd_vert_src);
30    shader_manager→unuse();
31 }
```

Line 4–15, adapt the source of the existing displacement vertex shader to make it compatible with the vertex shaders of Apelles. In the gathering fragment shader Apelles generates texture coordinates for accessing the shadow maps. Therefore, the displaced vertex position must be input as a vector with four components to the fragment shaders of Apelles (Line 6–9). The displaced normal must be input as a normalised vector (Line 11–14). Line 10 and 15, remove conflicting varying declarations because the shaders of Apelles declare position and normal shader inputs already. Line 16–19, copy and link the library shader programs of both scattering and gathering. The gathering program is copied with respect to the utilised PCSS quality setting and the number of light sources being rendered (Line 17). Line 20–23, defines the source of a simple shader for texturing the displaced surface. Line 24 and 25, create the user shader for scattering. Line 26–30, create the user shader for gathering. Note that the adapted vertex shader source is reused for creating the user shaders of both scattering and gathering (Line 25 and 29). According to the scattering user shader, the declaration of the varying variable `N_` is ignored because the scattering shader only declares the vertex position as input (Listing 4.1). Line 28, by default the merging of fragment shaders ensures that the colour of the texture is modulated with the result of the lighting computation of Apelles.

The steps to render an image using Apelles mainly correspond to the steps of the simple example (Section 4.2.1). First, Apelles is initialised. Non progressive rendering is enabled by default. The example uses regular depth biasing instead of second-depth shadow mapping. Second, the user shaders are created. Third, two light sources are created: spot light and area light. Accordingly, specific depth biasing settings are applied to the shadow map of the area light with respect to the size of the light and the spatial configuration of the scene. Fourth, the texture is initialised for covering the deformed surface with a grid. Finally, rendering is initiated by calling the function display(). In function draw() the user shaders are enabled accordingly with respect to both scattering and gathering. The user shaders are only enabled upon drawing the plane. In particular, it has to be noted that the values of the uniform variables of the library shaders must be transferred to the uniform variables of the user shaders (not shown). To do so, an observer is registered on the ShaderManager to get notified if uniforms of library shaders change. Then, at the beginning of draw(), a UniformSnapshot is used to transfer the values from the library shaders to the according user shaders (Section 4.1.5). Of course, uniform transfer must be done always if the uniforms of library shaders change continuously due to animation.

Listing 4.21: Pseudocode of the example demonstrating how to combine existing shaders with the shaders of Apelles.

```
1  //------------------------------------------------------------------
2  #include "Apelles/render_manager.h"
3  using namespace Apelles_;
4  RenderManager *render_manager;
5  LightManager *light_manager;
6  ShaderManager *shader_manager;
7  ShaderProgramPtr vd_depth_prog;
8  ShaderProgramPtr vd_main_prog;
9  TexturePtr grid_tex;
10 //------------------------------------------------------------------
11 void init()
12 {
13    glClearDepth(1.0);
14    glEnable(GL_DEPTH_TEST);
15    render_manager = &RenderManager::getInstance();
16    light_manager = &render_manager→getLightManager();
17    shader_manager = &render_manager→getShaderManager();
18    createVDShaders(2, PCSS_QUALITY_LOW, vd_depth_prog, vd_main_prog);
19    light_manager→createSpotLight();
20    AreaLight *area_light = light_manager→createAreaLight(...);
21    ShadowMap sm = render_manager→getShadowMap(area_light);
```

```
22     sm→setSlopeBias(...);
23     sm→setConstBias(...);
24     render_manager→getTextureUnitAcquirer().acquire(GL_TEXTURE0);
25     initTexture(grid_tex, GL_TEXTURE0);
26 }
27 //------------------------------------------------------------------
28 void draw()
29 {
30   // if necessary, transfer uniforms of library shaders to user shaders
31   if (shader_manager→get() == shader_manager→getDepthProg())
32     shader_manager→use(vd_depth_prog);
33   else shader_manager→use(vd_main_prog);
34   drawMeshPlane();
35   if (shader_manager→get() == vd_depth_prog)
36     shader_manager→use(shader_manager→getDepthProg());
37   else shader_manager→use(shader_manager→getMainProg(2, PCSS_QUALITY_LOW));
38   drawHoveringObject();
39 }
40 //------------------------------------------------------------------
41 void display()
42 {
43     glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
44     establishViewingTransformation();
45     establishProjectionTransformation();
46     establishViewportTransformation();
47     render_manager→render(&draw);
48 }
```

Line 15, initialise Apelles. Depth biasing and non-progressive rendering is enabled by default. Line 18, create the user shaders for applying vertex displacement in both scattering and gathering (Listing 4.20). The number of light sources and the PCSS quality settings must be specified to create the user shader for gathering. Line 19 and 20, create the spot light and the area light. Line 21–23, increase the slope and constant depth bias of the shadow map of the area light for hiding incorrect self-shadowing. Note that the magnitude of depth bias necessary to hide incorrect self-shadowing sufficiently well depends on the size of the area light (filter size) and the current spatial configuration of the scene. Line 24, reserve the first texture addressing unit (TAU) for the grid texture. Hence, Apelles does not use the first TAU for addressing scatter data. Line 25, initialise the grid texture and bind the texture to the first TAU. Line 31–37, enable the user shaders for drawing the plane only and use the library shaders for drawing the hovering object. Line 30, the uniforms of the library shaders must be transferred to the user shaders at least once (not shown). Line 34, note that if a gather pass is rendered, the grid texture is applied to the deformed plane because the user shader performs texturing (Listing 4.20, line 27 and 28). Line 37, if a gather pass is rendered, use the according library shader to render two light sources with a low PCSS quality.

Chapter 5

Results and Discussion

All of the pictures and performance numbers in this section were generated on a dual-core AMD Athlon 6000 X2 machine with a single NVIDIA GeForce 250 GTS running openSUSE 10.2-AMD64 and OpenGL v3.0. Pictures were rendered at a resolution of 640×480 pixels. The performance numbers were recorded while rendering both scattering and gathering.

5.1 Results

Visual Results To demonstrate the shadow generation capabilities of Apelles a detailed model of the Kölner Dom is illuminated with different types of light sources. Figure 5.1 shows the exterior of the model being illuminated by a distant SampledAreaLight. Figure 5.2 shows the interior of the model being illuminated by an indoor PointLight and an outdoor SampledAreaLight. Figure 5.3 shows the interior of the model being illuminated by an indoor PointLight and an outdoor AreaLight. Figure 5.4 shows the exterior and interior of the model being illuminated by an indoor PointLight, an outdoor AreaLight and a GlobalDirectionalLight.

Performance Results Figure 5.1 and Figure 5.2 were rendered at 0.16 frames per second (SampledAreaLight). Figure 5.3 was rendered at 11.04 frames per second. Figure 5.4 was rendered at 10.45 (left) and 8.20 (right) frames per second. With the GlobalDirectionalLight enabled only, the model was rendered at 77 frames per second. When *Second-Depth Shadow Mapping* [WM94] was enabled, performance increased by 30% on average. While decreasing the size of shadow maps, performance increased linearly. While increasing the number of samples or single-sample light sources, performance decreased exponentially. While increasing the complexity of geometry, performance decreased exponentially.

The 512 samples of the SampledAreaLight were rendered with eight samples per pass. The AreaLight was rendered with PCSS quality high (8×8 blocker search, 16×16 PCF). The shadow maps of local light sources were of size 4096×4096. The shadow map of the GlobalDirectionalLight was of size 8192×8192. The mesh of the model of the Kölner Dom consisted of 100k vertices and 200k triangles. Vertex buffer objects were utilised to accelerate vertex processing.

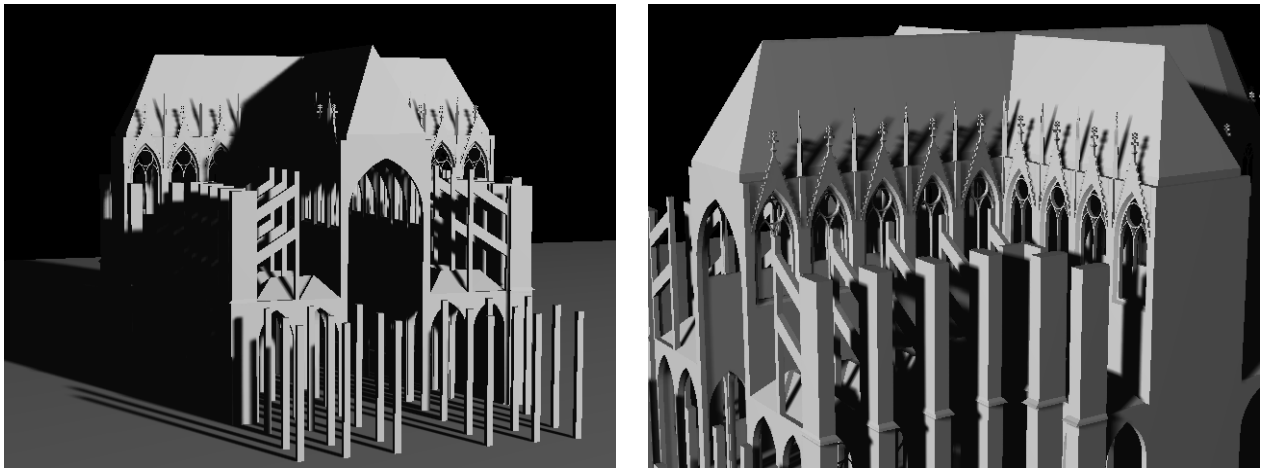


Figure 5.1: Visual Results One. The exterior of a model of the Kölner Dom is illuminated by a distant SampledAreaLight with 512 samples.

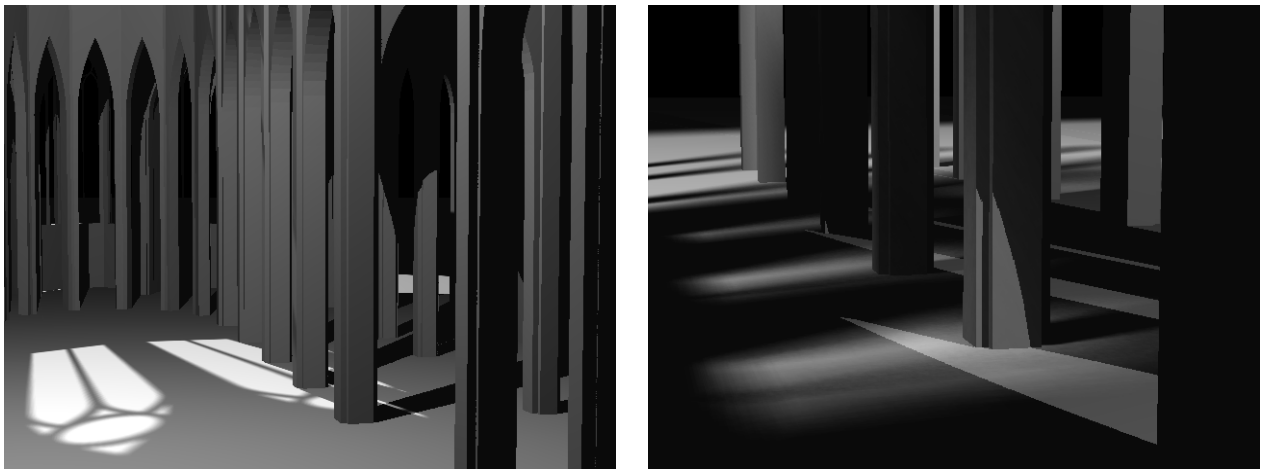


Figure 5.2: Visual Results Two. The interior of a model of the Kölner Dom is illuminated by an indoor PointLight and an outdoor SampledAreaLight with 512 samples.

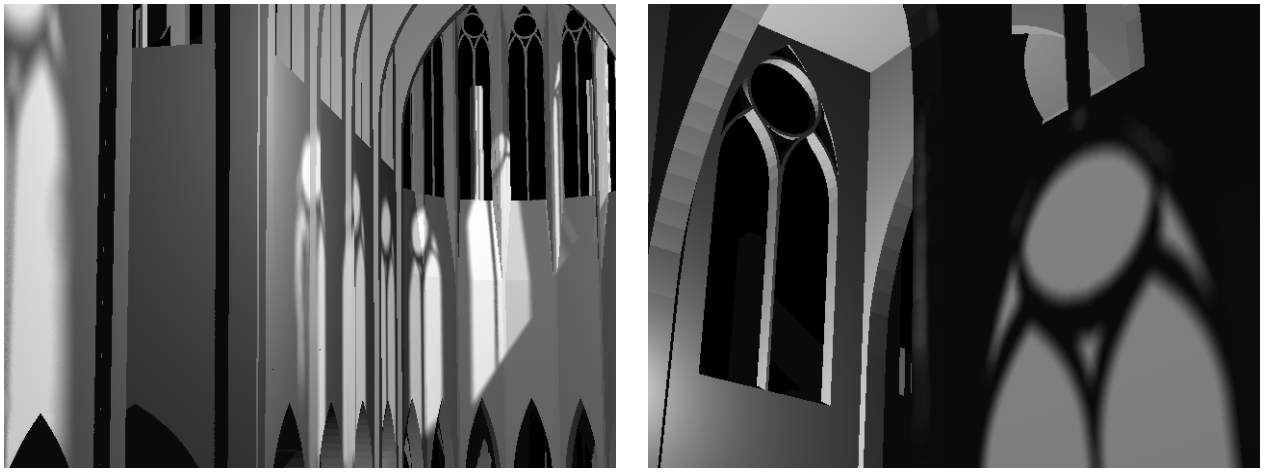


Figure 5.3: Visual Results Three. The interior of a model of the Kölner Dom is illuminated by an indoor PointLight and an outdoor AreaLight.

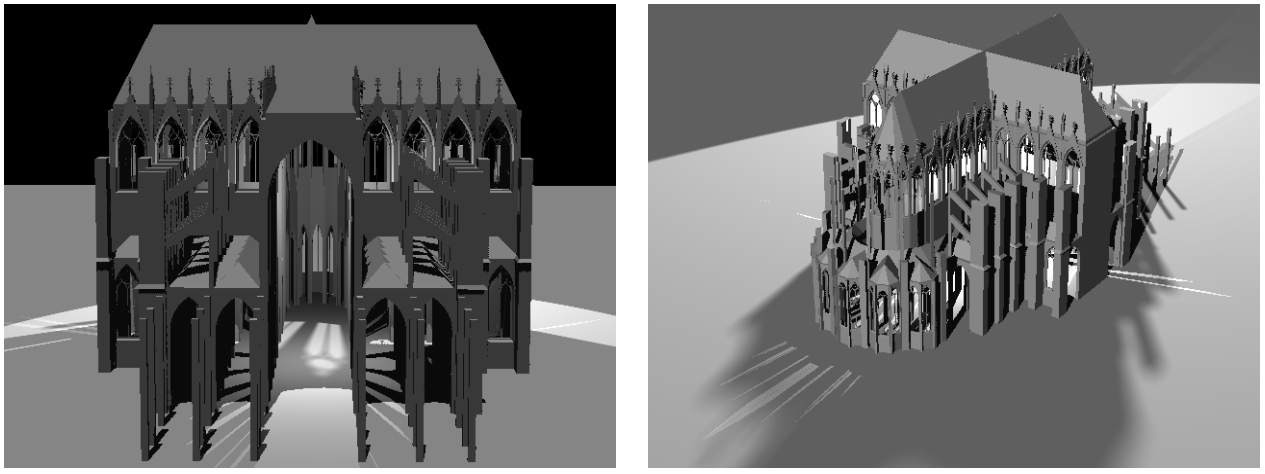


Figure 5.4: Visual Results Four. The interior and exterior of a model of the Kölner Dom is illuminated by an indoor PointLight, an outdoor AreaLight and a GlobalDirectionalLight.

