Guided Sampling of Illumination for Real-Time Path Tracing

Zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

von der KIT-Fakultät für Informatik des
Karlsruher Instituts für Technologie (KIT)

genehmigte

Dissertation

von

Addis Dittebrandt

aus Baden-Baden

Tag der mündlichen Prüfung: 23.07.2025
1. Referent: Prof. Dr.-Ing. Carsten Dachsbacher
2. Referent: Prof. Dr.-Ing. Hendrik P. A. Lensch

Acknowledgements

Here, I would like to express my gratitude and thanks to many people who accompanied me throughout my academic journey. They provided lots of advice and feedback, as well as emotional support and companionship. I would have never achieved this without you!

First, I would like to thank my supervisor, Carsten Dachsbacher. He gave me the opportunity for this thesis in the first place. For my research, he provided the necessary guidance to stay on track, but, crucially, also the freedom to explore different avenues and to become an independent researcher.

I would also like to thank Hendrik P. A. Lensch for serving as external reviewer.

I had the pleasure of co-authoring with fantastic people, namely Johannes, Sebastian, Carsten Benthin, Lorenzo, Lukas, Vincent, Michael, and Alexander. I want to express particular gratitude to Johannes, Sebastian, as well as Carsten Benthin, who have mentored me and taught me in many technical and methodical aspects. I really enjoyed working with you all.

Next, I would like to thank all my colleagues at the KIT computer graphics group. Diana was always there to help and chat with, and she kept the group together. I shared an office with Max for most of my time there, and it was a great time. His determination spilled over to me, and I learned a lot about visualization. We also had a lot of fun with sysadmin-related endeavors. I had a lot of insightful discussions with Vincent, and Reiner gave me many new perspectives on GPU programming. I also want to thank Bene for basically financing me, since he was the best customer of my beverage service. Also, thanks to my other colleagues, including the ones who left the group (long) before me, Alisa, Christoph Peters, Colin, Daniel, Dmitrii, Emanuel, Hisanari, Jasmin, Killian, Lucas, Mahmoud, Mikhail, Nathan, Moritz, Tobias Rapp, and Tobias Zirr. Together, we went on numerous excursions, to movie (and programming) nights, went bouldering, complained about mensa food, and solved jigsaw puzzles. I enjoyed the jam sessions with Killian and Vincent, although my piano playing skills are still quite subpar.

I also want to thank my colleagues during my internship at Intel, particularly Florian and Johannes Meng.

A special thanks goes to Christoph Schied, for sparking my interest in doing a PhD in computer graphics. I am really grateful for having done my Master's thesis under the guidance of you and Johannes.

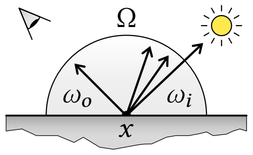
Finally, I want to thank my family and friends, Halida and Manfred, as well as Caroline, Christoph, Davide, Felix, Max, Sara, Timo Lorenz, and Timo Tränkel. Your constant encouragement made all the difference.

Contents

1. Zusammenfassung

Die fotorealistische Bildsynthese ermöglicht die Erzeugung physikalisch korrekter Bilder von virtuellen Szenen. Neben Produktvisualisierung und Virtual Reality wird Sie vor allem in der Unterhaltungsindustrie in animierten Filmen oder interaktiven Videospielen angewendet. Der Kern ist dabei der Lichttransport, wo die Ausbreitung des Lichts ausgehend von Lichtquellen über potenziell mehrere Indirektionen zur Kamera simuliert wird.

1.1. Grundlagen



Der fotorealistische Lichttransport wird durch die Rendergleichung [79] formalisiert:

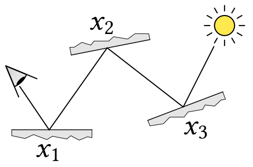
$$L(x, \omega_o) = L_e(x, \omega_o) + \int_\Omega f_r(\omega_o, x, \omega_i) L_i(x, \omega_i) \cos\theta_i \mathrm{d}\omega_i. \tag{1.1}$$

Diese beschreibt die an einem Oberflächenpunkt x in Richtung ω_o ausgestrahlte Strahldichte, welche sich aus der lokalen Emission (L_e) und der reflektierten Strahldichte zusammensetzt. Letztere ist ein Integralausdruck, welcher das eingehende Licht L_i einsammelt und abhängig vom lokalen Materialmodell f_r anteilig nach ω_o weitergibt. Die Auswertung von $L_i(x, \omega_i) = L(\mathtt{ray}(x, \omega_i), -\omega_i)$ kann dabei eine erneute Auswertung der Rendergleichung am nächsten Oberflächenpunkt von x in Richtung ω_i erfordern (ausgedrückt durch den Strahltest `ray`). Durch diese rekursive Definition ergibt sich eine im Allgemeinen unendliche Dimensionalität des Integrals. Somit ist eine analytische Lösung oder eine numerische Integration mittels Quadraturformeln unpraktikabel.

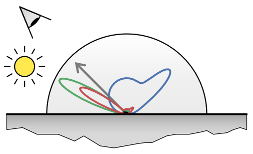
Stattdessen erlaubt Monte Carlo Integration eine effiziente Bestimmung des Integrals durch zufällige Abtastung des Integranden $f(x)$:

$$I \approx \frac{1}{N} \sum_{i=1}^{N} \frac{f(x_i)}{p(x_i)}. \tag{1.2}$$

Die Abtastpunkte sind dabei nach einer Wahrscheinlichkeitsdichte $p(x)$ verteilt. Aufgrund der zufälligen Abtastung besitzt das Ergebnis einen zufälligen Fehler, dessen Varianz mit Anzahl der Samples N zurückgeht. Zusätzlich wird die Varianz stark von der Wahl von $p(x)$ beeinflusst. Je ähnlicher $p(x)$ zu $f(x)$ ist, desto geringer ist die Varianz. Dementsprechend liegt ein großer Forschungsschwerpunkt im Finden guter Samplingdichten, die zielgerichtet beitragsstarke Teile des Integranden abtasten (Importance Sampling).

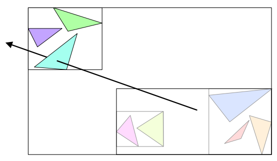


Ein klassischer Monte Carlo Algorithmus zur Lösung der Rendergleichung ist Path Tracing, bei der das Integral mit einem Sample ausgewertet wird. Intuitiv entsteht so ein Lichttransportpfad, der die Kamera über mehrere Oberflächeninteraktionen mit einer Lichtquelle verbindet. Bereits in einfachen Szenen kann der Integrand jedoch eine große Komplexität erreichen, etwa wenn nur wenige Lichttransportpfade einen nennenswerten Beitrag leisten. Entsprechend muss die zufällige Abtastung genau diese Pfade häufig treffen. Im Folgenden werden weitere Grundlagen zur verbesserten Abtastung, sowie zum Strahltest vorgestellt.

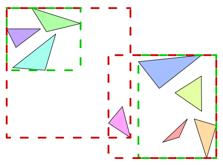


Path Guiding Samplingdichten werden meist nur auf Basis von lokal an einem Oberflächenpunkt verfügbarer Information konstruiert, z. B. das Materialmodell f_r (blaue Funktion). Die eingehende Radianz L_i (grüne Funktion) wird dabei vernachlässigt, obwohl diese einen ebenso großen Einfluss auf den Integranden (rote Funktion) hat, was zu starkem Rauschen führen kann. Path Guiding ist eine Methode, bei der bereits berechnete Samples verwendet werden, um optimierte Samplingdichten zu erzeugen. So kann insbesondere Information über L_i erlernt und einbezogen werden. Als Repräsentationen kommen z. B. Mixturmodelle infrage.

Radiance Caching ist eine Methode, bei der Teile der Beleuchtungsberechnung auf der Szene gespeichert werden: Ein von der Kamera ausgehender Lichttransportpfad kann, statt durch Verbindung mit einer Lichtquelle komplettiert zu werden, auch schon früher auf einer Oberfläche terminiert werden. Der verbleibende Beitrag kann dann durch Konsultation des Radiance Caches bestimmt werden. Eine frühere Terminierung in den Cache führt zu einer reduzierten Varianz im Bild. Gleichzeitig entsteht jedoch ein systematischer Fehler (Bias), da nicht die exakte Radianz an einem Oberflächenpunkt im Cache vorliegt, sondern nur ein approximativer Wert über eine Region. Dieser Fehler wird im Allgemeinen größer bei früherer Terminierung.



Ray Tracing Wie schon eingangs erwähnt, erfordert die Auswertung der Rendergleichung die Bestimmung des nächsten Punktes ausgehend von einem Startpunkt und einer Richtung (Strahl). Anstatt naiv alle geometrischen Primitive (meist Dreiecke) einer Szene zu testen, lässt sich diese Abfrage unter anderem mittels Hüllkörper-Hierarchien (engl. Bounding Volume Hierarchies, BVHs) beschleunigen. Hierbei werden die Primitive hierarchisch durch Hüllkörper zusammengefasst. So können zuerst die Hüllkörper getestet werden, bevor dann rekursiv weitere Hüllkörper und letztlich einzelne Primitive getestet werden.



Die Konstruktion einer solchen Hierarchie besitzt jedoch viele Freiheitsgrade und wirkt sich etwa auf die Geschwindigkeit der Abfragen aus. Die meisten Ansätze versuchen, die sogenannte Surface Area Heuristic (SAH) [53] zu minimieren, welche die Oberflächen der Hüllkörper bemisst. So werden bevorzugt dichtere Regionen isoliert. Ein klassischer Konstruktionsansatz ist die Top-Down Konstruktion [157]. Die Primitive werden rekursiv aufgeteilt, woraus sich dann implizit die Hierarchie ergibt. Die Aufteilung minimiert dabei die SAH, indem verschiedene Trennebenen als Kandidaten für die Aufteilung getestet werden.

1.2. Beiträge

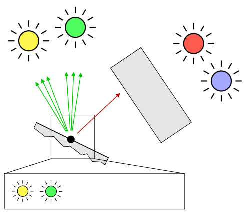
Die Beiträge dieser Arbeit liegen im Bereich von Bildsyntheseverfahren auf Basis von Monte Carlo Integration. Der Fokus liegt dabei auf interaktiven Verfahren, die auf parallelen Grafikprozessoren ausführen.

- Die Beiträge [35] und [36] stellen interaktive Guiding Verfahren für direkte & indirekte Beleuchtung vor. Für die direkte Beleuchtung werden Lichtquellen explizit in einer räumlichen Datenstruktur abgelegt [35] oder implizit durch ein Mixturmodell repräsentiert [36]. Für indirekte Beleuchtung ermöglicht ein komprimierter Quadtree effizientes Sampling [35].

- Der Beitrag [81] beschäftigt sich mit optimierter Pfadterminierung für Radiance Caching. Ermöglicht wird die explizite Kontrolle über die Varianz im finalen Bild und approximativ auch die optimale Abwägung zwischen Varianz und systematischem Fehler (Bias).
- Der Beitrag [146] stellt ein Verfahren zur Beschleunigung der Top-Down BVH Konstruktion vor. Die ersten Ebenen der Hierarchie werden hierbei durch eine zufällige und repräsentative Teilmenge der Primitive konstruiert.

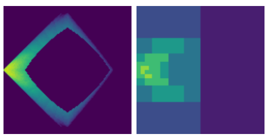
1.2.1. Interaktive Guiding Verfahren für direkte & indirekte Beleuchtung

Der Lichttransport in virtuellen Szenen kann oftmals eine große Komplexität erreichen. Dies Betrifft schon die direkte Beleuchtung, wenn eine große Anzahl von Lichtquellen unterschiedliche Teile der Szene beleuchten. Das macht eine effiziente Berechnung mittels Monte Carlo Methoden schwierig, da die wenigen relevanten Lichtquellen mit naiven Ansätzen (z. B. Abtastung des lokalen Materialmodells oder Abtastung der Lichtquellen proportional zu deren Helligkeit) oft verpasst werden, was sich in einer hohen Varianz der Schätzung ausdrückt.



Es wurden zwei Guiding Verfahren zum Importance Sampling direkter Beleuchtung entwickelt [35, 36]. In ersterem [35] werden sichtbare Lichtquellen explizit in einem räumlichen Cache abgespeichert (siehe Grafik). Für jede Region wird dabei eine Liste von sichtbaren Lichtquellen verwaltet. Durch ein initial uninformiertes Abtasten aller Lichtquellen, werden jene gesammelt, die Beiträge größer Null liefern. In späteren Iterationen werden dann verstärkt Lichtquellen im Cache abgetastet. Auf diese Weise wird die Sichtbarkeit explizit berücksichtigt. Im Vergleich zu existierenden Ansätzen zeigt sich eine deutliche Varianzreduktion, wenn die Anzahl der Lichtquellen, die eine Region beleuchtet, moderat ist. In Regionen mit vielen Lichtquellen ist die uninformierte Auswahl der Lichtquellen im Cache problematisch und führt zu erhöhtem Rauschen.

Im zweiten Verfahren [36] werden Lichtquellen implizit durch ein Mixturmodell repräsentiert. Dies ist insbesondere dann sinnvoll, wenn keine expliziten Informationen über Lichtquellen verfügbar sind. Dies ist z. B. bei texturierten Emittern der Fall. Das Mixturmodell wird implizit durch einen stochastischen Markovkettenprozess gesteuert, was eine schnelle Anpassung an dynamische Situationen erlaubt. Im Gegensatz zu Markov Chain Monte Carlo Verfahren wie Metropolis-Hastings [111], können Zustandsübergänge frei definiert werden, da weiterhin klassisches Importance Sampling des Mixturmodells betrieben wird.



Zusätzlich wurde ein Guiding Verfahren für indirekte Beleuchtung entwickelt [35]. Die entwickelte Repräsentation ist eine komprimierte Variante von direktionalen Quadtrees [116]. Die Samplingdichte (linke Grafik) wird hierbei durch einen Quadtree (rechte Grafik) repräsentiert, welcher auf einer 2D-Abbildung der Kugel definiert ist. Knoten und Blätter im Baum speichern hierbei die eingehende Radianz. Ein stochastischer Abstieg kann dann genutzt werden, um bevorzugt Blätter und damit Richtungen mit großem Beitrag zu erzeugen. Die entwickelte Variante speichert keine explizite Hierarchie, sondern repräsentiert den Baum implizit als Bitfeld [75]. Durch uniforme Abtastung der Blätter werden stärker unterteilte Bereiche bevorzugt abgetastet, ohne das explizite Gewichte benötigt werden. Die Bäume besitzen so Größen zwischen 32 und 128 Bits.

1.2.2. Optimierte Pfadterminierung für Radiance Caching

Radiance Caching ermöglicht eine Varianzreduktion durch Speicherung von Beleuchtungsberechnungen auf der Szene. Lichttransportpfade müssen auf diese Weise nicht erst bei Lichtquellen enden, sondern können schon vorher durch Zugriff auf den Cache terminiert werden. Ein früher Zugriff geht mit niedrigerer Varianz einher, da sich die im Cache gespeicherte Radianz aus vielen Samples zusammensetzt. Gleichzeitig entsteht aber ein größerer systematischer Fehler (Bias), da der Cache nur die durchschnittliche Radianz in räumlichen Regionen abbildet. Vorige Arbeiten steuern den Zugriff heuristisch, etwa durch Festlegung einer fixen Tiefe zur Terminierung [14], oder durch Approximation des Footprints eines Strahls [117].

Es wurde ein Verfahren [81] entwickelt, dass die Terminierung von Lichttransportpfaden so steuert, dass die Varianz im Bild begrenzt ist. Diese Zielsetzung ergibt sich in Kombination mit Rauschentfernung, welche üblicherweise als Nachbearbeitungsschritt angewandt wird. Diese erzeugt jedoch bei zu großer Varianz im Bild Artefakte. Durch Begrenzung der Varianz soll dem entgegengewirkt werden.

Die Pixelvarianz wurde hierzu in den lokalen Beitrag einzelner Pfade umformuliert, was die Auswertung während der inkrementellen Pfadkonstruktion erlaubt. Dadurch kann die Varianz für die Entscheidung der Terminierung verwendet werden. Konkret werden Pfade nur so lange verlängert, bis die lokale Varianz einen festgelegten Schwellwert überschreitet. Die Pixelvarianz bleibt dann auch unterhalb dieses Schwellwertes.

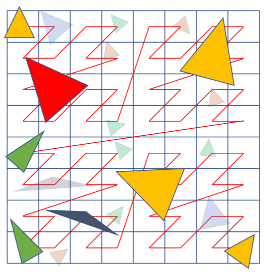
Die Pfadlänge wird unter Beschränkung des Schwellwertes jedoch maximiert: Eine frühere Terminierung könnte sinnvoller sein, um den Gesamtfehler zu reduzieren, wenn der systematische Fehler an vorigen Pfadvertices gering ist. Um dem entgegenzuwirken wurde zusätzlich eine Schätzung des systematischen Fehlers entwickelt. Im Gegensatz zur Pixelvarianz konnte jedoch nur eine Abschätzung des systematischen Fehlers gefunden werden. Dies führt dazu, dass Pfade weiterhin länger als notwendig sind.

1.2.3. Stochastische BVH Konstruktion

Die Top-Down BVH Konstruktion [157] erzeugt im Vergleich zu anderen Verfahren bessere Hierarchien (schnellere Abfragen) [1], jedoch ist die Konstruktion verhältnismäßig teuer: Die rekursive Unterteilung erfordert die mehrfache Betrachtung aller Primitive (proportional zur Rekursionstiefe). Daraus ergibt sich eine Gesamtlaufzeit von $O(n \log n)$. Besonders im interaktiven Kontext mit dynamischer Geometrie ist eine schnelle Konstruktionszeit relevant, da erzeugte BVHs meist nur für eine kurze Zeit verwendet werden.

Das Ziel der vorgestellten Arbeit [146] ist, die Konstruktion des Top-Down Verfahrens zu beschleunigen. Der zentrale Ansatz ist dabei, die wiederholte Betrachtung von Primitiven zu umgehen, welche den Hauptteil der Laufzeit ausmacht. Hierfür werden die ersten Ebenen der Hierarchie nur mit einer kleinen Teilmenge der Primitive konstruiert. Die Annahme ist dabei, dass aufgrund der großen Zahl an Primitiven in den ersten Ebenen die Auswahl der Trennebenen weniger von einzelnen Primitiven abhängig ist. Die restlichen Primitive werden anschließend in die unvollständige Hierarchie eingefügt und diese durch Fortführung der rekursiven Unterteilung fertiggestellt. Es ergeben sich somit zwei zusätzliche Bausteine im Vergleich zur Top-Down Konstruktion: die Bestimmung der Teilmenge und die Einfügung der restlichen Primitive.

Die Bestimmung der Teilmenge erfolgt durch eine räumlich stratifizierte Auswahl der Primitive (durch Anordnung entlang einer raumfüllenden Z-Kurve), gewichtet nach Größe der Primitive (siehe Grafik). Dies stellt eine repräsentative Auswahl der Primitive sicher, welche unter anderem auch für die Lastverteilung der Konstruktion relevant ist. Die Einfügung der restlichen Primitive erfolgt unter Minimierung der SAH. Dabei werden nicht alle Blattknoten der Hierarchie für jedes Primitiv in Betracht gezogen, sondern eine durch die raumfüllende Z-Kurve induzierte Nachbarschaft.



Der Ansatz zeigt eine Performancesteigerung gegenüber der klassischen Top-Down Konstruktion um einen Faktor von bis zu 1.8× (größere Szenen profitieren stärker). Allerdings schließt die Laufzeit weiterhin nicht zu schnelleren Konstruktionsalgorithmen auf Basis von Z-Kurven [92] oder PLOC [106] auf, die jedoch Einbußen in der Qualität aufweisen.

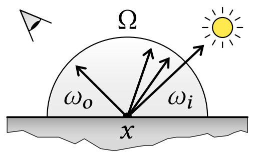
1.3. Zusammenfassung

Insgesamt ermöglichen die vorgestellten Arbeiten die effiziente Bildsynthese komplexerer Szenen unter der Laufzeiteinschränkung von interaktiven Anwendungen. Zunächst erlauben die vorgestellten Guiding-Verfahren den Umgang mit komplexerem Lichttransport, dessen Berechnung durch zuvor uninformierte Abtastung zu stark rauschen würde. Mittels der optimierten Pfadterminierung für Radiance Caching kann das Rauschen der synthetisierten Bilder nun explizit kontrolliert und begrenzt werden, was sich positiv auf die spätere Rauschentfernung auswirkt. Zuletzt ermöglicht die stochastische BVH Konstruktion den Umgang mit komplexerer animierter Geometrie, wo die Beschleunigungsstruktur durch Bewegung der Geometrie dauernd neu berechnet werden muss.

2. Summary

Photorealistic image synthesis enables the generation of physically correct images of virtual scenes. It is used in domains like product visualization, virtual reality and the entertainment industry in animated movies or video games. At its core lies the simulation of light transport, which models the propagation of light in virtual scenes starting from light sources to the camera through potentially many indirections.

2.1. Introduction



Light transport is modeled through the rendering equation [79]:

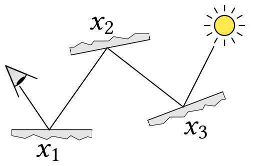
$$L(x, \omega_o) = L_e(x, \omega_o) + \int_\Omega f_r(\omega_o, x, \omega_i) L_i(x, \omega_i) \cos\theta_i \mathrm{d}\omega_i. \tag{2.1}$$

It describes the radiance leaving a surface point x in direction ω_o, which is composed of the local emission (L_e) and the reflected radiance. The latter is an integral expression which collects the incoming radiance L_i and forwards a portion of it towards ω_o depending on the material model f_r. The evaluation of $L_i(x, \omega_i) = L(\mathtt{ray}(x, \omega_i), -\omega_i)$ may necessitate another evaluation of the rendering equation at the closest surface point from x towards ω_i, expressed through the ray tracing query `ray`. Because of this recursive definition, the rendering equation possesses, in general, infinite dimensionality. This renders analytical solutions or numeric integration through quadrature impractical.

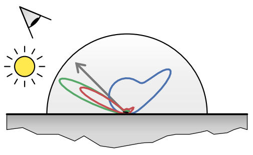
Instead, Monte Carlo integration allows efficient computation of the integral through random sampling of an integrand $f(x)$:

$$I \approx \frac{1}{N} \sum_{i=1}^{N} \frac{f(x_i)}{p(x_i)}. \tag{2.2}$$

The sample points are distributed according to a probability density $p(x)$. Due to the random sampling, the result possesses a random error (variance) that shrinks as the number of samples N increase. Variance is also influenced by the choice of $p(x)$. Variance generally decreases when $p(x)$ is more similar to $f(x)$. A large research focus therefore lies in finding suitable sampling densities that target parts of the integrand with high contribution.

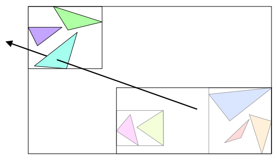


Path tracing is a classic Monte Carlo algorithm to integrate the rendering equation by recursively evaluating the equation with one sample. Intuitively, a light transport path is constructed that connects the camera with a light source through multiple surface interactions. Even in relatively simple scenes the integrand can reach a complexity such that only very few paths provide significant contribution. In the following we will introduce relevant approaches to improve sampling as well as perform the ray tracing query.

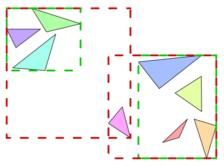


Path Guiding Probability densities are usually constructed using only locally available information, such as the material model f_r (blue function). The incoming radiance L_i (green function) is usually ignored, although it has just as much influence on the integrand (red function). This often results in larger variance. Path guiding is a method where already drawn samples are used to construct optimized probability densities. In particular, a representation of L_i, such as mixture models, can be learned and used for sampling.

Radiance Caching is a method where parts of the illumination are stored in the scene. Instead of completing a light transport path that originates from the camera through connection with a light source, it can also be terminated earlier on a surface point. The remaining contribution is then retrieved by consulting the radiance cache at the surface point. Earlier termination into the cache reduces variance, since the radiance cache itself is free of variance. However, the cache does not store the exact radiance at a surface point. Thus, a systematic error (bias) is introduced that generally increases with earlier termination.



Ray Tracing As already mentioned, evaluation of the rendering equation requires repeated computation of the closest surface point from a given starting point and direction (ray). Instead of naively testing all geometric primitives (usually triangles), bounding volume hierarchies (BVHs) can be used to speed up this query. Primitives are organized in a hierarchy of bounding volumes. Bounding volumes in the upper levels are tested for intersection first before recursively testing other bounding volumes and eventually individual primitives.



The construction of such hierarchies possesses many degrees of freedom with implications on query performance. Most approaches attempt to minimize the so-called Surface Area Heuristic (SAH) [53] that measures the surface area of bounding volumes. In particular, this leads to isolation of dense and sparse regions in the scene. A classic construction algorithm is top-down construction [157]. Primitives are recursively partitioned while minimizing SAH. The hierarchy emerges implicitly from the recursion.

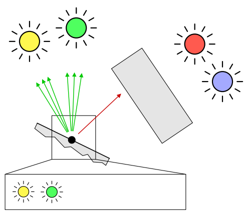
2.2. Contributions

My contributions lie in the field of photorealistic image synthesis using Monte Carlo integration. A particular focus is placed on interactive techniques that execute on parallel graphics processing units (GPUs).

- The contributions [35] and [36] present interactive guiding techniques for direct & indirect illumination. For direct illumination, visible light sources are explicitly maintained in a spatial data structure [35] or implicitly represented through a mixture model [36]. For indirect illumination, a compressed quadtree enables efficient sampling with low storage overhead [35].
- The contribution [81] optimizes path termination for radiance caching. It enables explicit control of the image variance and also an approximation for the optimal trade-off between variance and systematic error (bias).
- The contribution [146] presents a technique to accelerate top-down BVH construction. The first levels of the hierarchy are constructed with a representative subset of the primitives.

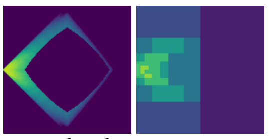
2.2.1. Interactive guiding for direct & indirect illumination

Light transport in virtual scenes can reach a great complexity. Even direct illumination can be challenging with scenes that contain large numbers of light sources that illuminate different parts of the scene. In such a case, Monte Carlo integration using naive methods (such as sampling the material model or light sources proportional to their brightness) can be insufficient, since relevant light sources are missed most of the time.



Two guiding techniques for direct illumination were developed [35, 36]. In the first contribution [35], visible light sources are explicitly stored in a spatial data structure (see figure). For each region in the spatial data structure, a list of visible light sources is maintained. Initially, an uninformed sampling of the light sources allows exploring which light sources provide any contribution. Later iterations then increasingly rely on the cache, resulting in more sampling budget being spent on visible light sources. We observe a significant variance reduction compared to existing techniques if the number of light sources that illuminate a region is moderate. Scalability issues emerge if the number of light sources increases, resulting in increased variance.

In the second contribution [36], Light sources are represented implicitly through a mixture model. This is reasonable in situations where no explicit information about light sources is available. This is often the case with textured emitters. The mixture model is controlled through a stochastic Markov chain process that is tuned to fast adaptation in dynamic situations. In contrast to Markov chain Monte Carlo approaches that rely on algorithms like Metropolis-Hasting [111] for correctness, we can freely define state transitions without introducing bias.



We additionally introduce a guiding technique for indirect illumination [35]. The representation is a compressed variant of directional quadtrees [116]. The probability density (left figure) is represented through a quadtree (right figure) that spans a 2D-projection of the sphere. We do not store an explicit hierarchy but a succinct representation [75]. Through uniform sampling of the leaves we can entirely encode importance in the topology (refined regions are thus sampled more often) without having to store explicit weights. The required storage space is between 32 and 128 bits.

2.2.2. Optimized path termination for radiance caching

Radiance caching enables variance reduction through caching of illumination on the scene. Light transport paths then do not need to be completed by ending on a light source, but can be terminated earlier on surface points. Earlier termination reduces variance, as the cache itself is an average of many samples. At the same time, a systematic error is introduced as the cache stores only the average radiance for a whole spatial region. Earlier works control termination heuristically, for example through a fixed termination depth [14] or by approximating the footprint (projection of the pixel size) of a ray [117].

We developed a technique that controls path termination such that image variance is bounded [81]. This goal stems from post-processing filters that are usually applied on the noisy output of path tracers to remove noise (denoiser). However, if variance is too large, artifacts emerge. Through explicit bounding of image variance, we circumvent such artifacts.

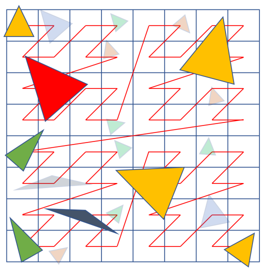
We reformulated pixel variance into a local contribution of individual paths, which allows evaluation during incremental path construction. This enables using variance as a criterion for the termination decision. Specifically, paths are extended as long as a user-defined variance threshold is reached. By construction, the resulting image variance will stay below this threshold.

However, path length is maximized with respect to the defined threshold. An earlier termination could be better to reduce the overall error if the introduced bias is negligible. We therefore introduce an estimation of the systematic error that a termination would introduce. In contrast to image variance, however, this estimation is only approximate, resulting in longer paths than necessary.

2.2.3. Stochastic BVH construction

Top-down BVH construction [157] produces hierarchies with higher quality (faster query performance) [1]. However, construction is more expensive compared to other methods, because the repeated partitioning of primitives requires repeated access, resulting in a total construction complexity of roughly $O(n \log n)$. This is particularly problematic for interactive applications with dynamic content, where constructed BVHs are used only for short periods of time and therefore require fast construction.

We present a method to accelerate top-down BVH construction [146]. We circumvent repeated access of primitives by constructing the first levels of the hierarchy with a small, representative subset of primitives. The assumption is that partitioning of the first levels of the hierarchy is less sensitive to missing primitives. After constructing an initial hierarchy based on the subset, the remaining primitives are inserted into the hierarchy and recursive construction is resumed with all primitives. As a result, two additional building blocks are introduced: Computing the primitive subset and inserting the remaining primitives.



The subset is computed based on a spatially stratified selection of primitives (by arranging the primitives along a space-filling Z-curve), weighted by size of primitives (see figure). This ensures representative selection of primitives and is also relevant for load balancing of the construction on the GPU. The remaining primitives are inserted by considering a neighborhood of leaf nodes induced through the space-filling Z-curve. The leaf node with the smallest impact on the SAH-metric is chosen.

Our approach shows a performance increase compared to top-down construction with a factor of up to 1.8× (larger scenes benefit more), while retaining the same level of quality. While execution time is still not competitive to faster construction algorithms like PLOC [109] or sorting-based methods [92], these algorithms tend to produce hierarchies with lower quality.

2.3. Summary

Our proposed contributions enable efficient image synthesis of more complex scenes under the constraint of interactive execution times. Our proposed guiding techniques can handle more complex light transport problems that would otherwise not be feasible with uninformed sampling techniques. Through optimized path termination for radiance caching, image variance can be explicitly controlled and bounded, which is advantageous for the usually following denoising step. Finally, stochastic BVH construction closes the gap towards faster construction algorithms, allowing the use in scenes with more complex geometry.

3. List of publications

The contributions of this thesis were already partially published. For some published works, I began research during my master's thesis, or I share first authorship with other authors. In such a case, the contributions relevant to this thesis are detailed here.

A. Dittebrandt, J. Hanika, and C. Dachsbacher. "Temporal Sample Reuse for Next Event Estimation and Path Guiding for Real-Time Path Tracing". In: *Eurographics Symposium on Rendering - DL-only Track* (2020). DOI: `10.2312/SR.20201135`

> I began research on this paper during my master's thesis "Light selection for real-time Rendering". In that thesis, I developed the visibility light cache algorithm. The paper refines the algorithm by (1) using a more accurate importance weight for lights [68], (2) improving scalability by subdividing large caches into buckets that are uniformly sampled and (3) maintaining light caches in an adaptive octree data structure to allow usage for surface points not directly visible from the camera. The compressed quadtree presented in the paper is fully novel.

L. Tessari, A. Dittebrandt, M. J. Doyle, and C. Benthin. "Stochastic Subsets for BVH Construction". In: *Computer Graphics Forum* 42.2 (2023). DOI: `10.1111/cgf.14759`

> I share first authorship with Lorenzo Tessari. In the following, our respective contributions are detailed. My contributions are:
> - subset sampling: histogram weight clamping and uniformity (mostly)
> - primitive insertion: window search and heuristics (equally)
> - implementation: CPU builder (mostly); GPU builder (equally with Carsten Benthin)
> - evaluation: build performance, build time breakdown and algorithmic evaluation
>
> Contributions of Mr. Tessari are:
> - subset sampling: spatial and importance stratification (mostly)
> - primitive insertion: cost model; window search and heuristics (equally)
> - analysis on theoretical complexity
> - implementation: test framework (mostly)
> - evaluation: variance analysis and comparison with deterministic clustering

A. Dittebrandt, V. Schüßler, J. Hanika, S. Herholz, and C. Dachsbacher. "Markov Chain Mixture Models for Real-Time Direct Illumination". In: *Computer Graphics Forum* 42.4 (2023). DOI: `10.1111/cgf.14881`

L. Kandlbinder, A. Dittebrandt, A. Schipek, and C. Dachsbacher. "Optimizing Path Termination for Radiance Caching Through Explicit Variance Trading". In: *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 7.3 (2024). DOI: `10.1145/3675381`

> I share first authorship with Lukas Kandlbinder. I contributed the theoretical derivation of variance-bounding and MSE-minimizing termination strategies. Mr. Kandlbinder contributed most of the implementation and evaluation. This paper is partially based on the master's thesis "Adaptive Path Space Filtering" from Alexander Schipek that I supervised. In that thesis, the variance-bounding termination strategy was developed.

4. Background

This section introduces the necessary background for the contributions in this thesis. The structure draws inspiration from Pharr [128] and Hanika [61]. We will begin with integration & measure theory in section 4.1, followed by probability theory in section 4.2. Then, we continue with Monte Carlo integration in section 4.3, a numeric integration method that is widely used in photorealistic image synthesis. We formalize light transport in section 4.4 and discuss basic algorithms to solve the light transport equations in section 4.5. More specific algorithms with greater relevance to the contributions are later introduced in section 4.6. In section 4.7 we introduce acceleration structures that are used to accelerate ray tracing queries, a fundamental building block to image synthesis. Finally, we give an overview on the architecture of graphics processing units and discuss the impact on implementing efficient algorithms in section 4.8.

4.1. Integrals on measure spaces

Integrals of 1D functions $f(x)$ over an interval $[a, b]$ can be defined in terms of Riemann integration:

$$\int_a^b f(x)\,\mathrm{d}x = \lim_{n\to\infty} \sum_{i=0}^{n-1} \frac{b-a}{n} f\left(a + i\frac{b-a}{n}\right). \tag{4.1}$$

The sum is a Riemann sum that approximates the area under the graph through the areas of a finite number n of rectangles. Each rectangle spans a subinterval of width $(b-a)/n$ and the height is set to a function value of f in the interval (here the leftmost). The area is thus the product of both lengths. The integral is then defined in the limit as n grows to infinity, i.e. the number of intervals becomes infinitesimally small.

This notion can be generalized using measure theory. Instead of defining subintervals to measure area, we construct patches that we assign area or volume in possibly higher dimensions. Measure spaces are defined as a triplet (X, A, μ). X is a set and A is a so-called σ-algebra on X. It is a subset of the powerset of X ($A \subseteq \mathcal{P}(X)$) with the following properties:

- $X \in A$
- $E \in A \Rightarrow X \backslash E \in A$ (closure under negation)
- $E_1, E_2, \dots \in A \Rightarrow E_1 \cup E_2 \cup \dots \in A$ (closure under unions)

This construction ensures that when applying operations such as unions or intersections on elements in A, the results will also be in A (A is closed under these operations). A special σ-algebra is the so-called borel algebra $\mathcal{B}(X)$. It is defined as the smallest σ-algebra that contains all open sets of X and is therefore unique to a given set X. Through negation and unions, closed sets are also contained in $\mathcal{B}(X)$. Finally, μ is the actual *measure* that assigns a value to elements in A. An integral on a measure space has the following form:

$$\int_\Omega f(x)\,\mathrm{d}\mu(x), \tag{4.2}$$

Angle (radian rad)
Solid angle (steradian sr)

Figure 4.1.: Solid angles are a natural extension to 2D angles. While angle is defined as the length of an arc segment on the unit circle with the unit radian rad, solid angle is defined as the area of a patch on the unit sphere with the unit steradian sr.

although $\mathrm{d}\mu(x)$ is often abbreviated with $\mathrm{d}x$ as with normal integrals.

In the following, we will detail how to convert integrals to different measures using change of variables and present important measure spaces that are relevant to this thesis.

4.1.1. Change of variables

For integration problems it is often useful to change the integral domain, which can lead to simpler integrands. This is referred to as change of variables. Given an initial integral of a function f over the domain X, we can reformulate the integral over another domain Y by defining a mapping $\phi : Y \longrightarrow X$:

$$\int_X f(x)\,\mathrm{d}x = \int_{\phi^{-1}(X)} f(\phi(y))\,|\det(D\phi(y))|\,\mathrm{d}y. \tag{4.3}$$

While the integral is now in the domain Y, f is still evaluated with values in X by applying the integration variable y to ϕ before invoking f. The change of variables introduces a distortion, since f is now evaluated with a different density than in the original domain. To compensate, the determinant expression, referred to as Jacobian determinant, is introduced. Assuming $X : \mathbb{R}^n, Y : \mathbb{R}^n$, $D\phi$ is the Jacobian matrix, which consists of partial derivates of ϕ for each output dimension along every input dimension, resulting in an $n \times n$-matrix. Intuitively, this expression measures the local rate of change of the output variables depending on the input variables, which is exactly the distortion that needs to be compensated.

4.1.2. Important measures

Solid angle measure The solid angle measure is most central to image synthesis, since the fundamental light transport equations are defined in it. Solid angles are a three-dimensional equivalent to two-dimensional angles (fig. 4.1). Angles are defined as the length of a given arc on the unit circle. The angle between two points, for example, is the length of the arc between them. Solid angles, on the other hand, are defined as the area of a given patch on the unit sphere. Solid angle integrals have the following form:

$$\int_\Omega f(\omega)\,\mathrm{d}\omega, \tag{4.4}$$

where Ω denotes the unit sphere as integration domain and ω are directions.

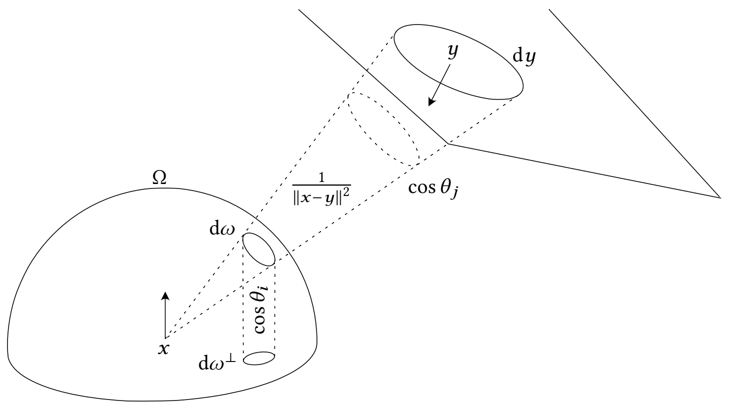


Figure 4.2.: Conversion between solid angle, projected solid angle and surface area measures.

Projected solid angle measure Light transport often considers transport between surface elements. In such cases it is oftentimes more convenient to consider the projection of the unit sphere onto the surface. Projected solid angle integrals and solid angle integrals can be converted through a change of variables with the Jacobian determinant $|\det(D\phi(y))| = \cos\theta$ (fig. 4.2), where θ is the angle between the given surface normal and direction ω:

$$\int_{\Omega^\perp} f\left(\omega^\perp\right) \mathrm{d}\omega^\perp = \int_{\Omega} f(\omega)\cos\theta \, \mathrm{d}\omega. \tag{4.5}$$

$\Omega^\perp$ and $\omega^\perp$ are the projected domain and projected directions, respectively.

Surface area measure In some instances it can be more useful to explicitly integrate over surface points instead of directions. For a given surface point x, the mapping $\phi(y) = (y-x)/\|y-x\|$ can be used to map surface points y back to directions ω. The resulting Jacobian determinant is given as (fig. 4.2):

$$|\det(D\phi(y))| = \frac{\cos\theta_j}{\|x-y\|^2}. \tag{4.6}$$

Intuitively, when moving over a surface, the resulting direction vector will move more slowly, when the surface is far away from x and when it is not directly facing x. Both effects would cause a distortion that is corrected with the inverse squared distance and cosine term, respectively. The integral then has the following form:

$$\int_{\Omega} f(\omega)\,\mathrm{d}\omega = \int_{A^V} f(x)\frac{\cos\theta_j}{\|x-y\|^2}\,\mathrm{d}y. \tag{4.7}$$

The integration domain needs to be constrained to the set of visible surfaces A^V from x. This domain can be fairly complex, therefore the domain is usually extended to be over all surfaces A by introducing a visibility term V:

$$\int_{A^V} f(x)\frac{\cos\theta_j}{\|x-y\|^2}\,\mathrm{d}y = \int_{A} f(x)V(x,y)\frac{\cos\theta_j}{\|x-y\|^2}\,\mathrm{d}y. \tag{4.8}$$

Finally, the conversion from projected solid angle to surface area integrals can be performed:

$$\int_{\Omega^\perp} f\left(\omega^\perp\right)\mathrm{d}\omega^\perp = \int_{\Omega} f(\omega)\cos\theta_i\,\mathrm{d}\omega = \int_{A} f(x)\,\underbrace{V(x,y)\frac{\cos\theta_i\cos\theta_j}{\|x-y\|^2}}_{\text{Geometry Term } G(x,y)}\,\mathrm{d}y. \tag{4.9}$$

The cosine, distance and visibility terms that form this Jacobian determinant are often referred to as the geometry term $G(x,y)$.

4.2. Probability theory

This section provides an introduction to concepts from probability theory as needed by the later sections. We will first introduce random variables as a fundamental stochastic quantity and from there construct concepts such as joint distributions, expectation & variance as well as simple distribution function and mixture models. In later section, we will cover fitting of distributions to data as well as methods to generate samples according to distributions.

4.2.1. Random variables

Randomness is formalized through *random variables.* Random variables are defined as functions $X : \Omega \longrightarrow E$ that map outcomes in a sample space Ω to values in another space E. We are usually interested in random variables that produce real numbers ($X : \Omega \longrightarrow \mathbb{R}$). Applying a function $f : E \longrightarrow F$ to a random variable yields another random variable $Y : \Omega \longrightarrow F = f(X)$.

In the following, we do not provide a rigorous derivation of random variables based on measure theory, and refer to existing works [52] for this purpose.

We typically differentiate between discrete and continuous random variables. For discrete random variables, E is countable finite or infinite. It can be fully described through a *probability mass function (PMF)* $p_X : E \longrightarrow [0, 1]$ that assigns occurrence probabilities to each value. A PMF is normalized:

$$\sum_{x \in E} p_X(x) = 1. \tag{4.10}$$

For $E \subset \mathbb{R}$, we can define a *cumulative distribution function (CDF)* $c_X(x) : \mathbb{R} \longrightarrow [0, 1]$:

$$c_X(x) = \sum_{x' \in E, x' < x} p_X(x'). \tag{4.11}$$

It describes for a given x the probability that produced values lie below x. c_X converges to one and zero as x approaches positive and negative infinity, respectively.