5.2 Discussion: Strengths and Limitations

OpenSG and Apelles were compared concerning differences in shadow generation. While OpenSG uses a single technique for all types of supported light sources, Apelles infers the technique to be utilised from the type of the supported light sources. Apelles improves convolution-based soft shadow generation with non-uniform sampling. Additionally, depth biasing is improved with a linear depth metric. Furthermore, Apelles supplies sampling-based soft shadow generation and adaptive rendering.

A comparison of convolution-based and sampling-based soft shadow generation revealed considerable visual limitations of convolution-based soft shadows. Results generally exhibited overestimated umbrae. Additionally, the overlapping shadows of separate occluders were combined incorrectly. Both limitations are a consequence of resolving visibility from a single point on the surface of the extended light source. Furthermore, peter panning arose from large magnitudes of depth bias necessary to hide incorrect-self shadowing. Of course, convolution-based soft shadow generation was considerably faster than sampling-based soft shadow generation. Visually pleasing results of acceptable quality were achieved with convolution-based soft shadow generation.

As a consequence of the image-based nature of *Shadow Mapping* [Wil78], shadows suffered from aliasing artefacts inevitably. Aliasing stems from undersampling, oversampling and reconstruction errors. Undersampling was caused by limited shadow map resolution, projection aliasing and perspective aliasing. However, projection aliasing was implicitly reduced. The implemented lighting computations ensured that the diffuse term decreased significantly where the surface normal and the direction of light became close to being orthogonal. In particular, if lights and objects moved vividly, temporal aliasing aggravated artefacts additionally. Of course, aliasing was hardly appearing with soft shadows if penumbrae exhibited sufficiently smooth transitions of partial illumination.

According to *Percentage-Closer Soft Shadows* (PCSS) [Fer05], non-uniform sampling was superior to uniform sampling in both quality and performance. However, performance significantly depended on the distance of the near plane of the view volume of the light source. Furthermore, performance decreased exponentially with increasing light sizes. In particular, the magnitude of depth bias necessary to eliminate incorrect self-shadowing had to be increased exponentially with linearly increasing light sizes. As a consequence, peter panning was amplified significantly. Additionally, light bleeding easily occurred for large light sizes.

Sampling-based soft shadow generation required multi-pass rendering because the number of samples which can be evaluated in a single pass is considerably limited. However, experiments showed that evaluating eight samples per pass achieved high performance.

The near and far distances for optimal light view volume culling were efficiently determined from an OpenGL floating-point texture instantaneously using an OpenCL-enabled *Parallel Reduction*. Several optimisation strategies were applied to improve performance significantly.

Adaptive rendering continuously measures performance and decreases shadow quality accordingly once the performance dropped below real-time. Measuring the frame rate was considerably improved by reducing spikes with a *Weighted Moving Average*. However, if the scene was highly dynamic, a highly unstable frame rate hindered to correctly determine when to decrease or increase shadow quality.

The magnitude of depth bias necessary to eliminate incorrect self-shadowing was significantly reduced by utilising a linear depth metric. As a consequence of sampling the distance to the light source, depth was sampled uniformly along the direction of light, if the reprojection from eye-space to light-space involved a perspective projection. Additionally, depth disparity of neighbouring fragments increased significantly. According to PCSS, peter panning was considerably reduced due to a reduced magnitude of depth bias.

For generating performance numbers of rendering simple geometry, a scene with three planar surfaces was rendered (Figure 5.5). For generating performance numbers of rendering complex geometry, the Utah Teapot of the GLUT[1] hovering above a single planar receiver was rendered.

5.2.1 Comparison of OpenSG and Apelles Shadow Generation

The real-time scenegraph system OpenSG supplies experimental shadow generation[2]. Both hard and soft shadows are generated for an arbitrary number of light sources. Three types of light sources are supported: directional, point and spot lights. All implemented algorithms are based upon *Shadow Mapping* [Wil78]. Incorrect self-shadowing is avoided by applying depth biasing (slope and constant).

Shadow generation is seamlessly integrated into OpenSG and enabled involving two steps only. First, a shadow map viewport is created and light sources are attached to the root node of the viewport. Second, the viewport is attached to a rendering context. The shadow map viewport specifies the algorithm to be used for shadow generation (shadow mode)

[1] The OpenGL Utility Toolkit, http://www.opengl.org/resources/libraries/glut (last access 2012-05-07).
[2] http://www.opensg.org/wiki/Gallery/Shadows (last access 2012-05-07)

and the parameter for trading quality for performance (smoothing parameter). The smoothing parameter specifies the filter size for soft shadow generation. In conclusion, a *single* algorithm and a *single* parameter setting is used to generate shadows cast by *all* lights of different types being attached to the root node of the shadow map viewport.

The supported shadow modes for hard shadow generation include

Standard shadow mapping Implements the initially proposed algorithm [Wil78]. Incorrect self-shadowing is avoided by applying depth biasing (slope and constant).

Perspective shadow mapping Warping is applied to reduce perspective aliasing of shadow boundaries close to the viewer. A robust reparameterisation of the shadow map is achieved by utilising *Light-Space Perspective Shadow Maps* [WSP04]. However, standard shadow mapping is applied in two cases. First, if the view volume of the light encompasses the entire scene (global light). Second, if the camera is located behind the light source. In particular, it has to be noted that the free parameter of the reparameterisation is chosen while ignoring an improved falloff function [LGQ*08]. Therefore, the error potentially exceeds the error of standard shadow mapping if the free parameter fails to increase fast enough where the directions of light and view become almost parallel. Furthermore, warping involves fitting which considerably aggravates temporal aliasing if the sampling rate changes between consecutive frames abruptly.

The supported shadow modes for soft shadow generation include

Dither shadow mapping Smoothes shadow boundaries by averaging four occlusion tests non-uniformly. The sampling pattern of the four depth samples is varied per fragment. The smoothing parameter is ignored.

PCF shadow mapping Smoothes shadow boundaries by averaging the results of several occlusion tests with *Percentage-Closer Filtering* (PCF) [RSC87]. The following PCF kernel sizes are used with respect to the specified smoothing parameter: 2×2, 3×3, 4×4, 5×5 and 6×6.

PCSS shadow mapping The size of a PCF kernel is varied per fragment with respect to both the smoothing parameter and the distance between occluder and receiver [Fer05]. The smoothing parameter specifies the size of the area of the extended light source. The sampling pattern is uniform for both blocker search and PCF. The blocker search averages 6×6 depth samples. The PCF kernel averages the outcome of 8×8 occlusion tests. Therefore, penumbrae considerably suffer from banding artefacts where insufficient intensity levels are available to yield smooth transitions.

Variance shadow mapping The result of PCF over a filter region is efficiently approximated from the first and second moments of a distribution of depths with *Variance Shadow Mapping* (VSM) [DL06]. The inequality of CHEBYCHEV is highly susceptible to numerical instability which is mitigated by two improvements. First, the distance to the light source is computed and mapped to $[-1,+1]$ to gain extra precision from the sign bit and to achieve a linear distribution of depth. However, if the light is directional, the linear clip-space depth being mapped to $[0,1]$ is used instead. Second, for adequately storing the two moments, the shadow map utilises the internal format `GL_RGBA16F`. As consequence, compared to a standard shadow map with a single 24 bit fixed-point depth component, the memory footprint of the shadow map increases considerably. However, the moments should be stored as 32 bit floating-point numbers instead to further improve numerical stability [DL06]. Additionally, if depth complexity is high, light bleeding easily appears. Approaches to mitigate light bleeding are ignored [Lau07, LM08].

According to soft shadow generation, PCSS is the only algorithm which generates visually pleasing soft shadows exhibiting contact hardening. In contrast, dithering, PCF and VSM only uniformly smooth shadow boundaries. Hence, soft shadows visually disconnect from occluders with large filter widths easily if shadows do not harden on contact.

In particular, it has to be noted that a non-linear depth metric is used; except for VSM. Therefore, precision of depth along the direction of light is highly non-uniform which aggravates depth biasing (Section 5.2.8). Furthermore, concerning PCSS, using the distance to the light source is highly recommended to avoid numerical instability upon estimating the varying width of penumbrae.

In contrast, Apelles infers the technique utilised for shadow generation from the type of the light source

Standard shadow mapping Hard shadows are generated for global directional light, directional light, spot light and point light. Incorrect self-shadowing is avoided with depth biasing (slope and constant). In order to increase the accuracy of occlusion testing and improve depth biasing, a linear depth metric is used (distance to light) and depth is stored as 32 bit floating-point numbers; for all types of light sources. Accordingly, shadow maps utilise the internal format `GL_R32F`. Furthermore, if the scene geometry is closed *Second-Depth Shadow Mapping* [WM94] is utilisable. Costly occlusion tests are confined to fragments which are potentially lit only. Of course, aliasing artefacts easily appear with standard shadow mapping in large environments (Section 5.2.3).

Convolution-based shadow mapping Visually pleasing soft shadows which exhibit contact hardening are generated for AreaLight. PCSS is extended with non-uniform sampling to hide banding artefacts while increasing both quality and performance [Isi06]. The kernels of both blocker search and PCF are randomly rotated per fragment. The sampling positions inside the filter window are located with respect to a POISSON disk distribution. Three shadow quality settings are supported (low, medium and high). The size of both blocker search and PCF kernels is freely specifyable for all quality settings to allow trading quality for performance with respect to the detail of the rendered geometry and the underlying hardware platform. Of course, large filter widths require large magnitudes of depth bias. As a consequence of gaps between receivers and occluders, which are attached to receivers, occluders easily appear to incorrectly hover (Section 5.2.4).

Sampling-based shadow mapping Physically accurate soft shadows are generated for SampledAreaLight. The planar surface of the extended light source is randomly sampled with spot lights using a POISSON disk distribution. If the number of spot lights is sufficiently high, the individual hard shadows unify accordingly while introducing smooth transitions of partial illumination (penumbrae). Apelles supplies several POISSON sampling patterns with a different number of samples. Furthermore, the planar surface of the extended light source is specifyable as being circular-shaped or square-shaped. Of course, sampling is costly which hinders real-time performance. Therefore, a progressive render mode is provided which successively refines soft shadows over consecutive frames.

In addition, Apelles provides an adaptive render mode. The frame rate is continuously measured and shadow quality is reduced accordingly once performance dropped below real-time (Section 5.2.7).

5.2.2 Comparison of Convolution-Based and Sampling-Based Soft Shadow Generation

Convolution-based shadows considerably suffered from overestimated umbrae, incorrect occluder fusion and incomplete shadows. This is a direct consequence of resolving visibility of objects being visible from a single point on the surface of the extended light source only. In addition, peter panning caused occluders to incorrectly appear as being detached from receivers due to large magnitudes of depth bias necessary to eliminate incorrect self-shadowing. In contrast, more accurate results were achieved if the surface of the extended light source was sampled with multiple spot lights. The samples were located with respect to a POISSON disk distribution on the planar surface of the extended light source. Accordingly, penumbrae were generated accumulatively while exhibiting smooth transitions of partial illumination if the number of samples was sufficiently high. All figures of this section showing soft shadows generated with AreaLight used the PCSS quality setting high (8×8 blocker search, 16×16 PCF).

The soft shadows of AreaLight exhibited overestimated umbrae in general (Figure 5.5). This is a direct consequence of resolving visibility of objects being visible from a single point on the surface of the extended light source only. In comparison, if visibility was sampled accurately (SampledAreaLight), the size of penumbrae varied more intensively with respect to the distance between light source, occluders and receivers. Shadows cast by occluders being closer to the extended light source exhibited larger penumbrae than distant (Figure 5.5, shadow on the middle plane).

Shadow generation of AreaLight suffered from both inaccurate occluder fusion and incomplete shadows (Figure 5.6). The individual contribution of occluders was combined incorrectly where shadows cast by separate occluders overlapped on receivers. Furthermore, shadows exhibited considerable defects for two reasons. First, only sampling depth of objects being visible from a single point on the surface of the extended light source restrains to resolve visibility correctly. Second, only sampling depth of front-facing or back-facing geometry with respect to the direction of light confines occlusion testing to a single layer of depth. However, PCSS estimates the size of penumbrae by averaging several depths of occluders in a neighbourhood of the fragment currently being shaded (blocker search). As a consequence, shadows remained complete while, however, exhibiting considerable defects (Figure 5.6, left, shadows on the plane), if compared to incomplete shadows (Figure 3.8, left, shadows on the plane). In particular, it has to be noted that utilising *Second-Depth Shadow Mapping* [WM94] significantly reduced shadow defects in this context. The overestimation of the penumbra of the shadow cast by the cube onto the sphere was considerably reduced because the depth of the bottom face of the cube was rendered into the shadow map. As a consequence of identifying a closer blocker (bottom face), the width of the penumbra is estimated to be considerably smaller. However, the computation of light reaching the surface of the sphere ignores that light is partly blocked by the cube if the light source exhibits spatial extend. Therefore, the surface of the sphere receives too much light incorrectly (Figure 5.6, left).

As a consequence of peter panning, AreaLight was considerably limited to small light sizes (Figure 5.7). The magnitude of depth bias necessary to eliminate incorrect self-shadowing depends on the size of the light. Large PCF kernel sizes inferred large magnitudes of depth bias to eliminate incorrect self-shadowing. As a consequence, shadows started too far behind which introduced gaps between receivers and occluders being attached to each other. Hence, occluders appeared to incorrectly hover above receivers. Peter panning was considerably reduced by utilising a linear depth metric

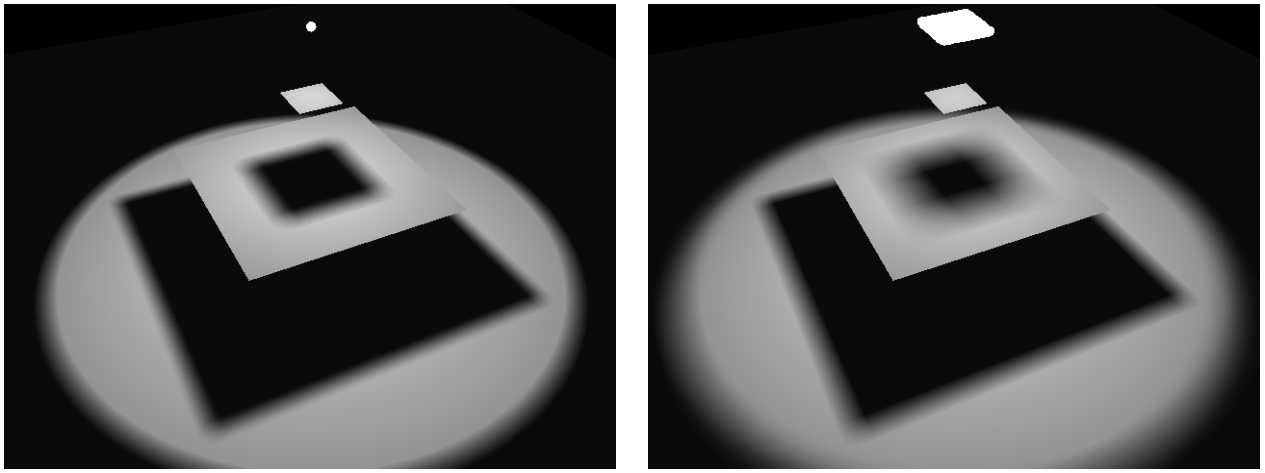


Figure 5.5: Comparison of AreaLight and SampledAreaLight. Umbrae overestimation. Three planar surfaces are illuminated from above. The surfaces increase in size from top to bottom with respect to the distance from the light source. Left: The umbra of the shadow cast by the top plane onto the middle plane is considerably overestimated. The umbra of the shadow cast by the middle plane onto the bottom plane is considerably less overestimated than the umbra on the middle plane. Right: Visibility is sampled accurately with 512 spot lights. The spot lights are located randomly, with respect to a POISSON disk distribution, on the square-shaped surface of the extended light source (SampledAreaLight). As a consequence, the umbra of the shadow on the middle plane is considerably smaller. The umbra of the shadow on the bottom plane is only slightly but noticeably smaller if compared to the AreaLight (left).

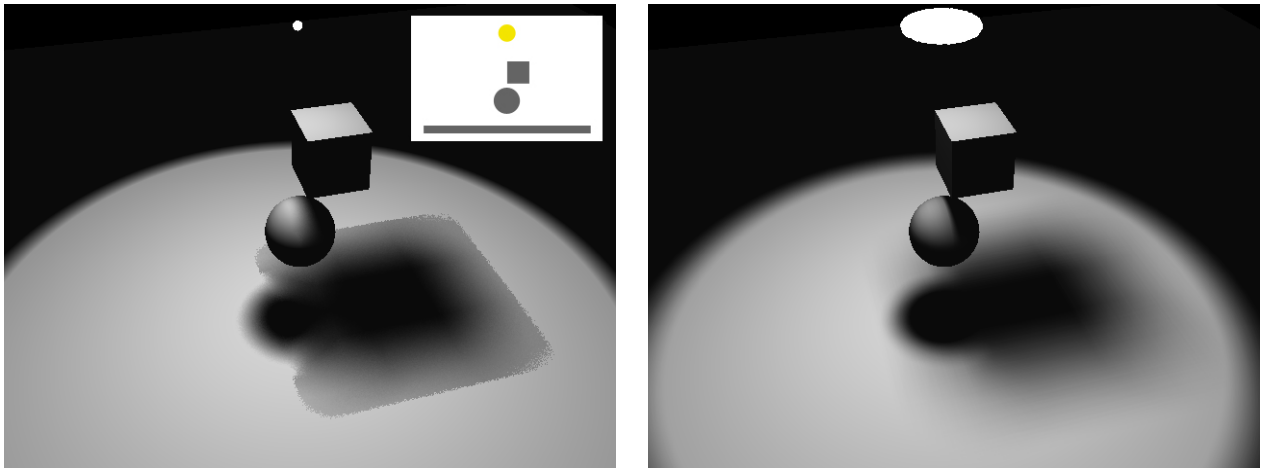


Figure 5.6: Comparison of AreaLight and SampledAreaLight. Incorrect occluder fusion. The small image in the middle illustrates the vertical laying out of the scene which consists of a light source, a cubical occluder, a spherical occluder and a large planar receiver. The shadow map stores depth of front-facing geometry with respect to the direction of light. Left: Incorrect occluder fusion appears on the plane because visibility is resolved from a single point on the surface of the extended light source only. Note that the frazzled and harsh shadow boundary on the plane stems from non-uniform PCSS with early out acceleration. Furthermore, the shadow cast by the cube onto the sphere exhibits overestimated penumbrae because depths of the bottom face of the cube are ignored due to rendering front-facing geometry into the shadow map only. Right: The circular-shaped surface of the extended light source is sampled randomly with 512 spot lights. On the plane the shadow of the sphere correctly combines with the shadow of the cube. The shadow cast by the cube onto the right hemisphere only exhibits a narrow penumbra. The intensity of light on the left hemisphere is noticeably attenuated because the light of several samples on the circular-shaped surface of the extended light source is blocked by the cube.

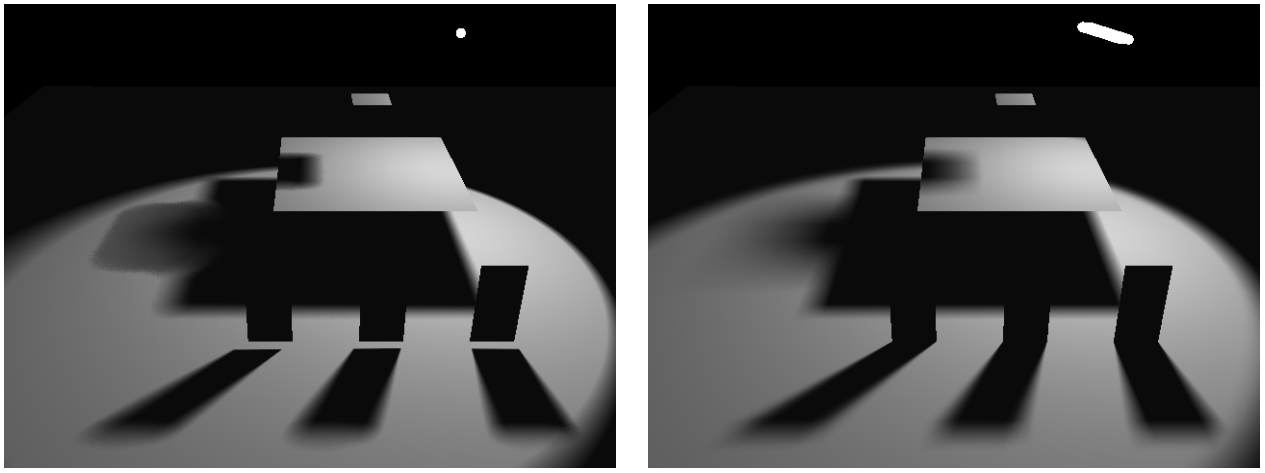


Figure 5.7: Comparison of AreaLight and SampledAreaLight. Peter panning causes gaps at contact shadows. The magnitude of depth bias necessary to eliminate incorrect self-shadowing depends on the size of the AreaLight. PCF infers large magnitudes of depth bias necessary to eliminate incorrect self-shadowing which introduces peter panning. Three black panels have been attached rectangularly to a large horizontal plane to expose peter panning. Left: As a consequence of depth biasing, the three black panels incorrectly appear to hover above the large horizontal plane (depth biasing: slope 20.0, const 0.05, light size: 0.5). Right: The square-shaped surface of the extended light source is sampled with 512 spot lights. The three black polygons correctly appear to be attached to the large horizontal panel regardless of the size of the SampledAreaLight (depth biasing: slope 0.5, const 0.0001). Note that peter panning disappears if SDSM is used because the black planes are back-facing with respect to the direction of light. However, the top and middle plane would no longer cast shadows because the planes only consist of front-facing geometry with respect to the direction of light.

because a smaller magnitude of depth bias was necessary to eliminate incorrect self-shadowing (Section 5.2.8). Furthermore, peter panning was avoided by utilising *Second-Depth Shadow Mapping* (SDSM) [WM94]. However, SDSM is only applicable if geometry is closed. Furthermore, geometry is required to exhibit sufficient thickness to ensure sufficient depth disparity for effectively eliminating incorrect self-shadowing. Otherwise, depth biasing is necessary additionally which reintroduces peter panning.

5.2.3 Aliasing

As a consequence of utilising shadow mapping, shadows generated with Apelles suffered from aliasing (Figure 5.8). Aliasing artefacts manifest in jaggies at the boundaries of shadows which stem from undersampling, oversampling and reconstruction errors. Undersampling is caused by limited shadow map resolution, projection aliasing and perspective aliasing. Both projection aliasing and perspective aliasing enlarge the shadow map locally and globally. Reconstruction errors are introduced due to nearest neighbour reconstruction. Oversampling occurs where the shadow map must be minified to reconstruct a depth value from multiple depth samples.

Projection aliasing was implicitly reduced because the lighting computations of Apelles ensured that the diffuse term decreased dramatically where the surface normal and the direction of light come close to being orthogonal. Both ambient term and specular term do not expose projection aliasing. The ambient term contributes uniformly to shadowed and unshadowed regions. The specular term significantly contributes to regions only where projection aliasing is negligible. As a consequence, surfaces exhibiting aliasing artefacts due to projection aliasing received a small amount of light only. However, with oblique light objectionable artefacts were eliminated only by increasing the sampling rate of the shadow map globally (Figure 5.8, dark regions, shadows of detailed decorations of transverse gables). Of course, the sampling rate must be adjusted locally with respect to the surface normal and the direction of light to eliminate projection aliasing completely which is hardly achievable in real-time [GW07a, GW07b].

In particular, if lights and objects moved vividly, movements of jagged shadow boundaries created the highly visual disturbing impression of shadows moving across receivers with frequent and abrupt stops. In addition, if the sampling rate of shadow maps changed abruptly between consecutive frames, temporal aliasing further aggravated artefacts.

Aliasing was hardly appearing with soft shadows. According to AreaLight, by averaging several occlusion tests, PCF introduces a refined reconstruction filter. The jagged boundaries of shadows are blurred with respect to the size of the filter window. Therefore, aliasing due to both undersampling and oversampling is effectively camouflaged in penumbrae. As a consequence, the resolution of shadow maps can be considerably smaller without introducing aliasing. Additionally, the memory footprint of shadow maps is considerably decreased; particularly for SampledAreaLight which allocates as many shadow maps as samples are evaluated in a single gather pass.

Figure 5.8: Shadow map aliasing in large environments. The scene is illuminated with a GlobalDirectionalLight having a shadow map of size 8192×8192. The intensity of light was increased additionally to expose aliasing artefacts in dark regions. Left: Shadows exhibit noticeable aliasing artefacts. As a consequence of oblique light, shadows in the darker regions are incapable of accurately reflecting the detailed decorations on the edges of traverse gables (projection aliasing). Right: Aliasing artefacts are decreased considerably by increasing the sampling rate of the shadow map globally. The GlobalDirectionaLight is replaced with a local DirectionalLight. The field of view is reduced to only encompass the model of the Kölner Dom instead of the large environment.

5.2.4 Percentage-Closer Soft Shadows

Percentage-Closer Soft Shadows (PCSS) [Fer05] varies the size of a PCF kernel based on the occluder-receiver distance to create visually pleasing soft shadows with contact hardening. In order to achieve acceptable results for large light sizes and large penumbrae, respectively, the PCF stage requires large kernel sizes. Therefore, numerous shadow map accesses limit performance considerably. As a consequence of inadequate PCF kernel sizes, large penumbrae suffer from banding where insufficient intensity levels are available to yield smooth transitions. Accordingly, I tested various combinations of uniform and non-uniform sampling for the blocker search and for the PCF at different kernel sizes. The influence on performance and inhibition of penumbra banding of the combinations was examined. The results showed that non-uniform sampling is capable of hiding penumbra banding effectively while enabling higher frame rates (Table 5.1). However, noise was introduced due to sampling the shadow map irregularly (Figure 5.9). The noise was significantly less noticeable than banding.

In addition to inhibiting penumbra banding, non-uniform sampling increased the accuracy of the blocker search heuristic significantly. The boundaries of penumbrae became more accurate but frazzled as well (Figure 5.9, bottom row). Moreover, the improved accuracy of the blocker search heuristic was beneficial to the efficiency of early out (Figure 5.10). Performance was raised because more fragments were correctly determined to be unoccluded and, therefore, missed the costly PCF stage. Accordingly, non-uniform sampling performed better than uniform sampling; despite the cost of instantly generating pseudorandom numbers to randomly rotate the POISSON disk kernel per fragment. As a consequence of the improved accuracy of the blocker search heuristic, non-uniform sampling allowed to take less samples for the blocker search compared to uniform sampling (Table 5.1, first column). Experiments showed that if less than 25 samples were taken for a uniform blocker search, salt and pepper noise was introduced due to incorrect early outs.

Blocker search	**PCF**	**Simple** fps	**Complex** fps
4×4 non-uniform	4×4 non-uniform	270	150
8×8 uniform	12×12 uniform	120	95
4×4 non-uniform	12×12 uniform	180	130
4×4 non-uniform	8×8 non-uniform	200	150
4×4 non-uniform	16×16 non-uniform	110	70

Table 5.1: Percentage-closer soft shadows. Comparison of uniform and non-uniform sampling for simple and complex geometry. Visual quality increases from top to bottom. Note that non-uniform sampling is superior to uniform sampling in both quality and performance.

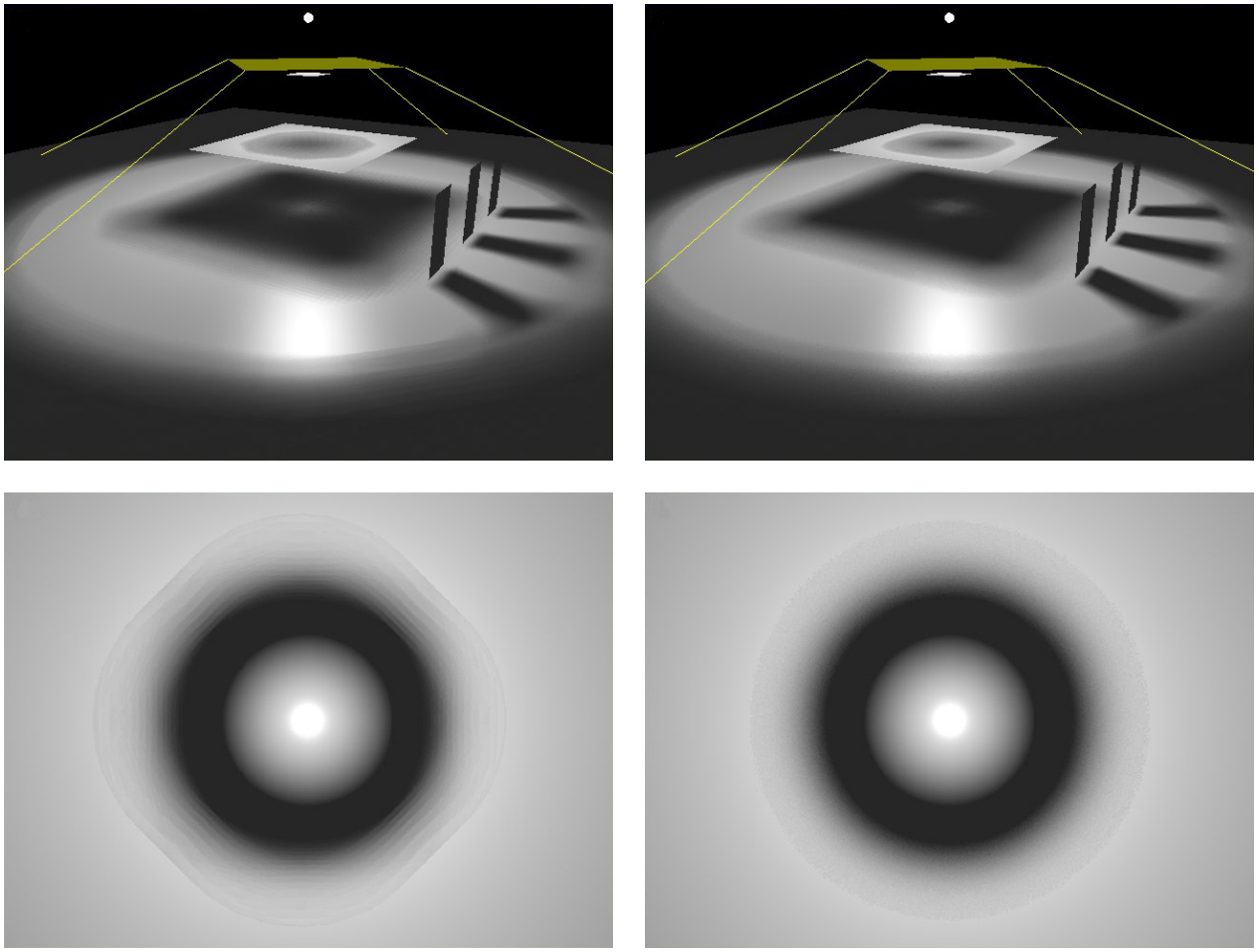


Figure 5.9: Percentage-closer soft shadows. Comparison of uniform and non-uniform sampling (8×8 blocker search and 8×8 PCF). Left: Uniform sampling. Shadow boundaries are nonnatural due to uniform sampling in blocker search. Penumbra suffers from banding. Right: Non-uniform sampling. Shadows are more visually pleasing especially if regions with banding are compared. However, banding is traded for noise. Top row: Note that the three black polygons are attached to the ground plane but due to a large magnitude of depth bias they appear to be not. Note that the large umbra on the ground plane shows light bleeding as a consequence of a large light size (large filter width). Bottom row: A sphere is lit directly from above to expose the artefacts of uniform sampling.