For continuous random variables, E is uncountable. Similar to discrete random variables, they can be fully described through a corresponding *probability density function (PDF)* $p_X : E \longrightarrow [0, \infty)$. PDFs do not describe direct occurrence probabilities of individual values. Instead, integrals of the PDF over given intervals give the probability that produced values fall within the interval. A PDF also needs to be normalized:

$$\int_E p_X(x) \, \mathrm{d}x = 1. \tag{4.12}$$

For $E = \mathbb{R}$, we can define a CDF with the same properties as in the discrete case:

$$c_X(x) = \int_{-\infty}^{x} p_X(x') \, \mathrm{d}x'. \tag{4.13}$$

4.2.2. Joint distributions

Random variables can also depend on each other. Dependence means that outcomes of a random variable have an effect on the outcomes of another random variable. The dependence can be expressed through a joint distribution. For two random variables X and Y, for example, we can define a joint PDF $p_{X,Y}(x,y)$. If the two random variables are actually independent, then we can define PDFs $p_X(x)$ and $p_Y(y)$ with $p_{X,Y}(x,y) = p_X(x)p_Y(y)$ for all x, y. If the random variables are indeed dependent, we are often still interested in analyzing the density for only one of the random variables. This necessitates the other random variable to be removed in some fashion.

One approach is to define a *marginal* density by integrating over one of the random variables. As an example, we can marginalize out Y in $p_{X,Y}$ with the following definition:

$$p_X(x) = \int_\Omega p_{X,Y}(x,y)\,\mathrm{d}y. \tag{4.14}$$

This density gives the overall distribution of X without regard for Y.

Another approach is to define a *conditional* density, where one of the random variables is fixed instead of being integrated over. We can, for example, conditionalize to $Y = y$ in $p_{X,Y}$, resulting in the following definition:

$$p_{X\,|\,Y}(x \mid y) = \frac{p_{X,Y}(x,y)}{\int_\Omega p_{X,Y}(x,y)\,\mathrm{d}x} = \frac{p_{X,Y}(x,y)}{p_Y(y)}, \tag{4.15}$$

where the denominator is used to ensure normalization of the conditional density (it is equal to the marginal density $p_Y(y)$ without X). This density gives the distribution of X under the assumption that the outcome of Y is already known to be y. Naturally, $p_Y(y)$ needs to be greater than zero for the conditional density to be well-defined.

Marginalization and conditionalization can also be applied to multivariate density functions (density functions with more than two random variables) by sequentially performing either operation on different random variables.

4.2.3. Expectation & variance

Oftentimes, we are not interested in specific outcomes of a random variable, but rather aggregate information about its distribution. The most basic quantities are the *expectation* (or expected value) and *variance*.

The expectation for a discrete random variable X with PMF p_X is defined as the sum of all outcomes, weighted by their respective probability mass:

$$E[X] = \sum_{x \in E} x p_X(x). \tag{4.16}$$

For a continuous random variable X with PDF p_X, the expectation is defined as an integral over the space of outcomes:

$$E[X] = \int_E x p_X(x)\,\mathrm{d}x. \tag{4.17}$$

The expectation is a linear operator:

$$E[aX + Y] = aE[X] + E[Y]. \tag{4.18}$$

Additionally, it is monotonous:

$$X \leq Y \Rightarrow E[X] \leq E[Y]. \tag{4.19}$$

Given two (potentially dependent) random variables X and Y, where realizations of X are always less than realizations of Y, then the expectation of X is less than the expectation of Y.

Variance is used to describe the spread of values produced by a random variable. It is by itself defined as an expectation of squared differences of a random variable to its mean:

$$V[X] = E\big[(X - E[X])^2\big]. \tag{4.20}$$

This term can also be decomposed further:

$$\begin{aligned} V[X] &= E\big[(X - E[X])^2\big] = E\big[X^2 - 2XE[X] + E[X]^2\big] && (4.21)\\ &= E\big[X^2\big] - E[2XE[X]] + E[X]^2 = E\big[X^2\big] - 2E[X]E[X] + E[X]^2 && (4.22)\\ &= E\big[X^2\big] - E[X]^2. && (4.23)\end{aligned}$$

The expectation of the random variable is referred to as the first moment, while the squared expectation is referred to as the second moment. These terms are usually collected numerically to estimate the variance of a distribution.

Variance is not a linear operator but can also be decomposed, here for independent random variables X and Y:

$$V[aX + Y] = a^2 V[X] + V[Y]. \tag{4.24}$$

Since variance is defined as squared differences to the mean of a random variable, it does not possess the same unit as values of the original random variable. In contrast, the standard deviation σ_X of a random variable X is defined as the square root of the variance of X:

$$\sigma_X = \sqrt{V[X]}. \tag{4.25}$$

Conditional expectation & variance Since expectation and variance are directly defined based on probability densities, conditional expectations & variances can be readily defined:

$$\begin{aligned} E[X \mid Y] &= \int_E x p_{X|Y}(x, y)\,\mathrm{d}x, && (4.26)\\ V[X \mid Y] &= E\big[(X - E[X \mid Y])^2 \;\big|\; Y\big]. && (4.27)\end{aligned}$$

Note that since Y is a random variable, the overall expression becomes a random variable as a function of Y.

Variance decomposition Given two dependent random variables X and Y, we can decompose variance of Y into variances that are or are not "explained" by X:

$$V[Y] = E[V[Y \mid X]] + V[E[Y \mid X]]. \tag{4.28}$$

The first term is the "unexplained" variance. It is an expectation over a conditional variance to X. The conditional variance is the variance that remains even if we have knowledge about X. The outer expectation thus denotes the mean variance over all realizations of X. The second term is the "explained" variance. It is a variance over a conditional expectation to X. The conditional expectation represents the distribution of Y if we have knowledge about X. The variance of this term is therefore attributed directly to X.

The decomposition can be proven by applying the moment definition for variance in eq. (4.23) and exploiting linearity of the expectation operator:

$$\begin{aligned}
V[Y] &= E[V[Y \mid X]] + V[E[Y \mid X]] && (4.29)\\
&= E\left[E\left[Y^2 \mid X\right] - E[Y \mid X]^2\right] + E\left[E[Y \mid X]^2\right] - E[E[Y \mid X]]^2 && (4.30)\\
&= E\left[E\left[Y^2 \mid X\right]\right] - E\left[E[Y \mid X]^2\right] + E\left[E[Y \mid X]^2\right] - E[E[Y \mid X]]^2 && (4.31)\\
&= E\left[Y^2\right] - E[Y]^2 && (4.32)\\
&= V[Y]. && (4.33)
\end{aligned}$$

4.2.4. Distribution functions

This section introduces important probability densities relevant to this thesis.

Normal distribution The normal distribution function is a very fundamental probability density defined on $\mathbb{R}$. It is characterized by a mean μ and standard deviation σ:

$$\mathcal{N}\left(x \mid \mu, \sigma^2\right) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x_i-\mu)^2}{2\sigma^2}}. \tag{4.34}$$

It possesses a characteristic bell shape that is centered around μ and with its width proportional to σ.

von Mises-Fisher distribution While the normal distribution is defined on $\mathbb{R}$, we also need distributions that are defined on the unit sphere S^2. The von Mises-Fisher distribution can be seen as en equivalent of the normal distribution on the sphere. It is characterized by a mean direction μ and a concentration parameter κ:

$$p_{\text{VMF}}(\omega \mid \mu, \kappa) = \frac{\kappa}{4\pi \sinh \kappa} e^{\kappa \mu \cdot \omega}. \tag{4.35}$$

4.2.5. Mixture models

Complex sample distributions can often not be faithfully represented through simple distribution functions. In such cases, a composition of simple distribution functions may be used instead. Given a distribution function $p(x \mid \theta)$, distribution parameters $\theta_1, \ldots, \theta_n$ and component weights $\pi_1, \ldots, \pi_n$, we can define a mixture as follows:

$$p(x) = \sum_{k=1}^{n} \pi_k p(x \mid \theta_k). \tag{4.36}$$

The mixture is thus a linear combination of the underlying distributions with weights π_k. To ensure that the mixture is a valid probability density, the component weights must sum to one $\left(\sum_{k=1}^{n} \pi_k = 1\right)$.

4.2.6. Maximum-likelihood estimation (MLE)

Given a set of independent and identically distributed samples $x_1, \ldots, x_n$, we want to find a suitable parameterization for a parametric model $p(x \mid \theta)$ with parameters $\theta \in \Theta$. We can define a likelihood function that quantifies for a given parameterization how well it explains the given samples:

$$\mathcal{L}(\theta) = \prod_{i=1}^{n} p(x_i \mid \theta). \tag{4.37}$$

Due to sample independence the overall probability of producing the given samples according to the model is the product of probabilities for each individual sample. The goal is then to find the parameterization which maximizes this likelihood (that best explains the given samples):

$$\hat{\theta} = \arg\max_{\theta \in \Theta} \mathcal{L}(\theta). \tag{4.38}$$

To simplify derivation, the goal is often reformulated to maximize the log-likelihood, since this does not change the maximum (the logarithm operator is monotonous):

$$\hat{\theta} = \arg\max_{\theta \in \Theta} \ln \mathcal{L}(\theta) = \arg\max_{\theta \in \Theta} \ln \left(\prod_{i=1}^{n} p(x_i \mid \theta) \right) \tag{4.39}$$

$$= \arg\max_{\theta \in \Theta} \sum_{i=1}^{n} \ln \left(p(x_i \mid \theta) \right). \tag{4.40}$$

The maximum-likelihood estimation can be extended to incorporate weights w_i to individual samples x_i. That is, the objective is biased to better explain samples with higher weight:

$$\hat{\theta} = \arg\max_{\theta \in \Theta} \mathcal{L}(\theta) = \prod_{i=1}^{n} p(x_i \mid \theta)^{w_i} \tag{4.41}$$

$$\Leftrightarrow \hat{\theta} = \arg\max_{\theta \in \Theta} \ln \mathcal{L}(\theta) = \sum_{i=1}^{n} w_i \ln \left(p(x_i \mid \theta) \right). \tag{4.42}$$

In the following, we will introduce (weighted) maximum-likelihood estimators for relevant distributions.

Normal distribution function The log-likelihood objective for the normal distribution is as follows:

$$\ln \mathcal{L}(\mu, \sigma^2) = \sum_{i=1}^{n} \ln \left(p\left(x \mid \mu, \sigma^2\right) \right) = \sum_{i=1}^{n} \ln \left(\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x_i-\mu)^2}{2\sigma^2}} \right) \tag{4.43}$$

$$= \sum_{i=1}^{n} \ln \left(\frac{1}{\sqrt{2\pi\sigma^2}} \right) + \ln \left(e^{-\frac{(x_i-\mu)^2}{2\sigma^2}} \right) = \sum_{i=1}^{n} -\frac{1}{2} \ln \left(2\pi\sigma^2 \right) - \frac{(x_i - \mu)^2}{2\sigma^2} \tag{4.44}$$

$$= -\frac{n}{2} \ln \left(2\pi\sigma^2 \right) - \frac{1}{2\sigma^2} \sum_{i=1}^{n} (x_i - \mu)^2 . \tag{4.45}$$

The maximum of the log-likelihood is found by searching its extreme point for each parameter. The partial derivative is zero at that location. We therefore solve each partial derivate for zero, beginning with the mean σ:

$$0 = \frac{\delta}{\delta\mu} \ln \mathcal{L}\left(\mu, \sigma^2\right) = \frac{\delta}{\delta\mu} - \frac{n}{2} \ln\left(2\pi\sigma^2\right) - \frac{1}{2\sigma^2} \sum_{i=1}^{n} (x_i - \mu)^2 \tag{4.46}$$

$$= -\frac{1}{2\sigma^2} \sum_{i=1}^{n} 2\left(x_i - \mu\right)(-1) = \frac{1}{2\sigma^2} \sum_{i=1}^{n} 2\left(x_i - \mu\right) \tag{4.47}$$

$$\Leftrightarrow 0 = \sum_{i=1}^{n} x_i - \mu = \sum_{i=1}^{n} x_i - n\mu \Leftrightarrow n\mu = \sum_{i=1}^{n} x_i \Leftrightarrow \mu = \sum_{i=1}^{n} \frac{x_i}{n}. \tag{4.48}$$

We can do the same for the variance σ^2:

$$0 = \frac{\delta}{\delta\sigma^2} \ln \mathcal{L}\left(\mu, \sigma^2\right) = \frac{\delta}{\delta\sigma^2} - \frac{n}{2} \ln\left(2\pi\sigma^2\right) - \frac{1}{2\sigma^2} \sum_{i=1}^{n} (x_i - \mu)^2 \tag{4.49}$$

$$= -\frac{n}{2} 2\pi \frac{1}{2\pi\sigma^2} - \frac{-1}{\sigma^4} \sum_{i=1}^{n} (x_i - \mu)^2 = -\frac{n}{2\sigma^2} + \frac{1}{\sigma^4} \sum_{i=1}^{n} (x_i - \mu)^2 \tag{4.50}$$

$$\Leftrightarrow \frac{n}{2\sigma^2} = \frac{1}{\sigma^4} \sum_{i=1}^{n} (x_i - \mu)^2 \Leftrightarrow \sigma^2 = \sum_{i=1}^{n} \frac{(x_i - \mu)^2}{n}. \tag{4.51}$$

von Mises-Fisher distribution A von Mises-Fisher distribution is parametrized by a mean direction $\mu \in S^3$ and a concentration parameter $\kappa \in [0, \infty)$. The weighted maximum likelihood estimate of samples with weight and direction w_i, ω_i is defined as: [7]

$$\hat{\mu} = \frac{r}{\|r\|} \qquad \hat{\kappa} \approx \frac{3\bar{r} - \bar{r}^3}{1 - \bar{r}^2}, \tag{4.52}$$

with r and $\bar{r}$ defined as:

$$r = \sum_{i=1}^{n} w_i \omega_i \qquad \bar{r} = \frac{\|r\|}{\sum_{i=1}^{n} w_i}. \tag{4.53}$$

4.2.7. Markov chains

Markov chains belong to the family of stochastic processes. A stochastic process models a form of time dependence, where future realizations depend on past states. Markov chains are a special case where the next state depends directly only on the current state. A Markov chain is defined in a state space Ω with a transition density $p(y \mid x)$ that is a conditional density on the current state x.

Certain Markov chains will exhibit the property that when performing many transitions from a given starting state, the distribution of possible states will always be the same, irrespective of the starting state. This distribution is referred to as stationary distribution $pi(x)$ and is reached in the limit as the number of transitions grows to infinity. The stationary distribution is invariant under state transitions:

$$\pi(y) = \int_{\Omega} \pi(x) p(y \mid x)\, \mathrm{d}x. \tag{4.54}$$

Another property is reversibility. That is, when the stationary distribution is reached, the probability of being in state x and transitioning to state y is the same as being in state y and transitioning to state x. This is formalized through the detailed balance condition:

$$\pi(x)p(y \mid x) = \pi(y)p(x \mid y). \tag{4.55}$$

4.2.8. Inverse CDF sampling

We often want to draw samples according to a given probability density $p_X(x)$. However, random number generators often only produce canonical random numbers, that is, random numbers uniformly distributed in the range $[0, 1)$. Therefore, a mapping between canonical random numbers and a desired probability density needs to be defined. One approach to achieve this is the inverse CDF sampling method. The central observation is that the CDF P_X of p_X defines for each possible outcome the probability that produced outcomes are smaller than it. Inverting this CDF will produce for a given probability the value that sits at the quantile.

An example is to produce samples according to the density:

$$p(x) = 2x. \tag{4.56}$$

We derive its CDF $P(x)$ and invert it:

$$P(x) = \int_0^x p(x')\,\mathrm{d}x' = \int_0^x 2x'\,\mathrm{d}x' = \left[x'^2\right]_0^x = x^2 \tag{4.57}$$

$$\Leftrightarrow x = \sqrt{\underbrace{P(x)}_{\xi}}. \tag{4.58}$$

Canonical random numbers ξ can then be inserted into this expression to produce samples according to $p(x)$.

Sampling joint densities For higher dimensions, the sampling must be performed in multiple stages by relying on marginal and conditional densities. Given a random variable $p_{X,Y}(x, y)$, the approach is to first derive the marginal density of one of the random variables. In this instance we choose X with the density $p_X(x)$, from which the inverted CDF can be derived. After that, the conditional density of Y given X is considered.

A practical example is to generate samples on the hemisphere according to a cosine distribution:

$$p(\omega) = \frac{1}{\pi}\cos\theta. \tag{4.59}$$

To allow parameterizing directions, we convert to spherical coordinates:

$$p(\phi, \theta) = \frac{1}{\pi}\sin\theta\cos\theta. \tag{4.60}$$

Next, we marginalize θ to retrieve a distribution $p(\phi)$ depending only on ϕ:

$$p(\phi) = \int_0^{\theta/2} p(\phi, \theta')\,\mathrm{d}\theta' = \int_0^{\theta/2} \frac{1}{\pi}\sin\theta'\cos\theta'\,\mathrm{d}\theta' \tag{4.61}$$

$$= \left[\frac{1}{2\pi}\sin^2\theta'\right]_0^{\theta/2} = \frac{1}{2\pi}sin^2\frac{\theta}{2} - \frac{1}{2\pi}\sin^2 0 = \frac{1}{2\pi}. \tag{4.62}$$

Finally, we derive the CDF $P(\phi)$ and invert it:

$$P(\phi) = \int_0^{\phi} p(\phi')\,\mathrm{d}\phi' = \int_0^{\phi} \frac{1}{2\pi}\,\mathrm{d}\phi' = \left[\frac{\phi'}{2\pi}\right]_0^{\phi} = \frac{\phi}{2\pi} \tag{4.63}$$

$$\Leftrightarrow \phi = 2\pi \underbrace{P(\phi)}_{\xi_0}\,. \tag{4.64}$$

Into $P(\phi)$ we can now insert canonical random numbers ξ_0 to angles ϕ.

We then continue with generating samples for θ by considering the conditional distribution $p(\theta \mid \phi)$:

$$p(\theta \mid \phi) = \frac{p(\phi, \theta)}{p(\phi)} = \frac{2\pi \sin\theta \cos\theta}{\pi} = 2 \sin\theta \cos\theta. \tag{4.65}$$

In the same fashion as with $p(\phi)$, we compute its CDF and invert it:

$$P(\theta \mid \phi) = \int_0^{\theta} p(\theta' \mid \phi)\,\mathrm{d}\theta' = \int_0^{\theta} 2 \sin\theta' \cos\theta'\,\mathrm{d}\theta' \tag{4.66}$$

$$= \left[\sin^2\theta'\right]_0^{\theta} = \sin^2\theta - \sin^2 0 = \sin^2\theta \tag{4.67}$$

$$\Leftrightarrow \theta = \arcsin\left(\sqrt{\underbrace{P(\theta \mid \phi)}_{\xi_1}}\right). \tag{4.68}$$

This mapping is more interesting due to the distortions introduced by the density and the spherical coordinates. The final direction is computed by inverting the spherical coordinate mapping:

$$\omega = \begin{bmatrix} \sin\theta\cos\phi \\ \sin\theta\sin\phi \\ \cos\theta \end{bmatrix}. \tag{4.69}$$

4.3. Monte Carlo integration

This section presents Monte Carlo integration, a numerical integration method for solving general integration problems, where we integrate a function f over a domain D:

$$I = \int_D f(x)\,\mathrm{d}x. \tag{4.70}$$

We will first introduce the Monte Carlo estimator. We will then show that convergence behavior of this estimator is independent of the dimensionality of the integral and then present extensions that improve convergence behavior and robustness.

4.3.1. Monte Carlo estimator

The basic idea of Monte Carlo integration is to define a random variable that has the sought integral value as expectation. This means that a plain sample mean of this estimator will approximate the integral value.

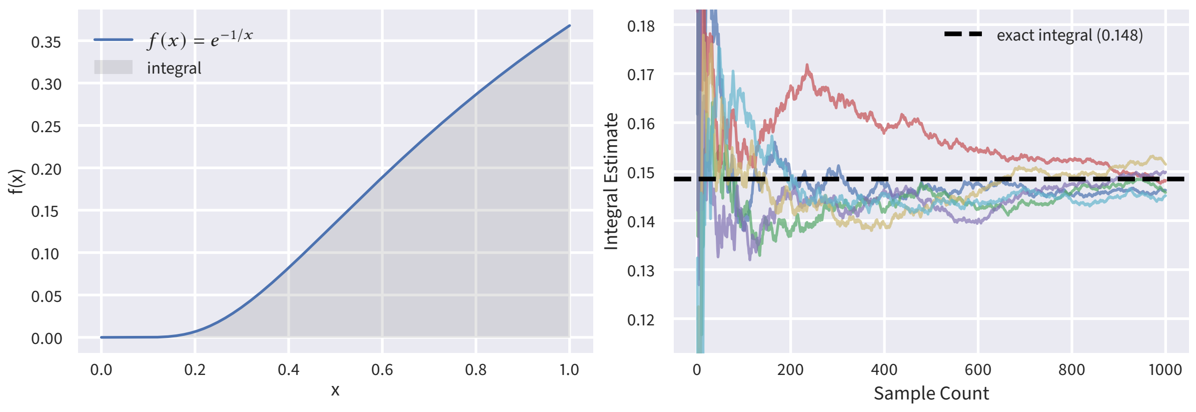


Figure 4.3.: Monte Carlo integration of function $f(x) = e^{-1/x}$ (left plot) over the range $[0, 1]$. Right plot shows convergence behavior of the integration with uniform density depending on sample count over multiple independent runs.

We initially define a random variable $X : \Omega \to D$ that maps from the sample space Ω to the integral domain D with density $p_X(x)$. We now introduce another random variable $g(X)$ based on a composition of a function $g : D \to \mathbb{R}$ with X. The expectation of this random variable is:

$$E[g(X)] = \int_D g(x) p_X(x)\, \mathrm{d}x. \tag{4.71}$$

Since the expectation is by itself defined as an integral in eq. (4.17), we only need to define $g(x)$ such that this integral corresponds to the sought integral. We can achieve this with $g(x) = f(x)/p_X(x)$ by canceling the density $p_X(x)$. Note that this derivation assumes that $p_X(x) > 0$ whenever $f(x) > 0$.

The final estimator is given as the sample average of N independent runs:

$$I \approx \frac{1}{N} \sum_{i=1}^{N} \frac{f(x_i)}{p_X(x_i)}. \tag{4.72}$$

In fig. 4.3, we show convergence behavior when using Monte Carlo integration to integrate the function $f(x) = e^{-1/x}$ with a uniform density over multiple independent runs. The variation of produced values is initially large, but shrinks as the sample count increases and converges to the integral value.

4.3.2. Convergence & importance sampling

In this section we will quantify the error of the Monte Carlo estimator through its variance and show how the individual components affect the error. First, we can relate the variance of the N-sample estimator to the variance of the one-sample estimator:

$$V\left[\frac{1}{N} \sum_{i=1}^{N} \frac{f(x_i)}{p_X(x_i)}\right] = \frac{1}{N^2} \sum_{i=1}^{N} V\left[\frac{f(x_i)}{p_X(x_i)}\right] = \frac{N}{N^2} V\left[\frac{f(x_i)}{p_X(x_i)}\right] = \frac{1}{N} V\left[\frac{f(x_i)}{p_X(x_i)}\right]. \tag{4.73}$$

The variance is therefore proportional to the inverse of the sample count and the variance of the one-sample estimator. We can further analyze the variance of the one-sample estimator:

$$V\left[\frac{f(x)}{p_X(x)}\right] = E\left[\left(\frac{f(x)}{p_X(x)} - E\left[\frac{f(x)}{p_X(x)}\right]\right)^2\right] = E\left[\left(\frac{f(x)}{p_X(x)} - I\right)^2\right]. \tag{4.74}$$

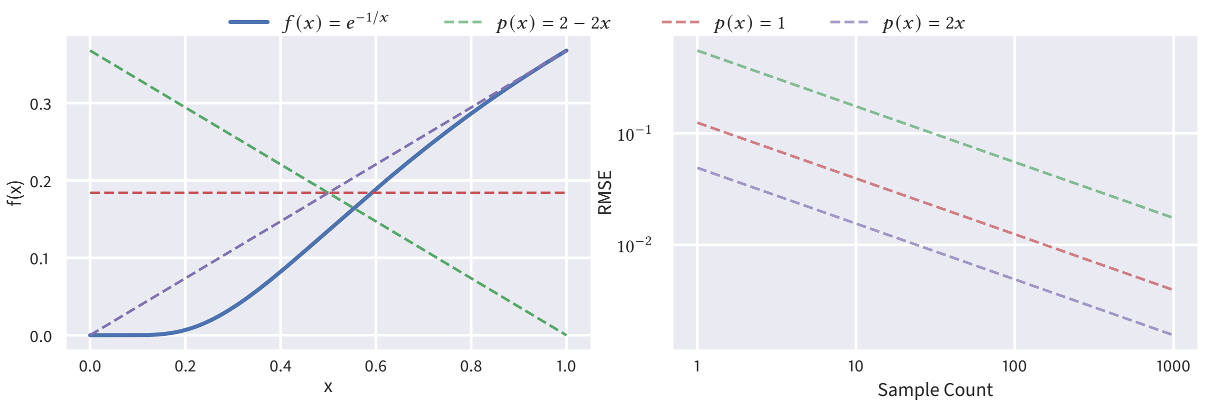


Figure 4.4.: Variance of Monte Carlo estimator depending on sample count and probability density for integrating function $f(x) = e^{-1/x}$. Left plot shows f as well as the used probability densities. Right plot shows the resulting variance when using each probability density depending on sample count. Probability densities that better match the integrand result in estimators with lower variance at the same sample count.

Here, variance shows a dependency on both the integrand and the probability density p_X, although the exact dependency is not as clear as with the sample count for the N-sample estimator. Intuitively, when the probability density is more similar to the integrand, the resulting ratio will be closer to the integral value, thus resulting in a smaller squared difference and therefore lower variance.

Due to squaring of the errors, variance has a different unit than the random variable. We therefore usually consider the standard deviation σ, which is defined as the square root of the variance:

$$\sigma_X = \sqrt{V[X]}. \tag{4.75}$$

For the one- and N-sample estimators, the following standard deviations emerge:

$$\sigma_1 = \sqrt{V\left[\frac{f(x)}{p_X(x)}\right]}, \sigma_N = \frac{\sigma_1}{\sqrt{N}}. \tag{4.76}$$

The standard deviation therefore decreases inversely proportional to the square-root of the sample count, thus requiring four times as many samples to halve the error.

This has the consequence that accumulating more and more samples will have diminishing returns on error reduction. Thus, more effort is invested in using better probability densities that resemble the integrand. This approach is referred to as importance sampling.

An example is shown in fig. 4.4 where we use different probability densities to integrate the function $f(x) = e^{-1/x}$. The densities are shown on the left plot and convergence, measured through root mean squared error (RMSE), is shown on the right plot with log-log scaling. The inverse square root dependency of the error on sample count manifests itself in the plot as a straight line with descent rate of $-1/2$. Different probability densities only affect the vertical offset of the line. But even in this simple example it can be observed that better probability densities can be much more effective than increasing the sample count, given that the best density (purple line) achieves with one sample the same error that the worst density (green line) achieves with around one hundred samples.

4.3.3. Multiple importance sampling

As we have observed in the previous section, choosing suitable probability densities can have a large impact on convergence. In many cases, however, no single known probability density provides the

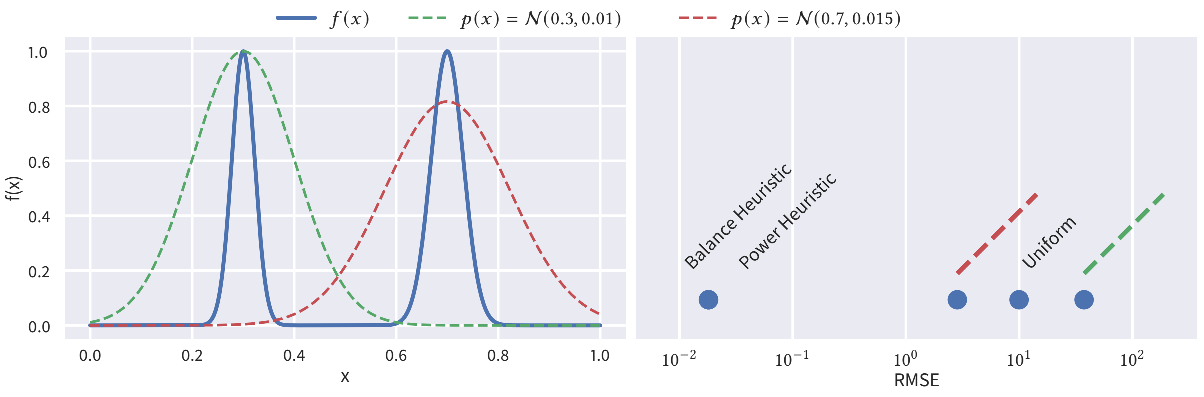


Figure 4.5.: Monte Carlo integration of function with multiple peaks over range $[0, 1]$ using multiple importance sampling (MIS). Left plot shows function and two probability densities. Right plot shows RMSE of using either probability density or MIS combinations of both using different MIS weight functions. While the uniform weight function produces error similar to the original probability densities, balance and power heuristic provide a significant error reduction.

required error reduction. An example is shown in fig. 4.5, where a function with multiple peaks cannot be adequately importance sampled using a single normal distribution that is centered around any of the peaks. Multiple Importance Sampling (MIS) [150, chapter 9] allows drawing samples from multiple probability densities $p_{X_1}, \ldots, p_{X_K}$ and combine them in a meaningful way. The general estimator is defined as:

$$I = \sum_{j=1}^{K} \frac{1}{N_j} \sum_{i=1}^{N_j} \frac{f(x_{i,j})}{p_{X_j}(x_{i,j})} w_k(x_{i,j}). \tag{4.77}$$

In this formulation, we draw an individual sample count N_j from each of the K probability densities. Overall, it is very similar to the original definition of the Monte Carlo estimator, except that a weighting function w_k is introduced. Defining a suitable weighting function is key to achieving further error reduction. Crucially, the weight distribution among the densities can be defined individually for each sample, since the weight function depends on the sample itself. For the estimator to be unbiased, the weighting function must meet the following conditions for all $x \in \Omega$:

- Normalization: $f(x) \neq 0 \Rightarrow \sum_{j=1}^{K} w_j(x) = 1$
- Support: $p_{X_j}(x) = 0 \Rightarrow w_j(x) = 0$

The first condition ensures normalization of the overall estimator, since each technique is only individually normalized by its local sample count N_j. The second condition ensures that only densities that can produce a sample have a non-negative weight. From both conditions, it also follows that for each x with $f(x) \neq 0$, there must exist at least one probability density that can produce the given sample. Otherwise, either of the conditions would need to be violated. In the following, we will introduce well-known weighting functions and discuss their tradeoffs.

Constant Weights A simple weighting scheme would be to set uniform weights $w_j^{\text{Uniform}}(x) = 1/K$. As we can see in fig. 4.5, it does not provide a noticeable improvement over exclusively using either probability density. The problem is that the individual estimators are effectively blended linearly between each other, thus largely inheriting variance of the original estimators.

Balance Heuristic The balance heuristic is based on the intuition that a good density for a given sample (which should therefore be given higher weight) produces the sample with high probability. The density itself quantifies this, and we can weight them proportionally to their density:

$$w_j^{\text{Balance}}(x) = \frac{p_{X_j}(x)}{\sum_{k=1}^{K} p_{X_k}(x)}. \tag{4.78}$$

One major consequence of this weighting scheme is that the evaluation of a sample requires evaluating all other densities with it. For one, this requires explicit routines to compute densities (oftentimes sampling routines directly cancel the density from the integrand, so no explicit density is computed) and also increases computational overhead, as K^2 density evaluations are required. However, as can be seen in fig. 4.5, error reduction can be dramatic, making the computational overhead worthwhile.

Power Heuristic The power heuristic is an extension to the balance heuristic, in that it puts additional weight to samples with higher density by raising the density to a power of P:

$$w_j^{\text{Power}}(x) = \frac{p_{X_j}(x)^P}{\sum_{k=1}^{K} p_{X_k}(x)^P}. \tag{4.79}$$

It can give a subtle improvement, although in fig. 4.5 we can see that it does not provide any substantial improvement in that specific case. In general, no substantial improvement should be expected from any weighting scheme compared to the balance heuristic [150].

4.3.4. Continuous multiple importance sampling

A generalization of multiple importance sampling is continuous multiple importance sampling (CMIS) [164], where the set of sampling techniques is extended to be uncountable. Sampling techniques are defined in a technique space $\mathcal{T}$. A one-sample estimator using CMIS then has the following form:

$$\langle I \rangle = \frac{w(t,x)f(x)}{p(t)p(x \mid t)}, \tag{4.80}$$

where $p(t)$ is the probability of drawing a sampling technique with parameters t and $p(x \mid t)$ is the probability of drawing x given t. $w(t,x)$ is the weight density. It must uphold similar conditions as the weight function in MIS for all $x \in \Omega$:

- Normalization: $f(x) \neq 0 \Rightarrow \int_{\mathcal{T}} w(t,x)\,\mathrm{d}t = 1$
- Support: $\forall t \in \mathcal{T} : p(x \mid t) = 0 \Rightarrow w(t,x) = 0$

Stochastic multiple importance sampling While an estimator can be directly defined based on CMIS, the technique density $p(t)$ is often intractable. Thus, CMIS cannot be evaluated in such contexts. To counteract this, stochastic multiple importance sampling can be employed, which does not consider the entire technique space, but rather a representative selection based on $p(t)$:

$$\langle I \rangle = \sum_{i=1}^{N} \frac{f(x_i)}{\sum_{j=1}^{n} p(x_i \mid t_j)}. \tag{4.81}$$

The techniques t_j are now produced according to $p(t)$. But since $p(t)$ is no longer present in the estimator, it does not need to evaluated explicitly, but can also be implicitly defined through a stochastic process.

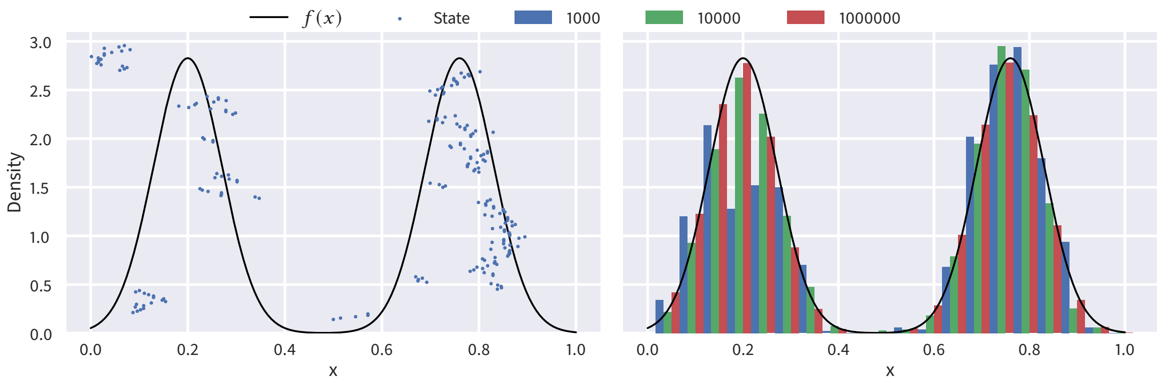


Figure 4.6.: Visualization of Metropolis-Hastings algorithm producing samples according to target density $f(x)$. Left plots shows the function and evolution of the Markov chain process over time (points are individual states and vertical axis is time). Right plot shows the resulting histogram of Markov chain states after an increasing number of iterations.

4.3.5. Markov chain Monte Carlo

While the previously introduced sampling schemes generate probability densities that only approximate the integrand, Markov chain Monte Carlo algorithms aim to produce samples directly proportional to the integrand f. Of particular importance is the *Metropolis Hastings* algorithm, that defines state transitions to steer the stationary distribution π of a Markov chain to be proportional to f.

The transition density of the Markov chain is composed of a proposal density $g(y \mid x)$ and an acceptance probability $A(y,x)$: $p(y \mid x) = A(y,x)p(y \mid x)$. The proposal density is a parameter of the algorithm and allows supplying expert knowledge to explore the domain of f. The acceptance probability is used to stochastically reject the proposal state and keep the current state. This probability is defined through Metropolis-Hastings and is necessary to ensure that the Markov chain converges to the sought stationary distribution. Recall the detailed balance condition, into which we can insert our specific definition for the transition density:

$$\pi(x)p(y \mid x) = \pi(y)p(x \mid y) \tag{4.82}$$

$$\Leftrightarrow \pi(x)A(y,x)p(y \mid x) = \pi(y)A(x,y)p(x \mid y). \tag{4.83}$$

The stationary and proposal distribution are fixed. To uphold detailed balance, only the acceptance probability can be defined accordingly. One possible solution is the metropolis choice:

$$A(y,x) = \min\left(1, \frac{\pi(y)g(x \mid y)}{\pi(x)g(y \mid x)}\right). \tag{4.84}$$

A side effect of this acceptance probability is that π does not need to be normalized. It only considers ratios of π evaluated at different points, which are scale invariant. Therefore, in the context of Monte Carlo integration, the integrand itself can be used here.

Figure 4.6 shows an example of how Metropolis-Hastings generates a density according f. The left plot shows f and the resulting samples over time (y-axis) to show the correlated nature of Markov chain Monte Carlo. The right plot shows the resulting histogram of samples after 1000, 10000 and 1000000 samples, respectively. While the distribution is initially not very close to the integral, it later follows the integrand more accurately.

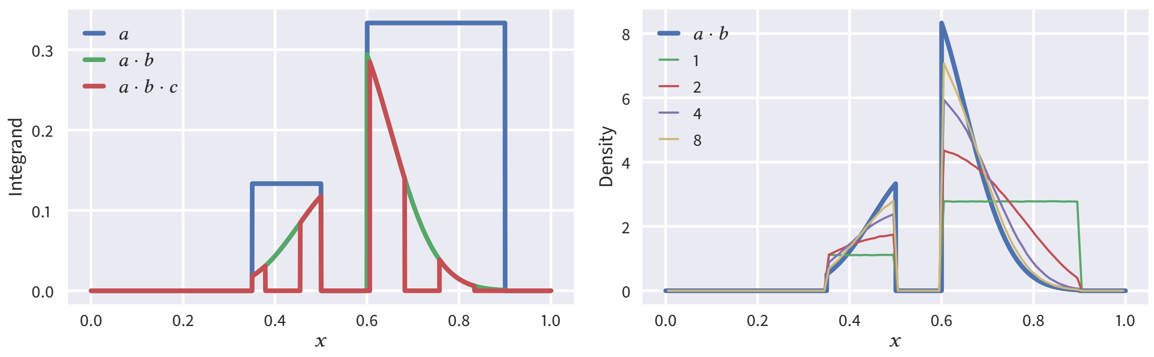


Figure 4.7.: Visualization of resampling to integrate function that is product of three functions a, b and c. Samples are drawn from a, while $a \cdot b$ is used as the target function for resampling. Left plot shows the integrand by successively adding more terms to the product. Right plot shows the density of produced samples depending on the number of samples for resampling. With more samples, the density converges to the target function.

4.3.6. Importance resampling

In some instances, we want to directly draw samples proportional to a target function $g(x)$, but cannot do so, because we either do not know a sampling procedure and/or do not even know the corresponding probability density (requiring us to compute the integral of g over the domain). In such cases, importance resampling [144] can be used to approximate g by drawing multiple (M) samples from a different probability density p that can be sampled from and selecting one of the samples proportional to a weight $w(x) = g(x)/p(x)$. The probability density is then defined as follows:

$$\hat{p}(x) = \frac{g(x)}{\frac{1}{M}\sum_{j=1}^{M} w(x_j)} = \frac{g(x)}{\frac{1}{M}\sum_{j=1}^{M} \frac{g(x_j)}{p(x_j)}}. \tag{4.85}$$

The nominator is the unnormalized target function, while the denominator normalizes it by computing the average target density over all the considered samples. Since M is finite, this is only an approximation with stochastic error. Therefore, $\hat{p}$ itself is only an estimate of the true density. The overall Monte Carlo estimator then has the following form:

$$I = \sum_{i=0}^{N} \left(\frac{f(x_i)}{g(x_i)} \frac{1}{M} \sum_{j=1}^{M} \frac{g(x_{ij})}{p(x_{ij})} \right). \tag{4.86}$$

Figure 4.7 shows an example for an integrand that is the product of three functions $f = a \cdot b \cdot c$. We only know a sampling procedure for $p = a$ but want to generate samples from $g = a \cdot b$. Through resampling, we can steer the probability density towards g by increasing M.

Importance resampling is most useful for integration problems where evaluation of parts of the integrand is computationally expensive. In the example case, if c is expensive, this approach allows to effectively evaluate the other parts ($a \cdot b$) at a higher rate.

Another aspect is that since the probability density $\hat{p}$ is only approximate, combining this approach with other sampling techniques through multiple importance sampling (section 4.3.3) will not yield optimal results.

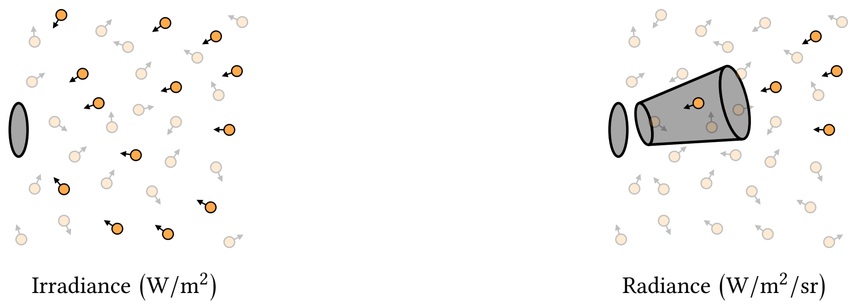

Irradiance $(\mathrm{W}/\mathrm{m}^2)$
Radiance $(\mathrm{W}/\mathrm{m}^2/\mathrm{sr})$

Figure 4.8.: While irradiance (left) measures radiant flux per surface element, radiance (right) further distinguishes by direction. This can intuitively be interpreted as an oriented funnel that only accepts light from specific directions.

4.4. Light transport

Before describing actual algorithms to compute light transport in the next section, we will use this section to present its formal framework. We will first introduce radiometry as the basic physical units that we use. Then, we continue with the rendering equation and its extension to path space. We also introduce some basic material models that are typically used in the context of rendering.

4.4.1. Radiometry

In this section we will introduce the basic radiometric quantities that are of interest to light transport algorithms.

Radiant energy Energy is a fundamental physical quantity that physical bodies posses in various forms. Examples include kinetic, potential or thermal energy. Radiant energy is the energy stored within the electromagnetic waves that constitute light and other forms of electromagnetic radiation. All forms of energy are convertible between each other. For example, kinetic energy can be converted to thermal energy through friction and radiant energy can be converted to electrical energy through the photovoltaic effect. This has the consequence that all forms of energy can be expressed in one common unit, namely Joule J. One Joule is equivalent to applying a force of one Newton over a distance of one meter ($\mathrm{J} = \mathrm{N} \cdot \mathrm{m} = \mathrm{kg} \cdot \mathrm{m}^2/\mathrm{s}^2$).

Radiant flux Light is in constant movement and is continuously emitted by light sources and absorbed by surfaces. This results in an equilibrium where the radiant energy entering a system is the same as the energy leaving a system. This continuous introduction of radiant energy can be quantified as a time rate (J/s) and is referred to as radiant flux with the unit Watt W. Sensors like the human eye do not measure radiant energy but radiant flux.

Irradiance Radiant flux received or emitted by a surface inherently depends on the area of the surface itself as more photons can be collected with a larger surface area. The radiant flux received by an individual surface point on the other hand is measured with irradiance, which is radiant flux per square meter (W/m^2) (fig. 4.8 left). It is a differential quantity and is therefore used in integration problems over surfaces.

Radiance Irradiance measures light received or emitted by surface points from all directions. We can further distinguish by direction, by accounting for the spread of directions where light is coming from. We can measure this spread using solid angle in steradian (fig. 4.1). Radiance is therefore radiant flux per square meter *per steradian* ($\mathrm{W/m^2/sr}$). It is a differential quantity both in area and solid angle. Intuitively, it measures incident or exitant flux at a small surface patch through a narrow funnel oriented towards a specific direction (fig. 4.8 right). This unit is very central to computer graphics as it allows incorporating view-dependence. Additionally, all the previous units can be derived from radiance through integrals over directions and surfaces.