On the contrary, if non-uniform sampling was used, 16 samples sufficed. Finally, the accuracy of the blocker search heuristic highly depended on the distance between the fragment (receiver) and the near plane of the view frustum of the light (Figure 5.11). The accuracy decreased dramatically as the distance between the near plane and the receiver increased. Therefore, the near plane should be moved as close as possible towards occluding geometry.

According to large light sizes, large filter widths had a negative effect on the blocker search and the PCF. The efficiency of early out was dramatically decreased because more fragments were incorrectly determined to be occluded (Figure 5.12). Moreover, light bleeding was introduced because PCF incorrectly computed penumbra intensity levels for umbrae fragments (Figure 5.9, top row, the large umbra on the ground plane is incorrectly brightened). Furthermore, a large filter width required a large PCF kernel, otherwise banding was introduced. As a result of a large PCF kernel, a huge magnitude of depth bias was necessary to eliminate incorrect self-shadowing (Figure 5.9, top row, the three black polygons appear to hover incorrectly). In conclusion, for large filter widths non-uniform sampling allowed to use smaller PCF kernels which required less biasing. Smaller PCF kernels can be used because banding is traded for noise which is less noticeable. Of course, if geometry is closed, *Second-Depth Shadow Mapping* [WM94] is utilisable to eliminate incorrect self-shadowing without biasing if geometry exhibits sufficient thickness. However, light bleeding still occurred for large PCF kernels.

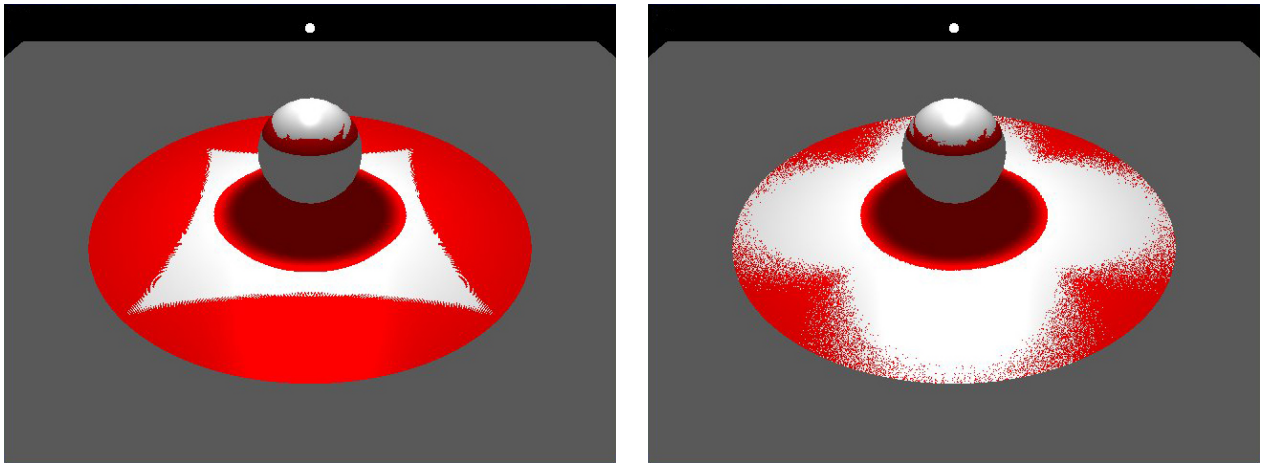


Figure 5.10: Percentage-closer soft shadows. Effect of non-uniform sampling on efficiency of blocker search. Non-uniform sampling considerably improves the efficiency of early out. Red fragments are determined to be occluded (blocker search) and sent to the PCF stage. Left: Uniform sampling is used for the blocker search only. Non-uniform sampling is used for PCF. As a consequence of using the unclamped distance from light as a linear depth metric together with slope-based depth biasing, the region of early out fragments on the ground plane is pillow-shaped. Right: Non-uniform sampling is used for both, blocker search and PCF. Less fragments are incorrectly determined to be occluded which yields a higher frame rate.

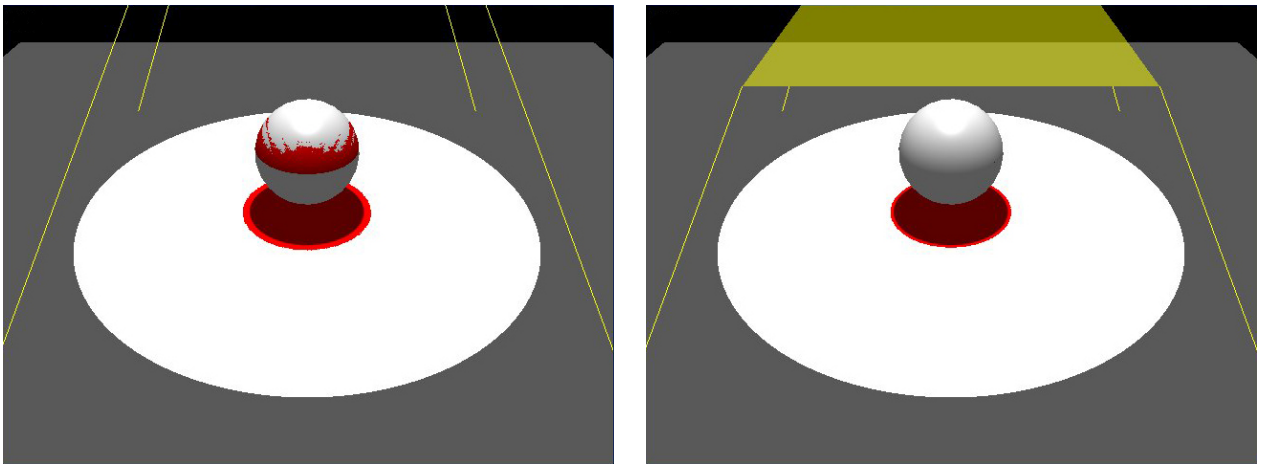


Figure 5.11: Percentage-closer soft shadows. Effect of near plane distance on efficiency. Efficiency of early out depends highly on the distance between the fragment (receiver) and the near plane. Red fragments are determined to be occluded (blocker search) and sent to the PCF stage. Left: PCF is performed unnecessarily for fragments on the sphere which are unoccluded. Right: On the sphere no fragments are incorrectly determined to be occluded. Therefore, moving the near plane closer to the sphere results in a higher frame rate.

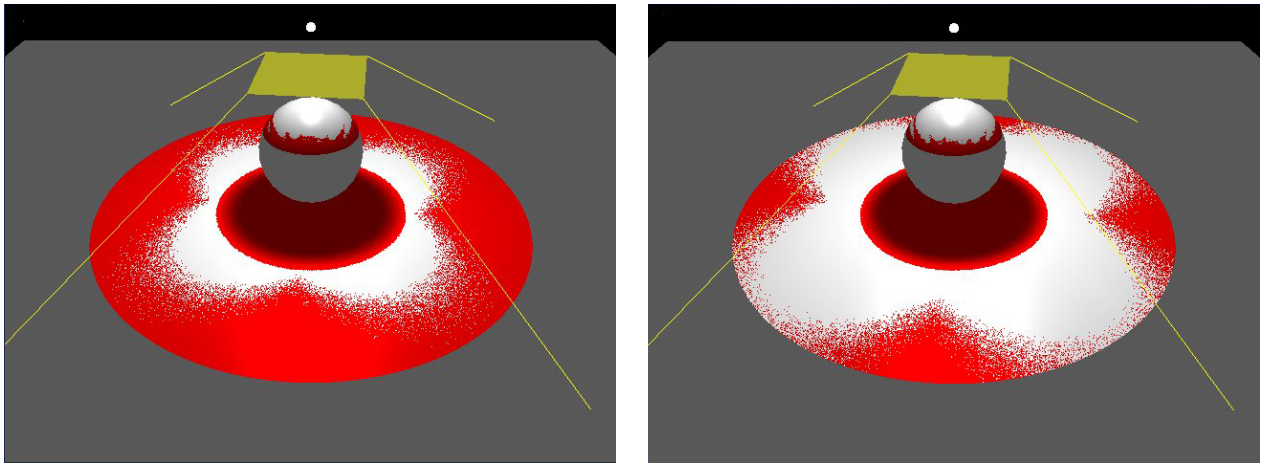


Figure 5.12: Percentage-closer soft shadows. Effect of light size on efficiency. Efficiency of early out depends highly on the size of the light. Red fragments are determined to be occluded (blocker search) and sent to the PCF stage. Left: On the sphere and the ground plane PCF is performed unnecessarily for more fragments which are unoccluded. If the size of the light is increased, the width of the blocker search is increased as well. Right: As a consequence of a smaller light size (smaller penumbra), less fragments are incorrectly determined to be occluded which increases performance significantly.

5.2.5 Optimisation of Sampling-Based Soft Shadow Generation

The area of the extended light source is irregularly sampled with multiple spot lights. The number of shadow maps, which can be evaluated for occlusion testing within a single pass, is constrained by three limits.

First, the number of available texture addressing units (TAU) limits the number of shadow maps being accessible. Each sample requires a shadow map for occlusion testing. Therefore, only as much samples can be evaluated within a single pass as unoccupied TAUs are available. However, only a single TAU was required by aggregating individual shadow maps of equal size in a single texture array which is addressable with a single TAU. As a consequence, the shadow map of the according sample was simply accessed through an index (layer).

Second, the capabilities of the underlying hardware platform limit the size of uniform arrays. Using the GeForce 250 GTS, a maximum of 98 samples were evaluated within a single pass. The compiler of the OpenGL Shading Language (GLSL) failed upon assembling the fragment program for 99 samples. According to aggregating the positions of the spot lights in a uniform array, the GLSL assembler emitted an *array out of bounds* error.

Third, the size of available texture memory limits the number of resident shadow maps (Figure 5.13). A single shadow map with 2048×2048 texels, which represent 32 bit floating-point depth values, is 16.78 MByte large. Hence, an area light with 32 samples consumes 537 MByte of GPU memory (VRAM). Accordingly, a shadow map array with 61 layers consumed the entire VRAM of the GeForce 250 GTS. However, 94 samples could be evaluated in a single pass because the shadow map array was allocated in RAM by the GL driver implicitly. Of course, performance decreased dramatically due to costly DMA transfers over PCI-express between the CPU and the GPU.

Soft shadows of sampled area light sources were generated by utilising ping-pong offscreen rendering. First, the samples were divided into smaller batches of equal size. In each pass a single batch of samples is evaluated only. Using ping-pong offscreen rendering the results of passes were successively accumulated to yield the final result. For example, 512 samples are broken down into 16 batches with 32 samples each. A significant gain in performance was identified upon evaluating eight samples per pass (Figure 5.14). Accordingly, rendering eight single-sample light sources in one pass achieved high performance as well.

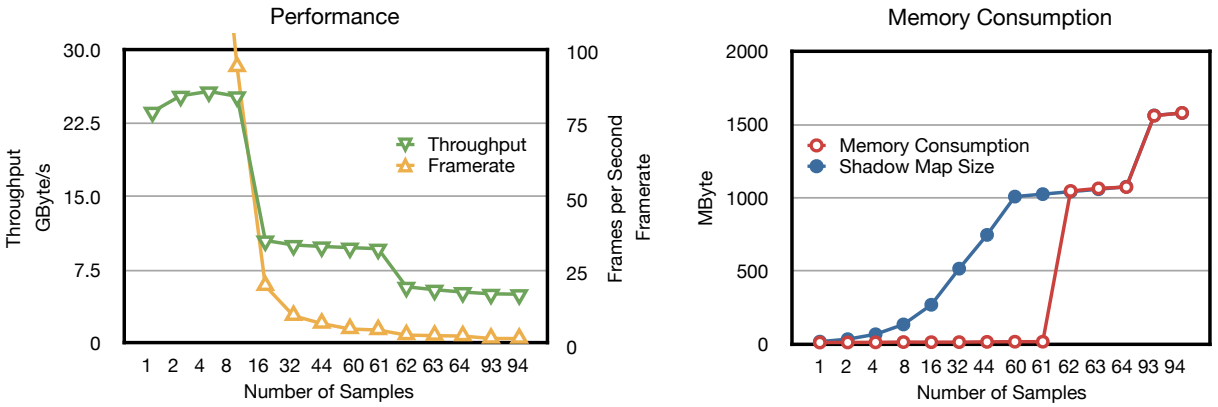


Figure 5.13: Comparison of rendering a varying number of area light samples in a single pass. Each sample has a shadow map 2048×2048 texels large (16.8 MByte). Geometry is kept simple to minimise the influence of vertex processing. Left: Frame rate and throughput is measured while increasing the number of samples being rendered in a single pass. Note that, first, throughput slightly rises but, then, considerably decreases beyond 8 samples. Despite the simple geometry, the frame rate drops below real-time at 16 samples already. Right: Memory consumption of RAM is measured while increasing the number of samples. With 62 samples, the shadow map array becomes too large to fit into VRAM and, therefore, is allocated in RAM by the GL driver implicitly. Hence, the memory consumption corresponds to the size of the shadow map array. Note that, accordingly, throughput decreases significantly due to costly DMA transfers over PCI-express between the CPU and the GPU (left, 61 and 62 samples).

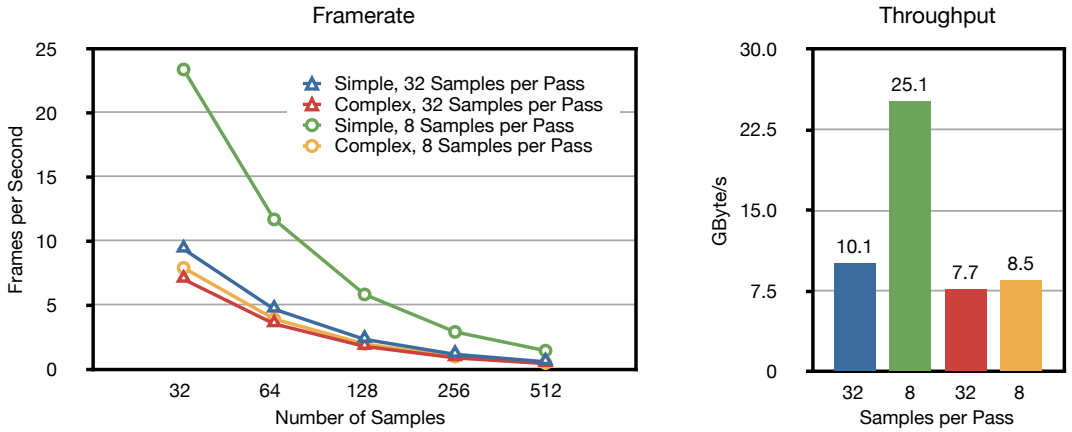


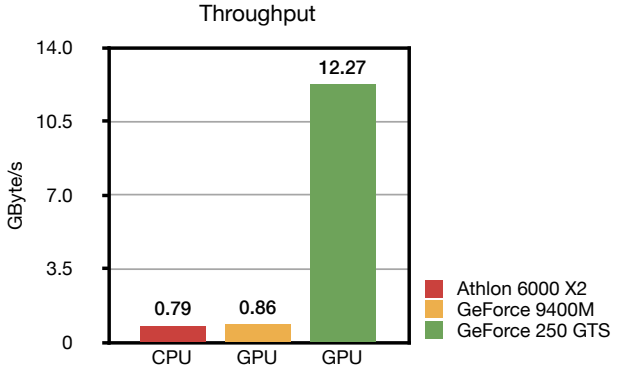
Figure 5.14: Comparison of rendering a sampled area light with 32 and 8 samples per pass. Each sample has a shadow map 2048×2048 texels large. The shadow maps are aggregated in a texture array with 32 and 8 layers, respectively. Therefore, each pass renders both scattering and gathering. In order to illustrate the influence of repeated vertex processing during multi-pass rendering, simple and complex geometry is rendered. Left: Frame rate is measured while increasing the number of samples. Right: Throughput is averaged over the increasing number of samples with respect to rendering 32 and 8 samples per pass. Note that even with complex geometry throughput is significantly higher if 8 samples per pass are rendered (right, second and fourth bar).

5.2.6 Optimisation of Light View Volume Culling with OpenCL

The optimal magnitude of the near and far distances of the view volume of the light corresponds to the minimum and maximum of the light clip-space depth values in the shadow map (Section 3.5). Accordingly, experiments were conducted using an OpenCL-enabled implementation of a *Parallel Reduction* to retrieve the minimum and maximum from an OpenGL floating-point texture instantaneously. The results showed that the GPU performed the computation approximately 15 times faster than a dual-core desktop CPU. Even a mobile GPU, which is rather feeble and optimised to save battery, outperformed the CPU (Figure 5.15).

The OpenCL-enabled computation of minimum and maximum of a shadow map 2048×2048 texels large using a parallel reduction required three passes and took 2.65 ms (throughput 12.3 GByte/s, Figure 5.16, third bar). For comparison, rendering the Utah Teapot of the GLUT once into that shadow map took 3.58 ms (throughput 9.37 GByte/s, Figure 5.16, open bar). As a consequence of the limits of the OpenCL implementation of the underlying hardware platform, the two-dimensional problem domain (2D texture) was partitioned into 16384 square blocks 16×16 elements large. Therefore, the first pass of the reduction yielded 16384 minima and maxima respectively. After the first pass the problem domain became one-dimensional (array). This enabled to take advantage of the maximum block size of 512 elements. Accordingly, the array was partitioned into 32 blocks 512 elements long. Therefore, the second pass of the reduction yielded 32 minima and maxima, respectively. Finally, in the last pass a single multiprocessor reduced the remaining 32 elements to yield the minimum and maximum. In particular, it has to be noted that the partition of the shadow map into square blocks inhibited to use the maximum block size supported by the OpenCL implementation of the underlying hardware platform. In addition, the shadow map served as a direct input to the algorithm (OpenGL and OpenCL context sharing). As a consequence, both costly on-chip and off-chip data transfers between the CPU and GPU were circumvented.

The OpenCL-enabled implementation of the parallel reduction applied several optimisation strategies to improve efficiency (Figure 5.16). For increased throughput input data was loaded into fast local memory of a work-group. Hence, warps of work-items, which are executed in lockstep within a work-group (SIMD), had faster access to the data they worked on. The initial implementation used the modulo operator to interleave active work-items. In particular, modulo is a costly operation on a GPU and, therefore, limits throughput significantly. As a consequence of eliminating the modulo operation, the addressing of local memory resulted in many local memory bank conflicts. These local memory bank conflicts were avoided by ensuring that adjacent work-items accessed different local memory banks which increased throughput even further. However, sequential addressing required an integer division. This particularly expensive instruction was replaced with an efficient bitwise operation.



Compute Device Type	Name	Memory MByte	Clock speed MHz	Compute units	Work-group size
CPU	AMD Athlon X2 6000	1568	2970	2	2
GPU	NVIDIA GeForce 9400M	268	1100	2	512
GPU	NVIDIA GeForce 250 GTS	1027	1836	16	512

Figure 5.15: Comparison of different compute devices for the determination of the minimum and maximum from an OpenGL floating-point texture. Top: Throughput of a texture 2048×2048 texels large. Note that the mobile GPU (middle bar), which is feeble and optimised to save battery, is faster than a desktop CPU (left bar). Of course, the desktop GPU easily outperforms the other devices due to a larger number of parallel compute units (right bar). Bottom: Properties of the compute devices. Note that the compute unit of a GPU comprises eight processing elements.

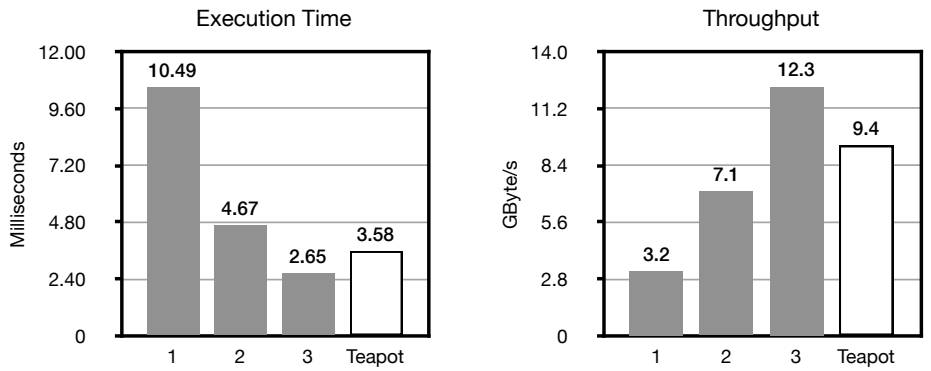


Figure 5.16: Comparison of two optimisations for the determination of the minimum and maximum from an OpenGL floating-point texture. Efficiency increases significantly after applying two optimisations to the computation of the minimum and maximum from a texture 2048×2048 texels large. The change in execution time (left) and throughput (right) was measured after eliminating an expensive modulo operation (second bar) and after resolving local memory bank conflicts (third bar). For comparison, the performance of rendering the Utah Teapot of the GLUT into that shadow map has been added (open bar).

5.2.7 Adaptive Rendering

Adaptive rendering continuously measures the frame rate and reduces shadow quality once the performance dropped below real-time. Experiments showed that adaptive rendering was capable of recovering real-time performance (Figure 5.17). After the setting time had elapsed (frame 0–12), the average frame rate was continuously measured (frame 12–30). Then, the frame rate dropped below real-time due to switching to fullscreen rendering (frame 30). Three counter actions were applied to raise the frame rate (frame 31–48, upward arrows). As a consequence, the size of shadow maps was decreased from 2048×2048 to 1024×1024 and, finally, to 512×512, respectively. After applying each counter action, a setting time of five frames had elapsed before the effect of the counter action was measured. As a consequence of taking counter actions, the frame rate raised and real-time was recovered (frame 49–54). Particularly, the last counter action was undone because the frame rate had raised sufficiently high (frame 67). As a consequence, the size of shadow maps was increased from 512×512 to 1024×1024 respectively. Finally, the frame rate stabilised close to real-time (frame 70 and beyond).

Measuring the frame rate was improved considerably by averaging the raw frame rate with a *Weighted Moving Average*. As a consequence, spikes were reduced while maintaining sufficient signal response sensitivity. Of course, if the frame rate is highly unstable due to abruptly changing workloads, adaptive rendering is inapplicable. Furthermore, after applying a counter action, the frame rate was given a setting time to reflect the effect of the counter action. If the frame rate changes due to additional causes during the setting time, the effect of the counter action is measured incorrectly.

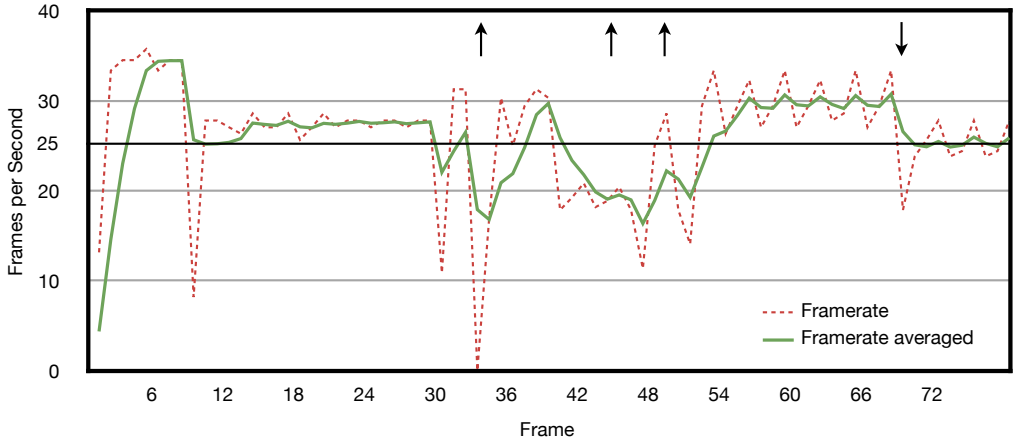


Figure 5.17: Effect of adaptively changing render quality to maintain real-time performance. The measured frame rate (dashed line) is averaged over five consecutive frames using a *Weighted Moving Average* (solid line). As a consequence, spikes are reduced significantly while maintaining sufficient signal response sensitivity. Once the measured frame rate drops below real-time (frame 30), three counter actions are applied to raise performance (upward arrows). Note that after applying each counter action, a setting time elapsed before measuring the effect of the counter action. The last counter action is undone because the frame rate raised sufficiently high (downward arrow). Finally, the frame rate stabilises close to real-time.

Hence, adaptive rendering fails at both undoing previous counter actions and applying new counter actions correctly. No further counter actions were applied if three subsequent counter actions had been identified to be ineffective. In particular, it has to be noted that undoing of counter actions carries the risk of a feedback loop where the same counter action is applied and subsequently undone repeatedly. The feedback loop was avoided by disabling the undoing of counter actions once detecting the sequence *apply undo apply* of the same counter action.

5.2.8 Effect of Depth Metric on Depth Biasing

Shadow mapping is prone to incorrect self-shadowing artefacts due to a resampling problem. This problem was circumvented with depth biasing (slope and constant). Filtering the shadow map with PCF to create visually pleasing soft shadows using PCSS induced a greater magnitude of slope bias. As a consequence of large offsets, shadows started too far behind giving the wrong impression of hovering objects which were attached to each other (peter panning). However, the necessary magnitude of slope bias was reduced by using the distance to the light as a depth metric (Figure 5.18).

Although a uniform distribution of depth values had insignificant impact on biasing (Figure 5.18, first and second bars), a linear depth metric is preferable [BAS02a]. If the transformation into light-space involves a perspective projection, depth is sampled highly non-uniform along the direction of light. Objects close to the near plane of the view frustum of the light are oversampled while objects situated in the farther half of the view frustum of the light are being sampled extremely coarsely. In contrast, a linear depth metric distributes depth values uniformly between the near plane and the far plane. According to directional lights, light-space depth values are already distributed uniformly due to orthographic projection.

If PCF was used to generate soft shadows more slope bias was needed. The softness of the shadows is determined by the size of the PCF kernel. The larger the PCF kernel, the more slope bias was necessary to eliminate incorrect self-shadowing. In addition, the larger the shadow map, the more slope bias was needed. In this context, a uniform distribution of depth values had little effect on biasing (Figure 5.18, second bar). Moreover, upon changing the size of an unfiltered shadow map (hard shadows), no influence on biasing was observable.

According to PCF, a linear depth metric which computes the distance to the light proved to be capable of reducing the necessary magnitude of slope bias significantly (Figure 5.18, third bar). Furthermore, the accuracy of the occlusion test was improved because the depth disparity of neighbouring fragments increased significantly. Hence, the distance from light is preferable to a uniform distribution of depth values; even for a directional light. Of course, computing the distance involved a higher computational cost. Additionally, slope-based biasing required to evaluate the partial derivative of z with respect to x as well as y. Experiments showed that the performance degradation in the scatter pass was negligible (data not shown). In the gather pass no additional computations were necessary. The distance to the light source was already computed for evaluating the spatial attenuation of light.

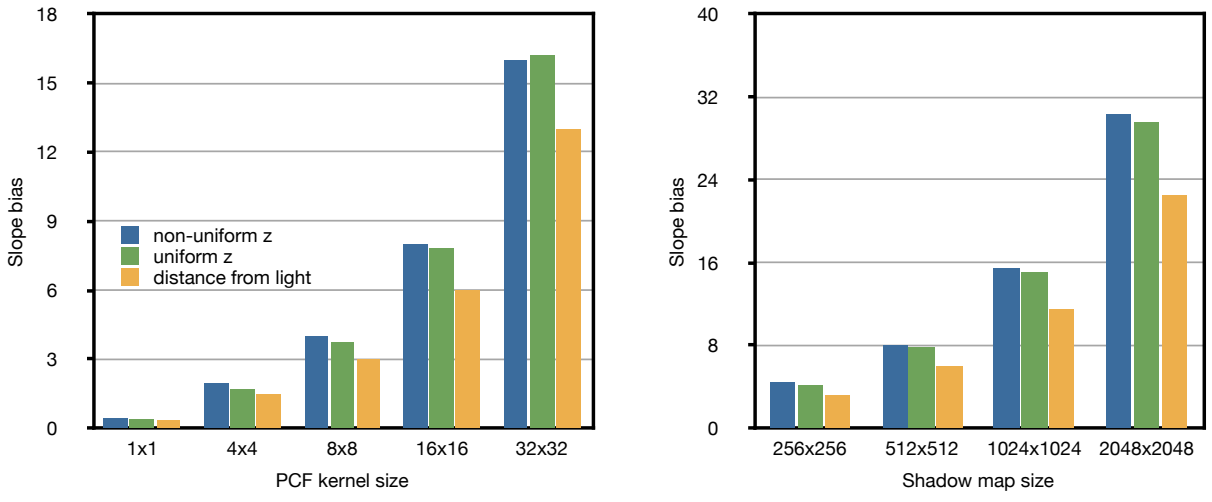


Figure 5.18: Effect of depth metric on depth biasing. Comparison of three different depth metrics: non-uniform z, uniform z and distance from light. Left: For a fixed shadow map size of 512×512 the PCF kernel size is increased. Note that a PCF kernel size of 1×1 corresponds to hard shadowing. A larger PCF kernel requires more slope bias. Right: For a fixed PCF kernel size of 16×16 the shadow map size is increased. A larger shadow map requires more slope bias. Note that using the distance from light as a linear depth metric (third bar) reduces the necessary magnitude of slope bias significantly.

Biasing was influenced by the accuracy of the occlusion test which is affected by the numerical representation of depth values considerably. According to storing depth values in a depth buffer, a floating-point representation was preferable to a fixed-point representation. In the range $[0,1]$ a 32 bit fixed-point representation provides less accuracy than a 32 bit floating-point representation. Concerning the IEEE 754 standard format [IEE08], the units of last precision vary in size. Therefore, representable values are distributed unequally in the number range. Distances between small numbers are very small. With the magnitude of numbers the distance increases expeditiously. For example, the distance between two numbers in the range $[0,1]$ is approximately 10^{-45} and starts to increase beyond 1. Instead, the constant distance between two numbers in fixed-point representation is approximately 10^{-9}. Therefore, a 32 bit floating-point representation identifies 10^{45} numbers in the range $[0,1]$. In contrast, a 32 bit fixed-point representation identifies 10^{9} numbers within the same range only. In short, to maximise the accuracy of the occlusion test, depth values should be represented with floating-point numbers and the depth metric should map depth values to the range $[0,1]$. Since depth values are always positive, mapping depth values to $[-1,+1]$ utilises the idle sign bit for extra precision. However, negative depth values were inapplicable for PCSS. In addition, experiments with a depth metric in the range $[0,1]$ showed that numerical instability problems arose upon estimating the blocker search width and penumbra width. As a consequence, PCSS required to use the unmapped distance from light as depth metric. In conclusion, the accuracy of the depth metric decreases dramatically with large distances from the light because the floating-point representation infers large gaps between large numbers.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

Apelles is capable of rendering both geometric hard and soft shadows cast by opaque occluders. An arbitrary number of light sources can be setup in a scene naturally. The supported types of light sources include directional lights, point lights, spot lights and area lights. Apelles exposes a simple and concise interface to facilitate both integration into existing applications and development of new applications. Visual quality is decreased adaptively to ensure interactivity if necessary. Of course, Apelles is open for supplementary extension to easily combine additional GPU-accelerated rasterisation techniques which add missing global illumination effects (see Section 6.2.5).