Spectral units All the aforementioned units forego dependence on wavelengths. Depending on the context, the relevant spectrum can be implied, e.g. when computing radiance individually for each RGB color channel. All units can also be defined *per-wavelength*. For example, Spectral radiance is radiant flux per square meter per steradian *per nanometer* $\left(\mathrm{W/m^2/sr/nm}\right)$.

Photometric units All the aforementioned units are physical units that measure (the flow of) electromagnetic energy. Photometric units, on the other hand, try to account for human perception, where brightness is perceived differently depending on the wavelength. There exist equivalent photometric units for the presented radiometric units. For example, the equivalent of radiant flux is luminous flux and the equivalent for radiance is luminance.

4.4.2. Rendering equation

The rendering equation [79] formalizes the light transport problem and is defined as follows:

$$\underbrace{L_o(x,\omega_o)}_{\text{Outgoing Radiance}} = \underbrace{L_e(x,\omega_o)}_{\text{Emission}} + \int_\Omega \underbrace{f_r(\omega_o,x,\omega_i)}_{\text{BRDF}}\, \underbrace{L_i(x,\omega_i)}_{\text{Incoming Radiance}} \cos\theta_i \,\mathrm{d}\omega_i. \tag{4.87}$$

It is composed of two parts: The local emission and the reflected light.

The reflected light is denoted as an integral over the local hemisphere at x, and is composed of the actual incoming radiance (L_i), the foreshortening term $\cos\theta_i$ and the local material model (f_r). The material model, also referred to as bidirectional reflectance distribution function (BRDF), defines the local scattering properties of the surface. Through the material model, different surface appearances like mirrors, glossy or diffuse surfaces can be defined. It defines the portion of radiance received from ω_i that is reflected towards ω_o. The foreshortening term accounts for the projection of the incoming radiance onto the surface. At grazing angles, the projection becomes increasingly larger, thus less radiance is received at individual surface points. When looking from x in direction ω_i, oftentimes there lies another surface point y. In such cases, the incoming radiance at x in direction ω_i is the outgoing radiance at another surface point y in direction $-\omega_i$. Formally, we can introduce a function $\mathrm{ray}(x,\omega)$ that computes the closest point on the scene from x in direction ω. Therefore, $L_i(x,\omega_i) = L_o(\mathrm{ray}(x,\omega_i), -\omega_i)$. The main consequence is that the integral is inherently recursive, where integrals of reflected light are nested within each other.

4.4.3. Material models

Material models describe the local surface reflectance. For a given outgoing and incident direction, they describe the fraction of light that is transported between those directions.

While reflectance models can be defined quite freely, some restrictions are often imposed based on real-world observations:

- *Non-negativity* $\forall \omega_o, \omega_i : f_r(\omega_o, \omega_i) > 0$
- *Energy conservation* $\forall \omega_o : \int_\Omega f_r(\omega_o, \omega_i) \cos\theta_i \, \mathrm{d}\omega_i \leq 1$
- *Helmholtz Reciprocity* $\forall \omega_o, \omega_i : f_r(\omega_o, \omega_i) = f_r(\omega_i, \omega_o)$

These conditions are not only necessary to ensure synthesis of physically plausible images, but they are often also needed for consistency in rendering algorithms. Helmholtz Reciprocity in particular is necessary for light transport method that generate light paths from both the camera or light sources. In such cases, arguments to the BRDF are swapped, but still need to resolve to the same quantity. Energy conservation is necessary for rendering algorithms to actually converge, since otherwise more and more energy would be added through each indirection.

In the following, specific reflectance models are introduced that are often used.

Lambert model The most basic model is the Lambert reflection model, it scatters all incoming light evenly into all outgoing directions, resulting in the following definition:

$$f_r(\omega_o, \omega_i) = \frac{\rho}{\pi}, \tag{4.88}$$

where ρ is the surface albedo that gives color to the surface and π is the normalization constant (f_r needs to conserve energy together with the cosine term).

Mirror Perfect mirror reflections are modeled through Dirac impulses, that have their impulse at the reflected direction:

$$f_r(\omega_o, \omega_i) = \frac{\delta(\omega_i - \omega_r)}{\cos\theta_i}, \tag{4.89}$$

where ω_r is the reflected direction:

$$\omega_r = \omega_o - 2\langle \omega_o, n \rangle n. \tag{4.90}$$

Note that the integral in the rendering equation (eq. (4.87)) vanishes when using a mirror reflectance model due to the Dirac impulse function:

$$\int_\Omega f_r(\omega_o, x, \omega_i) L_i(x, \omega_i) \cos\theta_i \, \mathrm{d}\omega_i = \int_\Omega \frac{\delta(\omega_i - \omega_r)}{\cos\theta_i} L_i(x, \omega_i) \cos\theta_i \, \mathrm{d}\omega_i \tag{4.91}$$

$$= \int_\Omega \delta(\omega_i - \omega_r) L_i(x, \omega_i) \, \mathrm{d}\omega_i = L_i(x, \omega_r). \tag{4.92}$$

Microfacet models A physically-based reflectance model is introduced with microfacet models [29]. A surface is modeled as a collection of specular facets. The general formulation has the following form:

$$f_r(\omega_o, \omega_i) = \frac{F(\omega_o, \omega_h) D_\alpha(\omega_h) G(\omega_i, \omega_o)}{4 \cos\theta_o \cos\theta_i}, \tag{4.93}$$

with $\omega_h = (\omega_i + \omega_o) / \|\omega_i + \omega_o\|$ being the half vector between ω_i and ω_o. The distribution of microfacets is defined through the normal distribution function D_α. Given incident and exitant directions ω_i and ω_o, only the microfacets with normal ω_h will transport light between those directions. As such, the normal distribution is evaluated with ω_h. F is the Fresnel term and defines the fraction of light that is reflected or transmitted when interacting with a microfacet. Finally, the distribution of microfacets causes them to obstruct each other, resulting in less light being transported. This is captured through the geometry term G.

The Fresnel term depends on the complex-valued indices of refraction η_t, η_i of the media that lie at the interface of the surface interaction. Therefore, the Fresnel term inherently depends on wavelength. It is composed of a perpendicular and parallel reflectance [128]:

$$R = \frac{1}{2}(R_s + R_p) \qquad R_s = \left|\frac{\eta_t \cos\theta_i - \eta_i \cos\theta_t}{\eta_t \cos\theta_i + \eta_i \cos\theta_t}\right|^2 \qquad R_p = \left|\frac{\eta_i \cos\theta_i - \eta_t \cos\theta_t}{\eta_i \cos\theta_i + \eta_t \cos\theta_t}\right|^2, \tag{4.94}$$

where the cosine of the transmitted direction is given as follows:

$$\cos\theta_t = \sqrt{1 - \frac{\sin^2\theta_i}{\eta_t/\eta_i}}. \tag{4.95}$$

Note that this term can be complex-valued. The wavelength-dependence is what gives metallic objects (e.g. gold, copper) their colored appearance, since it decides what fraction of light is transmitted into the material and readily absorbed.

An example of a commonly used normal distribution function is the GGX distribution [149, 161], which models the normal distribution of an ellipsoid:

$$D_{\alpha_x,\alpha_Y}(\omega_h) = \frac{1}{\pi \alpha_x \alpha_y \left(\frac{x_h^2}{\alpha_x^2} + \frac{y_h^2}{\alpha_y^2} + z_h^2\right)^2}, \tag{4.96}$$

with $\omega_h = [x_h, y_h, z_h]$ and $\alpha_x, \alpha_y \in [0, 1]$ modelling anisotropic roughness of the surface. The surface becomes perfectly specular or diffuse for $\alpha = 0$ or 1, respectively.

For GGX, various geometry terms exist that have differing assumptions about the underlying geometry of the surface or approximations. For example, the height-correlated geometry term [67] is given as:

$$G(\omega_o, \omega_h, \omega_i) = \frac{1}{1 + \Lambda(\omega_o) + \Lambda(\omega_i)}. \tag{4.97}$$

The auxiliary function $\Lambda(\omega)$ is given as:

$$\Lambda(\omega) = \frac{-1 + \sqrt{1 + \frac{1}{a^2}}}{2}, \tag{4.98}$$

with $a = 1/(\alpha \tan\theta_o)$.

4.4.4. Measurement equation

Before defining light transport, we need to formalize the camera model that performs measurements in the scene. For each pixel, we want to compute the radiant flux it receives. The pixel sensor can be thought of as a planar surface which collects light from all directions. Formally, this means that we need to jointly integrate over the surface of the sensor and all incoming directions:

$$I_p = \int_A \int_\Omega W_p(x,\omega) L_o(x,\omega) \cos\theta \,\mathrm{d}\omega \,\mathrm{d}x. \tag{4.99}$$

W_p is the sensor responsivity function, which defines for a given pixel how much it responds to the given surface point and direction. This allows the integral to be defined in terms of all scene surfaces A (including the sensor), while W_p picks surface points that actually lie on the sensor surface that is relevant to the given pixel. W_p can also include an explicit model of the projection, thus picking specific directions ω. It is also possible to explicitly model the camera lenses as scene geometry. L_o is the incoming radiance ($\mathrm{W/m^2/sr}$) incident to x from ω. Through integration, only radiant flux (W) remains.

4.4.5. Path integral

The recursive formulation of the rendering equation only permits a local view on the light transport problem, as each recursion step is only concerned with one surface point. This section will introduce a flat version of the rendering equation that integrates over whole light transport paths connecting the camera to light sources [150, Chapter 8].

To express an integral over paths, it is easier to directly operate on path vertices in surface area measure (eq. (4.9)). In solid angle measure, a path would be given implicitly as an array of directions, requiring ray tracing to reconstruct the actual path vertices. Furthermore, a change of one direction can possibly affect many path vertices, depending on the reconstruction. We begin with the measurement equation (eq. (4.99)) and also transform it to surface area measure:

$$I_p = \int_A \int_A W_p(x_1,x_2) L_o(x_1,x_2) G(x_1,x_2) \,\mathrm{d}x_1 \,\mathrm{d}x_2. \tag{4.100}$$

Note that all functions are adapted to receive path vertices directly instead of directions for brevity. Due to the mapping, directions can always be reconstructed from surface positions. L_o is the recursively defined rendering equation. To flatten it, we leverage that the integral in L_o is a linear operator, thus the rendering equation can be expressed as $L_o = L_e + TL_o$, where T is the integral operator. Solving for L_o:

$$L_o = L_e + TL_o \Leftrightarrow (1-T)L_o = L_e \Leftrightarrow L_o = (I-T)^{-1}L_e. \tag{4.101}$$

The last term is equivalent to a Neumann series, which is an infinite sum of an increasing number of repeated applications of the integral operator:

$$L_o = (1-T)^{-1}L_e \underset{\substack{\uparrow \\ \text{Neumann series expansion}}}{=} \sum_{i=0}^{\infty} T^i L_e. \tag{4.102}$$

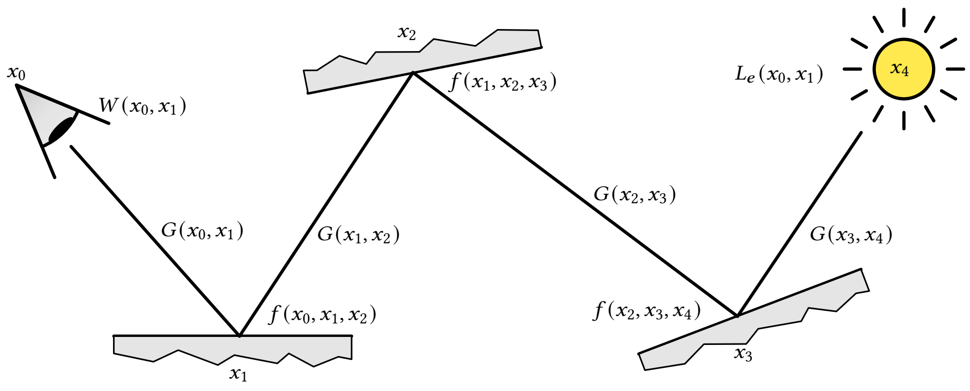


Figure 4.9.: The measurement contribution function is defined in product vertex area measure and consists of a series of BRDF- (f_r) and geometry-terms (G), terminated on one side by sensor responsivity W and by light emission L_e on the other side.

Inserting the Neumann series expansion of L_o into the measurement equation results in the following expression:

$$I_p = \sum_{i=2}^{\infty} \underbrace{\int_A \cdots \int_A}_{i \text{ times}} W_p(x_1, x_2)G(x_1, x_2) \tag{4.103}$$

$$f_r(x_1, x_2, x_3)G(x_2, x_3) \ldots f_r(x_{i-2}, x_{i-1}, x_i)G(x_{i-1}, x_i) \tag{4.104}$$

$$L_e(x_{i-1}, x_i)\, \mathrm{d}x_1 \ldots \mathrm{d}x_i. \tag{4.105}$$

Each summand represents a specific path length and is composed of a corresponding number of surface integrals and a chain of geometry terms and BRDF evaluations, terminated on either side by sensor responsivity and emission, respectively. The integrand terms can be extracted into the so-called *measurement contribution function* (fig. 4.9):

$$\underbrace{f_p(\boldsymbol{X})}_{(x_1, \ldots, x_n)} = W_p(x_1, x_2)G(x_1, x_2) \left(\prod_{i=2}^{n-1} f_r(x_{i-1}, x_i, x_{i+1})G(x_i, x_{i+1}) \right) L_e(x_{n-1}, x_n). \tag{4.106}$$

The infinite sum can further be eliminated by defining the integration domain as the union of all path lengths $P = A \times A \cup A \times A \times A \cup \ldots$, resulting in the path space integral:

$$I_p = \int_P f_p(\boldsymbol{X})\, \mathrm{d}\boldsymbol{X}. \tag{4.107}$$

4.4.6. Path classification

The light transport algorithms we will present in the following sections have different strengths and weaknesses with regard to reliably sampling certain light transport paths. The sampling behavior mostly depends on the material models present at each path vertex.

A coarse classification that is sufficient for this purpose has been introduced by Heckbert [66]. If differentiates between four vertex types: Eye E, light L, diffuse D and specular S. D-vertices broadly scatter incident light, where the extreme case is a perfectly lambertian surface. For S-vertices, the scattering is much narrower, where the extreme case is a perfect mirror surface that amends only a single discrete exitant direction for a given incident direction. Most surface interactions lie in a

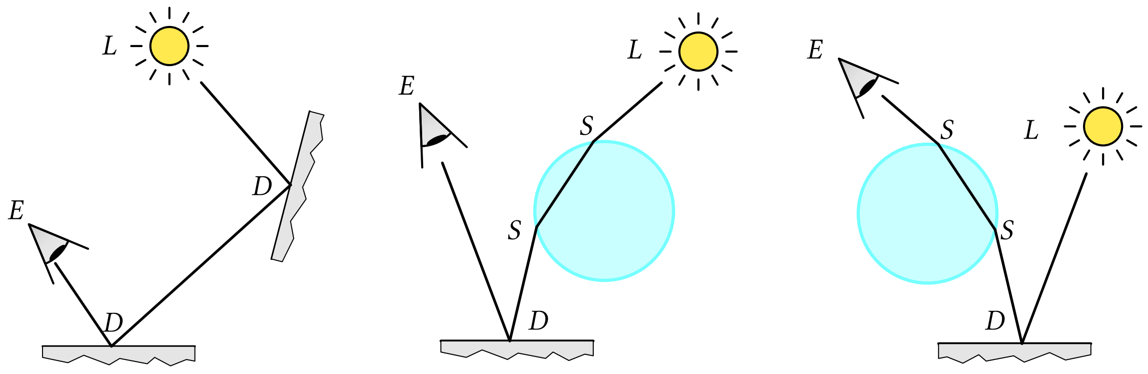


Figure 4.10.: Classifying light transport paths with Heckbert notation [66] using regular expressions on path vertex types (Eye E, light L, diffuse D and specular S).

$ED^{+}L$	Chain of diffuse surface interactions
$ES^{+}DL$	Directly illuminated surface visible through lenses
$EDS^{+}L$	Surface illuminated by light through lenses (caustic)

Figure 4.11.: Classifying light transport paths with Heckbert notation [66] using regular expressions on path vertex types (Eye E, light L, diffuse D and specular S).

continuum between being fully diffuse and fully specular. While performance of rendering algorithms is most pronounced at the extremes, it is still applicable to almost diffuse and almost specular surfaces.

A useful light transport path must have an E-vertex at one end, otherwise it would not contribute to the image. The other end must be an L-vertex, otherwise there would be no light to transport. In between there can be a variable number of D- and S-vertices that represent the surface interactions. This prose description can be formalized as a regular expression of the form $E\,(S|D)^{*}\,L$. More specific expressions can be constructed to define classes of light transport paths. Figure 4.10 visualizes example paths with annotated vertex types, while fig. 4.11 shows more specific examples of regular expressions to classify light transport paths.

4.5. Basic methods for solving light transport

In the following section, we will introduce basic methods for solving light transport. Basic in this context means that the techniques do not rely on auxiliary data to be computed before or during rendering. The following section the explicitly considers methods that make use of caching.

4.5.1. Forward path tracing

A straight-forward scheme to generate paths is to perform an incremental construction, starting out from the camera. This scheme is referred to as forward path tracing [79]. The probability density for generating the next path vertex can be conditional to all the previously generated path vertices. Most probability densities, however, only depend on the previous two path vertices. Usually, sampling is performed according to the local material model (fig. 4.12). The total probability density would have the following form:

$$p(\mathbf{X}) = p(x_1 \mid x_0)p(x_2 \mid x_1, x_0)\dots p(x_n \mid x_{n-1}, x_{n-2}). \tag{4.108}$$

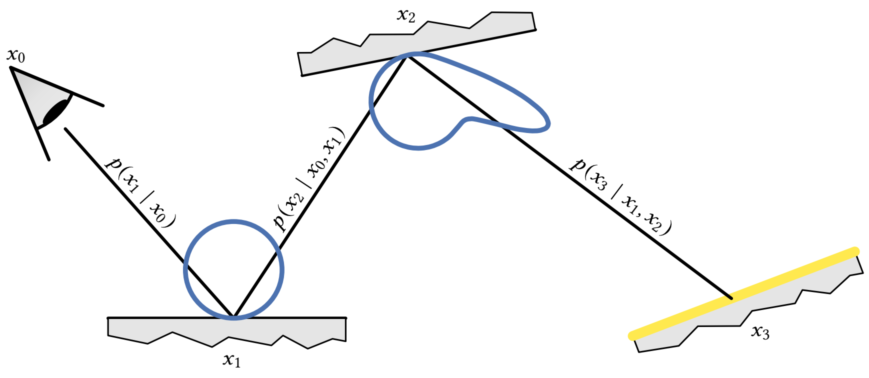


Figure 4.12.: Forward path tracing generates paths in an incremental fashion, starting from the camera. The probability density of the next path vertex is conditional on the previous path vertices (usually the last two) and is usually based on the material model. The total probability density of the resulting path is the product of all conditional probabilities.

One important property of path tracing is that it generates with each invocation multiple light transport paths, one for each indirection (fig. 4.13). To understand this behavior, we can split the path space integral back into individual path lengths:

$$I_p = \int_P f_p(X)\,\mathrm{d}\boldsymbol{X} = \int_{A\times A} f_p(X)\,\mathrm{d}\boldsymbol{X} + \int_{A\times A\times A} f_p(X)\,\mathrm{d}\boldsymbol{X} + \dots \quad . \tag{4.109}$$

Accordingly, we can define separate estimators for each indirection. Crucially, these estimators can be highly correlated without introducing any systematic error. This allows to re-use the same path prefix for different path lengths as is the case in path tracing.

A practical implementation of path tracing only requires constant space independent of path length, when summarizing most quantities into two accumulators: A path throughput variable that contains the Monte Carlo estimate of the path prefix so far (prefix of the measurement contribution function divided by the probability density), as well as a contribution variable that accumulates the pixel estimate for all indirections. The path vertices can be directly discarded after updating the accumulators and computing the next path vertex.

Russian roulette Path tracing can be naturally terminated when either no valid path vertex has been sampled (e.g. when sampling a direction which does not intersect any surface) or when the path throughput reaches zero (e.g. when sampling an occluded vertex). However, it is trivial to construct cases where either case does never occur (e.g. rendering the inside of a box) and path tracing would therefore never terminate. Furthermore, even in scenes where this extreme case does not arise, very long paths can have a negligible contribution, thus wasting a lot of computational time. Path lengths can be artificially limited in an implementation, but this will lead to a systematic error, since contributions from longer paths are not considered at all.

To remedy this, “Russian roulette” can be employed that randomly terminates paths while still being unbiased. Given a random variable X, we can define another random variable X_R in the following way:

$$X_R = \begin{cases} \frac{X}{p} & \text{if } \xi < p \\ 0 & \text{else} \end{cases} . \tag{4.110}$$

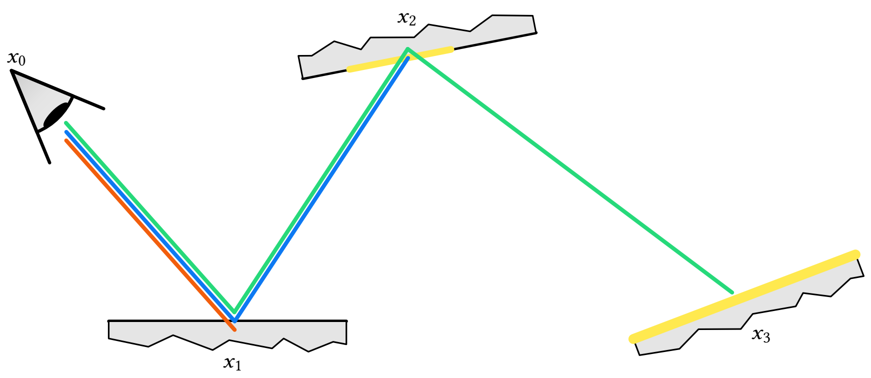


Figure 4.13.: While forward path tracing generates one path, each prefix is used to separately estimate illumination for all path lengths.

X will only be realized with a probability of p. To compensate, values of X are divided by p. As a result, the expectation of X_R is the same as X:

$$E[X_R] = pE[X/p] + (1-p) \cdot 0 = (p/p)\, E[X] = E[X]. \tag{4.111}$$

In the context of path tracing, we can employ random termination probabilities at each path vertex and scale the path throughput of surviving paths by the termination probability. As terminated paths are realized to zero, they do not need to be evaluated any further.

While the termination probability does not have an effect on the expectation, variance does in fact increase. When paths are often terminated too early. The surviving paths that end up finding significant contribution will be weighted even higher through termination probabilities, leading to fireflies. To counteract this, recent works try to estimate the remaining contribution of paths to make a more informed choice on whether to terminate paths or not [156, 130].

Discussion Forward path tracing can reliably sample many phenomena. In particular, lens systems in front of the camera can be sampled trivially by sampling the material model at each path vertex directly. Problems arise mostly in caustic paths ($LDS^{+}E$) where a small emitter illuminates a surface through specular interactions. In such a case, next event estimation is ineffective, since a direct connection from the surface to the light would be blocked by a specular object. Du to the small size, BSDF sampling would also not be able to reliably generate samples on the light source.

4.5.2. Next event estimation

So far, paths are generated with probability densities based on the local material model at a path vertex. This can be problematic in cases where light sources are either small or far away. That is, they subtend a small solid angle at a given path vertex. As a result, the probability of actually generating a path vertex on it becomes very small, leading to higher variance. The extreme case are point lights that have no physical extent and can therefore not be sampled through the local material model at all.

As already introduced, the integration domain of the rendering integral can be conveniently changed. In particular, path vertices can be sampled directly in surface area measure. We can therefore generate path vertices on light sources directly and connect with the previous path vertex using the geometry term.

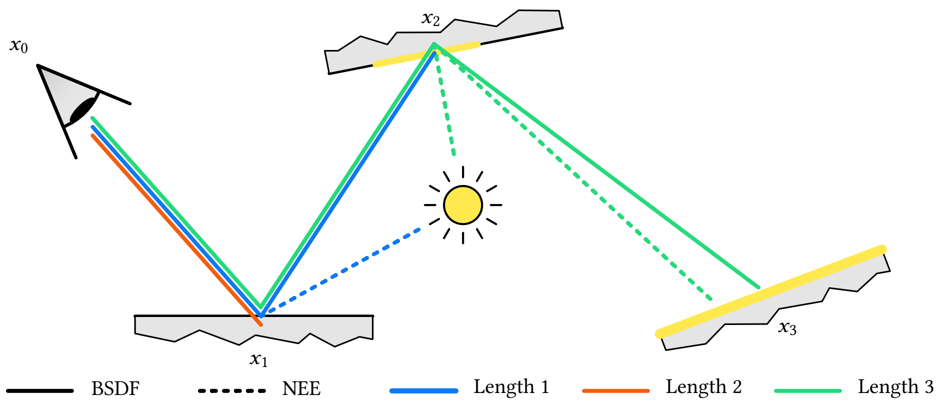


Figure 4.14.: Next event estimation (NEE) allows to directly sample light sources instead of relying on BSDF sampling to find potentially small light sources. In particular, point light sources that have no explicit area can now be incorporated. Since area light sources can be sampled by both techniques, they need to be combined using multiple importance sampling (MIS).

However, there are many instances where we would prefer to sample according to either BSDF or NEE, respectively. We effectively need both probability densities and multiple importance sampling provides the theoretical framework to combine them. Figure 4.14 shows example paths generated by NEE and BSDF sampling and which instances need to be combined via MIS.

Note that since the path prefix is the same, MIS weighting only incorporates the probability density for NEE and BSDF sampling.

While next event estimation allows to explicitly place points on light sources, we still need to define a suitable probability density. Given an area light source, we need to sample a point on it. Furthermore, scenes consist of multiple such light sources, therefore a light source needs to be selected first.

Sampling direct illumination can be divided into two problems: selecting a light source with probability mass $p(i)$ and sampling a point on the selected light source with probability density $p(x \mid i)$. The overall probability density is then:

$$p(x) = p(i)p(x \mid i). \tag{4.112}$$

$p(x \mid i)$ is often given as a uniform density over the surface area of the light source:

$$p(x \mid i) = \frac{1}{|A_i|}, \tag{4.113}$$

where A_i is the surface of light source i. Other works try to directly sample the solid angle subtended by triangles [4] and further incorporate the BRDF [127]. In the following, different probability mass functions for light selection will be introduced.

Global probability mass A simple probability mass can be defined based on globally constant weights associated with light sources:

$$p(i) = \frac{w_i}{\sum_j w_j}, \tag{4.114}$$

where w_i is the weight of light source i. Examples are uniform weights $w_i = 1$ (uniform sampling) or weights based on the total flux emitted of light sources $w_i = \Phi_i$ (power sampling). The latter can perform better in cases where light source have varying brightness. Globally constant weights have the advantage that sampling can be performed in constant time using alias tables [158]. However, they

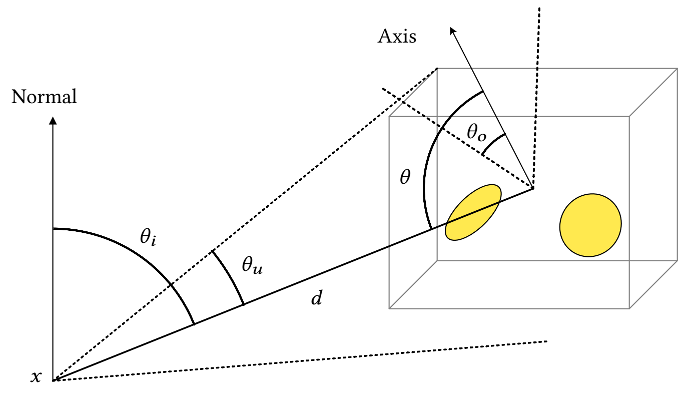


Figure 4.15.: The light hierarchy by Conty Estevez et al. [28] uses aggregate information in nodes to approximate expected contribution from light sources towards a given surface point.

fall short in scenes where specific parts of the scene are illuminated by only a small portion of lights, resulting in frequent selection of irrelevant light sources, leading to higher variance.

Light hierarchies To reduce selection of irrelevant light sources, the probability mass function needs to depend on the surface point that is to be illuminated. However, this would require accessing all light sources for every sample, since no precomputation can be done for individual surface points.

To remedy this, light hierarchies [28, 46] group related light sources in a hierarchy that is later used for sampling. Instead of considering all light sources, a stochastic traversal from the root of the hierarchy to a leaf node (a light source) is performed. Only direct children of the traversed inner nodes are accessed for sampling. This moves the complexity of sampling from linear closer to logarithmic with regards for the number of light sources. Similar to the global probability mass, a weight $w_i(x)$ is defined for each node. Crucially, this weight can also depend on the surface point x and is computed ad-hoc during the stochastic traversal. Also, each node is additionally augmented with aggregate information about the contained light sources.

In the case of Conty Estevez et al. [28], nodes are augmented with the total emission of all light sources E and two cones (common mean direction ω and two angular extents θ_o and θ_e) that bound the possible orientations and emission profiles of light sources. The weight function is defined as follows:

$$w_i(x) = \frac{f_a \cos\theta_i' E}{d^2} \begin{cases} \cos\theta' & \theta' < \theta_e \\ 0 & \text{otherwise} \end{cases}, \tag{4.115}$$

with $\theta' = \max(\theta - \theta_o - \theta_u, 0)$ and $\theta_i' = \max(\theta_i - \theta_u, 0)$. Figure 4.15 visualizes the used quantities for a given surface point and node. $f_a \cos\theta_i$ is a diffuse approximation to the material model, where θ_i is a bound for the minimum incident angle at the surface point and θ is a bound for the minimum exitant angle at a potential light source. d is the distance between the surface point and center of the node (as a coarse approximation to the true distance to any contained light source). The weight is therefore the best-case contribution that is to be expected from a light source contained in the given node.

Construction of light hierarchies is similar to construction of bounding volume hierarchies (as later discussed in section 4.7.2), although different cost metrics are used that also take into account the orientation and emission profiles of light sources.

4.6. Caching methods for solving light transport

This section covers caching methods in the context of light transport. We will first discuss relevant data structures that allow to store distinct caches for spatial and directional domains and then consider the relevant use cases of path guiding and radiance caching.

4.6.1. Spatial data structures

Caching methods need to organize cached data in such a way that it can be efficiently retrieved during rendering. This section introduces various spatial data structures that allow to map from surface point to a local cache record in the spatial data structure

Screen space For certain use cases it is sufficient for a cache to only be defined on surface points that are visible by the camera. In such a case, a screen-space data structure may be used that uses the camera projection to map positions to an intermediate space where cache records can be stored. Complications typically arise in fast movements, where previously not visible regions suddenly become visible, or moving objects cause other objects to become disoccluded. In that case, no previous information is available and can cause artifacts depending on how the cache structure is used.

Hashed grid Subdividing the scene into a dense three-dimensional grid is very efficient due to a simple index computation, but has prohibitive storage requirements due to cubic scaling with resolution. One thousand subdivisions along each axis already comprise a billion grid cells, resulting in memory consumption of more than a gigabyte depending on the size of cache records. Most of these cells are actually empty since no geometry is contained in them. The approach of hashed grids [58, 14] is therefore to only store occupied grid cells in a hash table that uses the three-dimensional cell index as key. The indexing scheme can also be adapted to employ some form of level of detail by enlarging grid cells as distance to the camera increases.

When accessing hash tables, conflicts arise because multiple grid cells may map to the same index of the hash table. Such collisions can either be ignored. This results in the hash table record being shared by possibly distant spatial locations with corresponding graphics artifacts. Hash table entries can also be atomically “claimed” by storing a secondary identifier of grad cell that can be used to disambiguate whether the given hash table index belongs to the cell that is being accessed.

Adaptive hierarchies The hashed grid subdivision scheme is relatively rigid, only allowing to control subdivision through metrics such as camera distance. Through hierarchical approaches it can also be controlled more explicitly. Examples include octree structures [116] or k-d trees [134].

Metrics for further subdivision are often based on collected samples inside cache records (one factor in determining how well-trained a cache record is).

Texture space [118]

One challenge in texture space mappings is to ensure that no overlap occurs (the mapping must be injective), to ensure that cache records are not shared between distant spatial regions. While this can be ensured, UV-maps authored by artists for texture mapping generally do contain overlap, thus requiring duplicate effort for separate UV-maps for caching purposes.

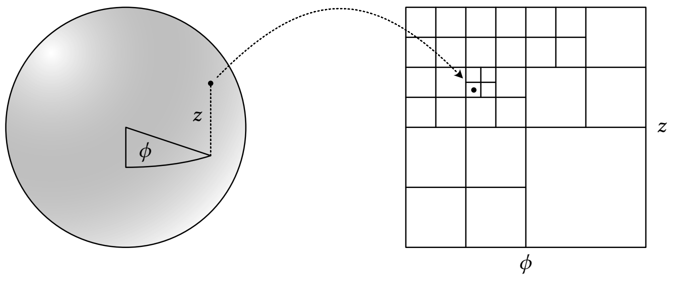


Figure 4.16.: Directional quadtree representation [116] that maps directions to a two-dimensional domain using cylindrical coordinates which embeds a quadtree structure to store incoming radiance.

4.6.2. Directional representations

This section presents relevant distributions to represent spherical functions of the form $f : \Omega \to \mathbb{R}$.

Directional quadtrees Directional information can be represented through a grid-structure after mapping three-dimensional directions to two dimensions using e.g. cylindrical coordinates. To reduce storage overhead, the dense grid can be replaced by a hierarchy where regions of interest are represented with higher resolution.

One example are directional quadtrees [116], where a quadtree on the 2D-mapping is employed (fig. 4.16). Learning of the structure is performed iteratively by splatting samples into the hierarchy and splitting cells that have more than 1% of the contribution of the entire quadtree. Sampling is performed by stochastically traversing the hierarchy from the root node by selecting children proportional to the contribution of their respective subtrees. Conversely, computation of the probability density can be performed by traversing the quadtree for a given sample. This approach has relatively high memory overhead since all nodes in the hierarchy need to store weights which are typically represented as 32-bit floating point numbers.

Parallax-aware von Mises-Fisher mixtures von Mises-Fisher mixtures have been successfully used in representing spherical functions in the context of path guiding for incoming radiance [134]. One complication is that this mixture is only really accurate for a single surface point. But in most systems such distributions are shared between points in a larger spatial region. Naively fitting the distribution to all surface points will cause certain features to become blurred. This is due to parallax: Features that are close to a surface will rapidly change direction when changing the reference point on the surface from which this feature is observed. To counteract this, the mixture is stored with respect to a reference point and samples are corrected for this changed position. For a given sample, we therefore also need to know the distance to the given feature. Given a feature that is in direction μ from x with distance t, the direction μ' for another surface point x' can be computed as:

$$d' = \frac{x + d \cdot t - x'}{\|x + d \cdot t - x'\|}. \tag{4.116}$$

This is also illustrated in fig. 4.17.

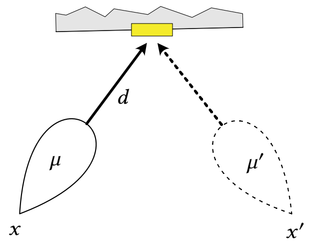


Figure 4.17.: Parallax-aware von Mises-Fisher distribution [134]. When maintaining the distance to the feature that a lobe represents, the mean direction μ can be adapted for different source positions x'.

4.6.3. Path guiding

When considering the rendering equation in eq. (4.87), the incoming radiance L_i in the integral expression is usually unknown and can only be evaluated using point samples. Therefore, sampling algorithms typically focus on producing samples according to the BRDF model, which is locally known when constructing paths. While next event estimation in section 4.5.2 provides some information about L_i (positions of light sources), it usually lacks visibility information, thus L_i is also only coarsely approximated.

Path guiding is a caching method where drawn samples are used to construct improved sampling densities. It is a relatively general approach and can be applied to various parts of the integrand from guiding selection probabilities of discrete distributions [151], to spherical functions (L_i in particular) [71, 116] and full paths [131].

Our work focuses mainly on discrete selections for light sources and representing incoming radiance. The directional and spatial representations presented in sections 4.6.1 and 4.6.2 may be used as guiding distributions. This section will cover additional concerns connected to path guiding.

Combining guiding with uninformed sampling Guiding is often used in conjunction with uniformed sampling techniques like BSDF sampling. Particularly in the beginning, no reliable guiding distributions are available, so only uninformed techniques are available then. Once enough sample data has been collected, most approaches allocate a fixed fraction of samples to guiding and uninformed sampling. This increases robustness in cases where the guiding distribution overfits certain features. MIS with the one-sample model is used to combine both approaches: Given a guiding density p_{guiding}, BSDF density p_{BSDF} and sampling fraction α that decides between BSDF and guiding, the combined density has the following form:

$$p_{\text{combined}}(\omega) = \alpha p_{\text{BSDF}}(\omega) + (1 - \alpha) p_{\text{guiding}}(\omega). \tag{4.117}$$

The sampling fraction is usually fixed between 25% to 50% uninformed samples, although some works try to explicitly optimize this quantity [154, chapter 5]

Sample data collection To account for indirect illumination, samples used in path guiding need to incorporate an estimate of incoming radiance due to indirect illumination. For approaches like unidirectional forward path tracing, this is not trivial to achieve, since intermediate vertices of constructed paths are usually not kept until the whole path has been constructed. Thus, incoming radiance of intermediate path vertices cannot be readily used for training.

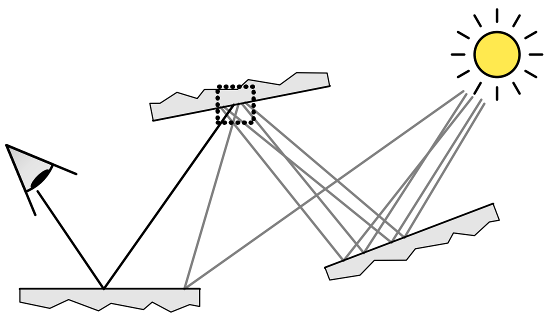


Figure 4.18.: Radiance caching [90] stores outgoing radiance spatially in cache records that can be used to terminate paths on surfaces while the cache record can be used to approximate the remaining contribution.

While full path state can in theory be stored [70], it is less practical for scheduling variants like wavefront path tracing, where each level of indirection is processed in parallel (as opposed to individual paths). In that case, full path state needs to be maintained for millions of paths, which can have too large of a memory footprint.

One approach to mitigate this is to combine path guiding with a form of radiance cache that can be used to approximate incoming radiance due to indirect illumination [125]. That way, path vertices that are currently processed can be directly used for training, although the approximation can make the guiding distribution itself more approximate, since an approximate cache is used instead of an unbiased path contribution estimate.

Iterative learning Most guiding techniques have a split between learning the distributions and using them to generate the final image. The rendering process is therefore divided into phases of execution. Depending on the training scheme, the training phase is additionally subdivided into iterations of various length.

Müller et al. [116] use an exponential learning scheme where each training iteration is executed twice as long as the previous iteration. This stems from the construction that guiding distributions are completely re-learned every frame. The used quadtree guiding distribution (section 4.6.2) increases in complexity as more samples are used to train it, therefore even more samples are needed in later iterations. The final rendering phase is also executed with twice the execution time as the last training iteration. This means that half of the total sample budget is spent on training the guiding distribution.

Ruppert et al. [134] use a batched learning scheme where small fixed-length sample batches are used to progressively adapt the guiding distributions using expectation maximization. Samples are not discarded in this case. This approach has the advantage that the guiding distributions are adapted more often based on the observed data compared to the exponential increase of iterations.

4.6.4. Radiance caching

Radiance caching [90] is a method where spatial data structures are used to store intermediate results of illumination (fig. 4.18). Instead of generating full paths that terminate on a light source, paths can be terminated earlier by consulting the cache structure and completing the path with the stored cache value.

Given the measurement contribution in eq. (4.106), a corresponding estimator has the form $g = f(x)/p(x)$. We can define a split version $g = g_l^{\text{pre}} g_l^{\text{suf}}$, which splits the terms of g before and after vertex l, with $0 < l < k - 1$:

$$g_l^{\text{pre}} = \frac{W(x_0)G(x_0, x_1)\left(\prod_{i=1}^{l-1} f_r(x_{i-1}, x_i, x_{i+1})G(x_i, x_{i+1})\right)}{p\left(x_0, \ldots, x_l\right)}, \tag{4.118}$$

$$g_l^{\text{suf}} = \frac{\left(\prod_{i=l}^{k-2} f_r(x_{i-1}, x_i, x_{i+1})G(x_i, x_{i+1})\right) L_e\left(x_{k-2}, x_{k-1}\right)}{p\left(x_{l+1}, \ldots, x_k \mid x_0, \ldots, x_l\right)}. \tag{4.119}$$

The radiance caching estimator is then defined as:

$$\bar{g} = g_{l(x)}^{\text{pre}} \operatorname{rc}\left(x_{l(x)}, x_{l(x)-1} - x_{l(x)}\right). \tag{4.120}$$

The estimator borrows a prefix of length $l(x)$ from the original estimator g but estimates the remaining suffix at $x_{l(x)}$ through the radiance cache, expressed through $\operatorname{rc}(x, \omega)$. x defines the spatial location and ω the outgoing direction. To allow for individual per-path decisions, $l(x)$ is a function of the path itself. In practice, terminating at vertex $x_{l(x)}$ should only depend on the prefix $\left(x_0, \ldots, x_{l(x)}\right)$ to allow for incremental construction.

For a real implementation of $\operatorname{rc}(x, \omega)$, it is impossible to maintain accurate radiance estimates between all vertex pairs. Instead, statistics inside spatio-directional partitions in a suitable spatial data structure are stored in discrete cache records to recover (weighted) averages of radiance. We define a partition as $(x_0, x_1) \in P \subseteq A^2$ (a subset of all vertex pairs). We define the radiance cache as the expectation over the suffix estimator for all vertex pairs in the same partition:

$$\forall (x, y) \in P : \operatorname{rc}(x, y - x) = E_{(x_0, x_1) \in P}\left[E\left[g_1^{\text{suf}} \mid x_0, x_1\right]\right] = E_P\left[g_1^{\text{suf}}\right]. \tag{4.121}$$

The occurrence probabilities of x_0 and x_1 (the joint probability of all path prefixes leading to these vertices) are not accounted for in this formulation, because they are intractable to compute. This means that this expectation gives higher weight to vertex pairs with higher occurrence probability.

Cache Termination Access to the cache can be controlled using various heuristics. In the simplest case, termination is performed directly at the primary path vertex. This is done often in literature where more focus is spent on achieving a good distribution of cache records. This is not trivial to achieve in interactive contexts, therefore many approaches defer access to later path vertices. Cache access can be deferred to the second path vertex [87, 14] or defined based on a heuristic of area spread [11, 117]:

$$a(x_1, \ldots, x_n) = \left(\sum_{i=2}^{n} \frac{\|x_{i-1} - x_i\|^2}{p(x_i \mid x_{i-1}, x_{i-2})|\cos\theta_i|}\right)^2. \tag{4.122}$$

Intuitively, a large probability density means that generated paths are very focused around a given path, thus being sampled with high resolution and making any errors more visible. This is compared against the approximated footprint at the first path vertex:

$$a_0 = \frac{\|x_0 - x_1\|^2}{4\pi\cos\theta_1}. \tag{4.123}$$

4.7. Ray tracing acceleration structures

Numeric integration of light transport requires repeated computation of the closest point on the scene from a starting point and direction. In particular, incremental path construction in path tracing needs to compute the next path vertex after having sampled a direction. This problem is referred to as ray tracing, where a ray is given as a starting point and direction. A naive solution would be to test all scene primitives against a given ray, although it is obvious that this would be prohibitively expensive given that practical scenes consist of hundreds of millions of primitives and a similar order of magnitude of ray tracing queries are made with algorithms like path tracing.