Although significant advances have been made in real-time shadow generation in the last few years, a universally applicable algorithm is still unavailable. Therefore, Apelles offers two different strategies for rendering soft shadows. The first strategy allows to render visually-pleasing but unreal soft shadows in real-time (convolution-based soft shadows). The second strategy achieves a more physically-correct result but comes at a higher cost (sampling-based soft shadows). Hence, a strategy can be selected depending on the context of the application appropriately.

Shadows convey inevitable visual information in three-dimensional virtual environments with both static and dynamic objects. The mere presence of objectionably coarse approximations of shadows is preferable to disregarding shadows totally. Shadows cast by objects on other surfaces provide critical visual cues about both the shape of the casting objects and the shape of the receiving surfaces. For example, the curvature of the surface of a convex object of interest can be evaluated by inspecting the shadow silhouette casted by oblique light on a nearby planar surface being rectangular to the direction of light. Furthermore, shadows enable to correctly perceive the spatial arrangement of both static and dynamic objects. If soft shadows harden on contact the proximity of objects can be perceived more finely than with stereo viewing; particularly if objects are moving. For example, upon visualising a polio virus infection with the computational microscope, the relative movements of long chain molecules with 100 million atoms are hardly traceable visually without dynamic cues for spatial layout.

6.2 Future Work

6.2.1 Aliasing

Despite the large texture resolutions being supported on contemporary hardware, uniform *Shadow Mapping* [Wil78] is hardly practicable in large environments.

Undersampling due to perspective aliasing is considerably reduced with warping and z-partitioning. *Light-Space Perspective Shadow Maps* (LiSPSM) [WSP04] provide a robust reparameterisation for all types of light sources. If the improved falloff function suggested by [LGQ*08] is utilised, the shadow quality of LiSPSM is ensured to never decrease below the shadow quality of uniform shadow mapping. However, if memory constraints allow to allocate more than one shadow map per single-sample light source, [LGQ*08] suggests to replace LiSPSM with z-partitioning [ZSXL06, ZZB09]. Explicitly tuneable split schemes allow to achieve a reasonable tradeoff between shadow quality and performance; particularly if geometry exhibits varying detail along the direction of light. In contrast, automatically optimising the splitting scheme every frame adaptively is achievable with *Sample Distribution Shadow Maps* (SDSM) [LSL11].

However, warping and z-partitioning inherently aggravate temporal aliasing in highly dynamic scenes. In order to maximise effectiveness warping requires fitting which aggravates temporal aliasing additionally. According to z-partitioning, artefacts easily appear close to the adjacencies of split planes. A combination of warping and z-partitioning mainly results in increased temporal aliasing while quality improvements remain negligible.

Projection aliasing is avoidable only with adaptive partitioning. The sampling rate of the shadow map is varied analytically with respect to the direction of the surface normal and the direction of light. This is hardly achievable in real-time due to excessive multi-pass rendering and costly read-backs to decide when to stop the iterative refinement of the shadow map [GW07a, GW07b]. However, the aforementioned approaches supply a parameter to fine-tune the tradeoff between quality and performance depending on the application requirements.

Oversampling is efficiently addressable with approaches which linearise the shadow map and, therefore, allow to utilise GPU-enabled filtering for the minification of shadow maps [DL06, LM08, AMB*07, AMS*08]. As a consequence, shadow maps become applicable to trilinear and anisotropic texture filtering. Furthermore, these approaches increase temporal coherence which reduces temporal aliasing significantly.

6.2.2 Convolution-Based Soft Shadow Generation

Despite improving *Percentage-Closer Soft Shadows* (PCSS) [Fer05] with stratified sampling [Isi06], performance is limited due to excessively accessing the shadow map for both blocker search and PCF. Furthermore, PCSS is susceptible to peter panning due to large magnitudes of depth bias necessary for eliminating incorrect self-shadowing. Peter panning is addressable with gradient-based depth biasing [Sch05, Isi06, Sch07] or normal-offset-based depth biasing [Hol11].

The performance of the filtering stage of PCSS is considerably improved with *Summed-Area Variance Shadow Maps* (SAVSM) [Lau07]. Adaptively varying the size of the filter window for each screen pixel is achieved accurately with summed-area tables which, however, infer a high cost of creation. In contrast, *N-Buffers* [Déc05] achieve a reasonable balance between cost of creation and quality if implemented efficiently on contemporary hardware utilising a hierarchical approach [ED06].

However, light bleeding due to high depth complexity in small scenes necessitates to partition the depth range into multiple layers [LM08]. For small lights *Layered Variance Shadow Maps* (LVSM) or *Exponential Variance Shadow Maps* (EVSM) [LM08] are a robust and fast alternative to PCF-based PCSS. In contrast, if lights are large, a VSM-based filtering stage easily leads to incorrect classification of occlusion.

Variance Soft Shadow Maps (VSSM) [YDF*10] address incorrect classification of occlusion with an efficient filter kernel subdivision scheme. Additionally, blocker search is performed efficiently by estimating the average blocker depth with a VSM-enabled formula. As a consequence, PCSS becomes a constant-time algorithm.

Furthermore, VSM-based approaches are inherently less susceptible to incorrect self-shadowing and, therefore, peter panning is mitigated implicitly. However, VSM-based PCSS is less precise than PCF-based PCSS because the outcome of PCF is estimated with a single depth sample only. Therefore, instead of replacing PCF-based PCSS with VSM-based PCSS in Apelles, both algorithms should be supplied side by side for increased flexibility upon using AreaLight. *Occlusion Textures* [ED06, ED08] is another real-time algorithm worth considering for convolution-based soft shadow generation. The performance is independent of the size of the light and the size of penumbrae. However, reasonable results are achieved for small scenes with low geometry detail only.

6.2.3 Environmental Shadows

Accurate environmental shadows can be generated by sampling the surrounding hemisphere non-uniformly [Pha04]. This is similar to SampledAreaLight where the planar surface of the light source is sampled non-uniformly using multiple spot lights having equal directions. In contrast, the direction of light of each sample on the hemisphere points towards the centre of the hemisphere surrounding the scene. Accordingly, the shader of SampledAreaLight only needs to be modified to support a varying direction of light per sample. Of course, the shadows of numerous spot lights need to be accumulated to sample environmental shadows densely enough in large environments.

As a consequence of the high computational cost of sampling-based environmental shadows, Apelles should include a real-time ambient occlusion algorithm for deformable objects additionally. *Screen-Space Ambient Occlusion* (SSAO) [SA07, Mit07, BSD08] is independent from scene complexity and exhibits low memory consumption while considerably enhancing the details of a scene. However, several issues need to be considered to achieve convincing results. First, thin geometry easily exhibits false occlusion halos. Second, distant occluders are easily undersampled in screen-space. Third, geometry must be tessellated sufficiently high. Finally, artefacts easily appear if occluders are nearly parallel to the direction of view.

6.2.4 Multi-Pass Rendering

According to scattering, in figure 5.2 rendering the spherical view of PointLight to the CubeShadowMap required six passes. Since the introduction of geometry shaders, cube maps can be rendered in a single pass. However, first, geometry is simply duplicated by the geometry processor for six layers. Then, each layer is rasterised to the depth texture of the

according cube map face using multiple render targets. Therefore, geometry is duplicated and rasterised repeatedly and redundantly in practice. Decomposing geometry into six disjoint subsets with respect to the cube map faces is non-trivial. As a consequence, significant gains in performance are potentially outweighed by excessive geometry processing and redundant rasterisation; particularly due to feeble geometry processing units on GPUs supporting Shader Model 4 only.

According to gather, in figure 5.1 the 512 samples of the SampledAreaLight were rendered with eight samples per pass. Accordingly, 64 passes were necessary to accumulatively evaluate partial illumination of the single area light source with sampling. In essence, multi-pass rendering infers to submit scene geometry repeatedly which scales poorly. Furthermore, lighting computations and occlusion testing is applied for a considerable amount of fragments which are potentially discarded due to visibility determination with z-buffering.

In contrast, *Deferred Shading* [HH04] decouples lighting from surface shading to allow for blending individual contributions of numerous lights in screen-space. First, while rendering geometry, depth, surface normal, diffuse colour, specular colour and specular exponent are aggregated in a screen-sized offscreen buffer (G-buffer). This requires to utilise multiple render targets because the G-buffer consists of several two-dimensional textures. Second, while looping over all lights the view volume of each light is rendered only. Lighting computations and occlusion testing are performed with data read from the G-buffer and shadow maps, respectively. As a consequence, computations are confined to pixels being affected by the currently processed light only. Finally, the individual contributions of lights are accumulated with additive blending. However, the G-buffer has a high storage cost and infers high bandwidth usage; particularly if lights overlap. Furthermore, G-buffer data must be stored with sufficient precision, otherwise artefacts are introduced.

Deferred Lighting [Hof09] minimises data stored in the G-buffer. First, while rendering geometry, the surface normal and the specular exponent are stored in a single offscreen buffer with four components only. Second, while looping over all lights the view volume of each light is rendered only. The results of lighting computations and occlusion testing are accumulated in separate offscreen buffers. Finally, while rendering geometry again, the diffuse and specular colours are modulated with lighting and shadowing terms read from offscreen buffers. As a consequence, less input data must be resident and the cost of computation for each light is considerably reduced.

6.2.5 GPU-Based Global Illumination

Missing global illumination effects in Apelles such as reflections, refractions, caustics and indirect illumination can be included with extended environment mapping while achieving convincing results efficiently.

Initially cubic environment mapping has been suggested for approximately tracing secondary rays while rendering objects which are perfect mirrors [Gre86]. The reflected view vector is used to look up the incoming radiance from one direction in a colour cube map that has been generated from the centre of the reflecting object (reference point). For including the increase in reflectance if light impinges at shallow angles the irradiance is multiplied by the FRESNEL term F which can be approximated well and efficiently [Sch94].

Recursive reflections are achievable by using a dedicated cube map per reflector [NC02]. First, all cube maps are initialised with the irradiance of the global environment. Then, while rendering a single reflector, cube maps of other reflectors are updated successively. Efficient but approximate recursive reflections are achievable, if only one cube map is updated per frame or if updating is postponed for several frames.

If the surface of objects exhibits roughness, glossy and diffuse reflections can be computed by filtering the cube map [Gre86]. In order to achieve more physically realistic results the cube map is filtered using BRDF lobes [KM00]. If surface roughness varies spatially, several cube maps filtered with BRDF lobes of varying sizes are aggregated in a MIP chain [GKD07]. While achieving reasonable performance, the quality of mipmapping is significantly improved with *N-Buffers* [Déc05], if implemented efficiently on contemporary hardware utilising a hierarchical approach [ED06]. However, if BRDF lobes are uniform and radially symmetric only, objectionable errors appear on flat surfaces. Using multiple cube maps aggregated in an array, arbitrary BRDFs can be approximated by accumulating filtering with multiple lobes [KM00]. In contrast, high quality results are obtained at a considerably higher computational cost by utilising *Monte Carlo quadrature with importance sampling* [CK07].

At the surface of transparent objects, the refracted view direction is computed with the SNELLIUS law of refraction which involves the refractive indices of the participating materials n_1 and n_2. The proportion of irradiance that is being transmitted depends on the FRESNEL term $1 - F$ and the ratio of the refractive indices n_2/n_1. Of course, for yielding more accurate results, the refracted view vector is bent a second time upon leaving the transmitter before looking up the irradiance from the cube map. Furthermore, if the index of refraction varies with the wavelength of light, *dispersion* occurs. Depending on the surface curvature of objects, both reflections and refractions introduce *caustics* where light is focused on diffuse surfaces.

Cubic environment mapping suffers from several limitations which hinders to yield convincing results [AMHH08]. First, both viewing vector and normal vector must vary sufficiently across surfaces. Otherwise, only a small region of the cube map is being sampled while shading the surface of the object. Second, self occlusion is ignored because the

environment is assumed to be fully visible from the reference point. Third, environment mapping produces only accurate results if distant objects have been rendered into the cube map only. As a consequence, the distance between the reference point and the surface point currently being shaded is negligible. Therefore, the cube map can simply be looked up with directional vectors at the reference point.

However, environment mapping can be extended with a sampled representation of local geometry (localisation). Convincing results are achievable efficiently by utilising approximate ray-tracing with distance impostors [SKALP05]. The cube map additionally includes the distance of surrounding geometry from the reference point (distance impostor). An iterative ray intersection method is applied to approximate a localised direction for looking up the cube map at the reference point. In essence, the approximated direction identifies the point on the environment surface that is hit by a given ray shot from a given surface point instead of the reference point. As a consequence, the quality of both reflections and refractions increase considerably; particularly with a few iterations. If an additional cube map stores the surface normal per texel (refractor map), secondary refractions can be approximated similarly. Furthermore, caustics can be computed using a two pass approach which is based on *Photon Mapping* [Jen96]. First, light paths are followed to obtain a texture that stores where a photon hits the closest diffuse surface. Additionally, the texture stores the power of the photon which passes through the texel along the path of light. Then, a caustic map is rendered by splatting the photons onto diffuse surfaces. Visually pleasing caustic patterns require two final steps. First, the contribution of individual photon hits must be combined appropriately with filtering. Second, the reflected proportion of caustic illumination must be computed with respect to the local BRDF. *Caustics Mapping* [SKP07] improves the efficiency of approximately tracing light paths and details how caustic generation can be combined with *Shadow Mapping* [Wil78].

Similar to approximating specular effects, indirect diffuse and moderately glossy illumination can be approximated efficiently with localised cubic environment mapping [SKL06]. The texels of the cubic environment map essentially correspond to small virtual lights whose radiance impinges at a surface point. Accordingly, the radiance and distance of numerous small virtual lights is sampled in a cube map. Then, the cube map is down-sampled while averaging radiance and distance of the small virtual lights. As a consequence, numerous small virtual lights are clustered into a few large area light sources. Hence, the last bounce irradiance of the area lights at a surface point can be determined efficiently with a few localised cube map lookups only. Of course, self-shadowing caused by indirect light is ignored.

Cascaded Light Propagation Volumes [KD10] supply a volume-based approach for approximating diffuse indirect illumination including occlusion efficiently and convincingly while achieving high temporal coherence. In particular, it has to be noted that the approach has been implemented into the deferred rendering pipeline of the CryENGINE® 3. Therefore, the algorithm is applicable to fully dynamic scenes even on feeble hardware. Furthermore, the approach is extensible to support multiple bounces of indirect light, glossy reflections and volumetric effects of participating media. First, in order to create a coarse volumetric representation of the spatial and angular distribution of light in the scene, a light propagation volume (LPV) is created by uniformly sampling a point cloud of secondary light sources into *Reflective Shadow Maps* (RSM) [DS05]. Multiple layers of depth are captured by utilising depth peeling [Eve02]. The resulting LPV is a volumetric texture which compactly approximates a three dimensional radiance field using low-order spherical harmonics coefficients [RH01] distributed in a grid. Additionally, a coarse representation of blocker geometry is created in a second volumetric texture (GV) to account for fuzzy occlusion. Then, light transport is propagated between voxels in the LPV using an iterative scheme while including blocking contributions of the GV. In large scenes the number of iterations necessary to propagate medium and long distance light transport is considerably reduced with a cascaded approach. Instead of using very large grids, LPV and GV grids are partitioned into multiple cascaded grids of small size to adaptively increase the sampling density for visually important regions (close to the viewer). Light transport at short distances is approximated with screen-space techniques. Finally, the distribution of light is obtained by accumulating the results of all iterations. The approach is mainly applicable for computing low-frequency indirect lighting due to the limitations of spatial discretisation, low-order spherical harmonics and the proposed propagation scheme. Particularly, as a consequence of a coarse volumetric representation and low-frequency approximations of the lighting, the approach is susceptible to light bleeding and incomplete self-illumination.

References

[ADM*08] ANNEN T., DONG Z., MERTENS T., BEKAERT P., SEIDEL H.-P., KAUTZ J.: Real-time, all-frequency shadows in dynamic scenes. *ACM Trans. Graph. 27*, 3 (2008), 1–8. 28, 34, 35

[AHL*06] ATTY L., HOLZSCHUCH N., LAPIERRE M., HASENFRATZ J.-M., HANSEN C., SILLION F.: Soft shadow maps: Efficient sampling of light source visibility. *Computer Graphics Forum 25*, 4 (December 2006), 725–741. 29, 30, 33

[AHT04] ARVO J., HIRVIKORPI M., TYYSTJÄRVI J.: Approximate soft shadows with an image-space flood-fill algorithm. *Computer Graphics Forum 23*, 3 (2004), 271–279. 26

[AL04] AILA T., LAINE S.: Alias-free shadow maps. In *Proceedings of Eurographics Symposium on Rendering 2004* (2004), Eurographics Association, pp. 161–166. 22

[AMB*07] ANNEN T., MERTENS T., BEKAERT P., SEIDEL H.-P., KAUTZ J.: Convolution shadow maps. In *Rendering Techniques 2007: Eurographics Symposium on Rendering* (2007), Eurographics, pp. 51–60. 20, 25, 28, 104

[AMHH08] AKENINE-MÖLLER T., HAINES E., HOFFMAN N.: *Real-Time Rendering*, 3 ed. A. K. Peters, Ltd., 2008. 41, 105

[AMS*08] ANNEN T., MERTENS T., SEIDEL H.-P., FLERACKERS E., KAUTZ J.: Exponential shadow maps. In *GI '08: Proceedings of Graphics Interface 2008* (2008), Canadian Information Processing Society, pp. 155–161. 21, 25, 104

[ARHM00] AGRAWALA M., RAMAMOORTHI R., HEIRICH A., MOLL L.: Efficient image-based methods for rendering soft shadows. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques* (2000), SIGGRAPH '00, ACM Press/Addison-Wesley Publishing Co., pp. 375–384. 31

[Arv04] ARVO J.: Tiled shadow maps. In *Proceedings of the Computer Graphics International* (2004), IEEE Computer Society, pp. 240–247. 16

[Arv07] ARVO J.: Alias-free shadow maps using graphics hardware. *Journal of Graphics, GPU, and Game Tools 12*, 1 (2007), 47–59. 23

[ASK06] ASZÓDI B., SZIRMAY-KALOS L.: Real-time soft shadows with shadow accumulation. In *Eurographics 2006 Short Papers* (September 2006), pp. 53–56. 29

[BAS02a] BRABEC S., ANNEN T., SEIDEL H.-P.: Practical shadow mapping. *Journal of Graphics Tools 7* (December 2002), 9–18. 9, 24, 48, 101

[BAS02b] BRABEC S., ANNEN T., SEIDEL H.-P.: Shadow mapping for hemispherical and omnidirectional light sources. In *Advances in Modelling, Animation and Rendering (Proceedings of Computer Graphics International 2002)* (2002), Springer, pp. 397–408. 24

[Bav08] BAVOIL L.: Advanced soft shadow mapping techniques. Presentation, Game Developers Conference, 2008. 30

[BCS08] BAVOIL L., CALLAHAN S. P., SILVA C. T.: Robust soft shadow mapping with backprojection and depth peeling. *Journal of Graphics Tools 13*, 1 (2008), 19–30. 30, 31

[Bli77] BLINN J. F.: Models of light reflection for computer synthesized pictures. *SIGGRAPH Computer Graphics 11* (July 1977), 192–198. 41

[BMR*96] BUSCHMANN F., MEUNIER R., ROHNERT H., SOMMERLAD P., STAL M.: *Pattern-Oriented Software Architecture, Volume 1: A System of Patterns*. John Wiley & Sons, 1996. 80

[BS02] BRABEC S., SEIDEL H.-P.: Single sample soft shadows using depth maps. In *Proceedings of Graphics Interface 2002* (May 2002), pp. 219–228. 26

[BS06] BAVOIL L., SILVA C. T.: Real-time soft shadows with cone culling. In *ACM SIGGRAPH 2006 Sketches* (July 2006), SIGGRAPH '06, ACM. 29, 31

[BSD08] BAVOIL L., SAINZ M., DIMITROV R.: Image-space horizon-based ambient occlusion. In *ACM SIGGRAPH 2008 talks* (2008), SIGGRAPH '08, ACM, pp. 22:1–22:1. 35, 36, 104

[Bun05] BUNNELL M.: Dynamic ambient occlusion and indirect lighting. In *GPU Gems 2 – Techniques for Graphics and Compute-Intensive Programming*, Pharr M., (Ed.), vol. 2 of GPU Gems. Addison-Wesley, 2005, pp. 223–234. 35

[Bur06] BURLEY B.: Shadow map bias cone and improved soft shadows: Disney bonus section. In *ACM SIGGRAPH 2006 Courses* (2006), SIGGRAPH '06, ACM. 23

[C++11] *ISO/IEC 14882:2011 Information Technology – Programming Languages – C++*. ISO International Organization for Standardization, September 2011. 80

[Cat74] CATMULL E. E.: *A subdivision algorithm for computer display of curved surfaces*. PhD thesis, University of Utah, 1974. 4

[Chr06] CHRISTENSEN P.: Ray tracing for the movie 'cars'. Presentation, Ray Tracing Symposium, 2006. 4

[CK07] COLBERT M., KŘIVÁNEK J.: Gpu-based importance sampling. In *GPU Gems 3 – Programming Techniques for High-Performance Graphics and General-Purpose Computation*, Nguyen H., (Ed.), vol. 3 of GPU Gems. Addison-Wesley, 2007, pp. 459–476. 105

[Cro77] CROW F. C.: Shadow algorithms for computer graphics. In *Proceedings of the 4th Annual Conference on Computer Graphics and Interactive Techniques* (1977), SIGGRAPH '77, ACM, pp. 242–248. 6

[CT81] COOK R. L., TORRANCE K. E.: A reflectance model for computer graphics. *SIGGRAPH Computer Graphics 15* (August 1981), 307–316. 40

[DBB06] DUTRÉ P., BALA K., BEKAERT P.: *Advanced Global Illumination*, 2nd ed. A. K. Peters, Ltd., 2006. 37, 41

[Déc05] DÉCORET X.: N-buffers for efficient depth map query. *Computer Graphics Forum 24*, 3 (2005), 393–400. 28, 32, 104, 105

[DH06] DUNBAR D., HUMPHREYS G.: A spatial data structure for fast poisson-disk sample generation. In *ACM SIGGRAPH 2006 Papers* (2006), SIGGRAPH '06, ACM, pp. 503–508. 54, 60, 72

[DL06] DONNELLY W., LAURITZEN A.: Variance shadow maps. In *Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games* (2006), I3D '06, ACM, pp. 161–165. 19, 25, 56, 90, 104

[DS05] DACHSBACHER C., STAMMINGER M.: Reflective shadow maps. In *Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games* (2005), I3D '05, ACM, pp. 203–231. 106

[DU07] DMITRIEV K., URALSKY Y.: Soft shadows using hierarchical min-max shadow maps. Presentation, Game Developers Conference, 2007. 30

[EASW09] EISEMANN E., ASSARSSON U., SCHWARZ M., WIMMER M.: Casting shadows in real time. In *ACM SIGGRAPH ASIA 2009 Courses* (2009), SIGGRAPH ASIA '09, ACM. 6, 25, 26, 27, 28, 30, 31, 33, 41, 43, 45, 53, 54

[ED06] EISEMANN E., DÉCORET X.: Plausible image based soft shadows using occlusion textures. In *Proceedings of the 19th Brazilian Symposium on Computer Graphics and Image Processing* (October 2006), SIBGRAPI, pp. 155–162. 28, 32, 35, 104, 105

[ED08] EISEMANN E., DÉCORET X.: Occlusion textures for plausible soft shadows. *Computer Graphics Forum 27*, 1 (March 2008), 13–23. 32, 35, 104

[Eng07] ENGEL W.: Cascaded shadow maps. In *ShaderX*^{5} *– Advanced Rendering Techniques*, Engel W., (Ed.), vol. 5 of ShaderX. Charles River Media, 2007. 14

[Eve02] EVERITT C.: *Interactive order-independent transparency*. Tech. rep., NVIDIA Corporation, 2002. 24, 30, 33, 106

[Fer05] FERNANDO R.: Percentage-closer soft shadows. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Sketches* (2005), ACM, p. 35. 26, 35, 51, 52, 56, 89, 90, 94, 104

[FFBG01] FERNANDO R., FERNANDEZ S., BALA K., GREENBERG D. P.: Adaptive shadow maps. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques* (2001), SIGGRAPH '01, ACM, pp. 387–390. 16

[GBP06] GUENNEBAUD G., BARTHE L., PAULIN M.: Realtime soft shadow mapping by backprojection. In *In Proceedings of Eurographics Symposium on Rendering* (June 2006), pp. 227–234. 29, 30

[GBP07] GUENNEBAUD G., BARTHE L., PAULIN M.: High-quality adaptive soft shadow mapping. *Computer Graphics Forum (Proceedings of Eurographics 2007) 26*, 3 (September 2007), 525–533. 30, 31, 35

[Ger04] GERASIMOV P. S.: Omnidirectional shadow mapping. In *GPU Gems – Programming Techniques, Tips and Tricks for Real-Time Graphics*, Fernando R., (Ed.), vol. 1 of GPU Gems. Addison-Wesley, 2004. 24, 25, 50

[GHFP08] GASCUEL J.-D., HOLZSCHUCH N., FOURNIER G., PÉROCHE B.: Fast non-linear projections using graphics hardware. In *Proceedings of the 2008 Symposium on Interactive 3D Graphics and Games* (2008), I3D '08, ACM, pp. 107–114. 24

[GHJV94] GAMMA E., HELM R., JOHNSON R., VLISSIDES J.: *Design Patterns: Elements of Reusable Object-Oriented Software*, 1 ed. Addison-Wesley Professional, 1994. 60, 63, 65, 66, 77

[GKD07] GREEN P., KAUTZ J., DURAND F.: Efficient reflectance and visibility approximations for environment map rendering. *Computer Graphics Forum (Proceedings of Eurographics 2007) 26*, 3 (2007), 495–502. 105

[Gre86] GREENE N.: Environment mapping and other applications of world projections. *Computer Graphics and Applications, IEEE 6*, 11 (Nov 1986), 21 –29. 105

[Gru10] GRUEN H.: Fast conventional shadow filtering. In *GPU Pro – Advanced Rendering Techniques*, Engel W., (Ed.), vol. 1 of GPU Pro. A. K. Peters, Ltd., 2010. 27

[GW07a] GIEGL M., WIMMER M.: Fitted virtual shadow maps. In *Proceedings of Graphics Interface 2007* (2007), GI '07, ACM, pp. 159–168. 17, 93, 104

[GW07b] GIEGL M., WIMMER M.: Queried virtual shadow maps. In *Proceedings of ACM SIGGRAPH 2007 Symposium on Interactive 3D Graphics and Games*. ACM Press, April 2007, pp. 65–72. 17, 93, 104

[Hei99] HEIDRICH W.: *High-quality shading and lighting for hardware-accelerated rendering*. PhD thesis, Universität Erlangen, 1999. 24

[HH97] HECKBERT P. S., HERF M.: *Simulating soft shadows with graphics hardware*. Tech. rep., Carnegie Mellon University, 1997. 44

[HH04] HARGREAVES S., HARRIS M.: 6800 leagues under the sea: Deferred shading. Presentation, 6800 Leagues Under the Sea Event, 2004. 105

[HJ07] HOBEROCK J., JIA Y.: High-quality ambient occlusion. In *GPU Gems 3 – Programming Techniques for High-Performance Graphics and General-Purpose Computation*, Nguyen H., (Ed.), vol. 3 of GPU Gems. Addison-Wesley, 2007, pp. 257–274. 35

[HKSB06] HADWIGER M., KRATZ A., SIGG C., BÜHLER K.: Gpu-accelerated deep shadow maps for direct volume rendering. In *Proceedings of the 21st ACM SIGGRAPH/EUROGRAPHICS Symposium on Graphics Hardware* (2006), ACM, pp. 49–52. 8

[HLHS03] HASENFRATZ J.-M., LAPIERRE M., HOLZSCHUCH N., SILLION F.: A survey of real-time soft shadows algorithms. *Computer Graphics Forum 22*, 4 (Dec 2003), 753–774. 26

[HN85] HOURCADE J., NICOLAS A.: Algorithms for antialiased cast shadows. *Computers Graphics 9*, 3 (1985), 259 – 265. 24

[Hof09] HOFFMAN N.: Deferred lighting approaches. Webpage, http://www.realtimerendering.com/blog/deferred-lighting-approaches/, June 2009. Last access 2012-03-21. 105

[Hol11] HOLBERT D.: Saying goodbye to shadow acne. Presentation, Game Developers Conference, 2011. 23, 104

[HSC*05] HENSLEY J., SCHEUERMANN T., COOMBE G., SINGH M., LASTRA A.: Fast summed-area table generation and its applications. *Computer Graphics Forum 24*, 3 (2005), 547–555. 28

[HTSG91] HE X. D., TORRANCE K. E., SILLION F. X., GREENBERG D. P.: A comprehensive physical model for light reflection. In *Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques* (1991), SIGGRAPH '91, ACM, pp. 175–186. 40

[Hua10] HUANG J.-H.: Opening keynote. Presentation, GPU Technology Conference, 2010. 4

[IEE08] *IEEE Standard for Floating-Point Arithmetic*. IEEE Std. 754-2008, August 2008. 102

[Isi06] ISIDORO J. R.: Shadow mapping: Gpu-based tips and techniques. Presentation, Game Developers Conference, 2006. 23, 27, 54, 72, 73, 91, 104

[Jen96] JENSEN H. W.: Global illumination using photon maps. In *Rendering Techniques '96 (Proceedings of the Seventh Eurographics Workshop on Rendering)* (1996), Pueyo X., Schröder P., (Eds.), Springer-Verlag, pp. 21–30. 106

[JLBM05] JOHNSON G. S., LEE J., BURNS C. A., MARK W. R.: The irregular z-buffer: Hardware acceleration for irregular data structures. *ACM Transactions on Graphics 24* (October 2005), 1462–1482. 23

[JMB04] JOHNSON G. S., MARK W. R., BURNS C. A.: The irregular z-buffer and its application to shadow mapping. Research paper, The University of Texas at Austin, 2004. 23

[Kaj86] KAJIYA J. T.: The rendering equation. In *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques* (1986), SIGGRAPH '86, ACM, pp. 143–150. 38

[KD03] KIRSCH F., DÖLLNER J.: Real-time soft shadows using a single light sample. *Journal of WSCG 11*, 2 (February 2003), 255–262. 26

[KD10] KAPLANYAN A., DACHSBACHER C.: Cascaded light propagation volumes for real-time indirect illumination. In *Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (2010), I3D '10, ACM, pp. 99–107. 106

[Kil01] KILGARD M.: Shadow mapping with today's opengl hardware. Presentation, Computer Entertainment Developers Conference, 2001. 23

[KL05] KONTKANEN J., LAINE S.: Ambient occlusion fields. In *Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games* (2005), I3D '05, ACM, pp. 41–48. 34