The general approach to accelerating ray tracing queries is to organize space or scene primitives in some form of hierarchy. Inner nodes of the hierarchy approximately (but conservatively) encompass the subtree. If a ray already has no intersection with the approximation, then the entire subtree can be ignored, thus saving computational time. This section first introduces various acceleration structures that can be used to efficiently fulfill ray tracing queries (section 4.7.1). We will then focus on construction of bounding volume hierarchy (BVH) acceleration structures in particular (section 4.7.2).

4.7.1. Acceleration structures

Acceleration structures can be coarsely divided into space and object partitioning schemes. Space partitioning acceleration structures subdivide space into disjoint regions. Each region then stores references to the primitives that intersect this region. Object partitioning acceleration structures, in contrast, subdivide the set of scene primitives directly. The space that primitive subsets enclose is then approximated with bounding volumes.

Space partitioning A simple form of space partitioning is a *uniform grid.* For each primitive, it is determined which grid cells the primitive intersects with and a reference to that primitive is stored in each cell. For a given ray tracing query, only the primitives contained in intersected grid cells are tested for intersection. A practical implementation marches the grid by visiting grid cells with closer intersections first. An important property of space partitioning schemes is that if space partitions are visited in order of the closest intersection, further partitions do not need to be considered if an intersection is found within an earlier partition, since potential intersections in later partitions are guaranteed to be further away.

The simple grid partitioning raises two problems, however: Depending on the grid resolution, primitives can intersect many grid cells, resulting in duplicate intersection tests. Furthermore, empty grid cells need to be explicitly intersected, since it is not known beforehand whether they contain primitives. As such, adaptive data structures are needed that have coarser resolution in regions with fewer primitives and higher resolution in regions with more primitives.

Octrees [105] are an example of such a data structure. It is a hierarchical data structure with a cube encompassing the entire scene at its root. Each inner node of the hierarchy subdivides its space into eight octants by halving the intervals along each axis. This construction allows more refinement in regions of high complexity and less refinement otherwise. The subdivision scheme is still somewhat rigid, since the intervals along each axis are always halved. *K-d trees* [13] are binary trees where space in each node is split along a selected axis. In contrast to octrees, the interval can be split arbitrarily. *Binary space partitioning (BSP)* [48, 147] is even more flexible in that arbitrary planes can be used instead of axis-aligned planes.

Duplicate intersection tests usually cannot be fully alleviated with space partitioning schemes if overlap between primitives occurs.

Object partitioning through Bounding volume hierarchies (BVHs) Instead of partitioning space, bounding volume hierarchies [27] partition objects directly in a hierarchical fashion. While encompassed space is inherent in a space partitioning scheme (and can be directly used for ray traversal), this is not the case in object partitioning. Therefore, the hierarchy needs to be augmented with so-called bounding volumes. They can be freely defined with the only requirement that a bounding volume of a node should contain all geometry in its subtree.

To ensure efficient ray traversal, bounding volumes should have low storage overhead, be efficient to test for intersection and enclose the contained geometry as tightly as possible to reduce false positives. Also, it should be efficient to compute (close to) minimal bounding volumes during construction given a set of primitives or other bounding volumes. Axis-aligned bounding boxes (AABBs) are used most prominently due to requiring low storage and being efficient to intersect. They are represented by six scalar values, each denoting one of two planes along each axis. They can be computed efficiently through the minimum and maximum extents of primitives or bounding boxes along each axis.

Other examples include oriented bounding boxes (OBBs) where the planes are not axis-aligned or bounding spheres. While oriented bounding boxes can fit geometry more tightly, more storage is required (an additional rotation needs to be stored compared to AABBs) and intersection tests are more expensive. Additionally, computation of the minimum oriented bounding box is not trivial [121].

Traversing BVHs for ray tracing queries is more complex compared to space partitioning, since bounding volumes of neighboring subtrees can overlap. This has the implication that even if an intersection inside a node is found, its sibling(s) may still need to be tested if their bounding volumes have overlap. More precisely, a node needs to be traversed if the closest intersection found so far is farther away than the intersection with its bounding volume.

4.7.2. Constructing bounding volume hierarchies (BVHs)

The construction of BVHs possesses many degrees of freedom. In particular, for a given scene there exist many valid BVH instances, although most of them will perform poorly. This section introduces construction algorithms for BVHs and is inspired by Meister et al. [110]. We will first introduce cost models that are used as a basis for BVH construction and then introduce relevant construction algorithms. In particular, top-down construction [157] and PLOC [106] are introduced.

Cost models Query performance of BVHs depends on many factors. Generally, the number of intersected nodes and primitives are a large factor, but also the resulting memory access pattern can have a large effect. The number of intersections is hard to determine in advance, since it is very dependent on the actual query instances. One ray might pass by a surface in proximity and therefore need to be tested against all the corresponding nodes and primitives. Another ray might be offset just a bit further away and the subtrees constituting the surface can be skipped entirely, leading to better performance. Cost models therefore usually do not consider individual rays, but assume a distribution of rays as a model.

The most widely-used cost model is the surface area heuristic [53]. Cost is measured through the likelihood that a ray will intersect nodes. This model assumes that all rays emerge uniformly from outside the scene and the intersection probabilities are approximated through the areas of bounding

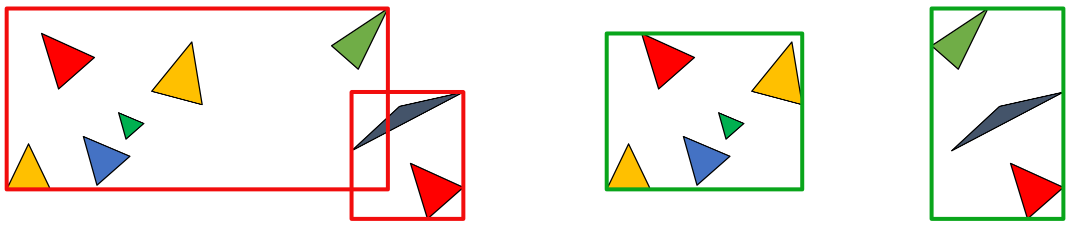


Figure 4.19.: The surface area heuristic (SAH) measures bounding box areas. This example shows two possible subdivisions, where the left one has larger bounding box area. The SAH tends to isolate dense regions.

box (fig. 4.19). Intuitively, a larger surface are increases the likelihood that a ray intersect with a box. Given that a ray intersects a node N, the likelihood that it also intersects a child node N_i is therefore approximated through the ratio of the respective surface areas:

$$p(N_i \mid N) \approx \frac{\mathrm{SA}(N_i)}{\mathrm{SA}(N)}, \tag{4.124}$$

where $\mathrm{SA}(N)$ is the surface area of the bounding volume belonging to node N. The heuristic for a subtree with root N is then defined as:

$$c(N) = \begin{cases} c_T + \frac{\mathrm{SA}(N_R)}{\mathrm{SA}(N)} c(N_L) + \frac{\mathrm{SA}(N_R)}{\mathrm{SA}(N)} c(N_R), & \text{if } N \text{ is inner node} \\ c_I |N|, & \text{otherwise} \end{cases}. \tag{4.125}$$

The heuristic recursively collects the cost of accessing a node c_T, weighted by the recursive probabilities (surface area ratios) of intersecting the node. For leaf nodes, all $|N|$ primitives contained in the node need to be intersected, which is expressed with the intersection cost c_I. The recursion can be unrolled to the following expression:

$$c(N) = \frac{1}{\mathrm{SA}(N)} \left[c_T \sum_{N_i} \mathrm{SA}(N_i) + c_I \sum_{N_l} \mathrm{SA}(N_l) |N_l| \right]. \tag{4.126}$$

The ratio of surface areas of every interior node to the root node is equivalent to the recursive expression above, since the surface area of all nodes between the root and an interior node cancel.

Construction algorithms There exist many construction algorithms that can be roughly categorized into top-down, bottom-up and more specialized schemes [110].

Top-down construction starts with the set of all primitives and recursively partitions this set. The topology of the final hierarchy directly corresponds to the recursion, where each invocation corresponds to a node in the hierarchy. The partitioning aims to minimize a cost model, usually the surface area heuristic. The set of all possible partitions of a set grows exponentially with the number of elements in a set. It is therefore unrealistic to naively consider all partitions for scenes with millions of primitives. Most approaches therefore constrain the search to a small subset of possible partitions:

- *Sweep SAH* For each axis, all primitives are sorted according to the respective component of their centroids (fig. 4.20 left). All splits along the sorted sequences are considered and their cost evaluated. Naively evaluating the cost of a single split requires accessing all primitives ($O(n^2)$ for all splits). Instead, surface areas of all prefixes and suffices can be precomputed in an incremental fashion. Evaluating the cost for each split then only requires accessing the surface area for the respective prefix and suffix ($O(n)$ for all splits). Execution time is then dominated by the sorting step $O(n \log n)$. The execution time for constructing the entire hierarchy is then $O(n \log^2 n)$.

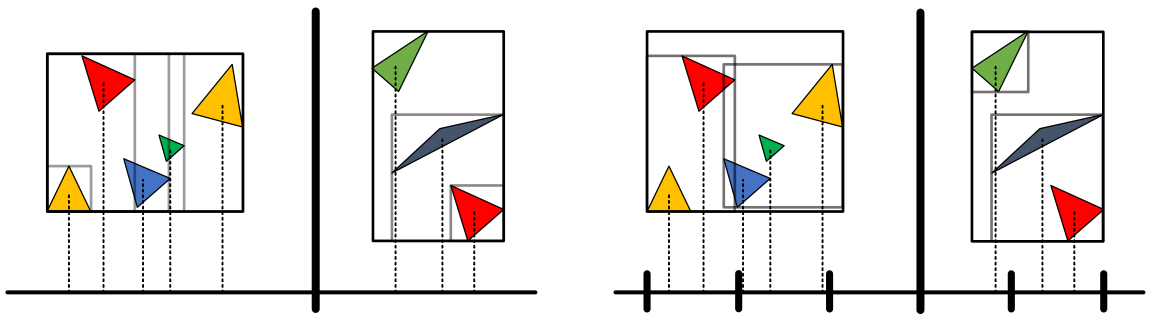


Figure 4.20.: Split methods used in top-down bounding volume hierarchy construction along a selected axis. Sweep SAH (left) sorts primitives along the axis by considering the projection of primitive centroids. Bounding boxes are constructed incrementally from both sides. Binning divides the axis into discrete bins into which primitives are inserted based on the axis projection.

- *Binned SAH* Since sorting is the most expensive part in sweep SAH, it can be replaced by binning [157]. For each axis, a fixed number of bins is maintained that store bounding boxes (fig. 4.20 right). Bins evenly subdivide the bounding box that spans all primitive centroids. Each primitive then expands the bounding box for each bin its centroid falls into. The border between each bin is then considered as a possible split plane and the SAH evaluated. The execution time is thus dominated by the binning $O(n)$, resulting in a total execution time of $O(n \log n)$.

Bottom-up construction is based on agglomerative clustering that initially treats each primitive as its own leaf node. Through some cost metric nodes are iteratively merged into inner nodes that reference the original nodes, leaving a hierarchy behind when only a single node remains. Naively evaluating all node pairs results in a worst-case cubic scaling behavior, which is infeasible for large scenes. Parallel locally ordered clustering (PLOC) [109] performs approximative clustering by only considering a small neighborhood around a given node using the surface are of the merged node as cost metric. The neighborhood is induced through a space-filling Morton curve. Multiple nodes are merged in parallel: Each pair of nodes that considers each other as the best candidate for merging can be directly merged without interfering with other nodes.

4.8. GPU architecture

The algorithms presented in this thesis are tailored for execution on graphics processing units (GPUs). This section introduces architectural principles that are the basis of current graphics hardware and the resulting implications on designing efficient algorithms for this hardware. We derive hardware-agnostic terminology from the Vulkan specification and hardware-specific terminology from NVIDIA.

Subgroup execution Threads are not executed individually. Rather, they are executed in groups, referred to as subgroups, that execute the same instructions. This allows to amortize instruction scheduling logic among a larger pool of threads and use more area on the die for arithmetic units or memory. Code branching can thus have a negative effect on performance if threads in a subgroup branch differently, since threads executing either one or another code path need to be paused. Additionally, memory accesses are grouped into single transactions if they access the adjacent memory (memory coalescing). This usually happens at the granularity of individual subgroups.

Latency hiding All executed instructions incur a latency overhead (memory accesses in particular can take hundreds of cycles). While CPU architectures employ deep pipelining with speculative execution to hide latency, GPUs use a different approach: Multiple subgroups are executed simultaneously on a single SM. When a subgroup executes an instruction with long latency, other subgroups are executed in the meantime.

This requires all subgroups to be readily executable. In particular all registers that are utilized by subgroups need to remain resident on the SM. A large register file (compared to CPUs) is used to hold the execution states of multiple subgroups at once to allow for fast context switches. Programs executed on the GPU can have different requirements on register usage. A high number of registers can impact performance, since fewer subgroups can be used for latency hiding.

5. Temporal Sample Reuse for Next Event Estimation and Path Guiding for Real-Time Path Tracing

This chapter is based on A. Dittebrandt, J. Hanika, and C. Dachsbacher. "Temporal Sample Reuse for Next Event Estimation and Path Guiding for Real-Time Path Tracing". In: *Eurographics Symposium on Rendering - DL-only Track* (2020). DOI: `10.2312/SR.20201135`

Good importance sampling is crucial for real-time path tracing where only low sample budgets are possible. We present two efficient sampling techniques tailored for massively-parallel GPU path tracing which improve next event estimation (NEE) for rendering with many light sources and sampling of indirect illumination. As sampling densities need to vary spatially, we use an octree structure in world space and introduce algorithms to continuously adapt the partitioning and distribution of the sampling budget. Both sampling techniques exploit temporal coherence by reusing samples from the previous frame: For NEE we collect sampled, unoccluded light sources and show how to deduplicate, but also diffuse this information to efficiently sample light sources in the subsequent frame. For sampling indirect illumination, we present a compressed directional quadtree structure which is iteratively adapted towards high-energy directions using samples from the previous frame. The updates and rebuilding of all data structures takes about 1ms in our test scenes, and adds about 6ms at 1080p to the path tracing time compared to using state-of-the-art light hierarchies and BRDF sampling. We show that this additional effort reduces noise in terms of mean squared error by at least one order of magnitude in many situations.

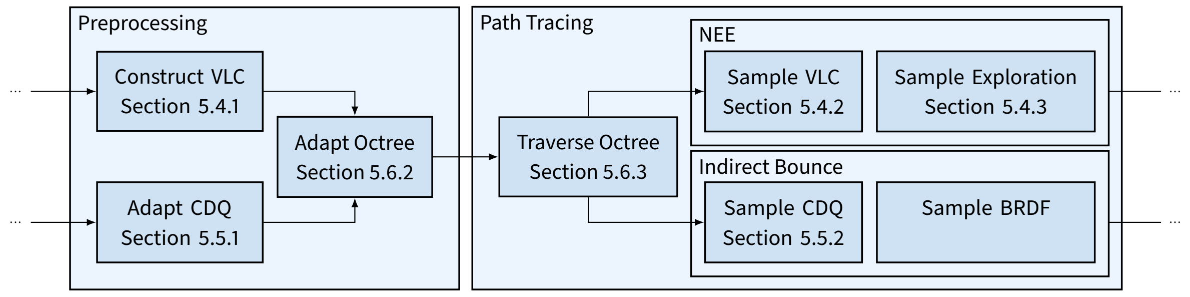


Figure 5.1.: Execution of components over a frame, split into a preprocessing and path tracing step. Arrows represent dependencies. Each leaf of the octree contains a unique Visibility Light Cache (VLC) and Compressed Directional Quadtree (CDQ) to account for spatial variation. During preprocessing, each VLC/CDQ and the octree itself is constructed/adapted based on samples of the previous frame. During path tracing, the adapted octree is traversed to access VLC and CDQ for importance sampling of visible lights (NEE) and high-energy directions (Indirect Bounce), respectively. Secondary techniques are used to explore new lights and to importance-sample directions according to BRDF. Figure taken from [35].

5.1. Introduction

Rasterization has been the standard real-time rendering algorithm for decades since its early implementation in hardware and efficiency. Current graphics processing units (GPUs) are capable of processing hundreds of millions of primitives per frame at interactive rates. However, rasterization determines primary visibility and approximating complex or global illumination is left to secondary techniques resulting in complex and deep pipelines. For example, shadow computation, although conceptually simple, resulted in myriads of techniques [42], shading with many lights requires efficient culling [122] and intricate shadow computation [123]. For indirect illumination, various approaches have been adapted to rasterization [82, 31, 132].

The recent advent of hardware ray tracing enables far more flexible real-time rendering algorithms. In particular, this includes the introduction of path tracing, which has been the standard in and exclusive to offline rendering for many years [46]. However, path tracing (or Monte Carlo techniques in general) comes at a cost: Noise. With real-time budgets, only few paths (or even just one) can be computed per pixel, leading to severely noisy images. Many recent approaches attempt to *denoise* these images by leveraging temporal coherence between successive frames as well as spatial coherence within a frame [100, 25, 136]. Furthermore, denoising generally introduces artifacts like lag and blurriness which stem from this temporal and spatial reuse. Instead of only “repairing” images after rendering, it is thus essential to reduce the image noise by improving the sampling quality. This importance sampling, however, is challenging even in offline rendering (see e.g. recent work [154]) as the shape of the integrand is generally unknown.

In this chapter we improve importance sampling for real-time path tracing in dynamic scenes, where the dynamic content requires the sampling to adapt over time. Obviously, these sampling techniques must not require precomputation exceeding the budget within one frame. To this end, we exploit, similar to many of the aforementioned approaches, temporal coherence by reusing the drawn samples (visibility queries) in one frame to construct improved sampling densities for the subsequent one. We focus on two pressing aspects in current real-time path tracing: next event estimation (NEE) with many light sources, and sampling of indirect illumination.

With many lights and complex geometry, the decision which light to sample for NEE is nontrivial. However, often the decisive factor is visibility, i.e. only light sources that are not blocked by scene geometry can contribute to the shading of a surface point. To this end, we store lights that had visible samples in the previous frame in a Visibility Light Cache (VLC, section 5.4). Sampling then focuses on this set of lights (but also includes previously occluded lights). With a high-quality importance sampled selection based on Linearly Transformed Cosines (LTCs) for area lights [68], we achieve sampling closer to product importance sampling.

When computing indirect illumination, standard BSDF sampling is inadequate and causes fireflies when incident light covers small solid angles. To improve this sampling step, we introduce a Compressed Directional Quadtree (CDQ, section 5.5) inspired by offline path guiding [116]. Our trees require only very few bits (32-128 bits). This compact representation trades memory usage for sampling quality, but is tractable for real-time and achieves significant variance reduction in many cases.

Obviously visibilities and sampling densities vary over the scene geometry. To adapt spatially, we partition the scene using an octree and provide individual estimators (VLCs and CDQs) per leaf node (section 5.6). The key is that we adapt this octree structure temporally based on the number of samples that fall within each leaf node. For this, we introduce a simple split/collapse algorithm with configurable thresholds. The advantage is that this structure automatically balances the available samples among all leaves, improving robustness of our techniques (see section 5.4.1 for caveats due to the fixed subdivision

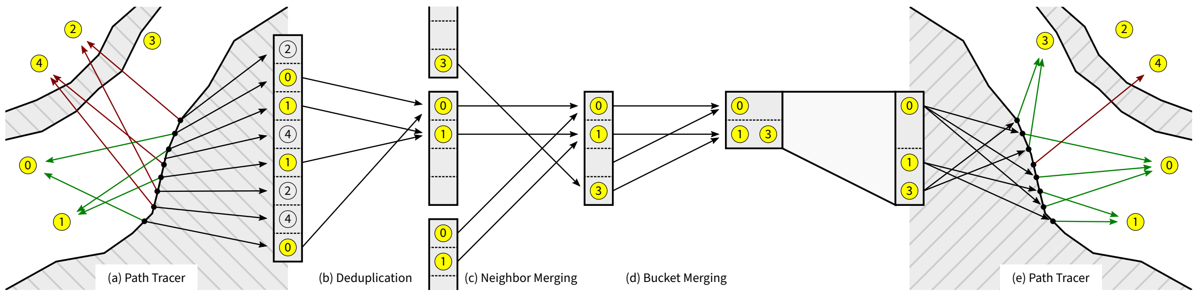


Figure 5.2.: Overview of Visibility Light Cache (VLC). (a) Samples from previous frame are placed into a region-specific buffer location. (b) Visible samples are deduplicated through a hash table. (c) Neighbor hash tables are merged to diffuse information (light 3 was not present in hash table). (d) Buckets of hash table are merged to minimize bucket count (capacity: 2). (e) Path tracer in next frame uses VLC. Buckets are selected uniformly. Lights in bucket are then importance sampled. For continued exploration, a fraction of samples is created with an exploration strategy (sample for light 2).Figure taken from [35].

scheme). With a readily available octree structure *before* path tracing, we can also preallocate space for storing samples, instead of rearranging them in memory afterwards.

To summarize, the key contributions of this chapter are:

- a Visibility Light Cache for product importance sampling NEE,
- a Compressed Directional Quadtree for guiding sampling of indirect illumination,
- a temporally- and sample-adaptive octree to organize estimators.

5.2. Overview

The VLC, CDQ and octree are mostly independent techniques and are discussed in the sections 5.4, 5.5 and 5.6, respectively. This section serves as a high-level overview on the combined interaction of their components as shown in figure 5.1. We divide execution of each frame into the two passes preprocessing and path tracing. During preprocessing, samples from the previous path tracing are used to construct the VLC and adapt the CDQ of each octree leaf node. For the VLC, the set of lights with unoccluded samples is computed (section 5.4.1) and the CDQ is adapted towards high-energy directions (section 5.5.1). The observed sample counts are used to adapt the octree with a simple threshold-based split-collapse scheme (section 5.6.2).

During path tracing, the octree is traversed at each surface interaction to access the contained VLC and CDQ (section 5.6.3). For Next Event Estimation (NEE), the VLC is used to select a visible light source (section 5.4.2). To discover new lights and ensure unbiasedness, a secondary exploration technique is used (section 5.4.3). The decision on which technique (VLC/exploration) to use is made stochastically based on a configurable fraction (typically around 10% for exploration). For the indirect bounce, the CDQ is used to sample a high-energy direction (section 5.5.2). Just as with NEE, a secondary technique is used. The BRDF is used in this instance with a configurable fraction (typically around 50%) to bound noise in the case of highly specular materials. All techniques are combined with MIS and the balance heuristic. To achieve this, they are specifically designed for efficient PDF computation (the VLC in particular).

5.3. Related Work

Direct illumination Next event estimation has been studied extensively to estimate direct illumination contribution [140]. Further work has refined this to build hierarchical acceleration structures for the emitters [160] and used these to produce unbiased estimators [39]. This approach has been perfected in production renderers [28], extended with BRDF awareness [97] and refined for special cases like directional emission and volume scattering [46].

Another line of work to reduce the number of light sources to consider at every path vertex sorts light sources by contribution [162] and more recent work considers shorter, learned lists of lights [154, Keller's presentation]. Similar considerations lead to hierarchical structures [152, 151]. To gain real-time performance, the contribution of light sources has been evaluated analytically [65, 3, 68], and fast ways to sample a list of lights on the GPU have been devised [88, 158]. We use a combination of the above ideas, and propose to use a learned structure for next event estimation in conjunction with a *path guiding* method to improve indirect lighting contributions.

Path guiding These methods have been around for some time [77, 74] but only relatively recently gained some traction because improved algorithms [155, 116, 32] made path guiding useful in practical applications [154]. In these approaches, guide cache records are stored throughout the scene and are used to construct sampling densities that follow the incident radiance field. Ideally, these distributions also include the BSDF [71], but this comes at a performance penalty that is yet to be overcome for real time applications. We employ path guiding for indirect lighting, and the parts of the transport that are not handled by our next event estimation technique (non-geometric light sources such as an environment map).

Data structures Guiding caches store an approximation of a directional distribution, by means of Gaussian mixtures [155] or quad trees [116]. We use the quad tree approach, and use a special compression technique suitable for the GPU [75], though there are possible alternative encodings [41]. To store the cache records themselves in an index structure, we use a sparse octree. It has been shown that hierarchical hashed grids can be very effective on the GPU as well [14].

5.4. Visibility Light Cache for Next Event Estimation

Explicit sampling of light sources via NEE is a critical aspect for path tracers, since light sources can become very small and therefore only cover a very small solid angle (or none when handling point light sources) while still providing much contribution. This makes BSDF sampling ineffective, causing fireflies. Additionally, if light sources are many, we have to select one for sampling. This selection probability becomes quite important with increasing scene complexity. In particular, many light sources might not be visible, so selecting them would increase noise considerably. The limited path budget in real-time applications only aggravates this issue.

We therefore propose the Visibility Light Cache (VLC) which, for a given region, extracts set of visible lights from light samples in the previous frame. Figure 5.2 gives an overview of this process which is explained in more detail in section 5.4.1. From this set we select a light using high quality importance sampling with LTCs [68]. As we only consider visible light sources, we get closer to product importance sampling, but only under special circumstances; see section 5.4.2 for more details. As the VLC can only select lights that are already known, a secondary strategy is needed to explore new lights. This exploration strategy is presented in section 5.4.3.

5.4.1. VLC Construction

The construction algorithm takes light samples from the previous frame (figure 5.2 (a)) and extracts the set of visible lights to form the VLC used for importance sampling.

Deduplication (figure 5.2 (b)) is the primary task of extracting visible lights. A light sample consist of an identifier for the light, as well as a single bit that indicates whether the light was visible or not. A visible light can be represented by multiple visible samples, so we need to remove duplicates. We dispatch a compute shader where each work group handles one region. Samples are inserted into a hash table in shared memory with the light identifier as key. To handle collisions, we use a hash table with buckets (not shown in the figure). Invisible samples are discarded directly. We also tried to improve sampling by estimating fractional visibility from these samples, but this did not significantly improve the result in difficult situations. To bound computation time, only a fixed maximum number of the available samples is processed per region. To ensure representative selection, we gather samples all across the buffer.

We then merge the caches of neighbor regions (figure 5.2 (c)), to counteract that samples are not divided uniformly among the leaves. Due to our octree subdivision strategy (section 5.6) one leaf might receive almost all samples, while another leaf receives almost none. The former will just subdivide again to distribute the samples among more leaves, but the latter cannot compensate the low sample count. This is especially problematic with surfaces that are not exactly axis-aligned. As a result, the constructed VLC would be of poor quality (i.e. instability, as lights compete for few samples). Merging has the additional effect that newly discovered lights can propagate between neighboring caches. For occasions where the exploration strategy rarely samples an important light, this ensures that the light needs to be sampled by only one region, instead of by every region individually, leading to higher robustness.

Finally, the hash table is compacted (figure 5.2 (d)) to form the VLC which can then be used to sample a light source in the next frame (figure 5.2 (e)). This is described in more detail in section 5.4.2.

5.4.2. Importance Sampling of VLC

With the VLC, we want to perform high-quality selection of the contained area light sources. We leverage linearly transformed cosines (LTCs [68]) to compute sampling weights for each light. This technique was originally proposed for approximate shading with area light sources in closed-form. It can just as well be used for high-quality importance sampled selection of area lights with low variance, since the sampling weights are approximately proportional to the light contributions. Since only visible lights are considered, it gives almost perfect product importance sampled selection, assuming uniform emitters and full visibility of all lights to all contained surface points.

To sample a light source for a shading point, we construct a *cumulative distribution function* (CDF). As our importance measure incorporates information about the surface point itself, we cannot precompute the CDF for each region, but we instead have to do so on-demand for each surface point by linearly scanning through the lights. This, however, is very expensive even with modest light counts. Instead of iterating through all of them, the VLC is divided into a variable number of buckets with fixed capacity which the lights are distributed into. When selecting a light, a bucket can then first be chosen with uniform probability and only the lights contained in the single bucket are iterated over.

Algorithm 1: Branch-friendly VLC and Exploration sampling

Input: vlc, surface
Output: light, pdf

```
doExploration ← rng() < fraction;
if doExploration then
    light ← sampleExploration();                                   // $\mathcal{O}(1)$
    bucket ← light.idx mod vlc.bucketCount;                         // calc bucket
else // sample VLC
    light ← null;
    bucket ← $\lfloor$rng() · vlc.bucketCount$\rfloor$;             // sample bucket
// first VLC traversal to accumulate weights
acc ← 0, weight ← 0;
for light′ ∈ vlc.buckets[bucket] do
    weight′ ← calcLightImportance(surface, light′);
    acc ← acc + weight′;
    if light = light′ then // find light for explor.
        weight ← weight′;
if ¬doExploration then
    // second VLC traversal to select light
    light, weight ← sampleVLC(vlc, surface, bucket, acc);
pdf ← fraction · pdfExploration(light);                             // $\mathcal{O}(1)$
pdf ← pdf + (1 − fraction) · weight/(acc · bucketCount);
```

For PDF computation, we need to efficiently retrieve the bucket that a light is possibly contained in. For that, we distribute lights deterministically among the buckets via hashing. In our instance, we use a plain modulo operation against the light identifier. Obviously, with many lights, the sampling quality degrades considerably with this technique, but it ensures a fixed computation budget, which is paramount for interactive rendering. To keep register pressure low, we do not store the CDF explicitly but traverse the bucket twice (once to accumulate all sample weights for normalization, and then to select a light source).

The hashing method has the disadvantage that the buckets may not be filled evenly. As such, it is not clear how many buckets are needed, given just the number of lights. We therefore construct the VLC by taking the hash table from section 5.4.1 and merge buckets bottom-up, as long as the capacity limit of each bucket is not exceeded (figure 5.2 (d)). Additionally, the light count may vary slightly from frame to frame, due to almost unimportant lights being rarely sampled. Normally, this is not problematic. But if this small variation causes a variation in the number of buckets, flickering artifacts appear. To combat this, we use a hysteresis approach: When merging buckets to a bucket count lower than in the previous frame, the capacity is artificially reduced to, e.g., 75%. After the final buckets are constructed, we sort the lights inside a bucket by their identifier to improve sampling stability.

5.4.3. Exploration Strategy

The VLC is only capable of selecting lights which were already selected in the previous frame. As such, new or yet unknown lights are never selected, resulting in a biased technique. A secondary "exploration" technique is needed which we combine with the VLC through MIS. The basic requirement for this technique is to be cheap to evaluate, ideally in constant time, since we already put a lot of computational effort into sampling the VLC with high quality. The sampling quality does not need to be especially good either, since we combine with the VLC right away. We thus opted for a constant-time sampling technique by precomputing a global PDF on the CPU over all light sources using the alias method [158]. This is done every frame. The major problem of this approach is that the relationship between surface point and light source (e.g. distance, relative angles) is completely lost. We therefore use the following heuristic weights:

1. Energy of the light
2. Change of the light (e.g. energy and/or position)
3. Distance from light to camera

(1) gives higher weight to brighter lights. (2) gives higher weight to dynamic and/or appearing lights for improved reactivity. (3) is used as a cheap proxy for the distance between light and surface point. This is obviously only reasonable for points that are close to the camera. We compute separate PDFs for each heuristic. They are then blended with configurable weights and normalized to form the final PDF given as input to the alias method.

At each surface interaction, a stochastic decision is made to sample either the VLC or exploration technique through a configurable fraction. In most instances, no more than 10% of samples should be generated by the exploration technique, because the sampling quality is typically vastly inferior to the VLC, resulting in increased noise.

Since we combine both techniques through MIS, we need to efficiently compute the PDF of the other technique. For the exploration strategy, this requires a traversal through the bucket in which the sampled light might reside. Modern GPUs execute groups of threads in lockstep, but the technique is selected with a random number. Therefore, virtually all groups have to execute both techniques. Thus, a naive implementation through separate code paths for VLC and exploration sampling would effectively cause three traversals of the VLC. But VLC traversal is the most expensive aspect, as it is computationally linear in bucket size. We resolve this issue by performing combined sampling of both techniques as shown in algorithm 1. The general idea is that we hijack the first traversal of the VLC, so that threads that have elected to use the exploration strategy use this traversal to also find the sampled light source in the VLC and write out the respective weight. Together with the accumulated weights, the PDF of the VLC can be computed. Just as with plain VLC sampling, two traversals are needed.

5.5. Compressed Directional Quadtree for Path Guiding

Guiding of indirect rays promises great variance reduction, especially if features with high contribution cover a small solid angle. But most techniques are exclusive to offline rendering due to needed precomputation and a large computational and memory overhead. We base our work on [116] to achieve real-time path guiding. Directions are mapped to 2D through cylindrical coordinates. In this space, a directional quadtree structure is used for importance sampling which approximates the radiance field of a region. Regions are also embedded in the leaves of an octree structure over all surface points.

The problems for adopting this technique into a real-time context as-is are twofold: First, the technique does a lot of precomputation by drawing many samples just to build this quadtree structure. In fact, this is done iteratively with geometrically growing sample counts, such that the paths of these "learning"

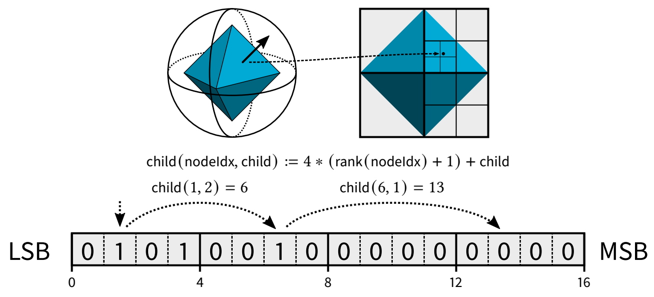


Figure 5.3.: Mapping of Sphere onto octahedron space, topology of quadtree as represented by a bit field and traversal of quadtree for a given direction. Quadtree is stored breadth first with a single bit per node. Rank operations are used to locate children of nodes. Figure taken from [35].

samples are guided as well. This approach is not feasible in real-time. Second, the octree occupies a large and varying amount of memory. Each node contains pointers to its immediate children as well as stochastic weights to guide traversal to important leaves. We already have to pay the cost of traversing the octree structure, so this can become prohibitively expensive.

We approach the memory problem by storing the quadtree in implicit form using a rank & select data structure [75] with few bits (on the order of 32 - 128 bits). We use octahedron mapping [45] to map directions to 2D. Figure 5.3 shows an example instance. The tree is stored in breadth first order with each node being represented by a single bit. It denotes whether the node in question is an inner (1) or leaf node (0). We assume the root node to always be an inner node (a single leaf is not very useful for guiding). It is therefore always zero and does not need to be stored explicitly. This ensures that for bit field sizes which are a multiple of four (e.g. a 32-bit integer), all bits can actually be used. Rank & select operations are used to traverse the tree in both directions (see figures 5.3 and 5.5). As for the sampling weights, we simply omit them, and each leaf is given a uniform sampling weight instead. As such, a region is given higher sampling weight just by refining the quadtree towards this region. Of course, the resulting quadtree will be inferior to the original, but it will still improve sampling in difficult situations so that denoising can reconstruct the image without striking artifacts. Adaptation and sampling of this data structure are described in section 5.5.1 and 5.5.2, respectively.

5.5.1. CDQ Adaptation

We leverage temporal coherence to adapt a given quadtree with samples from the previous frame. We assume a fixed allocation of nodes (the maximum number of nodes that fit in the bit field). Adaptation happens by collapsing a selected inner node with only leaves as children and splitting a selected leaf node, i.e. the leaves are migrated between the selected nodes. As such the number of nodes always stays the same. We implement this adaptation directly on the bit field as depicted in figure 5.4, so that no uncompressed intermediate representation of the tree is needed. Essentially, we use two partial bit shifts to remove the old and to insert the new leaves in their respective location.

The nodes are selected based on the samples of the last frame. The samples are composed of their contribution and the direction (mapped into 2D by the octahedron map). To avoid adaptation towards light sources (they are already handled by NEE), we ignore light samples entirely. We run a compute shader with work groups consisting of a single subgroup (the collection of threads that execute in lockstep on the GPU). Each work group is responsible for one region. The samples are distributed among the leaves based on their respective direction and the squared contributions are accumulated in shared memory. We aggregate leaf contributions of the currently processed samples in the subgroup

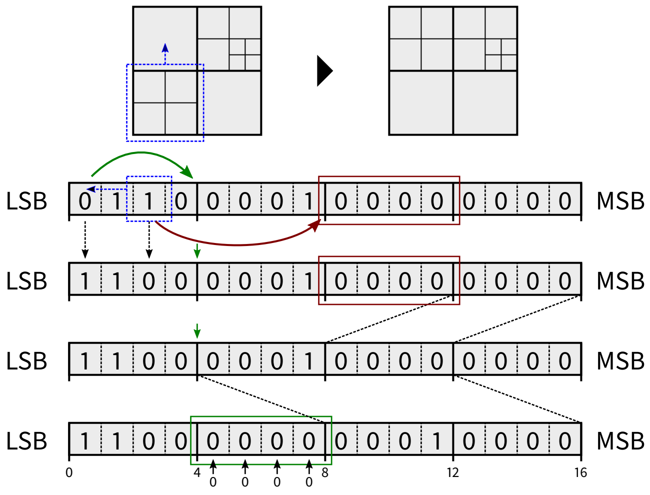


Figure 5.4.: CDQ adaption by leaf migration and implementation on bit field. First, children of selected nodes are computed. Then, bits of selected nodes are flipped. Deletion of old and insertion of new leaves is done by shifting all following bits. Finally, inserted leaves are zeroed. Figure taken from [35].

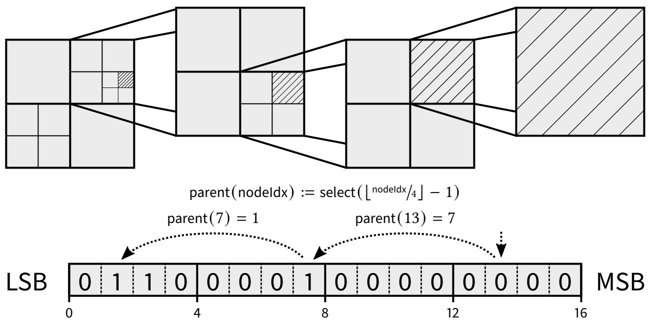


Figure 5.5.: Reconstruction of bounding box for a selected leaf. Successive select operations are used to rescale the bounding box into the parent frame until the root node is encountered. Figure taken from [35].

first. One thread per leaf is selected to accumulate the value in shared memory to avoid costly atomic float operations. We use squared contributions to have the CDQ adapt towards high-variance as well as high-energy regions (see [151] eq. 5). We then run a min-max search: We select the leaf node with maximum contribution and the inner node with minimum combined contribution of its leaf children. Leaf migration is done only if the contribution of the leaf is higher than the combined contribution of the inner node. For stability, a configurable factor (i.e. 0.1) on the leaf contribution is used such that the leaf contribution must be significantly higher in order to warrant migration.

5.5.2. Importance Sampling of CDQ

We do not explicitly store any additional weights for stochastic CDQ traversal. But traversing all children with uniform probability would result in overall uniform sampling. We instead approach this problem in a bottom-up fashion: We select a leaf node with uniform probability and then reconstruct its bounds by traversing its parents. Thus, a region in the quadtree that is more refined will be sampled more often, simply because more leaves are present in that region. The number of leaves is constant and the bit of leaf i can be computed through select(i) on the inverted bit field. Reconstruction is depicted in figure 5.5. With each traversed parent, the bounding box of the leaf is rescaled into the parent frame, until we arrive at the root node. The PDF is given as:

$$\text{PDF} = \frac{1}{A_{\text{leaf}}|\text{leaves}|}\frac{\|q\|_2^3}{4}. \tag{5.1}$$

The first term is the probability of sampling a point on the octahedron map. A_{leaf} is the area of the containing leaf node and |leaves| is the total number of leaves. The second term is the correction factor which maps our PDF from the octahedron map to solid angle. q is a 3D vector obtained by mapping the sampled direction vector d onto the octahedron ($q = d/(|d_x| + |d_y| + |d_z|)$). To compute the PDF for a given direction (e.g. for MIS), we only need to access the bit field to compute A_{leaf} of the containing leaf node. This is achieved through a top-down traversal with which the bounding box is computed.

5.6. Temporal Sample-adaptive Octree

Visibility of lights as well as the direction of incoming contribution greatly depend on which surface point is considered. This makes the VLC and CDQ greatly depend on local information. We therefore cannot construct one global estimator for each technique, but instead have to do so for many regions in the scene. We need to organize these regions in a data structure with the following considerations: First, there must be enough samples available per region in order to construct stable estimators. Second, regions must be adapted under dynamic conditions in order to uphold the first consideration. Third, it must be accessible for both direct and indirect surface hits, so that the latter also benefits from the estimators.

We opted for an octree structure situated in world-space, where each leaf node defines a region for our estimators. The layout of this data structure in memory and the design decision leading to it are given in section 5.6.1. Based on the collected per-leaf sample counts from the previous frame, the current octree is adapted through a simple split/collapse scheme as presented in section 5.6.2. Given a surface point, the octree is traversed top-down in order to access the corresponding estimators. Details are given in section 5.6.3.

5.6.1. Memory Layout

We optimized the memory layout such that most of the traversal only touches a small memory region (figure 5.6). We use two separate buffers for storing inner and leaf nodes. During traversal, only the first buffer needs to be considered until a leaf node is encountered. An inner node is composed of an 8-bit mask to encode which children are leaves (children are implicitly mapped to bits) and a 24-bit offset for the location of inner children. The inner children are stored at this location in a compacted way, i.e. there are no gaps if the current node also contains leaf children. To avoid storing another offset, leaves are indexed implicitly with the parent node index. While the node buffer remains small this way, this creates holes in the leaf buffer where the corresponding node is actually an inner node, ultimately leading to wasted space. But for a full tree with N children (each node has either zero or N children), given the number of inner nodes I, the total number of nodes is $N \cdot I + 1$. The wasted space, which is the ratio of inner nodes to the total number of nodes, is $I/(N \cdot I + 1)$. It converges to $1/N$ in the limit. As such, at most $1/8$ of storage is wasted in the leaf buffer. But overall, this waste is not the prime factor: Since the octree organizes surface samples, around half of all leaf nodes are empty anyway. We therefore focused on reducing the size of the inner node buffer, as it is accessed many

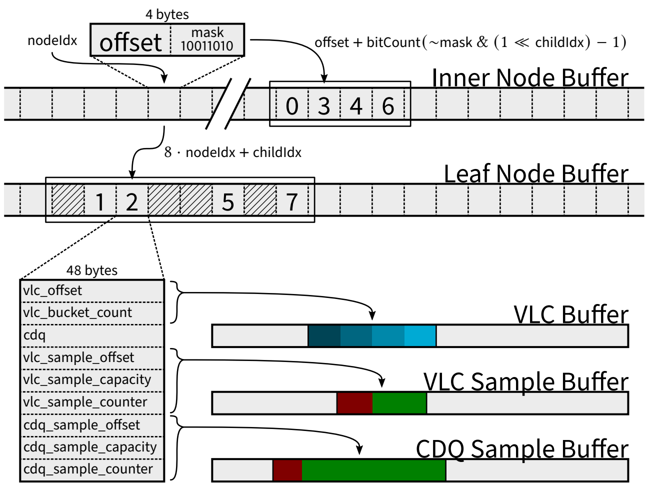


Figure 5.6.: Octree memory layout. We use two separate buffers for inner and leaf nodes. Inner nodes are composed of a 24-bit offset for the inner children and an 8-bit mask to encode leaves. Leaves are indexed implicitly instead of storing another offset per node. Inner children are stored compacted at the offset. Additional buffers (parent, neighbor and correspondence information) are not shown. Figure taken from [35].

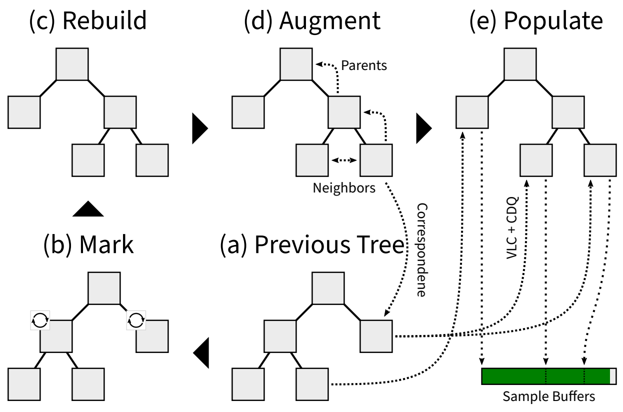


Figure 5.7.: Octree construction steps (visualized as binary tree) based on (a) previous tree. (b) Nodes are marked if they need to be split or collapsed based on the observed sample count. (c) The topology is rebuilt to account for marked nodes. (d) An augmentation pass adds parents to each node as well as neighbors and correspondence (equivalent node in previous tree) to each leaf. (e) This information is later used to populate the leaf structures (copy VLC pointers and CDQ, allocate space in sample buffers). Figure taken from [35].

times per traversal, while the leaf node buffer is accessed only once at the very end, thus giving the biggest potential in cache utilization.

Leaves store all necessary information for importance sampling and sample placement, namely offset and bucket count for the VLC buffer, the CDQ itself and offset, capacity and size triplets for the VLC and CDQ sample buffers. This amounts to a size of 48 bytes. We store leaves in an Array of Structures (AoS) format. We strongly recommend moving to a Structure of Arrays (SoA) format, as often only a single member (i.e. the CDQ) is needed for a particular purpose. Due to time constraints, we did not perform this rearrangement in our implementation.

5.6.2. Octree Adaptation

The octree needs to be continuously adapted in dynamic situations in order to ensure that each leaf node receives an adequate amount of samples (i.e. not too few and not too many). By observing the number of samples that each leaf received in the last frame, an adapted octree is constructed by splitting and/or collapsing nodes. The rough process is depicted in figure 5.7. It consists of four passes: Mark, rebuild, augment and populate. Mark decides which nodes need to be split and/or collapsed, rebuild constructs the new topology, augment adds additional information to nodes and leaves. Finally, the leaf structures are populated.

The mark step (figure 5.7 (b)) is responsible to decide for whether a node needs to be toggled, i.e. split or collapsed. Only leaf nodes can be split and only inner nodes with only leaves as children can be collapsed. A leaf node is marked for splitting if the observed sample count goes above a configurable upper threshold (i.e. 2048). An inner node is marked for collapsing if the combined sample count over all leaves goes below a configurable lower threshold.

The rebuild step (figure 5.7 (c)) creates the adapted topology based on the results of the mark step. A breadth first construction scheme is used, so we only construct one level in parallel at a time. This would require many dispatches, so we leverage shared memory in compute shaders to construct entire subtrees per work group. This is the reason why we defer augmentation of the tree and population of leaves to later passes: These would need to be stored in shared memory as well, decreasing the effective size of the tree that can be constructed in one work group. Also, once the topology is set, these steps can be done in parallel over all nodes, instead of just in parallel over each level.

The augmentation step (figure 5.7 (d)) adds additional information to the nodes and leaves of the octree. The first dispatch augments nodes by adding parent pointers to each child in a secondary buffer. The second dispatch augments leaves. This includes neighbor information as well as corresponding nodes in the previous tree. Parent pointers are used to reconstruct the position of a leaf in the tree. This position is used to traverse the previous tree for correspondence as well as to traverse an offset position in the new tree for the six neighbors with a common face. If one side has a finer subdivision, then a potentially unbounded number of neighbors exist on that side. We currently ignore this case for simplicity.

Leaves are populated (figure 5.7 (e)) by copying the CDQ and pointers to the VLC from the previous tree (using correspondence information) and by allocating space for samples based on the observed sample count. The copy operation is needed because otherwise two trees would need to be traversed: The previous tree for sampling the VLC and CDQ and the current tree for placing samples. But these two trees are almost identical (they only possibly differ in some split or collapsed nodes). The copy operation is trivial for unchanged and split nodes, because they uniquely map to a single node in the previous tree. A collapsed node has eight potential leaves from which the VLC and CDQ may be copied from. We simply copy from the node which had the biggest observed sample count. Another possibility is to have VLC and CDQ construction be aware of collapsed nodes, but this adds further complexity.

5.6.3. Reprojection & Combined Traversal

Given a surface point, the octree is traversed top-down in order to access the containing leaf node and the corresponding estimator as well as pointers to the sample storage. If the surface point is dynamic, however, the leaf in which it is currently contained might be empty, i.e. the VLC is empty and the CDQ is in its initial configuration. This would result in visible artifacts. We thus have to traverse the octree twice: For sampling, we use the previous position. For placing samples, the current position is used. The previous position is typically readily available, as correct motion vectors for, e.g. TAA, need to know the exact location of surfaces in the previous frame. However, we do not want to pay the cost of two traversals (at least not every time). If these two positions are identical, or even almost identical, most of the traversal is redundant. We therefore perform a combined traversal until the paths diverge.

5.7. Evaluation

We use a modified version of Quake 2 RTX as our evaluation framework. A standard path tracer with NEE is used. We trace two indirect rays, but only do NEE for the primary and secondary hit. From the tertiary hit we only collect direct emission (e.g. for environment maps). In total, five rays per path are traced. After the primary hit, we perform two simplifications: First, all lights are treated as uniform emitters. As we focus on just the selection of lights and not sampling of lights themselves, we do not want the additional noise that comes from sampling textured emitters. Lights are sampled with plain area sampling. Second, all surfaces are assumed to be fully diffuse after the primary hit. Otherwise, our reference renders were left with fireflies even after many thousands of samples. Unless otherwise noted, the octree is configured with a lower threshold of 2048 and an upper threshold of 4096, the VLC with a bucket size of 32 and an exploration fraction of 10% and the CDQ with a bit count of 128 and a BRDF fraction of 50%. All images are rendered with one sample per pixel.

We compare the VLC against two other techniques. The first, we refer to it as Static Light Lists (SLL), is the NEE implementation that shipped with Quake 2 RTX. It uses potentially visible sets (PVS) [145] derived from the BSP structure [47] to collect potentially visible lights for each BSP leaf. Based on the leaf in which a light is contained, the light is distributed among the leaves that are in the PVS of the leaf. BSP and PVS construction are inherently static due to expensive computation. As such, only static scene elements can affect visibility. However, dynamic moving lights are possible, as finding the containing leaf and corresponding PVS is very fast. Just as with the VLC, this sampling procedure is linear in the number of lights. To bound the computation time, the lights are traversed with a variable stride and uniformly random initial offset such that only a maximum number of lights are considered. We configured this maximum to be 32, in line with the VLC bucket size. We augmented this technique with our LTC-based importance measure, so that the techniques mainly differ in how precisely they capture the set of visible lights.

The other technique is the Light Hierarchy (LH) as proposed by [28] and [113] but without splitting. A Bounding Volume Hierarchy (BVH) is constructed over all lights with each leaf containing a subset of the lights. This hierarchy is then traversed stochastically by evaluating an importance measure per child node which is used as a probabilistic weight to descend this child. We compressed and widened the structure in the spirit of [167] to improve traversal speed and sampling quality (flattening the hierarchy can be seen as brute-force splitting, albeit not considering the surface point). Construction of the hierarchy is done each frame using standard binary binning [157] and a final top-down traversal to collapse nodes. Thus, dynamic lights are handled transparently. We configured the width to be 8 children per node and the number of lights in each leaf node to be 2.

We compare the CDQ against plain BRDF sampling. To the best of our knowledge, there currently exists no other guiding technique capable to execute at interactive rates.

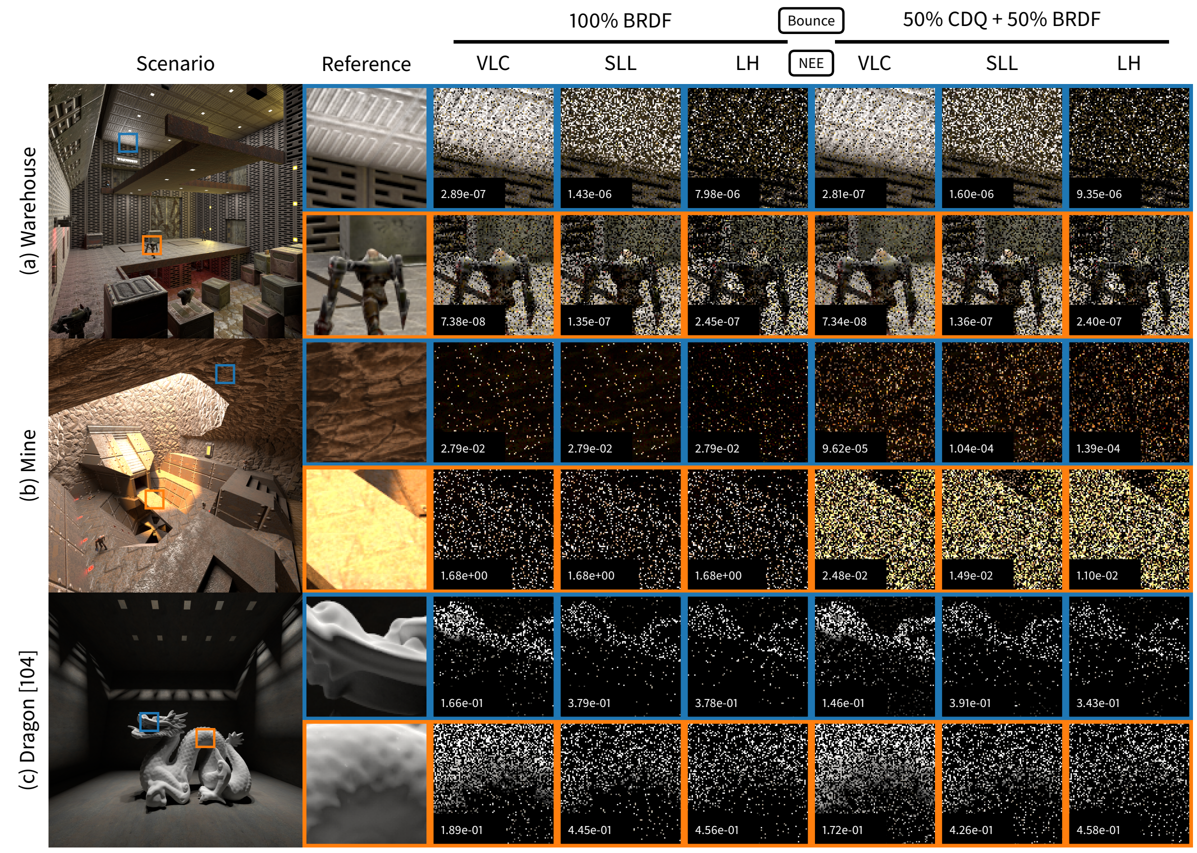


Figure 5.8.: Comparison of techniques with 1ssp frames in test scenes. Scenes (a) and (b) taken from Quake 2 RTX, © id Software, Inc. VLC consistently outperforms the other techniques in (a). Due to uniform illumination, CDQ provides no real benefit. CDQ is most useful with high contribution from a specific direction as with the sun in (b), improving both direct and indirect illumination. With very complex direct illumination as in (c) (area lights behind a cover), both techniques still improve sampling, but not by that much. Figure taken from [35].

5.7.1. Quality Comparison

We compare the image quality (quantified with MSE) of our presented techniques against the SLL, LH and plain BRDF sampling. For that we instantiate all combinations of NEE and indirect sampling techniques (6 in total). We use scenes from Quake 2 RTX as well as scenes with more complex geometry as shown in figure 5.8.

Scene (a) shows a warehouse with relatively many light sources and mostly uniform illumination. The full scene is considerably larger, containing even more lights. By focusing only on relevant lights, the VLC excels here, achieving almost on order of magnitude lower error compared to the SLL and LH. The LH is worst in this instance. We assume that due to missing visibility information, lights outside the room also receive a nonzero sampling weight. The CDQ does not provide any meaningful advantage here. But this is to be expected due to overall uniform illumination.

Scene (b) shows a mine which is open towards the sky. It is illuminated mostly by the environment map and the contained sun. Especially indirect illumination is affected by the spot which the sun directly illuminates. Our NEE implementation does not account for environment maps at all, rendering the VLC useless. With plain BRDF sampling, both directly and indirectly illuminated surfaces by the

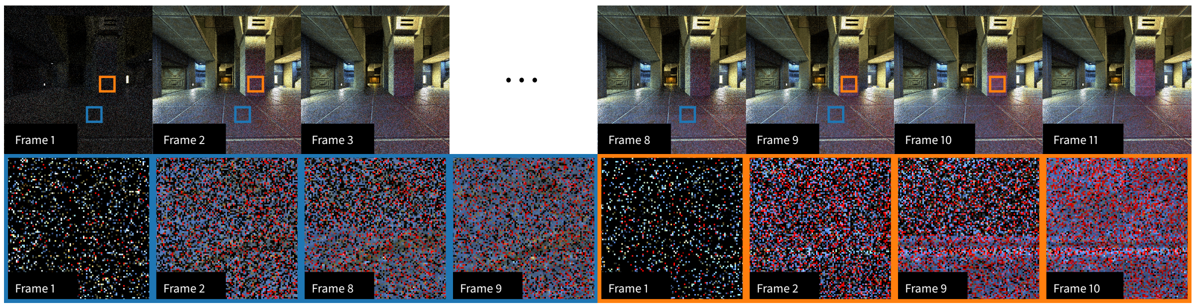


Figure 5.9.: VLC and octree adaptation over a frame sequence. Initially, only exploration sampling is used due to an empty VLC. Sampling is already improved after a single frame, but octree adaptation creates specialized VLCs with even better sampling in the subsequent frames. Figure taken from [35].

sun are severely undersampled. By refining towards the sun and the directly illuminated spot, the CDQ improves sampling by two to three orders of magnitude for these regions.

Scene (c) shows the Stanford dragon model illuminated by ten area lights behind a cover. This is a relatively complex situation as lighting is blocked by other geometry and the receiver is relatively complex. All techniques produce noisy results, but the VLC with the CDQ is still slightly better.

5.7.2. VLC & Octree Adaptation

In this section, we examine the temporal behavior of the VLC and octree. We are particularly interested in how long it takes for these techniques to fully adapt to a given situation from an uninitialized state (the worst case). We go over the frame sequence that is depicted in figure 5.9.

In the *initial state*, the octree is subdivided only once (because we cannot represent a single leaf node) and the VLC of each of the eight leaves is empty. As such, exploration sampling is used exclusively here. This leads to a very noisy image that lacks any detail of the scene. In the *second frame*, we directly see the VLC taking effect. If the octree were not to subdivide any further, the VLC would already be in a converged state. But this subdivision is still very rough with all the samples being deposited in a single leaf, leading to an imprecise VLC. This explains why there are still so many samples with zero contribution. There is no change in sampling until *frame eight*. The problem is that the octree has quite a large extent as it needs to cover the entire scene. The shown room is only a small part of it. As such, the octree needs multiple frames in order to subdivide towards this region. At frame eight, subdivision is refined enough so that we see first improvements at the floor, which continue with frame 9. The number of samples with zero contribution is now closer to the 10% exploration fraction. Up until frame 11, refinement continues with the pillar.

5.7.3. CDQ Adaptation

The main parameter of the CDQ is the number of bits with which the tree is encoded. We examine adaptation of the CDQ with different bit counts and rates of change. For this, we construct a scene which consists of a floor, a circular wall surrounding this floor and a rotatable light below the floor. The floor is illuminated indirectly by the light via the wall. The camera is pointed directly at the floor. We let the CDQ adapt to an initial position of the light. Then we perform a sudden, defined, rotational change of the light over one frame. We observe how long it takes for the CDQ to readapt to this new situation

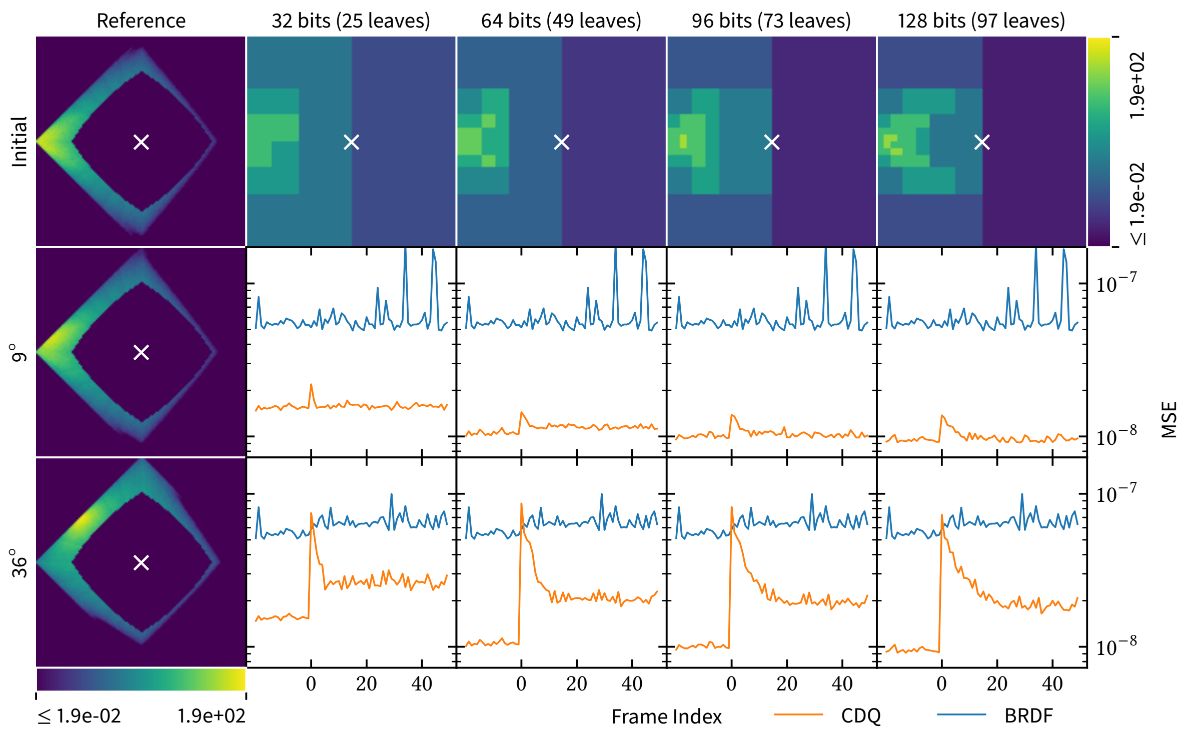


Figure 5.10.: CDQ static and dynamic adaptation through a rotatable light indirectly illuminating a surface via a circular wall. The left column visualizes the optimal sampling densities for the radiance field. The top row visualizes the CDQ sampling densities adapted to the initial configuration at different bit counts. The following rows show adaptation over frames quantified via MSE after a sudden rotation of the light source at different angles. A logarithmic Viridis color mapping with the same range across all images is used. Figure taken from [35].

and how much worse the sampling quality initially is after the change compared to the readapted state.

In figure 5.10, we compare bit counts of 32, 64, 96 and 128 bits with rotational changes of 9 and 36 degrees. We also provide visualizations of the optimal sampling densities of the radiance fields (in the 2D octahedron map) and CDQ sampling densities adapted to the initial configuration. The plots show MSE of the CDQ over a sequence of frames. At frame zero, the change is performed. All variations exhibit fast convergence, but overall adaptation greatly depends on both parameters. At nine degrees, the change is not all that big. Neither the peak error nor time to readapt are very large for all bit counts. The 128 bit variant still takes longest at around 10 frames. Both aspects deteriorate at an increasing angle. At 36 degrees, the CDQ is initially worse than plain BRDF sampling. Since we still use the BRDF 50% of the time, it effectively bounds the error of the CDQ. This is crucial in this instance, as the previous adaptation no longer has any meaningful relation to the new situation. While the readapted CDQ still provides much improvement, the error is higher than in the initial configuration. We assume that the rigid subdivision scheme of the quadtree cannot adapt to the new configuration as well as to the initial configuration. Adaptation time has increased to 20 frames for the 128 bit variant.

Another observation is that adding more than 64 bits does not improve variance any significantly. We assume that this is mostly due to the radiance field being very smooth (along the wall). It is thus approximated relatively well already by the 64 bit instance. More bits can still be advantageous for, e.g. multi-modal irradiance, but we did not explicitly evaluate this aspect.

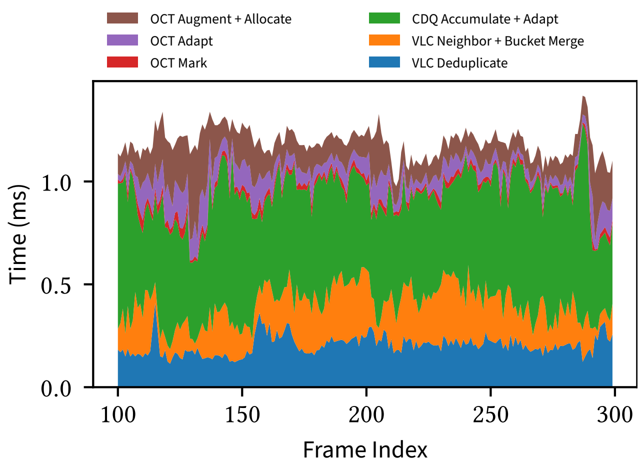


Figure 5.11.: Execution time of preprocessing operations over multiple frames (average of 200 runs). On average, around 1.2 ms are needed. Figure taken from [35].

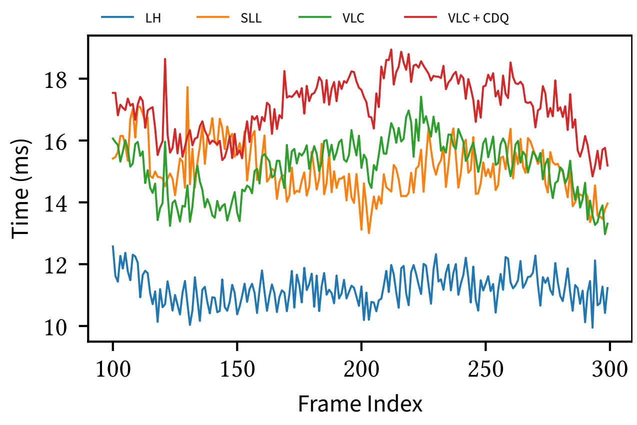


Figure 5.12.: Execution time of path tracer over multiple frames with different importance sampling strategies. On average, LH is fastest (11.2 ms) and VLC + CDQ is slowest (17.1 ms). Figure taken from [35].

5.7.4. Performance

For our performance evaluation, we use an Nvidia RTX 2080 Ti at a resolution of 1920 × 1080. A walk around the Quake 2 RTX scene "base1" is used as the reference scenario. A video of this scenario is attached in the supplemental material (frames 100 to 300). We consider both the required preprocessing and impact on the path tracer for the presented techniques.

Figure 5.11 shows the averaged timings of all preprocessing over 200 runs. In total, around 1.2 ms are needed. The entirety of all octree operations (mark, adapt, augment and allocate) take around 0.2 ms. While this timing is relatively fast, the octree is not large to begin with. This probably means that the GPU is not saturated during this time. Deduplication of the VLC takes 0.2 ms, while neighbor and bucket merging take around 0.2 ms. The CDQ accumulation and adaption takes around 0.5 ms. Both CDQ accumulation and VLC deduplication iterate over their respective sample sets, but the latter takes longer due to more complex processing: For each sample, the given quadtree leaf needs to computed, while for the VLC only few hash table accesses are needed.

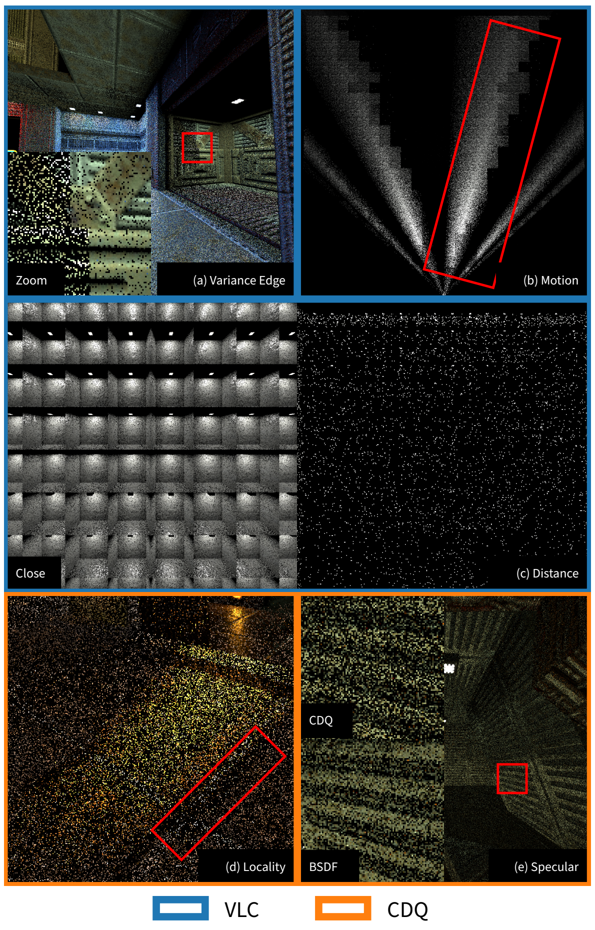


Figure 5.13.: Various artifacts encountered with the VLC and CDQ. (a) Variance edge due to varying number of lights in adjacent VLCs. (b) Fast moving shadows caused by lights traveling in front of a slit. (c) Heavily increased noise with larger distance from scene caused by rough octree subdivision (exaggerated by reducing VLC bucket size to 8 instead of 32). (d) Increased noise at the edge of a spot directly illuminated by the sun due to positionally dependent occlusion inside of octree leaf. (e) Increased noise when using CDQ on specular surfaces. Figure taken from [35].

The impact on the path tracer is shown in figure 5.12. We compare the impact against LH and SLL, taking around 11.2 ms and 15.1 ms, respectively. VLC is almost identical to SLL with 15.1 ms. Since both SLL and VLC are configured with the same bucket size of 32, this indicates that light selection is the limiting factor in this instance, and not octree traversal. Adding CDQ to VLC increases path tracing time to 17.1 ms.

5.8. Limitations & Future Work

While we have shown that both the VLC and CDQ are promising techniques for real-time path tracing, many technical and implementation problems remain that need to be addressed in the future.

The most fundamental problem is that there is an inherent delay of one frame when using samples from the previous frame. This can become problematic with very dynamic scenarios (e.g. strobing or fast-moving lights, see figure 5.13 (b)). A possible solution would be to use samples from the current frame as well, e.g., by running the path tracer in two phases (one for exploration and one for sampling).

Limitations associated to the temporal sample-adaptive octree are: (1) Oftentimes, the resolution is not enough to minimize variation within a leaf node (figure 5.13 (d)), e.g., due to positionally changing occlusion of lights and highlights. As resolution primarily depends on camera distance, this is especially problematic with distant geometry (figure 5.13 (c)). (2) We do not perform any interpolation between octree leaves. For the VLC, this is mostly not needed, as the point-specific knowledge introduced with the importance measure automatically smoothens the transition between regions. However, variations in contained lights and bucket count can be seen as variance edges (figure 5.13 (a)). This problem is even more pronounced with the CDQ, as it represents just a single CDF per leaf. (3) Large buffers are needed to store intermediate samples for VLC construction and CDQ adaptation. By directly deduplicating visible lights for the VLC and accumulating leaf contributions of the CDQ, memory consumption could be reduced significantly.

Limitations associated to the VLC are: (1) Scalability is limited, as it handles individual triangular area lights. This is an inherent disadvantage compared to, e.g., light hierarchies. While the bucket strategy always ensures a bounded computation time even with many lights, sampling quality deteriorates significantly. Selection on entire aggregates of lights (e.g. mesh lights) and/or a more informed distribution of lights among buckets might resolve this issue. (2) Lights with rare visible samples (e.g. due to partial occlusion) potentially need far more than a thousand samples per leaf to be stably part of the VLC. Persisting the VLC over multiple frames could be an option to increase the effective sample count. (3) The current global exploration strategy reverts to questionable heuristics to account for the missing relationship between surface point and light. Local exploration strategies per leaf, e.g. through well-known clustering techniques [122], could resolve this issue. (4) Both the VLC and exploration strategy are sampled at a fixed fraction. Exploring approaches to dynamically determine them seems worthwhile. In particular, if no new lights can be discovered, solely the VLC could be sampled.

Limitations associated to the CDQ are: (1) It does not account for the BRDF, giving higher noise to specular surfaces at the 50% default BRDF/CDQ split (figure 5.13 (e)). (2) It is not stratified (one random number for sampling a leaf plus two for the area contained by the leaf), making blue noise sampling much less effective. A top-down traversal with sample-warping of two random numbers would be needed. (3) Adaptation speed is relatively slow because only a single leaf migration is performed per frame. Obviously, migrating multiple leafs at a time could drastically improve adaptation speed.

5.9. Conclusion

In this chapter we have presented importance sampling techniques for next event estimation (VLC) and path guiding (CDQ) which construct improved sampling densities through sample reuse. With implicit consideration of visibility and high-quality importance sampling, the VLC is an important step towards full product importance sampling of direct illumination with many lights. By compressing a directional quadtree structure down to a few bits, the CDQ is a path guiding scheme that is tractable for real-time use. Both techniques are embedded in a temporally adapted octree structure situated in world-space which balances the available samples among leaves.

We have shown that both techniques provide significant variance reduction in various test scenes (section 5.7.1), while also adapting to entirely new situations in few frames (section 5.7.2 and 5.7.3). Performance for both techniques is within close reach for practical use (section 5.7.4). Still, we further wish to explore this field in order to free the techniques from the remaining artifacts (section 5.8).

6. Markov Chain Mixture Models for Real-Time Direct Illumination

This chapter is based on A. Dittebrandt, V. Schüßler, J. Hanika, S. Herholz, and C. Dachsbacher. "Markov Chain Mixture Models for Real-Time Direct Illumination". In: *Computer Graphics Forum* 42.4 (2023). DOI: `10.1111/cgf.14881`

We present a novel technique to efficiently render complex direct illumination in real-time. It is based on a spatio-temporal randomized mixture model of von Mises-Fisher (vMF) distributions in screen space. For every pixel we determine the vMF distribution to sample from using a Markov chain process which is targeted to capture important features of the integrand. By this we avoid the storage overhead of finite-component deterministic mixture models, for which, in addition, determining the optimal component count is challenging. We use stochastic multiple importance sampling (SMIS) to be independent of the equilibrium distribution of our Markov chain process, since it cancels out in the estimator. Further, we use the same sample to advance the Markov chain and to construct the SMIS estimator and local Markov chain state permutations avoid the resulting bias due to dependent sampling. As a consequence we require one ray per sample and pixel only. We evaluate our technique using implementations in a research renderer as well as a classic game engine with highly dynamic content. Our results show that it is efficient and quickly readapts to dynamic conditions. We compare to spatio-temporal resampling (ReSTIR), which can suffer from correlation artifacts due to its non-adapting candidate distributions that can deviate strongly from the integrand. While we focus on direct illumination, our approach is more widely applicable, and we exemplarily show the rendering of caustics.

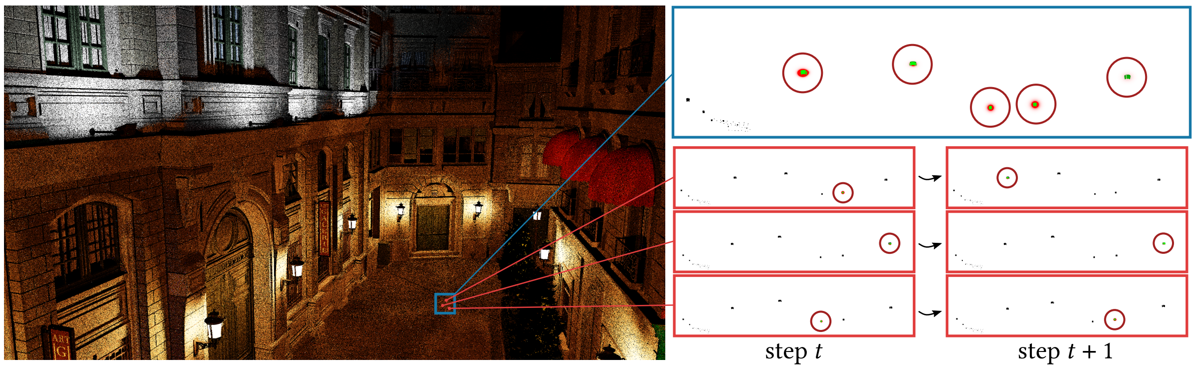


Figure 6.1.: Left: Direct illumination from many small light sources rendered using our mixture model with 8spp (10.8ms at 1080p), without explicit light source sampling. Right: a visualization of the mixture model in latitude/longitude projection. Red insets: At any time step, every pixel holds only a single vMF lobe (red), aimed at one of the light sources (black), the overlap is shown in green. Through a Markov chain process, lobes are adapted and exchanged locally. Blue inset: We can observe the mixture model as different slices of this construction: spatially over an image region, as well as temporally over multiple time steps. For better visibility, we surround lobes with a dark red circle. Figure taken from [36].

6.1. Introduction

The simulation of light transport in a scene is fundamental for photorealistic rendering and most often done using Monte Carlo integration [128]. It has found broad adoption in the visual effects industry [46] and due to the availability of fast ray tracing cores [120] and noise reduction filters [136] it is also becoming increasingly relevant for real-time applications with dynamic environments such as video games. This is boosted by recent improvements of direct illumination resampling [16].

Our work also primarily targets the sampling of direct illumination. To arrive at a general technique that can, in the future, be extended more easily to indirect illumination, we do not perform next-event estimation but use only hemispherical samples to discover lights. In the spirit of recent path guiding approaches [155], we use parametric mixture models to represent and sample a target distribution (e.g., the incident radiance field). In contrast to resampling schemes, our approach creates new samples by drawing from a continuous distribution instead of selecting from a fixed, previously discovered set of samples. However, to represent the target distribution sufficiently, the number of required mixture components can be large, and since it is unknown a priori, additional statistics need to be stored to refine or merge components [134] adaptively. Due to this storage and evaluation overhead, realizing an efficient guiding solution using discrete mixture models is challenging, especially if we want to store one mixture model per pixel. Our technique combines the simplicity and GPU-friendly fixed memory footprint of a single component per pixel [34] with the expressiveness of a mixture of components by replacing a deterministic mixture with the equilibrium distribution of a Markov chain (fig. 6.1), whose state space defines the parameter-tuple of a component. In detail, we use mixtures based on von Mises-Fisher (vMF) lobes. The Markov chains transition by exchanging the vMF lobe parameters between pixels, and adapting the mean and concentration parameters using a maximum likelihood estimate based on the history of samples. Large steps are facilitated by injecting independent hemisphere samples. Since each pixel hosts their own lobe, we adapt this data model after every sample without synchronization overhead.

Our contributions are:

- **A theoretical foundation of a Markov chain-controlled randomized vMF mixture model.** We use this foundation to sample the parameter space which defines vMF lobes approximating the incident radiance times material evaluation.
- **An efficient unbiased estimator based on stochastic multiple importance sampling (SMIS).** We are, thereby, leveraging the fact that, when using stochastic MIS, the equilibrium distribution cancels out from the estimator, and thus it is not required to know or compute the equilibrium distribution of the Markov chain process. This leaves us ample freedom when designing appropriate transition strategies to balance temporal adaptation in dynamic scenes and accurate representation of the incident light field.
- **An efficient way of reusing ray traced samples between the Markov chain and the estimator.** This allows us to build a light-weight, real-time path guiding technique for direct illumination, which is able to track dynamic lights, and an exemplary extension to underwater caustics.

We analyze the behavior of our method in simple experiments and demonstrate its performance in two open-source rendering systems, Nvidia's Falcor [80] and the Quakespasm engine to evaluate highly dynamic environments. Since the Markov chain process bears a resemblance to particle filters, it is well suited to track dynamic light sources moving at moderate speeds, which is also confirmed by our experiments.

6.2. Related Work

Light source sampling The large body of work on sampling of direct illumination in rendering can be roughly divided into techniques which are either concerned with selecting important light sources using a hierarchy [28, 114, 168], or with sampling points on a chosen light source, e.g. proportional to (projected) solid angle [4, 64, 127]. All of these rely on the availability of explicit light source information, which is sometimes hard to obtain, e.g. with textured emission or procedurals, and are lacking important information about occlusion. In contrast, we only require initial hemisphere or BSDF samples, and steer our mixture model towards highly-contributing directions.

Resampled importance sampling and ReSTIR Efficient resampling schemes [16] have shown impressive performance for sampling direct illumination of many light sources, and several extensions to global illumination have been proposed [124, 96, 20, 21]. However, since these works are based on resampled importance sampling (RIS) [144], they do have conceptual limitations [95]: resampling cannot discover new light sources, but only selects from a known set. That is, no new information is gained by resampling, and other techniques such as Markov chain transitions need to be mixed in to fulfill this task [135]. Our work is also based on selection by resampling and Markov chain transitions, but in a different order and different state space.

Dependent Monte Carlo Sampling Our technique constructs an unbiased Monte Carlo estimator on top of dependent samples from a Markov chain. Previous approaches achieved this by deterministic groups of samples [165, 62] or by explicit estimation of the reciprocal integral of the PDF [23, 169]. We employ continuous MIS [164] to achieve this.

Metropolis light transport Metropolis light transport [150, Chapter 11] simulates a certain target distribution by estimating a histogram of the posterior of the Markov chain. The posterior is a unique equilibrium distribution which does not depend on the initial state. To utilize this to estimate integrals, the mean image brightness has to be computed separately and the posterior of the Markov chain has to be known and is explicitly controlled by using transition densities to compute an acceptance probability after Metropolis-Hastings. We do not require either: our algorithm still works if the equilibrium distribution is not unique, and we can tune acceptance probabilities to focus on fast adaptation to dynamic content without evaluating transition densities or mean image brightness.

There exist other relevant extensions to MLT that share samples in image space through replica exchange [54] or ensembles of light transport paths to guide future transition kernels [9]. Nevertheless, these works still rely on detailed balance conditions in the same fashion as classic MLT.

Adaptive MCMC Adaptive MCMC [56] generally requires removing the adaptation of the transition densities in the limit. While our requirements on the Markov chain are lighter, our adaptation will also vanish if the scene remains static.

Ensemble and Population Monte Carlo There exist many Monte Carlo algorithms which utilize a population or ensembles of samples to improve an estimator [102, 143]. Our algorithm is similar to Population Monte Carlo (PMC) [24], which draws samples from a population of lobes. After the sampling step, the lobe means are selected from the sample locations via weighted resampling. Variants such as Optimized-PMC [43] also estimate the lobe covariances. In our work, the populations are tailored

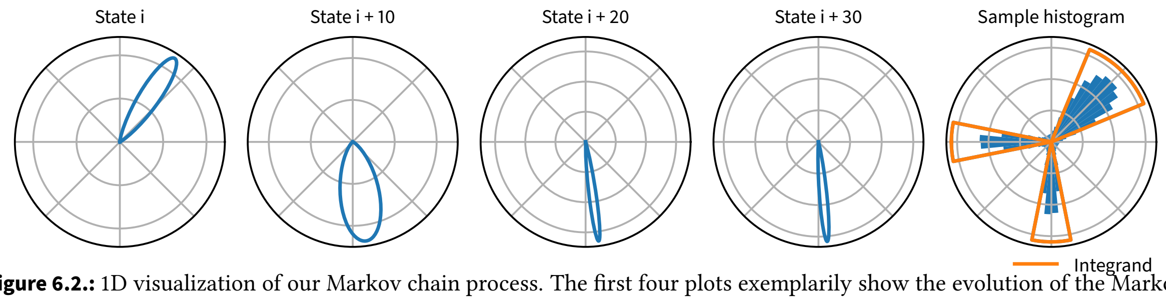


Figure 6.2.: 1D visualization of our Markov chain process. The first four plots exemplarily show the evolution of the Markov chain over the iterations. The last plot shows the target integrand and resulting sample distribution after 800000 iterations. Figure taken from [36].

to groups of pixels and threads. We explicitly have a population of lobe means and concentrations, and the mean will be updated from the samples via a maximum likelihood step.

PMC is known to reduce the diversity of the samples due to the resampling step. Elvira et al. [44] survey different resampling strategies (global, local, independent) and propose a reduced degeneracy strategy. Our resampling strategy is similar to what they call *independent*, but is designed to adapt quickly to dynamically changing target functions and for simple thread synchronization.

There exist prior approaches which employ MCMC sampling in combination with a Monte Carlo estimator [138, 133]. This is useful to draw more information from rejected samples. Our Markov chain operates on a parameter space controlling the lobes of the sampling PDF and is not directly used to estimate an integral.

Layered adaptive importance sampling [103] is a very related technique which estimates lobe means via MCMC and samples from these in combination with multiple importance sampling (MIS). Unlike our work, they use fixed lobe covariances and there is no sample reuse between the MCMC and the MC step.

Path guiding Data-driven adaptive sampling approaches such as *path guiding* have proven to be valuable tools in offline production rendering [154] to reduce variance when simulating complex global light transport. These gather information about light transport during rendering or in a pre-processing step to propose samples proportional to a guiding target function, e.g. incident radiance or its product with the BSDF. Various approaches have been presented, which differ in used spatial or directional data structures [77, 91, 74, 10, 116], domains [170, 131], or target functions [129]. Closely related to this work are approaches based on parametric mixture models [155, 71, 72, 134, 37, 137] of distributions from the exponential family, like Gaussian or von-Mises Fisher (vMF). They are usually fitted using weighted expectation maximization (wEM) [155]. Ruppert et al. [134] present a variance-aware split and merge algorithm to increase the robustness of wEM, while determining the optimal number of mixture components.

Due to the strict constraints on compute time and memory bandwidth, real-time guiding approaches [35, 125, 34] cannot afford complex data structures or fitting methods. Inspired by Derevyannykh [34], we store a single vMF component per pixel. However, we additionally combine vMF lobes of neighboring pixels to build a randomized mixture model.

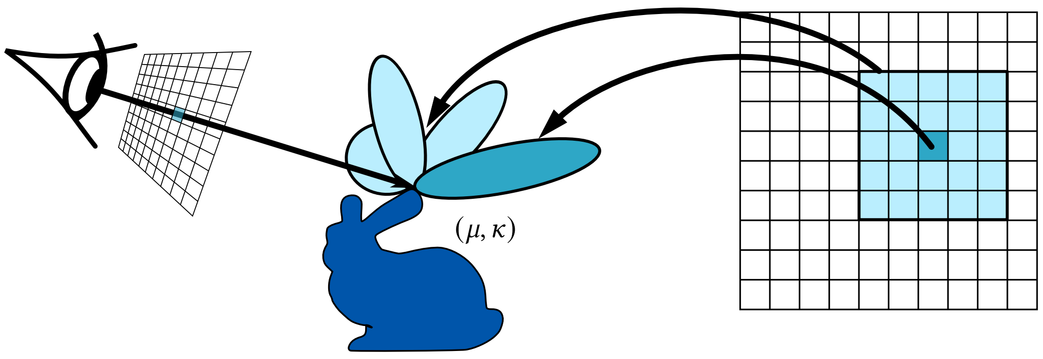


Figure 6.3.: The state space of our Markov chains defines the parameters of a single von-Mises-Fisher (vMF) lobe (μ, κ), and we run one Markov chain per pixel p. A random instance of the full mixture model for any pixel can be generated by collecting vMF lobes over the time progression of the Markov chain, or in a window around the query position (shown in light blue). Figure taken from [36].

6.3. An Estimator using a Randomized Mixture Model

The core idea pursued in this work is to replace a mixture model with a discrete and fixed number of components and their weights (section 4.2.5) by a continuous equivalent:

$$\mathrm{p}(\omega) = \int \mathrm{p}(\mathbf{t}) V(\omega \mid \mathbf{t})\, \mathrm{d}\mathbf{t}, \tag{6.1}$$

where $\mathbf{t}$ represents the component parameters and $\mathrm{p}(\mathbf{t})$ the component weight.