[KM00] KAUTZ J., MCCOOL M. D.: Approximation of glossy reflection with prefiltered environment maps. In *Proceedings of Graphics Interface 2000* (2000), pp. 119–126. 105

[Koz04] KOZLOV S.: Perspective shadow maps - care and feeding. In *GPU Gems – Programming Techniques, Tips and Tricks for Real-Time Graphics*, Fernando R., (Ed.), vol. 1 of GPU Gems. Addison-Wesley, 2004, pp. 217–244. 15, 24, 25

[Lam60] LAMBERT J. H.: *Photometria sive de mensure de gratibus luminis, colorum umbrae*. Eberhard Klett, 1760. 40

[Lau07] LAURITZEN A.: Summed-area variance shadow maps. In *GPU Gems 3 – Programming Techniques for High-Performance Graphics and General-Purpose Computation*, Nguyen H., (Ed.), vol. 3 of GPU Gems. Addison-Wesley, 2007, pp. 157–182. 19, 20, 23, 28, 35, 53, 90, 104

[LGQ*08] LLOYD D. B., GOVINDARAJU N. K., QUAMMEN C., MOLNAR S. E., MANOCHA D.: Logarithmic perspective shadow maps. *ACM Transactions on Graphics 27* (October 2008), 1–32. 11, 12, 13, 14, 15, 25, 47, 90, 103

[Lia10] LIAO H.-C.: Shadow mapping for omnidirectional lights using tetrahedron mapping. In *GPU Pro – Advanced Rendering Techniques*, Engel W., (Ed.), vol. 1 of GPU Pro. A. K. Peters, Ltd., 2010. 24

[LM08] LAURITZEN A., MCCOOL M.: Layered variance shadow maps. In *Proceedings of Graphics Interface 2008* (2008), GI '08, Canadian Information Processing Society, pp. 139–146. 20, 22, 90, 104

[LMCY11] LIANG X.-H., MA S., CEN L.-X., YU Z.: Light space cascaded shadow maps algorithm for real time rendering. *Journal of Computer Science and Technology 26* (January 2011), 176–186. 15

[LSL11] LAURITZEN A., SALVI M., LEFOHN A.: Sample distribution shadow maps. In *Symposium on Interactive 3D Graphics and Games* (2011), I3D '11, ACM, pp. 97–102. 14, 103

[LSO07] LEFOHN A. E., SENGUPTA S., OWENS J. D.: Resolution-matched shadow maps. *ACM Transactions on Graphics 26*, 4 (October 2007), 20:1–20:17. 16

[LV00] LOKOVIC T., VEACH E.: Deep shadow maps. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques* (2000), SIGGRAPH '00, ACM Press/Addison-Wesley Publishing Co., pp. 385–392. 8, 18, 32

[Mar96] MARTIN R. C.: *The Interface Segregation Principle*. Object Mentor, 1996. 58, 59

[MFS09] MÉNDEZ-FELIU À., SBERT M.: From obscurances to ambient occlusion: A survey. *The Visual Computer 25*, 2 (February 2009), 181–196. 34

[Mil94] MILLER G.: Efficient algorithms for local and global accessibility shading. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques* (1994), SIGGRAPH '94, ACM, pp. 319–326. 34

[Mit07] MITTRING M.: Finding next gen: Cryengine 2. In *ACM SIGGRAPH 2007 Courses* (2007), SIGGRAPH '07, ACM, pp. 97–121. 35, 104

[MKHS10] MOHAMMADBAGHER M., KAUTZ J., HOLZSCHUCH N., SOLER C.: Screen-space percentage-closer soft shadows. In *SIGGRAPH '10: ACM SIGGRAPH 2010 Posters* (2010), ACM, pp. 1–1. 27

[MMAH07] MALMER M., MALMER F., ASSARSSON U., HOLZSCHUCH N.: Fast precomputed ambient occlusion for proximity shadows. *Journal of Graphics Tools 12*, 2 (2007), 59–71. 35, 36

[MT04] MARTIN T., TAN T.-S.: Anti-aliasing and continuity with trapezoidal shadow maps. In *Proceedings of Eurographics Symposium on Rendering* (2004), pp. 153–160. 11, 24

[NC02] NIELSEN K. H., CHRISTENSEN N. J.: Real-time recursive specular reflections on planar and curved surfaces using graphics hardware. In *WSCG (Short Papers)* (2002), pp. 91–98. 105

[Nic70] NICODEMUS F. E.: Reflectance nomenclature and directional reflectance and emissivity. *Applied Optics 9*, 6 (Jun 1970), 1474–1475. 39

[NJH10] NGUYEN K., JANG H., HAN J.: Layered occlusion map for soft shadow generation. *The Visual Computer 26* (2010), 1497–1512. 30, 33, 36

[OCL10] *The OpenCL Specification*. Version 1.1. Khronos OpenCL Working Group, September 2010. 77

[Pha04] PHARR M.: Dynamic ambient occlusion and indirect lighting. In *GPU Gems – Programming Techniques, Tips and Tricks for Real-Time Graphics*, Fernando R., (Ed.), vol. 1 of GPU Gems. Addison-Wesley, 2004, pp. 279–292. 34, 104

[Pho75] PHONG B. T.: Illumination for computer generated pictures. *Communications of the ACM 18* (June 1975), 311–317. 41

[PSS98] PARKER S., SHIRLEY P., SMITS B.: *Single sample soft shadows*. Tech. rep., University of Utah, October 1998. 26

[RH01] RAMAMOORTHI R., HANRAHAN P.: An efficient representation for irradiance environment maps. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 2001), SIGGRAPH '01, ACM, pp. 497–500. 106

[RSC87] REEVES W. T., SALESIN D. H., COOK R. L.: Rendering antialiased shadows with depth maps. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques* (July 1987), SIGGRAPH '87, pp. 283–291. 7, 18, 23, 49, 90

[RT06] RONG G., TAN T.-S.: Utilizing jump flooding in image-based soft shadows. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology* (2006), VRST '06, ACM, pp. 173–180. 26

[RWS*06] REN Z., WANG R., SNYDER J., ZHOU K., LIU X., SUN B., SLOAN P.-P., BAO H., PENG Q., GUO B.: Real-time soft shadows in dynamic scenes using spherical harmonic exponentiation. In *ACM SIGGRAPH 2006 Papers* (2006), SIGGRAPH '06, ACM, pp. 977–986. 34, 36

[SA07] SHANMUGAM P., ARIKAN O.: Hardware accelerated ambient occlusion techniques on gpus. In *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games* (2007), I3D '07, pp. 73–80. 35, 36, 104

[SAPP05] ST-AMOUR J.-F., PAQUETTE E., POULIN P.: Soft shadows from extended light sources with penumbra deep shadow maps. In *Proceedings of Graphics Interface 2005* (2005), GI '05, Canadian Human-Computer Communications Society, pp. 105–112. 32

[SC97] SHIRLEY P., CHIU K.: A low distortion map between disk and square. *Journal of Graphics Tools 2* (December 1997), 45–52. 54, 72

[Sch94] SCHLICK C.: An inexpensive brdf model for physically-based rendering. *Computer Graphics Forum 13*, 3 (Sept 1994), 149–162. 105

[SCH03] SEN P., CAMMARANO M., HANRAHAN P.: Shadow silhouette maps. In *ACM SIGGRAPH 2003 Papers* (2003), SIGGRAPH '03, ACM, pp. 521–526. 7

[Sch05] SCHÜLER C.: Eliminating surface acne with gradient shadow mapping. In *ShaderX*^{4} *– Advanced Rendering Techniques*, Engel W., (Ed.), vol. 4 of ShaderX. Charles River Media, 2005. 23, 104

[Sch07] SCHÜLER C.: Multisampling extension for gradient shadow maps. In *ShaderX*^{5} *– Advanced Rendering Techniques*, Engel W., (Ed.), vol. 5 of ShaderX. Charles River Media, 2007. 23, 104

[SD02] STAMMINGER M., DRETTAKIS G.: Perspective shadow maps. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques* (2002), SIGGRAPH '02, pp. 557–562. 9, 10

[SEA08] SINTORN E., EISEMANN E., ASSARSSON U.: Sample-based visibility for soft shadows using alias-free shadow maps. *Computer Graphics Forum (Proceedings of the Eurographics Symposium on Rendering 2008) 27*, 4 (June 2008), 1285–1292. 23

[Sen04] SEN P.: Silhouette maps for improved texture magnification. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware* (2004), HWWS '04, ACM, pp. 65–73. 8

[SGHS98] SHADE J., GORTLER S., HE L.-W., SZELISKI R.: Layered depth images. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques* (1998), SIGGRAPH '98, pp. 231–242. 31

[SJW07] SCHERZER D., JESCHKE S., WIMMER M.: Pixel-correct shadow maps with temporal reprojection and shadow test confidence. In *Rendering Techniques 2007 (Proceedings Eurographics Symposium on Rendering)* (June 2007), Kautz J., Pattanaik S., (Eds.), Eurographics, Eurographics Association, pp. 45–50. 18

[SKALP05] SZIRMAY-KALOS L., ASZÓDI B., LAZÁNYI I., PREMECZ M.: Approximate ray-tracing on the gpu with distance impostors. *Computer Graphics Forum 24*, 3 (2005), 695–704. 106

[SKL06] SZIRMAY-KALOS L., LAZÁNYI I.: Indirect diffuse and glossy illumination on the gpu. In *Proceedings of the 22nd Spring Conference on Computer Graphics* (April 2006), pp. 29–35. 106

[SKP07] SHAH M. A., KONTTINEN J., PATTANAIK S.: Caustics mapping: An image-space technique for real-time caustics. *IEEE Transactions on Visualization and Computer Graphics 13* (March 2007), 272–280. 57, 106

[SKU08] SZIRMAY-KALOS L., UMENHOFFER T.: Displacement mapping on the gpu – state of the art. *Computer Graphics Forum 27*, 6 (2008), 1567–1592. 57

[SS98] SOLER C., SILLION F. X.: Fast calculation of soft shadow textures using convolution. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques* (1998), SIGGRAPH '98, ACM, pp. 321–332. 26, 32, 42, 44

[SS07] SCHWARZ M., STAMMINGER M.: Bitmask soft shadows. *Computer Graphics Forum (Proceedings of Eurographics 2007) 26*, 3 (September 2007), 515–524. 29, 30, 35

[SS08a] SCHWARZ M., STAMMINGER M.: Microquad soft shadow mapping revisited. *Eurographics 2008 Annex to the Conference Proceedings (Short Papers)* (April 2008), 295–298. 30

[SS08b] SCHWARZ M., STAMMINGER M.: Quality scalability of soft shadow mapping. In *Proceedings of Graphics Interface 2008* (May 2008), GI '08, Canadian Information Processing Society, pp. 147–154. 30, 31

[SSM11] SCHERZER D., SCHWÄRZLER M., MATTAUSCH O.: Fast soft shadows with temporal coherence. In *GPU Pro 2 – Advanced Rendering Techniques*, Engel W., (Ed.), vol. 2 of GPU Pro. A. K. Peters, Ltd., 2011. 33

[SSMW09] SCHERZER D., SCHWÄRZLER M., MATTAUSCH O., WIMMER M.: Real-time soft shadows using temporal coherence. In *Proceedings of the 5th International Symposium on Advances in Visual Computing: Part II* (2009), ISVC '09, Springer-Verlag, pp. 13–24. 28, 33, 36

[SWP10] SCHERZER D., WIMMER M., PURGATHOFER W.: A survey of real-time hard shadow mapping methods. In *State of the Art Reports Eurographics* (May 2010), Eurographics. 6, 25, 46, 49

[TM98] TOMASI C., MANDUCHI R.: Bilateral filtering for gray and color images. In *Proceedings of the Sixth International Conference on Computer Vision* (1998), ICCV '98, IEEE Computer Society, p. 839. 27

[Ura05] URALSKY Y.: Efficient soft-edged shadows using pixel shader branching. In *GPU Gems 2 – Techniques for Graphics and Compute-Intensive Programming*, Pharr M., (Ed.), vol. 2 of GPU Gems. Addison-Wesley, 2005. 27, 54

[VdB04] VALIENT M., DE BOER W. H.: Fractional-disk soft shadows. In *ShaderX*^{3} *– Advanced Rendering Techniques with DirectX and OpenGL*. Wordware Publishing, 2004. 23, 27

[WDB*06] WALD I., DIETRICH A., BENTHIN C., EFREMOV A., DAHMEN T., GÜNTHER J., HAVRAN V., SEIDEL H., SLUSALLEK P.: A ray tracing based framework for high-quality virtual reality in industrial design applications. In *Proceedings of the 2006 IEEE Symposium on Interactive Ray Tracing*. IEEE Press, 2006, pp. 177–185. 4

[WE03] WEISKOPF D., ERTL T.: Shadow mapping based on dual depth layers. In *Proceedings of Eurographics '03 Short Papers* (2003), pp. 53–60. 24

[Wil78] WILLIAMS L.: Casting curved shadows on curved surfaces. In *SIGGRAPH '78: Proceedings of the 5th Annual Conference on Computer Graphics and Interactive Techniques* (1978), ACM, pp. 270–274. 4, 6, 23, 44, 56, 89, 90, 103, 106

[Wil83] WILLIAMS L.: Pyramidal parametrics. *SIGGRAPH Comput. Graph. 17* (July 1983), 1–11. 16

[WM94] WANG Y., MOLNAR S.: *Second-Depth Shadow Mapping*. Tech. rep., University of North Carolina at Chapel Hill, 1994. 24, 49, 53, 61, 87, 90, 91, 93, 95

[Woo92] WOO A.: *The Shadow Depth Map Revisited*. Academic Press Professional, Inc., 1992, pp. 338–342. 24

[WSP04] WIMMER M., SCHERZER D., PURGATHOFER W.: Light space perspective shadow maps. In *Rendering Techniques 2004 (Proceedings of the Eurographics Symposium on Rendering)* (June 2004), Keller A., Jensen H. W., (Eds.), Eurographics, Eurographics Association, pp. 143–151. 10, 12, 25, 46, 90, 103

[YDF*10] YANG B., DONG Z., FENG J., SEIDEL H.-P., KAUTZ J.: Variance soft shadow mapping. *Computer Graphics Forum 29*, 7 (2010), 2127–2134. 28, 35, 104

[YFGL09] YANG B., FENG J., GUENNEBAUD G., LIU X.: Packet-based hierarchal soft shadow mapping. *Computer Graphics Forum (Proceedings of the Eurographics Symposium on Rendering 2009) 28*, 4 (June 2009), 1121–1130. 30, 31, 36

[ZSXL06] ZHANG F., SUN H., XU L., LUN L. K.: Parallel-split shadow maps for large-scale virtual environments. In *Proceedings of the 2006 ACM International Conference on Virtual Reality Continuum and its Applications* (2006), ACM, pp. 311–318. 14, 103

[ZZB09] ZHANG F., ZAPRIJAGAEV A., BENTHAM A.: Practical cascaded shadow maps. In *ShaderX*^{7} *– Advanced Rendering Techniques*, Engel W., (Ed.), vol. 7 of ShaderX. Charles River Media, March 2009. 14, 15, 103

List of Tables

List of Figures