While sampling such a mixture is typically trivial, estimating the reciprocal PDF to construct an estimator is often expensive [23, 169]. We instead use the n-sample *stochastic multiple importance sampling* (SMIS) estimator proposed in continuous multiple importance sampling [164, eq. (12)]:

$$\langle L_r(\mathbf{x}_1) \rangle_{\mathrm{SMIS}} = \sum_{i=1}^{n} \frac{f_r(\mathbf{x}_1) L_e(\mathbf{x}_{2,i}) \cos\theta_i}{\sum_{j=1}^{n} \mathrm{p}(\omega_i | \mathbf{t}_j)}. \tag{6.2}$$

We essentially approximate the full PDF through a stochastic selection of lobes. Note that $\mathrm{p}(\mathbf{t})$ is not present in the expression, since it canceled out: we do not need to explicitly compute or even know about the specific distribution of $\mathrm{p}(\mathbf{t})$. We exploit this property in the following.

We define $\mathrm{p}(\mathbf{t})$ through a Markov chain operating on a state space which allows us to compute the lobe's parameters $\mathbf{t} = (\mu, \kappa)$. A one-dimensional example is depicted in fig. 6.2, where the orange line represents the integrand. As the Markov chain transitions (first four plots), it deposits probability mass resulting in an (unknown) equilibrium distribution $\mathrm{p}(\mathbf{t})$. The resulting sample distribution $\mathrm{p}(\omega)$ can be observed in the last plot. Contrary to Metropolis-Hastings, which gives strong guarantees about the *exact* equilibrium distribution through detailed balance conditions, we do not depend on this reasoning. As a consequence, there is a lot of freedom designing transition steps in the Markov chain, to, for example, emphasize temporal adaptation without causing bias.

For our use-case (direct illumination), we store one Markov chain state per pixel in a simple screen space data structure (fig. 6.3). We make use of the flexibility in state transitions and perform them in an iterative fashion per frame, incorporating neighbor states to facilitate information sharing, as detailed in section 6.3.1.

West et al. [164] have made an additional assumption when proving unbiasedness of the SMIS estimator: All the pairs of lobes with their drawn samples need to be independent among each other. In particular, this means that (1) samples must be independent of each other and (2) techniques must be independent of drawn samples and (3) also from other techniques. These requirements are initially violated when using our inherently correlated Markov chain process, resulting in a biased estimator. In section 6.3.2, we show how the first two requirements can be upheld efficiently, by shuffling states between neighboring pixels. In appendix A, we prove that the last requirement is actually not needed at all, thus making our estimator unbiased even in the case of correlation. This is important when reusing Markov chain states from the framebuffer.

6.3.1. A Markov Chain Mixture Model

In this section, we detail the Markov chain process leading to a continuous mixture model of vMF lobes. This mechanism has two partially opposing goals: (i) the equilibrium distribution should faithfully represent the actual target distribution, (ii) it should adapt fast to dynamically changing content.

Extended State Space To address the first goal, we use a maximum a-posteriori (MAP) approach [6, 8] to compute accurate distribution parameters $\mathbf{t} = (\mu, \kappa)$. This is directly reflected in our state space definition of the Markov chain (table 6.1): it does not incorporate the parameter tuple, but rather sufficient statistics of a maximum likelihood estimator, from which the parameters are inferred (algorithm 3, with more details in about the MAP priors in section 6.4). This way, previous samples can be incorporated as well. Note that this makes the reduced state space over (μ, κ) non-Markovian, since transitions are affected by past states (section 6.6).

We additionally store a scoring term f. It is used to compute acceptance probabilities and is simply a one-sample estimate of the direct-illumination integral (eq. (4.87)).

State Transitions The maximum a-posteriori approach is sometimes at odds with the second goal which requires that newly appearing lights should be picked up very quickly. To facilitate this, we exchange Markov chain state between neighboring pixels. This results in a collaborative algorithm where information about new or undersampled features spreads quickly over the frame buffer.

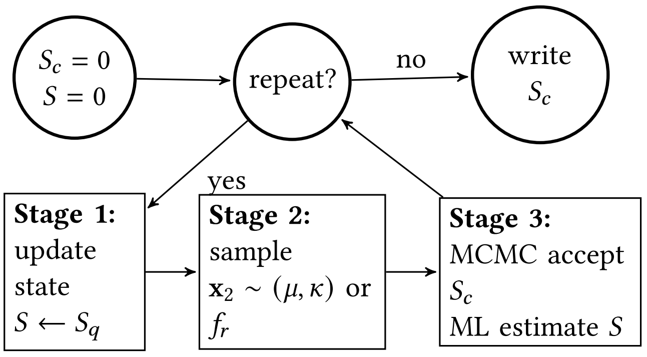

$S_c = 0$
$S = 0$
repeat?
no
write S_c
yes
Stage 1: update state $S \leftarrow S_q$
Stage 2: sample $\mathbf{x}_2 \sim (\mu, \kappa)$ or f_r
Stage 3: MCMC accept S_c ML estimate S

Figure 6.4.: Overview of the Markov chain transitions in a single pixel. An internal sampling state S is updated from neighboring pixels q. Then a new path vertex is drawn from the vMF lobe or a hemispherical PDF. In a third step, a current state S_c is stashed away if the state S passes a MCMC acceptance test. S_c will be written out to be shared with the other pixels in the next frame. Figure taken from [36].

Table 6.1.: Markov chain state $S = (\bar{\mathbf{y}}, \bar{r}, \bar{w}, N, f)$. We access elements of this tuple in the form of $S.\bar{w}$.

Symbol	Meaning
$\bar{\mathbf{y}}$	weighted mean of light source vertex positions
$\bar{r}$	mean cosine of weighted directions
$\bar{w}$	sum of weights (estimates of eq. (4.87))
N	number of samples which went into this state so far
f	score (estimate of eq. (4.87))

Within one frame, each pixel will independently (and in parallel) mutate their state in a loop with n iterations. Each iteration is composed of three stages as illustrated in fig. 6.4 (we refer to table 6.2 for symbol meanings):

1. We look at the states $S_q^{(i)}$ in other pixels q in a window around our pixel p and potentially use their state to overwrite ours.
2. We create a new path vertex $\mathbf{x}_2$ by drawing either an independent hemispherical sample or by using the current vMF lobe of this pixel S.
3. Through a mutation/acceptance step, we keep a current state S_c over all iterations which will eventually be written out as $S_p^{(i+1)}$.

In the following paragraphs we detail the three stages of the Markov chain transition steps.

Table 6.2.: Glossary of symbols used in the following stages.

Symbol	Meaning
S_p, S_q	Markov chain states for pixels p and q, respectively
S	Tentative state used for sampling
S_c	Current state to be written after rendering the frame

Stage 1: collaborative discovery/inter-pixel update Algorithm 2 details this stage: as mentioned, we potentially replace the pixel state S by those found in a framebuffer storing the states of the neighbor pixels written out in the last frame $S_q^{(i)}$. That means they have one frame lag and need to be accessed after correcting the pixel position q with estimated motion vectors to account for dynamic changes (e.g. camera or object motion) between frames. We randomly select a pixel q in a configurable window around p by sampling a B-spline kernel of degree 5. The offset is the sum of four random numbers [142].

The state found, S_q, provides a light position $\mathbf{y}$ but no information about the shading point that lead to the score f which is also provided in the buffer. This means that we have to assume some smoothness over both space and time such that this mutation type makes sense. This has been successfully exploited by similar approaches in the past [16][150, Chapter 11].

In practice, we use a window size of 91×91 at a resolution of 1920×1080. While it appears large, it is a trade-off between “spreading the news” quickly over the image, and the potential mismatch in score function between pixels due to occlusion. Cheap rejection heuristics (e.g. based on the shading point $\mathbf{x}$) are possible to use here without causing bias, but we simply use $\text{score}(S) = S.f$.

Algorithm 2: Collaborative discovery/inter pixel update

```
$S \leftarrow$ shuffleUp subgroups ;                        // see section 6.3.2
$\mathsf{sum} \leftarrow \mathrm{score}(S)$;
for some random neighbor pixels $q$, until $\mathrm{score}(S) > 0$ do
    $S_q \leftarrow$ load state $S_q^{(i)}$ at motion corrected $q$;
    $s_q \leftarrow \mathrm{score}(S_q)$;
    $\xi \leftarrow$ uniform random in $[0, 1)$;
    if $\neg\mathtt{valid}(S) \vee \xi < s_q/(s_q + \mathsf{sum})$ then
        $S \leftarrow S_q$;
    $\mathsf{sum} \leftarrow \mathsf{sum} + s_q$;
```

Algorithm 3: Sampling and ray casting

```
$\xi \leftarrow$ uniform random in $[0, 1)$;
if $\neg\mathtt{valid}(S) \vee \xi < 0.2$ then
    $\mathrm{p}(\omega) \leftarrow \frac{\cos\theta}{\pi}$ or $f_r(\mathbf{x}_1)$;         // cosine or BSDF sampling
else
    $\mu \leftarrow \frac{S.\bar{\mathbf{y}}-\mathbf{x}_1}{\|S.\bar{\mathbf{y}}-\mathbf{x}_1\|}$;         // convert to direction
    $r \leftarrow \frac{N^2 \cdot S.\bar{r}+N_p \cdot r_p}{N^2+N_p}$;         // apply prior
    $\kappa \leftarrow \frac{3r-r^3}{1-r^2}$;         // approx. lobe width
    $\mathrm{p}(\omega) \leftarrow V(\omega|\mu, \kappa)$;         // vMF distribution
sample $\omega \sim \mathrm{p}(\omega)$;
shoot ray to find $\mathbf{x}_2$;
```

Like population Monte Carlo methods, we perform weighted resampling of the neighbor states: we use the score function as weight and use reservoir-style sampling in a linear scan to randomly select a neighbor state proportional to this weight [26].

In the algorithm listings, a state S is considered *valid*, if it accumulated non-zero weight, i.e. $\bar{w} > 0$. Also note that in line 1 an additional shuffling step is performed. It is included here already to provide a coherent view of the program flow, but the necessity will later be explained in section 6.3.2.

Stage 2: sampling, ray casting To generate initial samples as well as large steps in the Markov chain we use hemispherical sampling. This can be achieved by sampling from the BSDF or simply by using a cosine distribution. We use either in our two separate implementations. This is performed if the state S holds no valid vMF lobe as well as at random. We use a fixed probability of 0.2. This also makes our sampling defensive, since the estimator now always contains BSDF or cosine sampling as a backup-strategy (see appendix A). In case the sample is drawn from the vMF, μ and κ are first reconstructed from the internal state representation (see algorithm 3).

Stage 3: internal pixel state update After tracing the new direction and finding a path vertex $\mathbf{x}_2$, we compute the score f of this connection. Then, we use a Metropolis acceptance step to decide whether to advance the current state S_c. After the MCMC step the maximum a-posteriori adaptation of the lobe is performed. Note that this includes zero-contribution samples which will make the lobe a bit narrower. Algorithm 4 shows this procedure.

Algorithm 4: Pixel state update, given a complete path $\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2$

```
$f \leftarrow \mathrm{lum}\Big(L_e(\mathbf{x}_2) \cdot f_r(\mathbf{x}_1)\Big)/\mathrm{p}(\omega)$;
$a \leftarrow \min\{1,\ f/S_c.f\}$;
$\xi \leftarrow$ uniform random in $[0, 1)$;
if $f > 0 \land (\neg\mathtt{valid}(S_c) \lor \xi < a)$ then                // MCMC transition
    if hemisphere sample then
        $S \leftarrow \{0\}$;                               // reset lobe if $\mathbf{x}_2$ is not from vMF
    $S.f \leftarrow f$;
    $S_c \leftarrow S$;
$S.N \leftarrow \min\{S.N + 1,\ 1024\}$;                     // maximum likelihood update of $S$
$\alpha \leftarrow \max\{1.0/S.N,\ 0.1\}$;
$\bar{w} \leftarrow \mathrm{mix}(S.\bar{w},\ f,\ \alpha)$;
$S.\bar{\mathrm{y}} \leftarrow \mathrm{mix}(S.\bar{w} \cdot S.\bar{\mathrm{y}},\ f \cdot \mathbf{x}_2,\ \alpha)/\bar{w}$;          // compute $\mu$ as in algorithm 3
$\omega \leftarrow \mathrm{mix}(S.\bar{w} \cdot S.\bar{l} \cdot \mu,\ f \cdot (\mathbf{x}_2 - \mathbf{x}_1)/\|\mathbf{x}_2 - \mathbf{x}_1\|,\ \alpha)/\bar{w}$;
$S.\bar{l} \leftarrow \|\omega\|$;
$S.\bar{w} \leftarrow \bar{w}$;
```

The score function f is evaluated as the luminance of the emission L_e times BSDF $f = \mathrm{lum}(L_e(\mathbf{x}_2) \cdot f_r(\mathbf{x}_1))/\mathrm{p}(\omega)$. We divide out the solid angle PDF to make f a coarse estimator for the pixel contribution. That is we include the BSDF which technically also depends on the incident direction. Since this varies slowly over the image it improves sampling of glossy objects considerably.

6.3.2. Sample decorrelation

Using a sample both to advance the Markov chain and construct the SMIS estimator leads to dependencies between a sample and subsequent techniques and samples (since the drawn sample influences the next lobe), resulting in a biased SMIS estimator. A naïve solution is to draw a separate sample each for both tasks. While this removes these dependencies, drawing a sample in our context is generally an expensive operation, since it requires evaluating complex shading models and to trace a ray. Thus, reusing all samples for the SMIS estimator is paramount.

In the following, we leverage that we are running multiple Markov chains in parallel - one for each pixel. By permuting Markov chain states between pixels after each step, we break any sample dependencies in the SMIS estimator. We discuss *temporal* and *spatial* schemes as depicted in fig. 6.5 that differ in where techniques constituting the SMIS estimator are sourced from.

Temporal SMIS (TSMIS) The mutation/acceptance loop that is executed n times in each pixel leaves a history of n vMF lobes that together form a mixture. A sample is drawn from each lobe and the SMIS estimator is evaluated accordingly. In practice, this requires us to store an array of n sample contributions, locations, and vMF parameters used to sample these. Computing the value of the estimator is then quadratic in n (eq. (6.2)). For moderate values of n we found this to be a negligible cost as compared to the ray tracing.

As mentioned, we cannot easily reuse the sample used to advance the Markov chain for the pixel estimate due to sample dependence. To decorrelate the two processes, we leverage the fact that we are running a whole population of Markov chains, with one state per pixel. We completely reuse all the

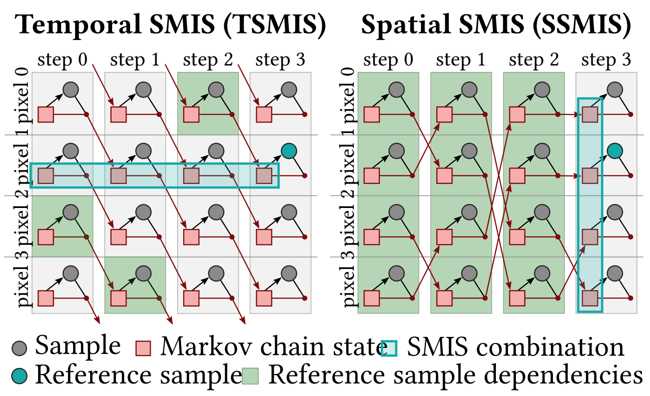


Figure 6.5.: Construction of our two unbiased estimators. After each step (sampling and state update, black arrows), we exchange Markov chain states between pixels (red arrows) to allow for unbiased SMIS. Temporal SMIS (left) decorrelates states by cyclic shifts between pixels and constructs a 4-SMIS estimator over the independent states of all 4 steps. Spatial SMIS (right) constructs a 1-SMIS estimator over the random set of states in the tile at a single step. It selects a state for each pixel to sample from by random permutation. In both cases, we can see for a given reference sample (marked blue) that its dependencies (green) do not overlap with the techniques in the SMIS estimator (blue box). Figure taken from [36].

MCMC samples, the large as well as the small step mutations, in the SMIS estimator. To make sure MCMC and SMIS are completely independent in one pixel, it is enough to simply shuffle up the MCMC state after mutation and before evaluating the SMIS estimator (fig. 6.5, left): we divide the frame into tiles and cyclically shift states between pixels in a tile, rotating the last one back to the first. As long as the tile size is larger than the SMIS sample set size n, the estimator for each pixel will be constructed from completely independent samples.

We choose tile sizes below or equal to 32 pixels (e.g. 8×4). On contemporary GPUs, the threads processing the individual pixels execute in groups of usually 32 (referred to as *subgroups*) and state can be exchanged very efficiently between them. We use this to exchange Markov chain states instead of slow video memory.

Spatial SMIS (SSMIS) For larger n the required storage and quadratic cost in n of temporal SMIS can be limiting factors. An alternate construction scheme can be derived by constructing an estimator over the set of vMF lobes at a given step in the pixel tile.

The resulting estimator is no longer a true n-sample SMIS estimator, but multiple realizations of a one-sample SMIS estimator over n techniques. Essentially, we perform a stratified selection of one technique using uniform probabilities, resulting in:

$$\langle L_r(\mathbf{x}_1)\rangle^1_{\mathrm{SMIS}} = \frac{f_r(\mathbf{x}_1)L_e(\mathbf{x}_2)\cos\theta}{\frac{1}{n}\sum_{j=1}^{n} \mathrm{p}(\omega|\mathbf{t}_j)}. \tag{6.3}$$

The estimator can be directly evaluated after a new sample is generated in each mutation/acceptance step. Additionally, the storage for all vMF lobes is distributed among the threads processing the individual pixels in the tile. As a result, the storage requirements are constant per thread. n is now also independent on the number of steps, which allows one to use fewer steps and still have a large SMIS set to reduce variance.

The selection is implemented as a permutation on the entire Markov chain state to allow using the generated sample to advance the Markov chain. Permutations are particularly important in this instance to avoid loosing states through duplicate selections. In practice, we implement the permutation by xoring a random mask on the subgroup invocation id.

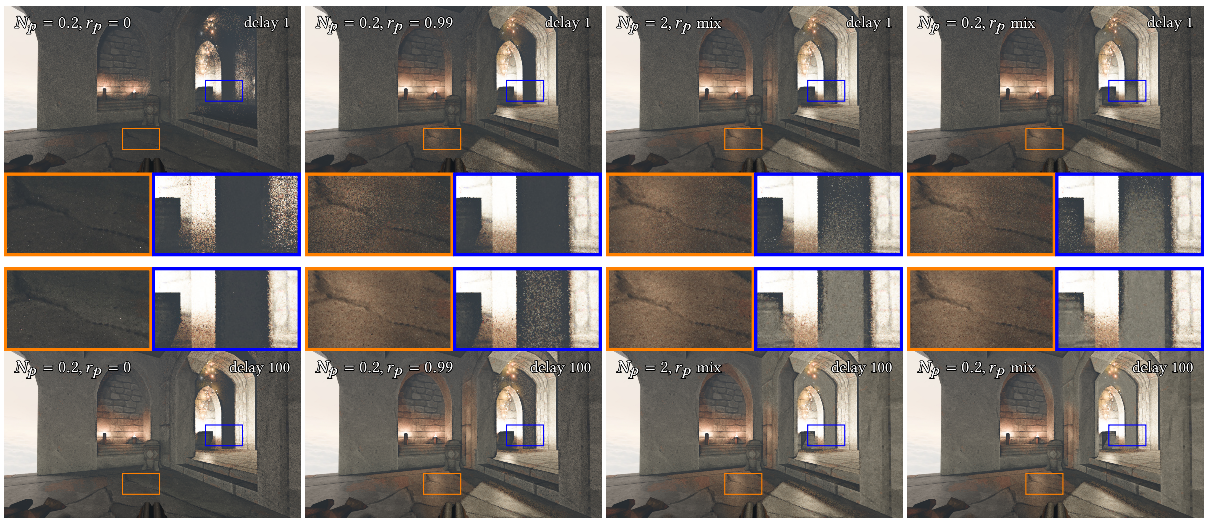


Figure 6.6.: The effect of different priors with mean cosines r_p and weight N_p. After a one frame they differ significantly (top row) but begin to look similar after 100 frames delay (bottom row). Here the lights are very small, so an informed prior with a focused mean cosine of $r_p = 0.99$ generally performs better than $r_p = 0$. The label "r_p mix" signifies a heuristic prior which becomes more diffuse ($r_p = 0$) as the initially discovered light comes closer to the shading point. Frame time was 20ms on an Nvidia RTX 2080 Ti and 10ms on the RTX 3080 Ti. Figure taken from [36].

6.4. Implementation details

We implemented our approach in two open-source rendering systems: The NVIDIA Falcor research renderer [80] and the classic Quakespasm engine. While Falcor allows us to evaluate with reference implementations of various algorithms, Quakespasm features scenes with highly dynamic content. Initial samples are generated using BSDF (Falcor) and cosine (Quakespasm) sampling. We use classic geometric motion vectors (Falcor) and optical flow [63] (Quakespasm) to gather Markov chain states from the previous frame. The Quakespasm framework does not provide light lists to be sampled. Lights are merely represented as (animated) *fullbright* textures, and we did not attempt to implement next event estimation (NEE) on top of this. While Falcor does contain explicit representations for emitters for NEE, we do not use them to demonstrate the technique's independence of such representations. We will release both implementations with the final version of the paper under a permissive license.

Sufficient statistics To account for parallax between different pixels towards a light, the sufficient statistics (table 6.1) incorporate a weighted mean of light source vertex positions $\bar{\mathbf{y}}$. The mean of the vMF μ can be recovered for a given position $\mathbf{x}$ as $\mu = (\bar{\mathbf{y}} - \mathbf{x})/\|\bar{\mathbf{y}} - \mathbf{x}\|$. This representation has the advantage that it does not depend on a common pivot point compared to using a direction and distance [134]. Therefore, the statistics can be exchanged directly between neighbors.

N is employed to compute a more accurate arithmetic average for the first few samples before switching over to an exponential average to better adapt to dynamic changes. It is also used to support custom priors which vanish over time. Such a prior is defined as N_p, the virtual number of samples it accounts for, as well as r_p, the mean cosine corresponding to the prior lobe width.

Uninformed vs. informed priors We use maximum a-posteriori parameter estimation with priors to stabilize lobe width estimation with very few samples. This is common due to the reset of lobe statistics when switching to different or new features. To get accurate fits quickly, we try to use all available

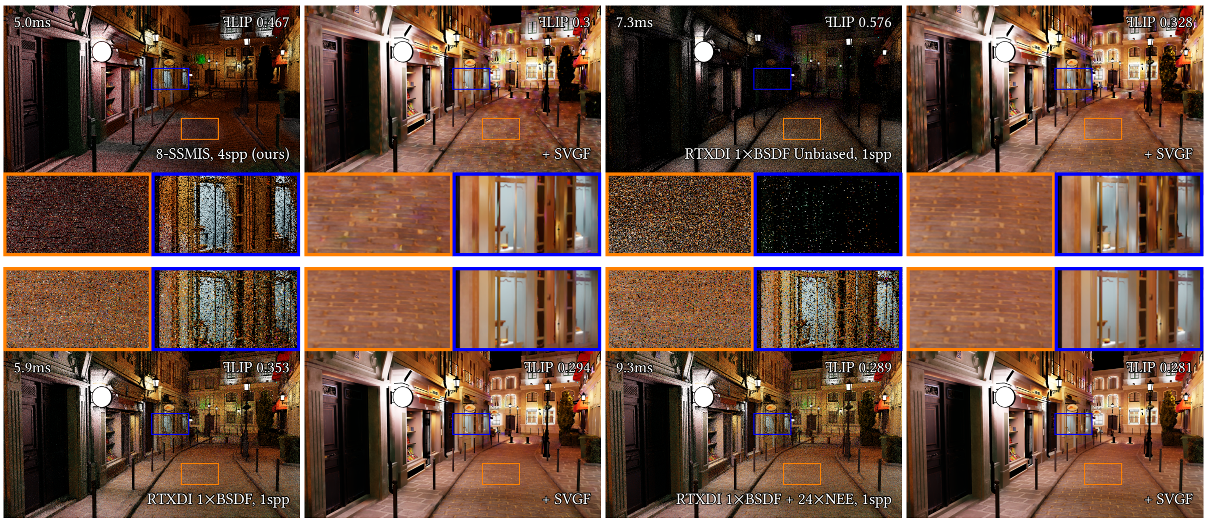


Figure 6.7.: Comparison of spatial SMIS against variants of RTXDI, a production implementation of ReSTIR [16] with and without application of a denoiser (SVGF [136]). Our approach is competitive in areas illuminated primarily by few light sources. Biased variants of ReSTIR retain image brightness better in complex regions. Figure taken from [36].

prior knowledge about the scene's light sources. In the Falcor scenes, the size of lights is unrestricted, and we thus resort to an uninformed (uniform) prior with a mean cosine of zero. For Quakespasm, we leverage the quantization of meshes to a size of 1.0 in world-space. There, we use an informed prior that fits the mean cosine to this size based on distance, resulting in a more uniform or peaky lobe for close or distant lights, respectively (see fig. 6.6). Tracking of emitting particles benefits most from this, since particles are sized exactly according to this quantization. In both implementations, the prior vanishes in a squared relationship to the number of samples. This is crucial for adaptation under dynamic conditions and also because fits are lost regularly due to initial samples, thus mandating fast recovery towards a feature.

Apparent targets For caustics rendering, retargeting a vMF lobe to the interface (the first intersection from the originating surface) produces suboptimal results. We use the concept of apparent distance to the emitter [134]. Since our vMF lobes are represented with respect to the target and not direction, we use an apparent target instead. Intuitively, we prolong the point at the interface in the originating direction using the distance from interface to emitter, scaled by a correction factor [134, eq. (35)].

6.5. Evaluation

The Falcor implementation is evaluated on an Nvidia RTX 3080 using openly available [98] and custom test scenes. The Quakespasm implementation is evaluated on an Nvidia RTX 2080 Ti and 3080 Ti and uses scenes from the *Arcane Dimensions* mod (`https://www.moddb.com/mods/arcane-dimensions`), which introduces much higher geometric complexity as compared to the original game. Since we mostly present rendering of direct illumination, we focus our comparisons to ReSTIR, which represents the state of the art in this area, using the publicly available RTXDI implementation in Falcor. In particular, we do not compare against works that cannot represent high-frequency direct illumination adequately [35, 34].

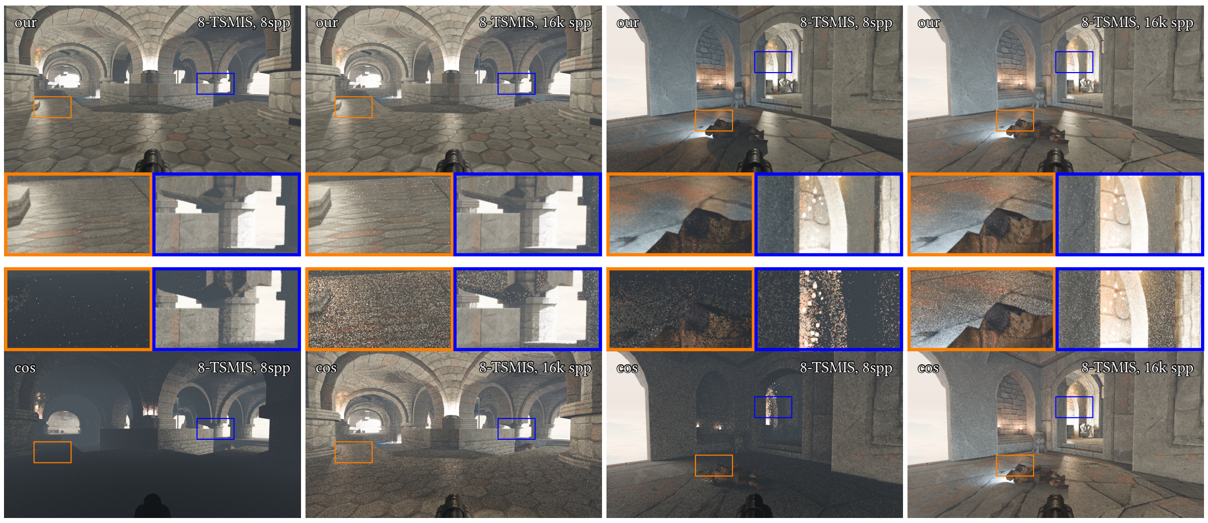


Figure 6.8.: Glossy materials rendered with vMF lobe or cosine-hemisphere sampling only, none of the approaches use BSDF importance sampling. Using the BSDF in the score function results in effective product sampling. Frame times on an Nvidia RTX 3080 Ti were 15ms and 9ms. Unlike fig. 6.11, these images are acquired after a few frames of adaptation, so the remaining image errors are due to SMIS estimating the PDF from a small subset of lobes. TAA is used to improve edge rendition. Figure taken from [36].

We report rendering times at display resolutions of 1920×1080. In Falcor, we only include the techniques themselves, not including rasterization of the G-Buffer or post-processing passes such as denoising. The Quakespasm render times include a ray traced G-Buffer and temporal antialiasing (TAA).

We refer to temporal and spatial SMIS as A-TSMIS or A-SSMIS, respectively. A is the tile size (e.g. 8-TSMIS operates on a 4×2 tile). We also denote the effective ray-traced samples per pixel (spp) for each image. For TSMIS, the spp are always a multiple of A.

Comparison to ReSTIR We compare our approach to ReSTIR [16] using the RTXDI production implementation in figure 6.7. RTXDI offers different candidate generation methods and also biased and unbiased variants. We show comparisons against RTXDI using just one BSDF as well as additionally 24 next event estimation (NEE) candidates. While the latter gives drastic quality improvements to ReSTIR, the former is more directly comparable to our approach, given that we rely purely on BSDF samples to find new features. We observe that our approach is competitive to ReSTIR in certain areas that are illuminated by few light sources (blue inset), while regions more evenly lit by many light sources are much darker in comparison. The unbiased variant of ReSTIR using a single BSDF candidate exhibits more darkening due to stronger noise. We also apply SVGF to all approaches. While the results are comparable, our approach tends to produce more splotches, likely induced by outlier samples. Execution times are generally similar, where our parameterization is slightly faster than the single BSDF variant. The other variants of ReSTIR are more expensive due to additional processing.

Glossy Surfaces Figure 6.8 shows how the algorithm adapts to product sampling situations with a glossy BSDF even when the initial samples are cosine distributed.

Caustics Our independent sampling scheme allows the application to rendering of caustics. This is shown with both our Quakespasm (fig. 6.9) and Falcor (fig. 6.10) implementations. The dynamic

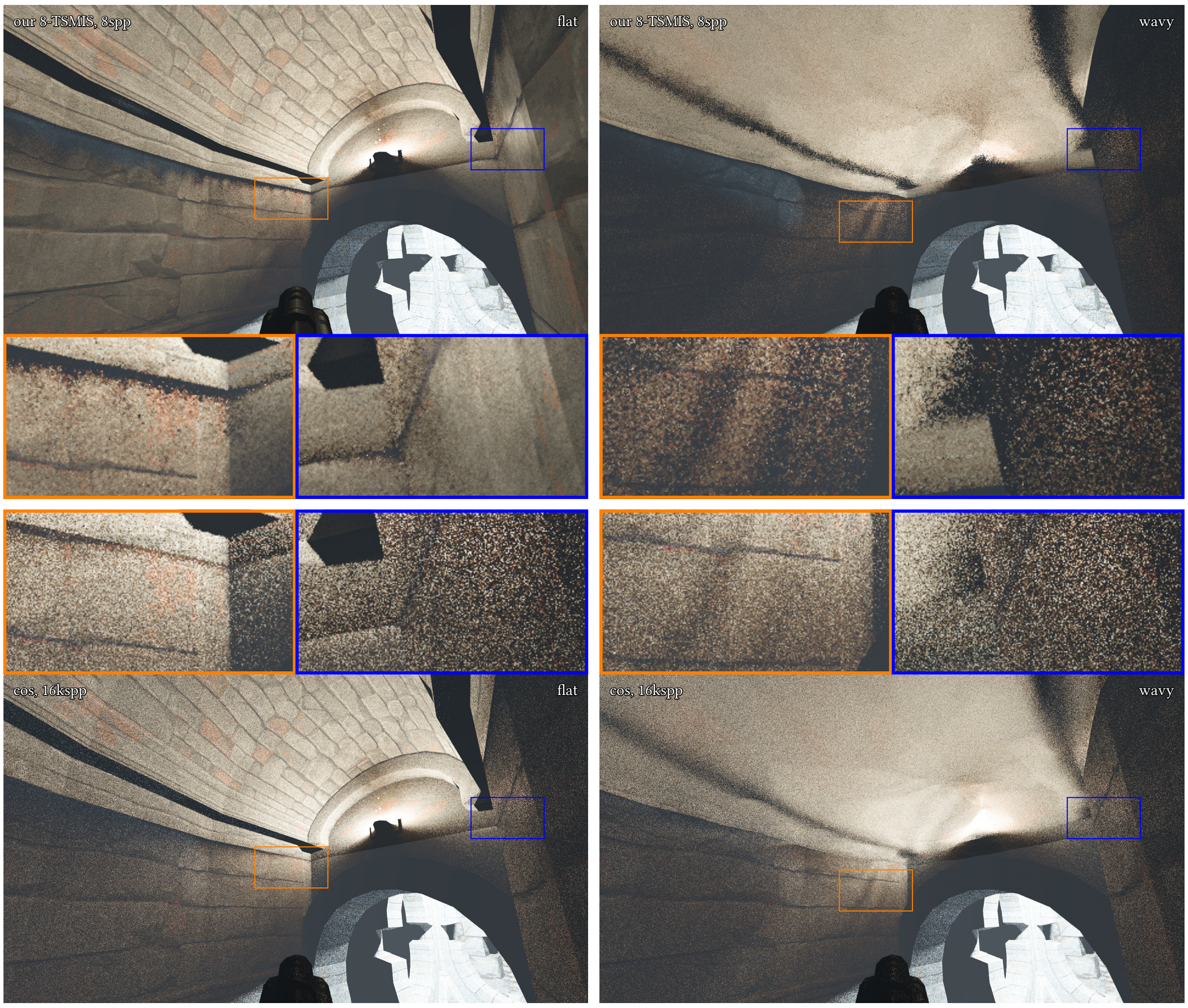


Figure 6.9.: Underwater renders showing caustics cast from the torchlight at the back wall to the underwater wall on the left. The wavy water surface consists of procedurally animated noise with 15 octaves, the torch emits animated particles. Frame time was 71ms on an Nvidia RTX 2080 Ti and 29ms on 3080 Ti. Figure taken from [36].

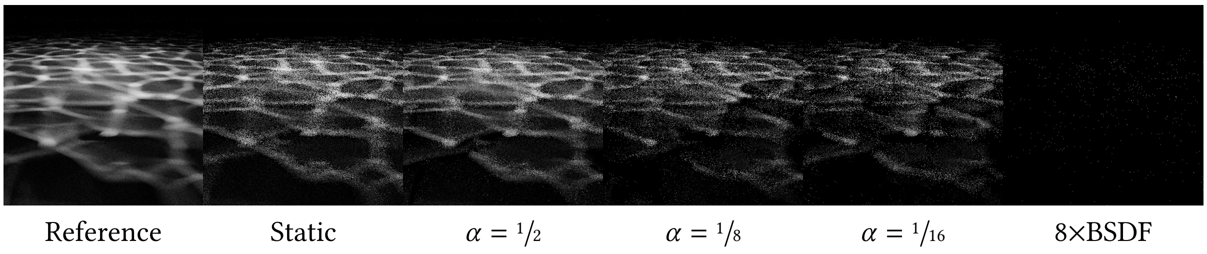


Figure 6.10.: Caustics induced by a small area light over a water surface. When the light is static, our technique (8-SMSIS) can resolve the caustics robustly. When the light is moving, adaptation depends on how long sample history is kept. Higher α-values (shorter history) allow the technique to readapt quicker. Figure taken from [36].

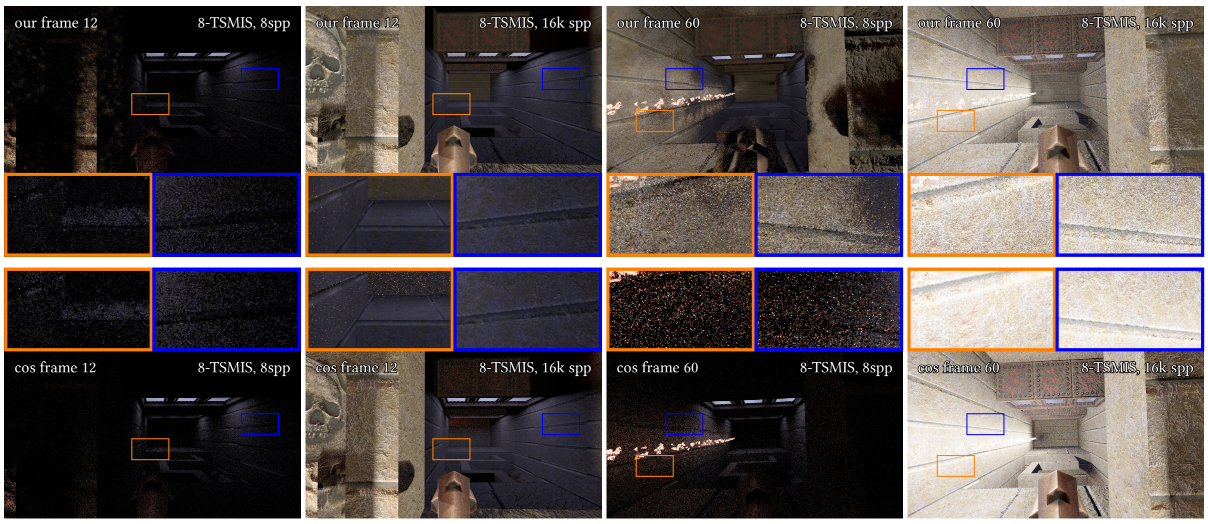


Figure 6.11.: Testing temporal adaptation: this figure shows the same scene at different points in time: frame 12 and frame 60. In frame 12 the rocket has not yet been fired, to illustrate the darkness of the environment. In frame 60, the rocket has not yet impacted in the back wall, so the emitting particles of the trail are a dynamic change. Our algorithm picked up the light sources already, but has not yet converged to a good sampling distribution: this can be observed by looking at the accumulated 2k frames versions which are notably brighter. Frame times were 15–16ms on an Nvidia RTX 2080 Ti and 8–9ms on 3080 Ti. Figure taken from [36].

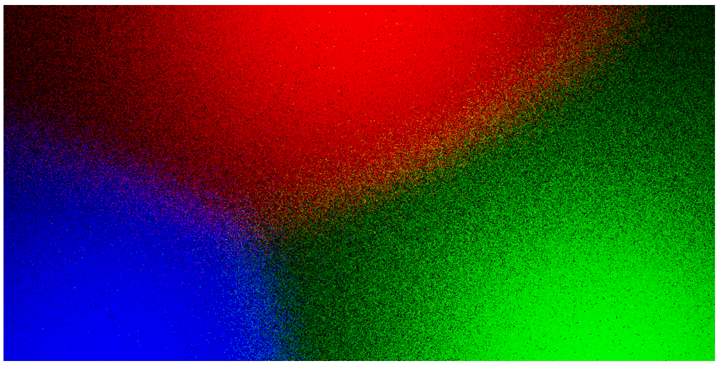


Figure 6.12.: Mixing behavior of three colored light sources. The Markov chain process has a stronger preference for the biggest contributor. This is hidden to some degree through neighborhood resampling as can be seen in the band that forms between the influence regions of the lights. Note that the asymmetry stems from using luminance in the scoring term. Figure taken from [36].

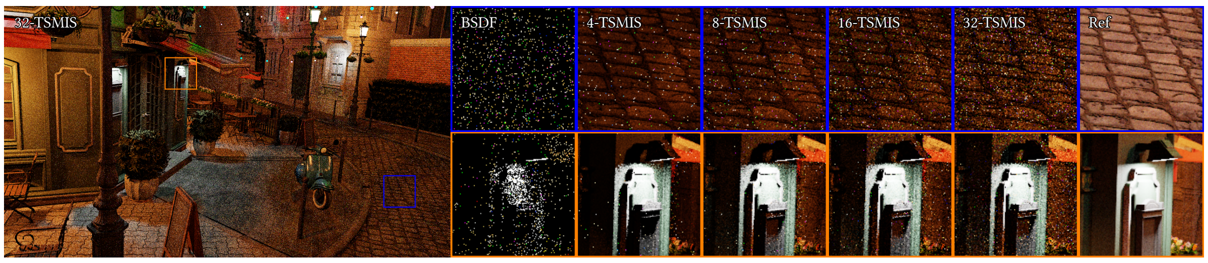


Figure 6.13.: 32spp renderings using different SMIS sample set sizes. Surfaces illuminated by many light sources benefit most from larger SMIS sample sets (blue inset), but smaller sets are sufficient otherwise (orange inset). Figure taken from [36].

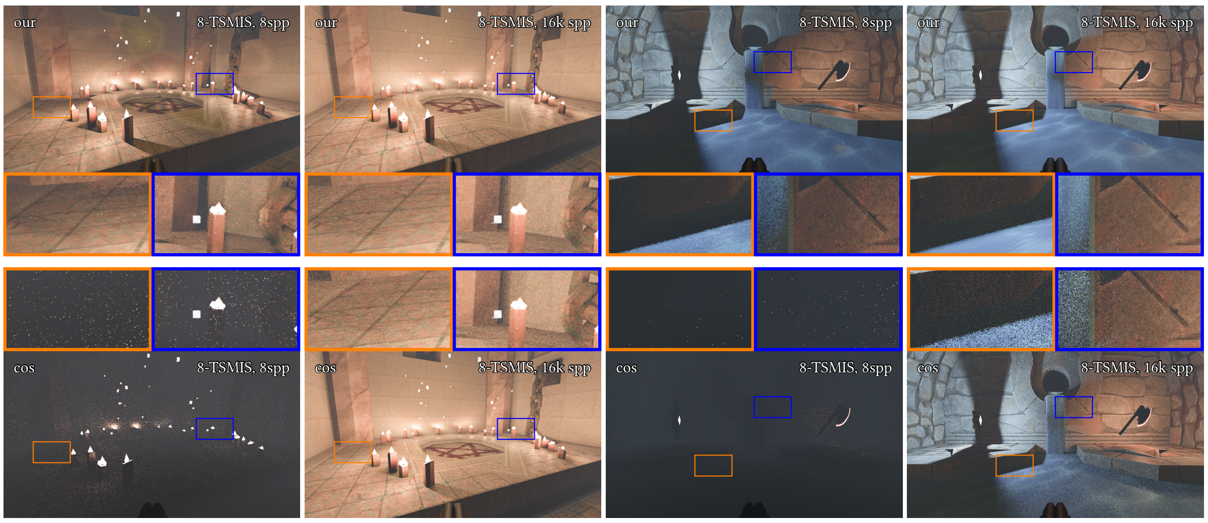


Figure 6.14.: Many light sources (left: candles and the animated spark particles emitted from them) pose a hard problem for the stochastic lobe mixtures. Right: two smoothly overlapping lights (blue and red) showing the transition between two dominant lobes. Unlike fig. 6.11, these images are acquired after a few frames of adaptation, so the remaining image errors are due to SMIS estimating the PDF from a small subset of lobes. TAA is used to improve edge rendition. Frame times were 14.9ms (candles) and 18ms (dual lights) on an Nvidia RTX 2080 Ti, and 8ms and 10ms respectively on a 3080 Ti. Figure taken from [36].

adaptation is fast enough to track moving caustics due to moving emitter (Quakespasm, Falcor) and animated water surface (Quakespasm).

Temporal behavior in dynamic scenes We have tailored the method to quickly react to dynamic changes. We demonstrate this with our Quakespasm implementation in fig. 6.11. Over the span of 48 frames, a rocket is launched (frame 12), leaving a trail of emitting particles. Just before the rocket impacts the wall (frame 60), the walls already receive stronger illumination compared to cosine sampling. However, the sampling distribution is still not optimal, given that the accumulated image is much brighter in comparison.

Mixing behavior We observe that the Markov chain process exhibits a stronger preference for the light source with the most contribution as can be seen in fig. 6.12. The light sources are colored to make correspondence of samples more obvious. We can clearly see that the color does not follow a smooth transition when moving between light sources, but rather a defined barrier, which is blurred to some degree through neighborhood resampling. This is a limitation of our current approach and leads to increased variance, especially where many light sources contribute similarly to a surface region.

Effect of SMIS set size in many-lights scenarios A core issue with SMIS is that every feature needs to be adequately represented in order to not produce outlier (firefly) samples. In a mixture model, this boils down to one component for every prominent feature. While our spatio-temporal mixture has a very large number of components, the use of SMIS effectively reduces this set to a small fixed-size stochastic selection during evaluation. This issue becomes apparent when surface regions are evenly lit by many light sources, as depicted in the blue insets of fig. 6.13 or the left part of fig. 6.14. The result is a general darkening of the image with an increase in outlier samples. Note that the technique is unbiased, thus more energy is focused in the outlier samples. Increasing the SMIS sample set size reduces this

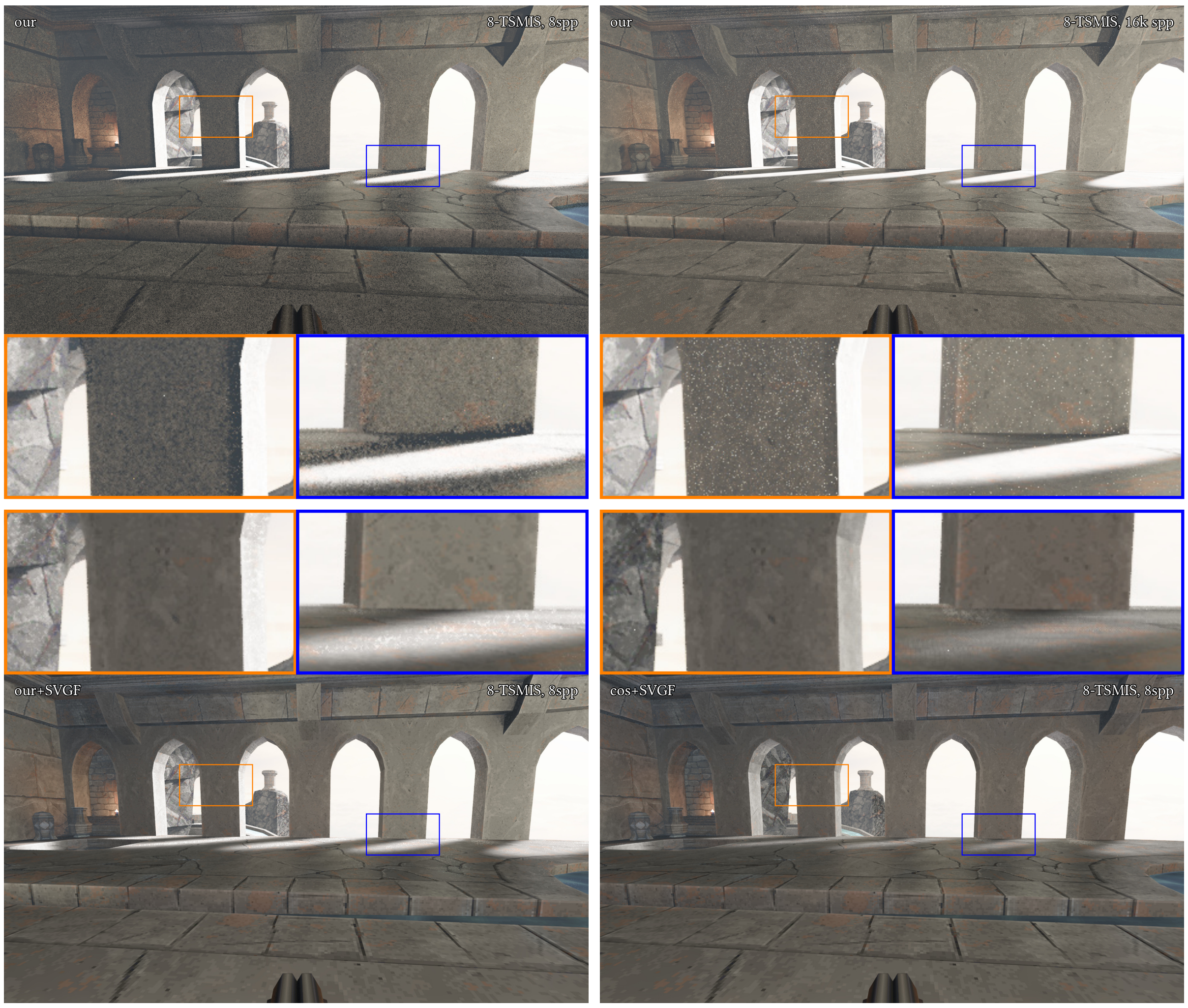


Figure 6.15.: Proximity bias when estimating lobes from neighbor pixels manifests itself as high-variance fringes around edges or shadow boundaries. A seemingly very important light propagates quickly to the neighbor pixels but then turns out to be occluded. In our estimator this does not cause bias but can lead to blurry contact shadows after applying SVGF (bottom images). Frame time was 20ms on an Nvidia RTX 2080 Ti and 10ms on a 3080 Ti. Figure taken from [36].

issue, such that the image approaches the brightness of the reference. Intuitively, the likelihood of a sample being represented by a vMF lobe increases when the set size is increased, thus leading to fewer outliers and an increase in brightness. When a surface is primarily illuminated by a single light source (orange inset), small SMIS sample set sizes suffice.

Proximity bias when sharing lobes between pixels Figure 6.15 illustrates a situation where bright lights are propagated between pixels but are occluded in some of them. In this case re-using the estimate from the neighboring pixels is suboptimal because sampling efforts will be spent on invisible lights. This degrades the quality of the estimator but does not cause bias: this can be observed by looking at the orange inset, where SVGF successfully averages the noisy fringe around the edge closely to the correct value. It fails to do the same for shadow edges since there is no auxiliary buffer to support this edge (blue insets).

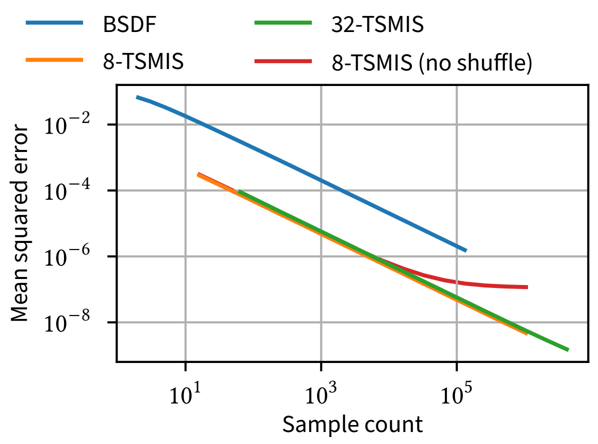


Figure 6.16.: Log-log plot showing the convergence of our technique when rendering a Cornell box scene (orange line). Note that due to the simple lighting, using a larger SMIS sample set size does not reduce the error further (green line). Performing no state shuffling results in a biased estimator (red line). Figure taken from [36].

Unbiasedness We prove unbiasedness of our technique in appendix A. In fig. 6.16 we accompany this proof with a log-log convergence plot when rendering the Cornell box scene. Note that local permutations of Markov chain states are needed to attain an unbiased estimator as discussed in section 6.3.2.

6.6. Discussion, Limitations and Future Work

In the following section we summarize our key findings and lay out the limitations of our approach and possible future work.

Optimality of importance sampling Since we do not apply Metropolis-Hastings, we do not know the actual distribution of the lobes $p(\mathbf{t})$. While it is great to enjoy the freedom to tweak the transition process to ones individual needs (prefer fast, reactive adaptation for dynamic environments) it would certainly be interesting to devise a Markov transition that obeys detailed balance and steers the equilibrium exactly towards optimal importance sampling. We expect this to perform worse in dynamic environments but better on static scenes in the long run.

Markov chain property Since we employ a maximum likelihood step which is storing intermediate values which are fed by multiple previous samples, the sampling is non-Markovian in path space (it is Markovian in a state space that includes the intermediates). As adaptive MCMC [56] we can see that the chain is still ergodic and converges to a unique equilibrium if the adaptation vanishes with increasing mutation count, i.e. the maximum likelihood estimate converges.

Many lights sampling While the use of SMIS is a crucial aspect in the evaluability of the model, any features not present in the set during evaluation can and do easily lead to outlier samples. This makes our approach less practical for scenes with millions of disjoint light sources. While an increase in the SMIS set size offsets this issue to some degree, it becomes increasingly expensive to evaluate, which results in a practical limit. With SSMIS, a set size of 32 posed a negligible overhead, so it can likely be increased a bit further without too much cost.

Next Event Estimation (NEE) Practical renderers usually incorporate some form of direct light sampling (NEE) for increased robustness. While we do not make use of NEE to stress test our algorithm, NEE can be combined in the same fashion as BSDF or cosine sampling. While the usage of SMIS increases variance compared to evaluating the full mixture (which is intractable in our case), cases that are reliably sampled by NEE (e.g. small, distant light sources) will still see large variance reductions. Furthermore, if NEE can reliably sample most relevant light sources, our technique could focus the mixture on the harder cases [84].

Indirect illumination In principle, our approach is not limited to guiding of direct illumination for primary surfaces. This is only due to our choice of a screen space data structure for simplicity. Guiding of direct illumination for deeper path vertices can be achieved by moving towards a world-space data structure similarly [20]. Guiding of indirect illumination additionally requires changes in data gathering (backtracking of paths to compute contribution for all path vertices). Furthermore, incoming radiance becomes a stochastic quantity which likely has side effects on the fitting process.

6.7. Conclusions

We have presented a lightweight guiding algorithm for sampling direct illumination based on a spatio-temporal randomized Markov chain mixture model. We laid the theoretical foundation to give the Markov chain process full flexibility in its state transitions if desired. We demonstrated this by steering it towards tracking of dynamic lights, at the cost of non-optimal variance reduction which manifests itself in this setting as a tendency of overrepresenting of important light sources. We hope to inspire future work where Markov chain transitions are tailored for other use-cases such as offline rendering. While we exemplarily demonstrate simple animated caustics using our algorithm, we believe that our randomized mixture model is also applicable to, and beneficial for sampling other lighting effects.

Appendix

A. Unbiasedness of the SMIS Estimator

Our derivation for unbiasedness closely relates to the proof found in the SMIS paper [164, Appendix B]. The sampling technique controlled by the Markov chain process is denoted t_i and x_i are the samples generated by this technique. In SMIS, the derivations start:

$$E[\langle I\rangle_{SMIS}] = E[\langle I\rangle_{SMIS}]_{(t_1,x_1),\dots,(t_n,x_n)} \tag{6.4}$$

$$= E[E[\langle I\rangle_{SMIS}]_{\boxed{x_1},\dots,x_n}]_{\boxed{t_1},\dots,\boxed{t_n}} \tag{6.5}$$

$$\begin{tikzcd} x_1 \arrow[r, bend left, "\times"] & t_1 \arrow[l, bend left] \arrow[r, bend left] & t_n \end{tikzcd}$$

which is valid in our case because, as indicated by the arrows, the x_i depend on their respective t_i, but not the other way around: the t_i do not depend on any x. They are instead advanced by their respective Markov chain process, i.e. there is a dependency of t_{i+1} on t_i. The independence of the t_i among each other is not required to perform the step between eq. (6.4) and eq. (6.5), as long as the x_i are independent between one another. This is also employed by previous work [103, Sec. 5.1], which explicitly states the importance of the independence of the mixture parameters t of the samples x. They also note two requirements for a consistent estimator. The first one is that the proposal densities $p(x|t)$ (in our notation) are *all* heavier-tailed than the target distribution. This is a simple way of expressing the need to cover the whole domain where the target is non-zero. In our case, every stochastic mixture always includes a defensive sampling lobe (BSDF or cosine hemisphere) [73]. Their second requirement is fulfilled by using MIS with the balance heuristic (deterministic mixture MIS in their terms).

We want to point out that the dependency of t_i on $t_j, j < i$ inside one pixel is weak in our case. The subgroup shuffle will eliminate any dependency completely. A small amount is reintroduced by the reuse of a neighborhood of states in a 91×91 window around a pixel. In the case of 8-TSMIS this means we will randomly select 8 out of a set of 8281 candidates, keeping the probability of collisions very low.

$$eq.\ (6.5) = E\left[E\left[\sum_{i=1}^{n} w(x_i,t_i)\frac{f(x_i)}{p(x_i|t_i)}\right]_{x_1,\dots,x_n}\right]_{t_1,\dots,t_n} \tag{6.6}$$

$$= E\left[\underbrace{\sum_{i=1}^{n}\int_X w(x,t_i)\frac{f(x)}{\cancel{p(x|t_i)}}\cancel{p(x|t_i)}\,\mathrm{d}x}_{=I,\ \text{discrete MIS}}\right]_{t_1,\dots,t_n} = I. \tag{6.7}$$

This last step reduces the problem to discrete MIS using the stochastically chosen techniques t_i. Note that it also shows that already the inner expression is the sought-for integral, taking the surrounding expectation does not change the value, i.e. there is no interplay between different sets of t_i to correct the outcome, in particular independence of t_i is not required. The one requirement it imposes is that every stochastic mixture (set of t_i) covers the full domain where $f(x) > 0$, which we achieve by defensive sampling.

7. Optimizing Path Termination for Radiance Caching Through Explicit Variance Trading

This chapter is based on L. Kandlbinder, A. Dittebrandt, A. Schipek, and C. Dachsbacher. “Optimizing Path Termination for Radiance Caching Through Explicit Variance Trading”. In: *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 7.3 (2024). DOI: `10.1145/3675381`

Radiance caching allows amortizing the cost of path tracing by sharing contributions of path suffices in a spatial data structure. This sharing generally introduces bias, but it can be traded with variance by terminating into the cache at deeper path vertices. We develop a framework to implicitly reduce bias by optimizing path termination towards a chosen variance bound. Importantly, this bound can be chosen large enough while still being amenable to denoising. This results in longer paths being sampled with unbiased path tracing as permitted by the estimator variance and variance bound. To that end, we reformulate the variance of a path tracing estimator as a quantity that can be expressed locally for a path, relying on auxiliary statistics that are shared through the radiance cache structure independently of the path prefix. This allows to perform the optimization locally during path construction, while still translating to a global bound. Our method is capable of maintaining the variance bound in complex scenes, resulting in lower bias compared to other techniques such as heuristics based on ray differentials. We additionally present first findings on directly optimizing the bias-variance tradeoff based on local bias estimates of individual cache records, although the optimization is only approximate, resulting in suboptimal termination decisions.

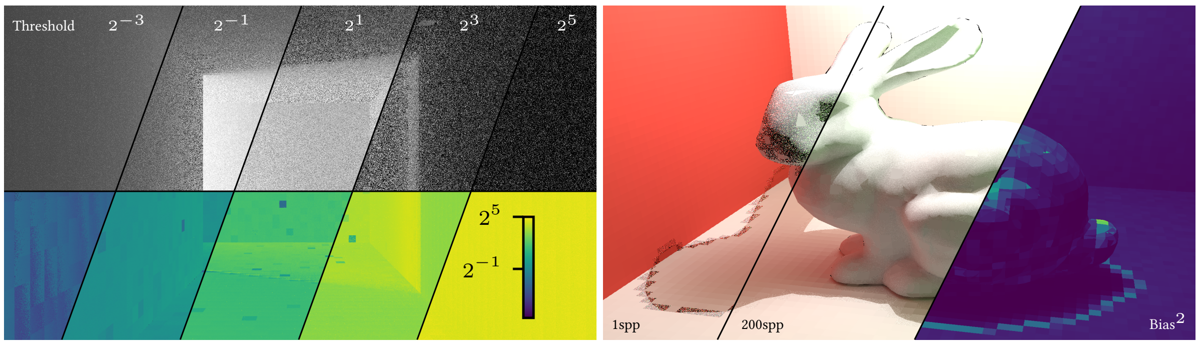


Figure 7.1.: Left: Winding corridor that requires multiple indirections to reach a light source. The variance bounding scheme is applied with increasing thresholds to allow for longer paths while bounding image variance (visualized below). Right: Bunny in a Cornell box that casts a shadow to the floor. The mean squared error minimization scheme detects cache records with high bias (right visualization) and avoids direct termination. Figure taken from [81].

7.1. Introduction

Photorealistic image synthesis based on light transport simulation provides a unified framework to handle a wide range of lighting effects. Monte Carlo integration techniques such as path tracing [79] in particular are used extensively in movie production [40]. With the advent of hardware ray tracing, Monte Carlo techniques are also starting to be adopted on the GPU for real-time rendering. However, the inherent noise due to the stochastic sampling remains an issue. Furthermore, the cost of generating individual paths is considerable due to incoherent memory accesses. Most applications therefore make only targeted use of path tracing to reproduce specific effects, such as soft shadows or diffuse global illumination.

Radiance caching [90] allows amortizing the cost of generating paths by storing cache records in a data structure that can be queried for surface points. Instead of tracing full paths per pixel, paths can be terminated early with the stored cache records used to estimate radiance of the suffix. The method is biased, however, as cache records do not store the exact radiance of a given surface point, but an average for surface points associated with a cache record. To hide bias artifacts, cache records are usually not queried at primary vertices but at deeper path vertices dictated by some heuristic.

In this chapter, we explicitly optimize radiance cache termination with two separate strategies: (1) Variance bounding termination (section 7.3.2) that bounds image variance to a configurable threshold. While bias is not explicitly accounted for, cache termination is deferred as much as possible under the variance constraint to minimize the effects of bias. An example scene of a corridor is shown in fig. 7.1 where the variance bound is varied, resulting in a corresponding image variance. (2) Approximate mean squared error minimizing termination (section 7.3.3) that, compared to variance bounding, additionally accounts for bias. This approach allows terminating paths earlier if a cache record represents its contained surface points well enough. Figure 7.1 right shows the Stanford bunny, where shadow edges are classified as regions with higher bias, forcing the path tracer to continue traversal. Both approaches build on the same principle to perform a global optimization (bounding variance or minimizing MSE) through local termination decisions. We first reformulate both variance and bias as terms that can be optimized locally for a path (section 7.3.1), allowing for evaluation inline while path tracing. We also show how to estimate variance and bias in a cache record by tracking three stochastic moments of the path tracing estimator (section 7.3.4), two of which are the first and second moment and one additional marginal second moment that distinguishes between variance and bias.

7.2. Related Work

Radiance caching The approach to amortize the cost of path tracing by storing cache records in a world-accessible data structure dates back to irradiance caching [163] which only considers diffuse interreflections. It was later extended with directional information to represent actual radiance [90]. In classic radiance caching literature, termination is performed directly at primary vertices. Therefore, many works focus on optimizing the representation to reduce visible error [90, 139, 101]. Other approaches use simpler representations but hide artifacts by deferring evaluation to deeper path vertices. Path space filtering [87] defers evaluation always to the second path vertex, which resembles final gathering. Neural radiance caching [117] uses a heuristic based on an approximate area-spread [11] of paths, that quantifies how much paths diverge. We compare against the area-spread heuristic in section 7.5.

Various cache representations were developed, such as hashed grids [58, 14], texture space [118], surfels [60], cards [166] and neural [117, 115, 59]. Our approach is largely orthogonal to those, since we only add additional quantities to the cache record. The exception are neural representations, as cache records are usually encoded opaquely in a latent vector space, making a direct application nontrivial. For simplicity, we implement our approach using hashed grids only.

Our variance estimates necessitate an iterative update scheme that is comparable to recent works [125, 33]. It transports radiance directly between caches instead of estimating radiance through path suffices.

Photon mapping Lighting effects involving specular-diffuse-specular paths are notoriously difficult to solve with path tracing. Photon Mapping [76] approximately solves this by distributing photon particles initiated from light paths across the scene and then estimate indirect illumination at a surface point using kernel density estimation. Final gathering is often applied to hide artifacts emerging from kernel density estimation by deferring photon map access after a diffuse interaction. Alternatively, the kernel width can be optimized, which is known as bandwidth selection in the wider literature and has been investigated in the context of photon mapping [83]. It is a tradeoff between bias in the form of blurring or variance due to an insufficient amount of photon particles. While radiance caching can be framed similarly, the mechanism by which we trade bias with variance (deferred path termination) needs to be formalized differently.

Adaptive sampling Initially introduced to reduce edge aliasing in image space [112], adaptive sampling allocates more sampling budget to challenging parts of the image or scene [57]. Controlled path termination can also be interpreted as an adaptive sampling technique, since early termination of paths has a similar effect.

Variance estimation Estimates of variance or second moment are also used for other purposes, for example, to steer guiding [129], control Russian roulette and splitting factors [130], or blurring radii for denoising [136].

Denoising Raw path tracer output is usually not presented directly to the user due to disturbing noise artifacts. It is instead fed into a post-processing pipeline to remove residual noise through controlled blurring. Classic approaches include edge-avoiding à-trous wavelets [136] and neural techniques [25, 148]. Our algorithms, variance bounding in particular, aim to provide output with predictable (bounded) noise characteristics.

7.3. Method

In the following, we derive algorithms to optimize radiance cache termination $l(\mathbf{X})$ based on screen-space variance and Mean-Squared Error (MSE). We do so using termination decisions that are entirely local to a path. To this end, we exploit monotonicity of the expectation operator:

$$X \leq Y \Rightarrow E[X] \leq E[Y]. \tag{7.1}$$

An inequality that holds between two random variables X and Y also holds in the expectation. By reformulating variance and MSE as expectations over path space (section 7.3.1), we can apply this

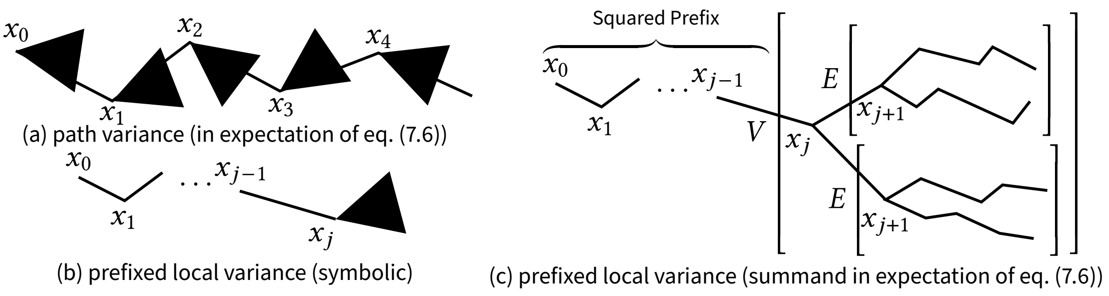

x_0
x_1
x_2
x_3
x_4
(a) path variance (in expectation of eq. (7.6))
x_0
x_1
$\dots x_{j-1}$
x_j
(b) prefixed local variance (symbolic)
Squared Prefix
x_0
x_1
$\dots x_{j-1}$
V
x_j
E
x_{j+1}
E
x_{j+1}
(c) prefixed local variance (summand in expectation of eq. (7.6))

Figure 7.2.: (a) Path variance is composed of (b) local variances of stochastic decisions at each path vertex with its prefix. (c) Local variance is defined over a marginalized suffix estimator, accounting only for the next path segment. Figure taken from [81].

reasoning: If we locally bound or minimize the inner term of the expectation, the expectation itself, and therefore the per-pixel variance and MSE, will also be bounded or minimized. In sections 7.3.2 and 7.3.3 we derive algorithms based on the reformulation to bound variance and minimize MSE on a per-pixel basis, respectively. Finally, in section 7.3.4 we show how to estimate the needed quantities for the aforementioned algorithms.

7.3.1. Reformulating variance & MSE

Reformulating variance We initially reformulate the variance for the path tracing estimator g and later relate this expression to the radiance caching estimator $\bar{g}$. We proceed as follows: We leverage the law of total variance:

$$V[g] = VE[g \mid x_1] + EV[g \mid x_1], \tag{7.2}$$

and apply it recursively by conditioning on successive path vertices:

$$V[g] = VE[g \mid x_1] + E\Big[V\big[E[g \mid x_1x_2] \;\big|\, x_1\big] + E\Big[V\big[E[g \mid x_1x_2x_3] \;\big|\, x_1x_2\big] + \dots\Big]\Big]. \tag{7.3}$$

Previous work [22] decomposes the recursion into variance terms for each level of indirection:

$$V[g] = VE[g \mid x_1] + EV\big[E[g \mid x_1x_2] \;\big|\, x_1\big] + EV\big[E[g \mid x_1x_2x_3] \;\big|\, x_1x_2\big] + \dots \quad . \tag{7.4}$$

We move all expectations out of the recursion instead:

$$V[g] = E\left[\sum_{j=0}^{\infty} V\Big[E\big[g \mid x_1 \dots x_{j+1}\big] \;\Big|\, x_1 \dots x_j\Big]\right]. \tag{7.5}$$

Note that the range $x_1 \dots x_j$ is empty for $j = 0$. This expression is already in the form of an expectation over path space, as required for the local optimization. We perform one additional step: The terms that only depend on conditioned vertices inside the variance can be factored as a squared prefix g_j^{pre} with the suffix g_j^{suf} remaining in the variance term:

$$V[g] = E\left[\sum_{j=0}^{\infty} \left(g_j^{\text{pre}}\right)^2 \underbrace{VE\left[g_j^{\text{suf}} \,\middle|\, x_{j+1}\right]}_{\text{Local variance (estimation in section 7.3.4)}}\right] = E\left[\sum_{j=0}^{\infty} P_j[g]\right]. \tag{7.6}$$

Due to the conditional expectation, the variance term expresses the local variance at vertex x_j. This is not to be confused with the variance of the suffix estimator. We refer to the combined term of squared prefix and local variance as prefixed local variance P_l. While the squared prefix can be readily computed for a path, the local variance needs to be estimated separately (section 7.3.4). We refer to the sum of prefixed local variances as path variance (fig. 7.2).

Comparing g to $\bar{g}$, it can be observed that $\bar{g}$ is a truncated version of g. This translates analogously to the variance, where the number of sum terms is bound by $l(\mathrm{X})$:

$$V[\bar{g}] = E\left[\sum_{j=0}^{l(\mathrm{X})-1} P_j[\bar{g}]\right]. \tag{7.7}$$

Reformulating bias The bias of $\bar{g}$ is defined with respect to the original estimator g:

$$\mathrm{Bias}(\bar{g}, E[g])^2 = (E[\bar{g}] - E[g])^2 . \tag{7.8}$$

We can factor out the common prefix of both estimators to retrieve a form that relates the bias to the local biases introduced through caching:

$$\mathrm{Bias}(\bar{g}, E[g])^2 = E\left[g^{\mathrm{pre}}_{l(\mathrm{X})}\left(\mathrm{rc}\left(x_{l(\mathrm{X})-1}, x_{l(\mathrm{X})}\right) - E\left[g^{\mathrm{suf}}_{l(\mathrm{X})}\right]\right)\right]^2. \tag{7.9}$$

This form is still not useful because the outer expectation is squared, so we bound this term from above using Jensen's inequality [78] and move the square into the expectation:

$$\mathrm{Bias}(\bar{g}, E[g])^2 \leq E\Bigg[\underbrace{g^{\mathrm{pre}}_{l(\mathrm{X})}\left(\mathrm{rc}\left(x_{l(\mathrm{X})-1}, x_{l(\mathrm{X})}\right) - E\left[g^{\mathrm{suf}}_{l(\mathrm{X})}\right]\right)^2}_{\text{Local bias (estimation in section 7.3.4)}}\Bigg] = E\left[B_{l(\mathrm{X})}[\bar{g}]^2\right]. \tag{7.10}$$

This approximation neglects cases where biases of different paths cancel each other out, since this effect is hard to determine when considering just individual paths. Similar to local variance, the local bias also needs to be estimated separately (section 7.3.4). In the following, we reference this bias term as $B_{l(\mathrm{X})}$.

Reformulating MSE Combining the reformulated variance and bias (eqs. (7.7) and (7.10)) gives the reformulated MSE:

$$\mathrm{MSE}\left(\sum_N \frac{\bar{g}}{N}, E[g]\right) = \frac{V[\bar{g}]}{N} + \mathrm{Bias}(\bar{g}, E[g])^2 \leq \frac{1}{N}E\left[\sum_{j=0}^{l(\mathrm{X})-1} P_j[\bar{g}]\right] + E\left[B_{l(\mathrm{X})}[\bar{g}]^2\right]. \tag{7.11}$$

Note that the reformulated MSE is an upper bound of the true MSE due to the approximate bias term. The respective expectations can be joined to receive an expression that can be evaluated individually for each path:

$$\mathrm{MSE}\left(\sum_N \frac{\bar{g}}{N}, E[g]\right) \leq E\left[\sum_{j=0}^{l(\mathrm{X})-1} \frac{P_j[\bar{g}]}{N} + B_{l(\mathrm{X})}[\bar{g}]^2\right]. \tag{7.12}$$

Algorithm 5: Basic path tracer augmented with variance trading

```
fn trace_path(pos, dir, thres, target, margin)
    thp ← 1; con ← 0;
    path_var ← 0; alpha ← 1;
    pos ← trace_ray(pos, dir); ;                                // terminate if no hit
    loop
        pl_var ← $thp^2$ · get_local_variance(pos, −dir);
        if path_var + pl_var > thres then
            alpha ← $\sqrt{\frac{thres - path\_var}{pl\_var}}$;              // eq. (7.15)
        pl_bias ← $thp^2$ · get_local_bias_sq(pos, −dir);
        pd_bias ← $thp^2$ · get_deferred_bias_sq(pos, −dir);
        pd_bias ← min (0.9 · margin · pl_bias, pd_bias);
        c_mse ← path_var/target + pl_bias;
        d_mse ← (path_var + pl_var)/target + pd_bias;
        if margin · c_mse < d_mse then
            alpha ← 0;
        con ← con + (1 − alpha) · thp · rc(pos, −dir);
        con ← con + alpha · next_event_estimation(pos, −dir);   // accounting for interpolation
        dir, weight ← sample_bsdf(pos, −dir);
        thp ← thp · weight;
        pos ← trace_ray(pos, dir);
        if alpha < 1 then                                        // terminate into next cache
            con ← con + alpha · thp · rc(pos, −dir);
            break;
        path_var ← path_var + pl_var;
    return con;
```

Variable Descriptions

Shared		Variance bounding	
pos	current surface position	thres	absolute variance threshold
dir	current incoming direction	MSE minimization	
thp	path throughput	target	sample target to optimize MSE for
con	path contribution	margin	margin factor
path_var	path variance	pl_bias	prefixed local squared bias
alpha	interpolation weight 1 = path tracer, 0 = current cache	pd_bias	prefixed deferred squared bias
pl_var	prefixed local variance	c_mse	deferred MSE when adding one more segment
		d_mse	current MSE when terminating at current surface

7.3.2. Variance bounding termination

The goal of the variance-bounding termination strategy is to terminate paths such that the resulting image variance is bounded. The key idea is that the reformulated variance in eq. (7.7) allows us to perform this bounding locally for individual paths while still translating to a global bound for the image. $l(\mathbf{X})$ can be defined directly based on the path variance:

$$l(\mathbf{X}) = \underset{l \in \mathbb{N}}{\arg\max} \left(\sum_{j=0}^{l-1} P_j[\bar{g}] \leq t \right) = \underset{l \in \mathbb{N}}{\arg\max} \sum_{j=0}^{l-1} \left(g_j^{\text{pre}}\right)^2 VE\left[g_j^{\text{suf}} \,\middle|\, x_{j+1}\right] \leq t. \tag{7.13}$$

This means that the termination depth is maximized under the variance constraint.

Variance interpolation Termination into the cache is performed even if the prefixed local variance P_j of the next vertex would increase the sum just slightly beyond the threshold t. For such cases it is

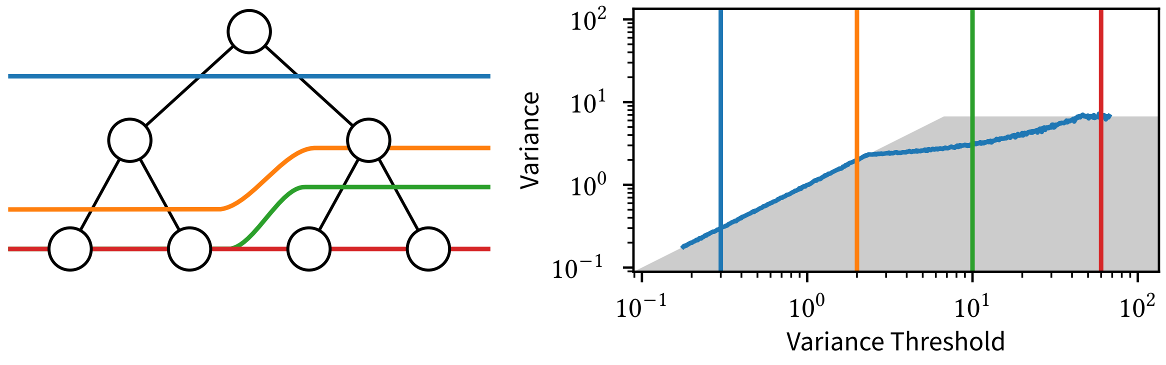


Figure 7.3.: Discrete test case of variance bounding path tracer implementation (algorithm 5). Left: Sampling tree consisting of leaf nodes that emit a constant energy of one unit. Left subtrees are sampled with 90% probability. Right: MSE of path tracer sampling the tree depending on the variance threshold. Colored lines: Termination of path tracer depending on variance threshold, which can be between a node and its children due to interpolation. Figure taken from [81].

sensible to blend the contribution of the cache with the contribution of one additional segment using an alpha factor α:

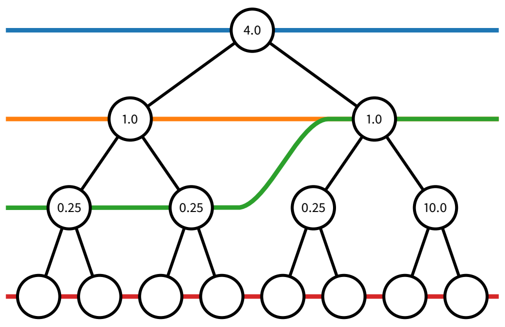
$$\sum_{j=0}^{l(\mathrm{X})-1} P_j[g] + P_{l(\mathrm{X})}[\alpha g] = \sum_{j=0}^{l(\mathrm{X})-1} P_j[g] + \alpha^2 P_{l(\mathrm{X})}[g] = t. \tag{7.14}$$

The additional $P_{l(\mathrm{X})}$ is scaled by α^2. We can solve this expression for α:

$$\alpha = \sqrt{\frac{t - \sum_{j=0}^{l(\mathrm{X})-1} P_j[g]}{P_{l(\mathrm{X})}[g]}}. \tag{7.15}$$

Path tracer integration We have sketched out a basic path tracer implementation with next event estimation in algorithm 5 that incrementally constructs a path by maintaining the current surface point (pos, dir) as well as the path throughput (thp) and contribution (con). The code with blue background depicts the variance bounding strategy. The path variance (path_var) can be constructed incrementally by adding the prefixed local variances (pl_var) of each sampled vertex to a cumulative sum. Path termination is decided in line 7 using the variance threshold (thres). When performing interpolation, the computed alpha (alpha) term also needs to be incorporated into any direct illumination (line 17) in addition to the reflected radiance at the next vertex as given by the cache.

Validation To validate the theory, we have implemented the variance bounding strategy to estimate the contribution of a discrete tree (fig. 7.3). The leaf nodes emit one unit of energy, which is transported along the edges towards the root node. Left edges are selected with 90% probability. The discrete nature of the test case allows computing the total reflected energy and local variances of each node in closed form. The plot shows the estimated root mean squared error depending on the variance threshold. We can see that the MSE initially closely follows the threshold. But eventually the variance detaches from the threshold and then converges to a maximum value. The latter happens because the underlying estimator has finite variance, i.e. no path termination takes place. The former is due to the left subtree contributing less variance, thus being fully expanded earlier. Since each path tries to match the variance threshold individually, the MSE will be smaller than the threshold as soon as some subtrees are completely expanded.



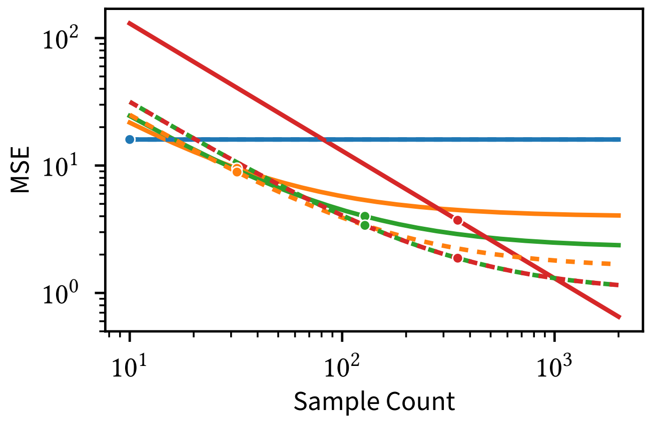


Figure 7.4.: Discrete test case of MSE minimizing path tracer implementation (algorithm 5). Left: Sampling tree consisting of leaf nodes that emit a constant energy of one unit and inner nodes with biased radiance caches (the value in the node denotes the bias). Left subtrees are sampled with 90% probability. Right: MSE of path tracer sampling the tree, depending on actual sample count for different sample targets (dots represents the sample target). The dashed lines represent the optimal termination, which the algorithm does not attain in general. Figure taken from [81].

7.3.3. MSE minimizing termination

The MSE minimizing termination strategy (algorithm 5, red background) strives to minimize the error in the image for a given number of samples to accumulate. We use the reformulated MSE (eq. (7.12)) and minimize the inner part of the expectation on a per-path basis. Due to incremental path construction, only information associated with the path prefix can be used. Therefore, minimization is performed in a greedy fashion: Before continuing a path, the current MSE (c_mse, line 12) is compared (line 14) against the deferred MSE (d_mse, line 13), which includes an estimate of average bias when terminating at the next vertex (pd_bias)

Heuristic factors In part due to the bounded bias term, the aforementioned comparison is invariant of the prefix - most terms cancel, including the path throughput. This means that termination is entirely dependent on the local statistics of a cache record. To counter this, we introduce a margin factor (line 14) that allows the current MSE to be slightly larger. Thus, the prefix does not cancel entirely and is used to gauge the relative impact on the image. Additionally, in the limit (for infinite sample counts), termination will always be performed if the deferred bias is larger than the local bias. We clamp the deferred bias to 90% below the local bias (line 11, also accounting for the margin factor) to ensure that for large sample counts, termination is always deferred. Note that these terms only heuristically solve some of the issues arising from our approximation. More principled approaches are discussed in section 7.6.

Validation Figure 7.4 shows a discrete test case of the MSE minimization. Compared to fig. 7.3, caches of inner nodes are now biased. The plot shows the MSE depending on the sample count for different sample targets (colored lines). The dot of each line represents the configured sample target, i.e. for that target it should have the lowest MSE among the other lines. We additionally plot the MSE for optimal termination decisions based on an exhaustive search (dashed lines). As can be seen, decisions are not optimal in general, since the optimal termination is often not found and the switch to another termination decision with more variance is made too early (discussed further in section 7.6).

7.3.4. Estimating local variance & bias

Our termination strategies depend on additional local statistics that are maintained for the individual cache records of the radiance cache structure. We will initially present how to estimate the local variance and later separate this quantity from the introduced bias due to averaging of contributions in a cache record.

Estimating local variance To reiterate, the reformulated variance (eq. (7.6)) consists of a sum of prefixed local variances that can be factored into a path-dependent prefix and a local variance $VE[g_j^{\mathrm{suf}} \mid x_{j+1}]$. This separation allows tracking the local variance individually for each partition, while the path-dependent prefix can be incorporated in an ad-hoc fashion. However, the nested expectation complicates direct estimation, especially for the second moment.

We leverage that the radiance cache itself can be used to approximate this expectation. Instead of collecting the moments of the suffix estimator, we construct an estimator that directly terminates at the next vertex and reads the stored radiance cache value. This leads to an iterative update scheme comparable to recent works [125, 33], where radiance cache values of previous frames are propagated forward (section 7.4).

Differentiating between variance and bias As a cache record is shared between vertex pairs in a partition, its radiance estimate is generally biased for all of them. The plain variance estimate based on the first and second moment consistently overestimates the true variance in such a case. This is precisely due to the bias induced by the shared accumulation.

To separate both quantities, we explicitly compute the average variance and squared bias (the plain average bias is always zero) in a partition as the expectation of variance and bias over all vertex pairs. Generally, the average variance of a random variable Y conditioned to y ($Y = E[g_1^{\mathrm{suf}} \mid x_0, x_1]$ and $y = (x_0, x_1)$ in our case) can be decomposed into the following moments (derivation in appendix A):

$$E\left[V\left[Y \mid y\right]\right] = E\left[Y^2\right] - E\left[E\left[Y \mid y\right]^2\right]. \tag{7.16}$$

Similarly, the average squared bias of a random variable Y conditioned to y with respect to the expectation of Y can be decomposed into these moments:

$$E\left[\mathrm{Bias}(Y \mid y, E[Y])^2\right] = E\left[E\left[Y \mid y\right]^2\right] - E\left[Y\right]^2. \tag{7.17}$$

Therefore, to distinguish between variance and bias, we need to track three moments: $E[Y]$, $E\left[Y^2\right]$ and $E\left[E\left[Y \mid y\right]^2\right]$. The first two are the usual first and second moments for variance or error computation. The latter is a marginal second moment. This term is what distinguishes between variance and bias, given that it is part of both the average variance and bias. Importantly, when adding both expressions together, the marginal second moment cancels, and we are left with only the first and second moment, which gives the original overestimated variance. Figure 7.5 shows a visualization of the three moments in the bunny scene. The region marked by the green square is uniformly lit. Here, the marginal second moment almost coincides with the squared first moment, thus all error is attributed to variance. In the region marked by the blue square, the marginal second moment shifts towards the second moment, thus some error is also attributed to bias. Note that in practice, the squared bias can often be much smaller than the variance. But since it does not shrink with sample count, it can still dwarf variance of the N-sample estimator at moderate sample counts.

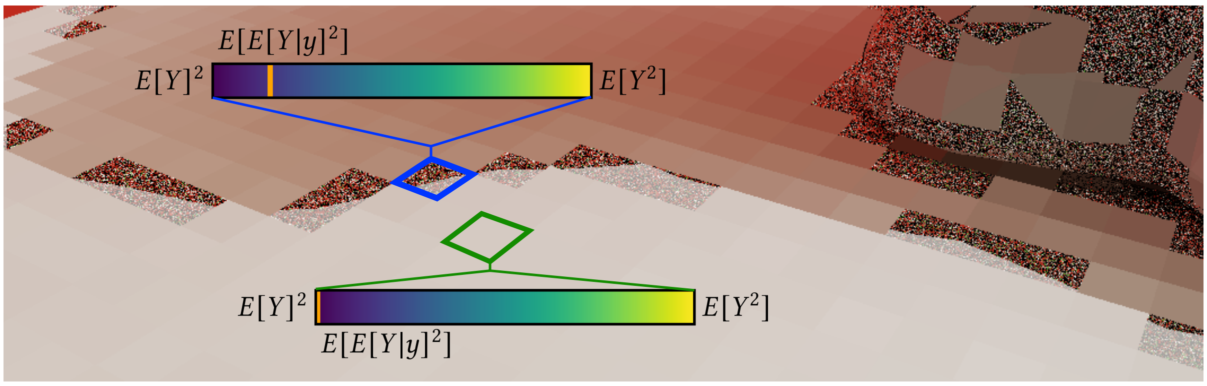


Figure 7.5.: Visualization of the marginal second moment $E\left[E\left[Y \mid y\right]^2\right]$ (orange line) in ratio of the squared first moment $E[Y]^2$ and the second moment $E\left[Y^2\right]$ for two different cache records. The green cell is uniformly lit, causing the marginal second moment to coincide with the squared first moment, indicating not much bias. The blue cell is crossed by a shadow edge, and the differently-lit regions cause the marginal second moment to shift more towards the second moment, indicating bias. Figure taken from [81].

Estimating marginal second moment The marginal second moment is constructed as an expectation over a squared expectation $E\left[E\left[Y \mid y\right]^2\right]$. This makes it challenging to directly estimate it. A naïve approach is to collect first moment estimates at a higher resolution with respect to the cache record. This allows us to quantify bias through the variation of the high-resolution first moment estimates, although this approach is both computation and memory intensive.

One approach to unbiasedly estimate this quantity without this memory overhead is to leverage that the expectation of the product of two independent realizations of a random variable is equal to the product of their expectations: $E[X_0X_1] = E[X_0]E[X_1]$. That is, we need to draw two independent samples from Y conditioned to y to obtain an unbiased estimate of $E\left[Y \mid y\right]^2$, the product is an unbiased estimate of squared expectation. This estimator tends to have very high variance, thus needing many samples. The application of U-statistics [94], which have also been applied before in computer graphics [89], allows amortizing the cost of taking many independent samples by computing the product between all sample pairs, effectively resulting in a quadratic sample count.

Another approach is to take many independent samples conditioned to one randomly chosen y upfront and estimate the inner expectation. This only approximates the inner expectation due to a finite sample count, which results in a consistent overestimation of the marginal second moment (and as a consequence over-estimated variance and under-estimated bias) depending on the sample count.

In either case, estimating this quantity requires taking at least two (but ideally more) independent samples, complicating direct estimation in a forward path tracer, but the storage overhead of collecting moments at higher resolution is mitigated. We therefore run a separate compute dispatch to update cache records (see section 7.4 for implementation details).

7.4. Implementation details

We implemented our termination strategies in the Falcor rendering framework [80] in a basic path tracer with next event estimation (NEE) and BSDF-sampling, combined with MIS for direct illumination [150]. The path tracer is implemented in a compute shader using ray queries. For the radiance cache, we use hashed grids [58, 14] as a spatial data structure.

Updating cache records Typical radiance caching implementations focus on updating cache records actually accessed by the path tracer (e.g. by sparsely generating suffix paths when termination is performed at a cache record [117]). In our instance, however, local variance and bias estimation require relatively stable radiance estimates at the next path vertex. We therefore update cache records in a separate compute shader to ensure that cache record updates are independent of how often the path tracer reaches a cache record. The same importance sampling methods as in the path tracer are used to ensure that variance estimates match the path tracer. The radiance cache maintains four vertices per cache record to be used by the compute dispatch to update the radiance estimate of the cache record, although other storage layouts independent of the radiance cache structure are also thinkable. The path tracer stores primary vertices with a 1% probability, while the compute dispatch itself also stores hit vertices in order to explore the scene and frequently refresh the set of representative vertices.

The update scheme is iterative: At each vertex location, a BSDF sample is drawn, and the incoming radiance is estimated by evaluating the radiance cache at the hit location, thus propagating radiance over multiple frames. This approach is comparable to recent works [125, 33]. We also exploit this iterative update scheme to compute the local variance (section 7.3.4), since the contained expectation over the path suffix is conveniently approximated with the radiance cache.

Computation of the marginal second moment requires drawing multiple samples at the same vertex location and computing products between all sample pairs. This is straightforward with a separate compute dispatch because we can leverage SIMT-layout of the hardware and draw 32 samples for a given vertex in one subgroup and use register permutations to exchange sample contributions, thus reducing register pressure and increasing coherence.

Relative variance bounding Variance is a scale-dependent quantity, which complicates finding good thresholds. If some estimate of pixel brightness is available (e.g. through denoising or the radiance cache itself), the variance threshold can be interpreted as a relative quantity and the absolute threshold that is needed by the algorithm in algorithm 5 can be recovered by multiplying with the squared pixel brightness. In our implementation, we use the radiance cache value of the first hit to approximate pixel brightness.

Interpolating cache records Directly using the radiance estimate of a single cache record corresponding to a vertex leads to noticeable grid artifacts (fig. 7.1 right). These artifacts also tend to be preserved by denoisers. A straightforward and inexpensive solution is to perform stochastic interpolation between neighboring cache records by jittering the vertex position before access [14]. This approach introduces additional variance that our variance estimates do not account for. However, since estimates of neighboring cells are usually similar in scale, this variance can likely be neglected when comparing to the variance of path tracing.

7.5. Evaluation

We initially analyze the behavior of our termination strategies (with parameters t for the variance threshold and s for the MSE sample target) and then continue with a comparative evaluation to the area-spread heuristic [117] (relative spread of $c = 0.01$, unless stated otherwise) with respect to quality and performance. We use both custom and openly available scenes [15] and modified them if needed to force more complex light paths that require more indirect bounces, as those are the main use case for our techniques. We chose to only use diffuse materials for now. While the radiance cache can be trained

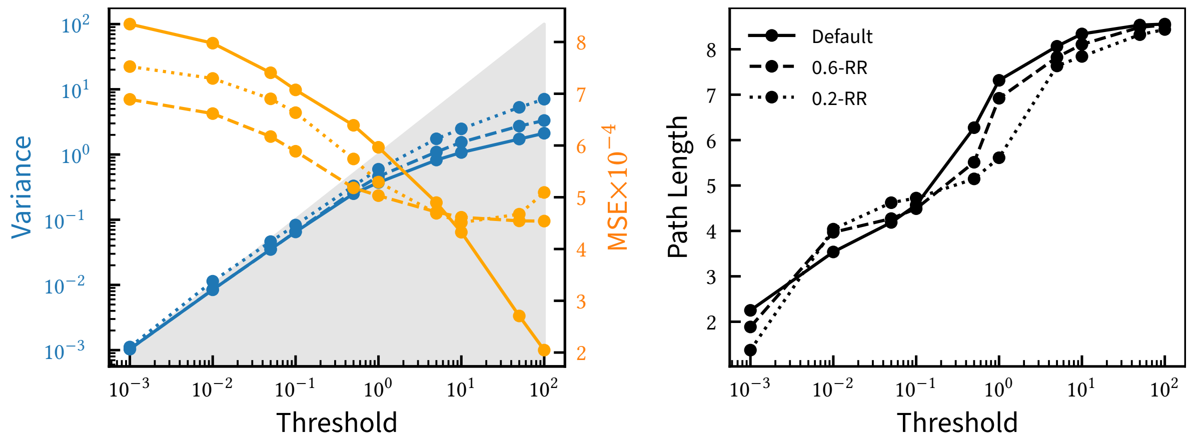


Figure 7.6.: Left: Variance bounding on the Veach Door scene for different Russian-Roulette survival probabilities. A lower variance threshold (blue) increases the MSE (orange) due to biased radiance estimates. Note that the MSE increases for low survival probabilities at high thresholds due to images not being completely converged. Right: Increased variance thresholds increase path length. Figure taken from [81].

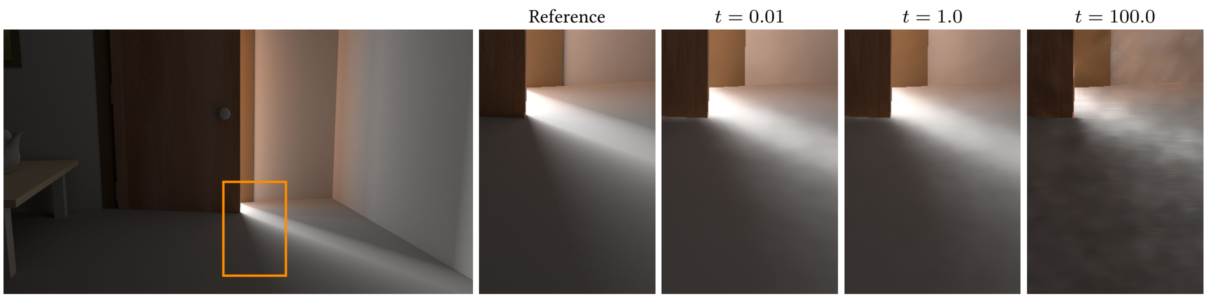


Figure 7.7.: Testcase of Veach Door scene showing the impact of variance bounding termination with stochastic interpolation on denoising. Higher thresholds increase variance, leading to more unstable output. Figure taken from [81].

for specular data, e.g., using a directional histogram, it becomes very memory-intensive. Performance measurements were taken on an Nvidia RTX 3070 Ti at a resolution of 1920×1080. As we focus on static scene configurations, we fill and propagate the radiance cache in a preprocess until it reaches an equilibrium state.

Variance bounding Similar to our detached test case (fig. 7.3) we evaluate behavior of the variance bounding strategy in the Veach Door scene (fig. 7.6). We measure variance and bias depending on the variance threshold (fig. 7.6, left) as well as path length (fig. 7.6, right). Note that path length accounts for variance interpolation in this case. To analyze termination behavior depending on different importance sampling strategies, we introduce probabilistic path termination through Russian-Roulette [5] with different survival probabilities (100%, 60% and 20%). We can generally observe a similar behavior as in the detached test case: For all survival probabilities, the variance bound is initially closely followed and eventually detaches because the underlying estimator has finite variance. Path length increases while bias decreases for increasing thresholds. Of note is that bias is dependent on the survival probability: Since low survival probabilities introduce more variance, paths are terminated earlier to maintain a given variance thresholds, increasing variance.

The controlled bias-variance tradeoff in our technique can be used to generate images that are more suitable for denoising. We use the SVGF [136] denoiser and use its result after four still frames. The output of the denoiser for different relative variance thresholds with stochastic interpolation are shown

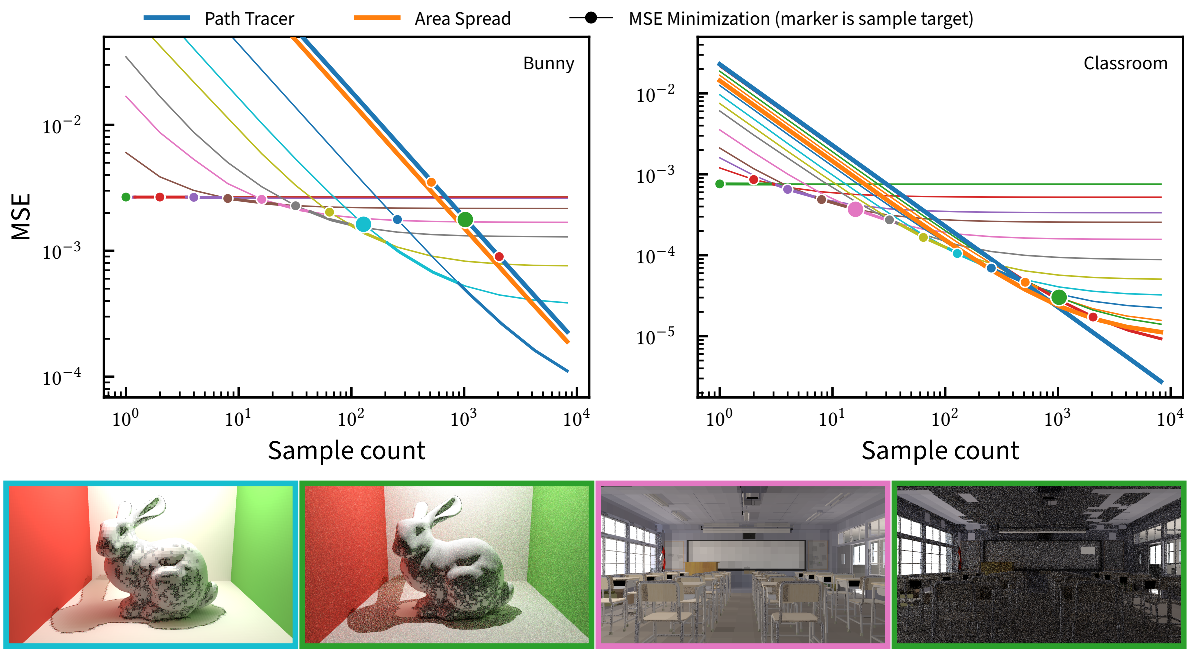


Figure 7.8.: Behavior of MSE minimization in two scenes (Bunny, Classroom) at different sample targets, plotted as MSE depending on sample count. Each line represents one sample target, signified by the marker. The plain path tracer and area-spread heuristics are also given for reference. For the enlarged markers, one-sample images are shown below. Figure taken from [81].

in fig. 7.7. A low threshold, e.g., taking the biased estimate from the radiance cache at the primary hit, produces structured artifacts. A high threshold, e.g., the noisy output of the plain path tracer, produces temporally unstable images. Choosing a suitable variance threshold achieves a balance that hides cache artifacts but also provides a more stable input image for the denoiser.

MSE minimization We evaluate how the MSE minimization strategy behaves in the Bunny and Classroom scenes in fig. 7.8, similar to the detached test case in fig. 7.4. The lines represent individual sample targets (signified by the marker), plotted as MSE over sample count. Ideally, each line should be globally minimal at its sample target, which can be partially observed. Especially the Bunny scene contains overshoots, where path terminations are made too late and results in overall higher error. However, for sample counts below 100, termination decisions are generally favorable. This is likely because around this region, deferring happens at primary vertices, where the approximation is more accurate.

Comparison to area-spread heuristic A comparison for different scenes between our techniques and the area-spread heuristic [117] is shown in fig. 7.9. All comparisons are with equal sample count and using stochastic interpolation. The main difference between our techniques and the area-spread heuristic is that we explicitly estimate error to optimize termination, while the area-spread heuristic estimates how well unquantified error is hidden through an approximate ray differential. The area-spread heuristic never terminates at a primary hit, which makes it less prone to discretization artifacts, but after that it generally terminates early, except for corner regions where distances to the next vertex can be very short. Earlier termination can sometimes be beneficial, as scenes like corridor show.

This aspect is even more pronounced in the Veach door scene (fig. 7.10), where the incoming lighting from the door gap is estimated with high variance, rendering the output image unusable for denoising.

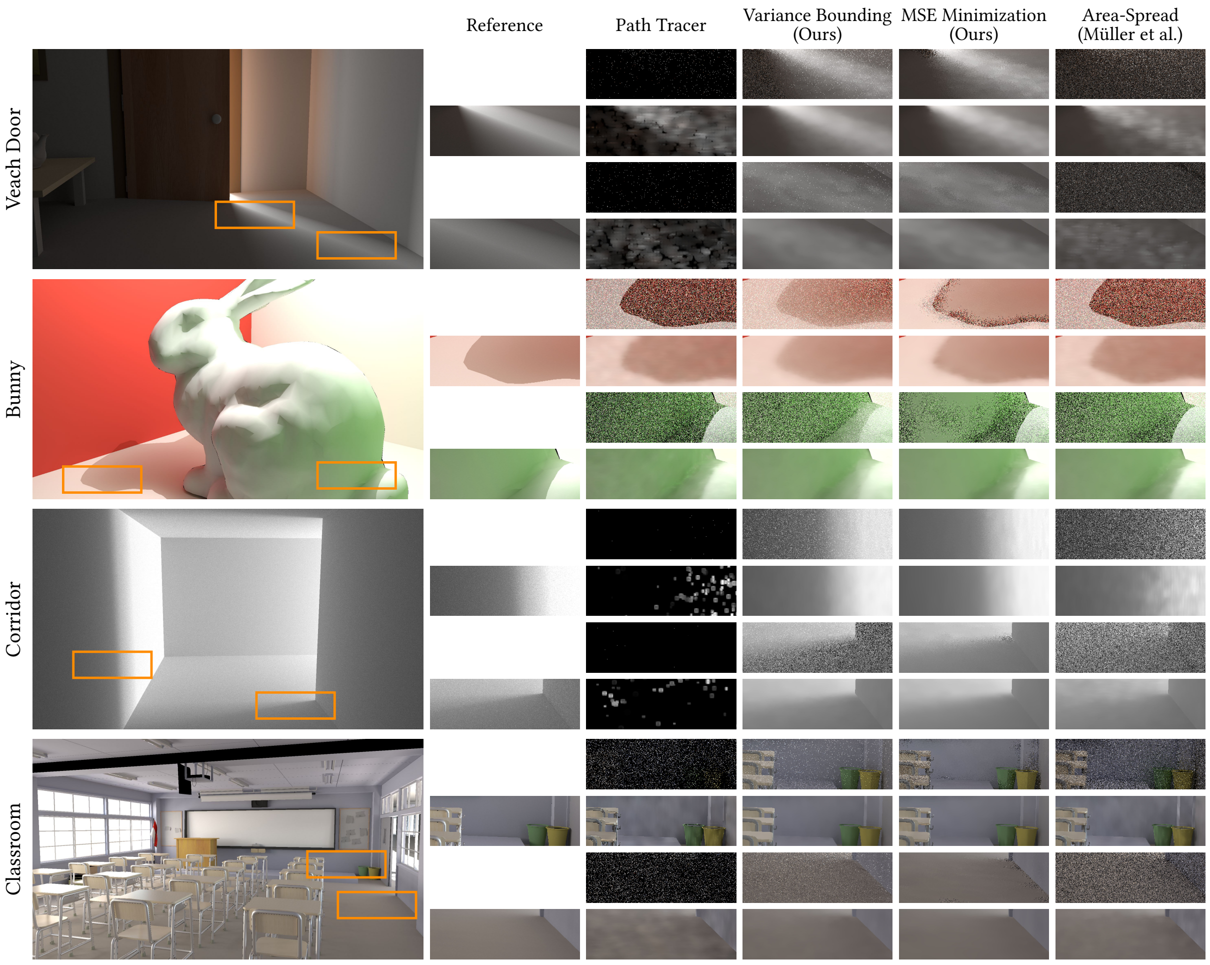


Figure 7.9.: Equal-sample comparison of termination strategies with and without denoising for different scenes: The plain path tracer vs. our variance bounding (relative variance threshold $t = 1.0$) vs. our MSE minimization (sample target $s = 100$) vs. the area-spread heuristic ($c = 0.01$). The first row shows the 1spp output, the second row the output of the denoiser [136] after four still frames. Figure taken from [81].

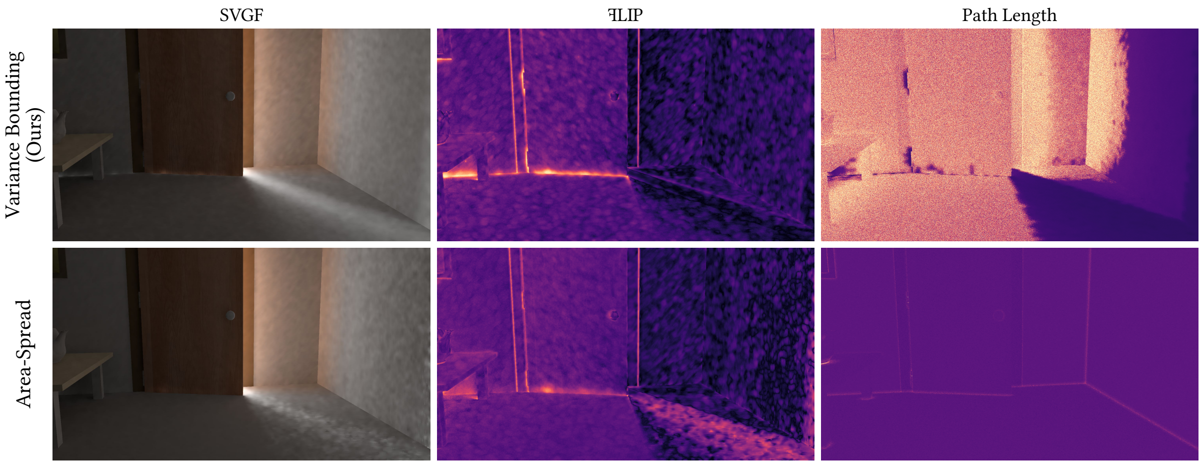


Figure 7.10.: Equal-sample comparison between our variance bounding (relative variance threshold $t = 10.0$) and the area-spread termination. Left: output of the denoiser. Middle: ꟻLIP error metric [2] compared to the reference. Right: visualization of the average path length per pixel. Apart from the edges of the room, the area-spread heuristic terminates quite early. Our variance bounding can adapt to the variance caused by indirect illumination. Figure taken from [81].

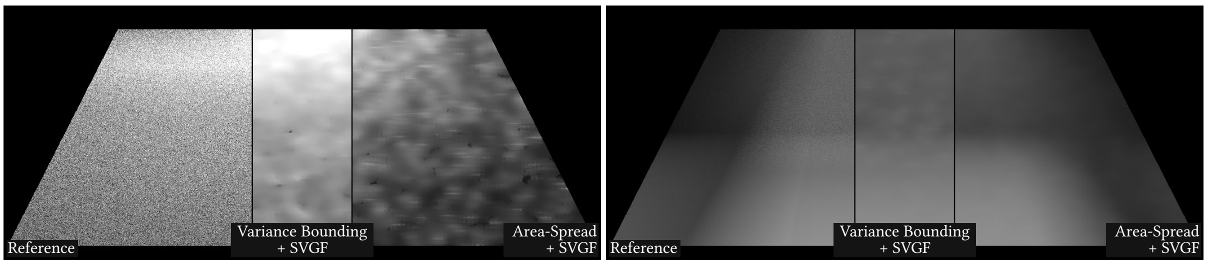


Figure 7.11.: Equal-sample comparison of variance bounding and area-spread. Left: scene indirectly illuminated by a small gap in a box containing a light source, causing high variance when not terminating at primary vertex. Right: surface indirectly illuminated by glossy reflection of an area light, causing bias when terminating at secondary vertex. Area-spread ($c = 0.01$) terminates at the second vertex in both cases, causing high variance and bias, respectively. Variance bounding ($t = 1.0$) terminates at the primary vertex in the left case and at the light source in the right case, resulting in lower variance and bias, respectively. Figure taken from [81].

Scene	Method	Total Frame [ms]	Tracer [ms]	Propagation [ms]	Path Length
Door	Var. Bound. ($t = 0.01$)	21.71	6.05	10.68	4.05
	Var. Bound. ($t = 0.10$)	22.64	6.94		4.86
	Var. Bound. ($t = 1.00$)	26.08	10.46		7.41
	MSE Min. ($s = 100$)	16.90	0.67		1.01
	Area-Spread ($c = 0.01$)	17.85	1.86	9.86	1.97
	Area-Spread ($c = 100$)	19.85	4.73		3.18
Corridor	Var. Bound. ($t = 0.01$)	12.46	1.39	6.22	1.97
	Var. Bound. ($t = 0.10$)	15.42	4.35		4.24
	Var. Bound. ($t = 1.00$)	15.70	4.63		4.88
	MSE Min. ($s = 100$)	13.23	0.53		1.00
	Area-Spread ($c = 0.01$)	11.96	1.48	5.64	1.98
	Area-Spread ($c = 100$)	15.02	4.49		4.18

Table 7.1.: Frame time comparison between our variance bounding (absolute variance threshold t), our MSE minimization and the area-spread heuristic. The threshold indirectly also bounds the path length. Our methods share the same radiance propagation algorithm. The area-spread heuristic does not require the additional statistics, resulting in lower execution times of the propagation pass. Figure taken from [81].

The variance bounding strategy has explicit knowledge of the local variance, and can therefore terminate directly at the primary vertex. On the other hand, the region under the door is better handled by the area-spread heuristic, since variance bounding estimates high variance in this region, leading to earlier termination.

Figure 7.11 summarizes the prime cases where termination at the secondary vertex is not beneficial: (1) when terminating at the secondary vertex causes high variance (left image). In this instance, it is caused by incoming radiance at the surface being confined to a small solid angle, which is challenging to estimate with plain BSDF sampling. (2) When terminating at secondary or later vertices introduces high bias (right image). In this instance, a glossy reflection at the secondary vertex introduces bias. No parameterization of area-spread can give satisfactory results for both cases, while the output of variance bounding gets closer to the reference.

Performance measurements Table 7.1 shows performance measurements of our variance bounding and MSE minimization strategies compared to the area-spread heuristic with different parameterizations. We report execution times of the path tracer and propagation pass separately, since updating the cache is mostly an orthogonal issue. We additionally report the average path length depending on the parameterization. We generally observe the area-spread heuristic (for its recommended parameter $c = 0.01$) to terminate paths mostly at the first indirection, which is consistent with an average path length of almost 2 in both scenes. Larger variance thresholds generally introduce longer paths, resulting in longer execution times. Since our methods need to access the radiance cache from global memory at every surface interaction to decide whether to terminate, we have an additional runtime overhead compared to the area-spread heuristic. The overhead seems to be small in practice, but more pronounced with more indirections, given that cache access at the primary vertex is likely faster due to memory coherence. The propagation pass in the case of area-spread has lower execution times, since it only needs to maintain radiance estimates, as opposed to additional variance and bias estimates. An additional speedup can be expected from lowering the sample count, since plain radiance estimates have lower variance.

7.6. Discussion & Future Work

Path termination The variance bounding termination strategy maximizes path length under a variance constraint. The rationale is that longer paths lead to lower bias, since errors due to radiance caching are reduced at deeper path vertices. However, since the algorithm is oblivious to the actual error reduction, there is no guarantee that error is reduced significantly. We have shown instances where this approach is indeed meaningful: Forcing early termination when local variance is very large (fig. 7.10) and continuing traversal to avoid local bias if the variance threshold permits (figs. 7.10 and 7.11).

Optimality of MSE minimization As we only track the second moment of bias as an upper bound of the squared first moment, the impact of bias is overestimated, leading to path termination being deferred further than necessary (this is already observable in the theoretical test case). This approximation foregoes the non-local cancellation of biases in unrelated parts of the path tree, since all biases are positivized through squaring. This especially affects gradient biases that are likely hidden well through cache interpolation. Finding better approximations to capture these interactions can lead to more informed termination decisions.

Efficient bias estimation While the estimator of the marginal second moment eliminates the prohibitive memory cost of a high-resolution intermediate representation, it possesses quite strong variance in general, requiring large sample counts for stable estimation. While the application of U-statistics offsets this cost to some degree, it is still not applicable in a dynamic context where short sample histories have to be maintained in order to react to changes in illumination. More approximate approaches could be conceivable, as well as additional filtering on the raw moment estimates to stabilize the output at lower sample counts.

Efficiency awareness Minimizing MSE has a similar issue as variance bounding in that paths are continued even if the error reduction is marginal. While we heuristically approach this problem with the margin factor (terminating even if the MSE when continuing would be slightly smaller), a more principled approach to optimize efficiency directly could prove fruitful.

Applications of bias estimation Knowing biases in radiance caching can be a useful tool for other purposes, too. One example is to control refinement of the cache structure towards regions of high bias. This might also allow for more efficient estimation, given that less cache records compared to a naïve grid would be present.

7.7. Conclusion

Optimizing path termination into radiance caches has the potential to give dramatic improvements in image quality for a given sample budget and reduce overall rendering times. We have conceived a mathematical framework to tackle the bias-variance tradeoff using local termination decisions and presented two algorithms, that optimize the tradeoff differently. The variance bounding termination strategy is capable of maintaining a given variance target with implicit bias reduction. With the MSE minimization strategy, we have made an initial step towards exploiting the cache structure to achieve lower error for a given number of samples to accumulate, which can be especially useful for highly limited sample budgets. While the results are promising, future work should focus on finding tighter approximations to achieve better tradeoffs.

Appendix

A. Moment derivation for average variance & bias

Here, we derive the moments of average variance and squared bias of a random variable Y conditioned to y (section 7.3.4). The squared bias is defined with respect to $E[Y]$:

$$\begin{aligned} E\left[V\left[Y \mid y\right]\right] &= E\left[V\left[Y \mid y\right]\right] \\ &= E\left[E\left[Y^2 \mid y\right] - E\left[Y \mid y\right]^2\right] \\ &= E\left[E\left[Y^2 \mid y\right]\right] - E\left[E\left[Y \mid y\right]^2\right] \\ &= E\left[Y^2\right] - E\left[E\left[Y \mid y\right]^2\right], \end{aligned}$$

$$\begin{aligned} E\left[\mathrm{Bias}(Y \mid y, E[Y])^2\right] &= E\left[\left(E\left[Y \mid y\right] - E\left[Y\right]\right)^2\right] \\ &= E\left[E\left[Y \mid y\right]^2 - 2E\left[Y \mid y\right]E\left[Y\right] + E\left[Y\right]^2\right] \\ &= E\left[E\left[Y \mid y\right]^2\right] - E\left[2E\left[Y \mid y\right]E\left[Y\right]\right] + E\left[E\left[Y\right]^2\right] \\ &= E\left[E\left[Y \mid y\right]^2\right] - 2E\left[Y\right]^2 + E\left[Y\right]^2 \\ &= E\left[E\left[Y \mid y\right]^2\right] - E\left[Y\right]^2. \end{aligned}$$

8. Stochastic Subsets for BVH Construction

This chapter is based on L. Tessari, A. Dittebrandt, M. J. Doyle, and C. Benthin. “Stochastic Subsets for BVH Construction”. In: *Computer Graphics Forum* 42.2 (2023). DOI: `10.1111/cgf.14759`

BVH construction is a critical component of real-time and interactive ray-tracing systems. However, BVH construction can be both compute and bandwidth intensive, especially when a large degree of dynamic geometry is present. Different build algorithms vary substantially in the traversal performance that they produce, making high quality construction algorithms desirable. However, high quality algorithms, such as top-down construction, are typically more expensive, limiting their benefit in real-time and interactive contexts. One particular challenge of high quality top-down construction algorithms is that the large working set at the top of the tree can make constructing these levels bandwidth-intensive, due to $O(n\log(n))$ complexity, limited cache locality, and less dense compute at these levels. To address this limitation, we propose a novel stochastic approach to GPU BVH construction that selects a representative subset to build the upper levels of the tree. As a second pass, the remaining primitives are clustered around the BVH leaves and further processed into a complete BVH. We show that our novel approach significantly reduces the construction time of top-down GPU BVH builders by a factor up to 1.8×, while achieving competitive rendering performance in most cases, and exceeding the performance in others.

8.1. Introduction

Bounding Volume Hierarchies (BVHs) are necessary to efficiently intersect rays with scene geometry. This, however, comes at a cost of preprocessing to generate such a hierarchy in the first place. This can easily become a limiting factor in interactive applications when large amounts of dynamic geometry

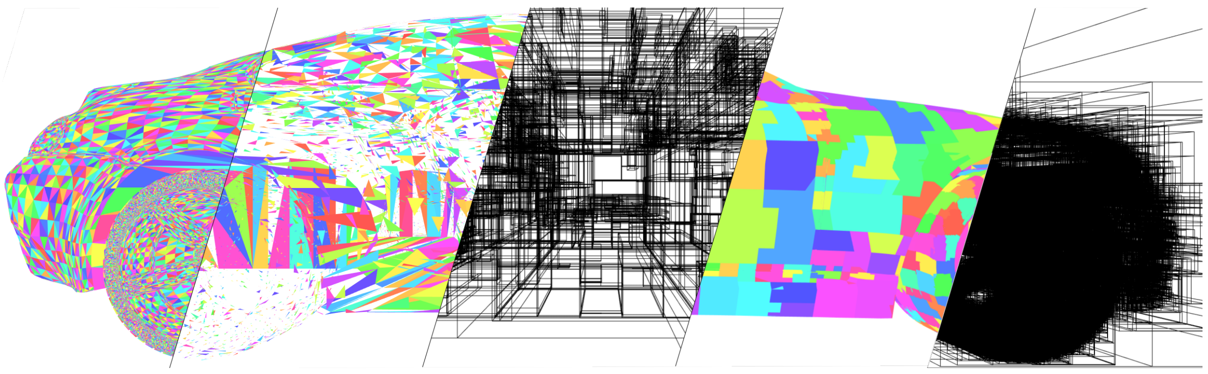


Figure 8.1.: Our proposed method first takes all scene primitives (a) and generates a representative subset of them (b). An initial hierarchy is constructed (b) into which the remaining primitives are inserted (d). The remaining hierarchy is the constructed (e). Figure taken from [146].

require regenerating acceleration structures every frame. While acceleration structures can be refitted efficiently to moving geometry by keeping the topology the same and only updating bound boxes [93], large changes can make the resulting hierarchy very inefficient, necessitating periodical full rebuilds.

Construction algorithms lie in a trade-off between efficient construction and efficient ray tracing queries. The *top-down* construction [157] produces hierarchies with high efficiency for ray tracing queries, but is generally not competitive in terms of construction time. Primitives are recursively partitioned, requiring repeated access to them. We propose a new construction scheme that averts the repeated access by operating on a representative subset of primitives for most of the construction.

The algorithm proceeds in four stages as depicted in fig. 8.1. Given the input primitives (a), a representative selection (b) is first made through spatial & importance stratification. We achieve this by ordering primitives along a space-filling curve and then performing a weighted & stratified random sampling. An initial BVH is then constructed based on the representative subset (c). The remaining primitives are then inserted (d) while minimizing the cost model and construction is then continued to produce the final BVH (e).

8.2. Related Work

A considerable variety of construction techniques have appeared in the literature. Some of the earliest high quality construction algorithms relied on a *top-down* approach to divisively cluster primitives [157]. To the present day, these builders are still recognized as delivering among the best traversal performance [1]. Later variants of top-down construction aim to improve and accelerate this general approach while retaining tree quality [49, 50, 69]. *Bottom-up* approaches operate in the opposite manner and construct from the leaves to the root [159, 55, 109, 12]. *Linear Bounding Volume Hierarchy* (LBVH) techniques differ from both categories in that the BVH construction is transformed into a sorting process, and an implicit hierarchy is extracted [92, 126, 51, 85, 153]. *Insertion-based* builders, while receiving little attention for many years [53], have re-emerged and work by incrementally inserting each primitive into an unfinished hierarchy [18, 106]. Finally, *treelet restructuring* methods draw upon LBVH methods to an extent, while introducing a second step where independent subtrees in an initial LBVH are restructured or optimized to refine the hierarchy towards a globally more efficient hierarchy [86, 38]. Other interesting approaches to BVH construction include *incremental construction* [19] and construction via k-means clustering [107]. Finally, for a full overview of BVHs for ray tracing, we refer to the STAR report [110].

Our method builds on some of these existing algorithms, but introduces stochastic techniques as a main feature. Concerning previous work on stochastic methods applied to BVH construction, the only approaches known to us are a randomized plane splitting decision [119], and a metropolis-hasting chain to select which nodes to reinsert for animation-optimized T-SAH cost[17]. In contrast, we base all of our construction on estimates and stochastic processes.

8.3. Method

This section details the individual steps of our new construction method. We avoid repeated access to all primitives by operating on a subset of the primitives for the initial levels of the hierarchy. We divide our method into four consecutive steps: Subset sampling, Subset BVH construction, Primitives insertion and Cluster BVHs construction.

In the first step, we perform stochastic sampling to compute a representative subset of scene primitives (*Subset Sampling*, section 8.3.1). In the second step, we use the top-down BVH construction method to compute an initial hierarchy (*Subset BVH construction*). This is the crucial step that benefits from operating on a representative subset. The computed hierarchy is obviously not the final one since most primitives are still missing. In the third step, we therefore insert the remaining primitives into the leaves of this hierarchy (*Primitives insertion*, section 8.3.2). After this insertion, the leaves of this hierarchy contain many primitives, thus it cannot be used directly for rendering. The leaves are therefore processed further in the fourth and last step (*cluster BVHs construction*) by continuing top-down construction from the leaf primitives using the same top-down construction algorithm as in the *Subset BVH construction* step.

8.3.1. Subset Sampling

The selected subset can have a large impact on the quality of the final hierarchy. For example, an uneven selection can lead to certain parts of the scene being underrepresented or completely missing with a large impact on the first levels of the topology. To counteract this, we rely on spatial ordering of primitives and weighted sampling on the ordered set to ensure a stratified selection. Spatial ordering is important to ensure that a stratified selection of primitives is also stratified in space, while a weighted sampling ensures that larger primitives end up in the subset with higher probability. Larger primitives have a bigger impact on the upper levels of the hierarchy and therefore need to be included in the subset.

We can divide the subset sampling step into the following phases: primitives sorting (section 8.3.1.1) and primitives importance sampling (section 8.3.1.2).

The weighted sampling is steered by the size of bounding box diagonals of primitives. However, typical scenes can exhibit large differences in primitive sizes. This means that the sample space can easily be consumed by few large primitives. Weight clamping (section 8.3.1.3) is a step prior to primitives importance sampling that clamps the sampling weights of primitives to ensure that no or few duplicates in sampling can occur.

Very densely tessellated regions tend to be underrepresented when sampling by primitive size. This is not only problematic with regard to the quality of the final hierarchy, but also for efficiency. It can also cause efficiency issues due to load balancing, since the resulting leaf nodes of the hierarchy will contain an overproportional amount of primitives. This causes To counteract this, we adapt the sampling further by mixing with a uniform weight (section 8.3.1.4).

8.3.1.1. Primitives Sorting

We first order primitives according to a Morton or Z-order space-filling curve. Other curves are also conceivable, like Hilbert, Moore or Peano curves, although computation of the respective sorting keys is more expensive, since Morton curves can be efficiently computed by interleaving the grid coordinates of each axis. We use a radix sort algorithm, since it lends itself well to efficient GPU implementation.

Reordering of primitives along a space-filling curve also possesses other advantages such as increasing spatial locality of data access, leading to improved access & cache coherency for the following processing stages.

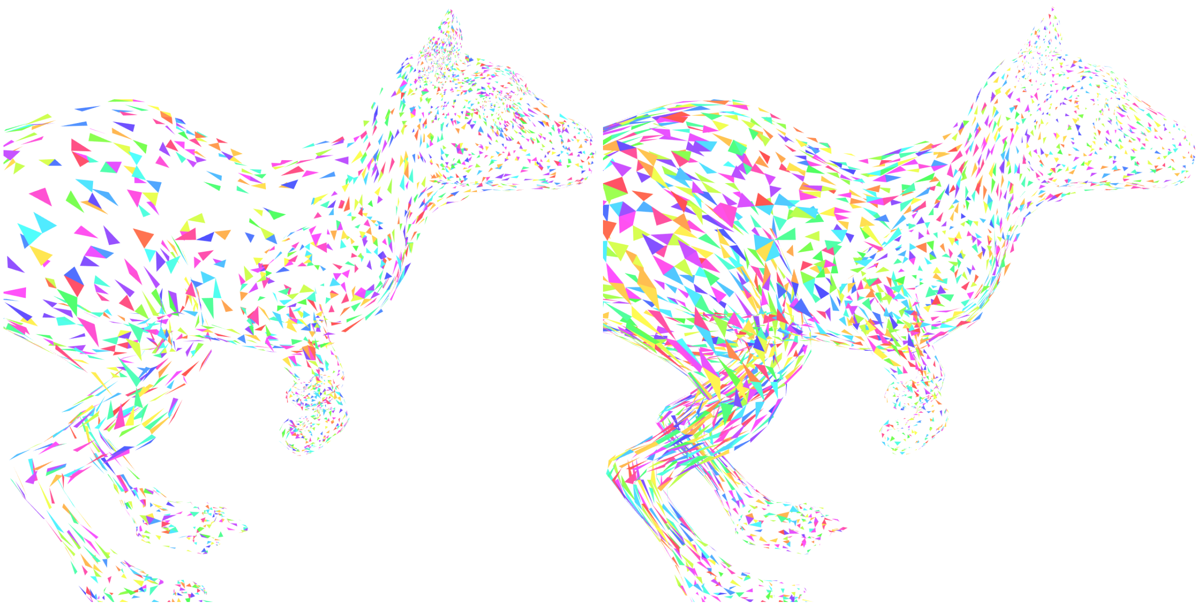


Figure 8.2.: Naïve random sampling (left) vs our method (right), equal number of primitives: our importance sampling concentrates both spatially and on the larger triangles. Figure taken from [146].

8.3.1.2. Primitives Importance Sampling

Large primitives tend to have a larger impact on the upper levels of the BVH topology compared to smaller primitives. Thus, we want to steer sampling to larger primitives by giving them higher weight. The most natural weight would be the area of the primitive, although we observed that this weighting would give too much weight to large primitives. The area scales in a squared relationship when scaling a primitive. We therefore use instead the diagonal of the axis-aligned bounding box that encloses the primitive instead.

We construct a cumulative distribution function (CDF) out of the primitive weights according to the spatial ordering in the previous section. Performing a stratified sampling on this CDF will ensure stratification both in space and according to the weight, reducing a 4-dimensional problem to a one-dimensional one. An example is given in fig. 8.2, where we compare uniform unstratified sampling to our weighted and spatially stratified sampling.

Generating Samples The actual samples from the CDF are drawn using stratified sampling. While low discrepancy random sequences like Sobol [141]. We resort to distributing equidistant samples across the sample space. Multiple samples can draw the same primitive for primitives with large weight. This can be resolved by either adaptively increasing the sample count until the desired number of primitives is attained, or by continuing with a smaller subset. Either approach can be problematic in pathological cases, since execution time can be practically unbounded or the subset will be very small. To combat this, we introduce a clamping technique in the following section.

8.3.1.3. Weight Clamping

Primitives with sampling probabilities larger than the normalized stratum size s (e.g. $s = 1/M$) are guaranteed to be sampled. However, large primitives can also cover numerous strata, resulting in many duplicate samples and a reduced efficiency of our sampling procedure. Prior to sampling the primitives, we clamp their weights such that the sampling probabilities of clamped primitives are equal to the stratum size. We define the clamped sampling probability of a primitive as

$$p_i = \frac{\min\left(w_i, c\right)}{\sum_j \min\left(w_j, c\right)}, \tag{8.1}$$

where i, j are primitives' indices, w is the sampling weight of a primitive and c is the clamping weight. To guarantee sampling of large primitives without duplicates, we only need to ensure that the sampling probability derived from the clamping weight itself is equal to the stratum size:

$$\frac{c}{\sum_j \min\left(w_j, c\right)} = s. \tag{8.2}$$

Note that depending on the stratified random number generator, random points can be placed arbitrarily inside their strata. This means that consecutive points can have distances ranging from 0 to $2s$. In that case, a greater stratum size of $2s$ must be used for the clamping to still guarantee the sampling of large primitives. This also means that duplicates are inevitable even when using weight clamping - However, they are still limited to up to 3 duplicates per large primitive in the worst case.

Efficient Computation of the Clamping Weight We can see in Equation 8.2 that the sum changes depending on c, complicating a direct computation of c. The naïve approach would be to order the primitives by weights and to evaluate each weight as a possible clamping weight. In the following section, we present an efficient algorithm (Algorithm 6) requiring only one pass over the weights to compute an *approximate* clamping weight that is slightly larger than the optimal one. Equation 8.2 then turns into an inequality, i.e. we only require the probability to be larger than s. We then find the lowest c that satisfies this constraint.

Our approach is to first build a distribution of weights through a histogram with b_c exponentially increasing bin ranges with base b (line 2). Mapping to the bin can be offset by the parameter o. We can then evaluate all boundaries between bins as potential clamping weights (line 7). The sum is approximated based on the collected distribution of weights before and after the boundary. We only track the number of weights that fall into a bin. While the clamped sum can be computed exactly, the unclamped sum is approximated through the upper bound of each preceding bin (line 11). In our implementation, we chose $b = \sqrt{2}$, $b_c = 128$ and $o = 64$.

This approach introduces two approximation errors: First, since we evaluate clamping weights only at bin boundaries, the resulting clamping weight can be off from the optimal one by up to a factor of b. Second, the overestimation of the unclamped sum can make the entire sum up to b times larger as well, therefore increasing the clamping weight by the same factor. The combined error bound is b^2. We found a bin base of $\sqrt{2}$ to already be sufficient for our use case, resulting in an error bound of 2. For more details on the impact of using the clamping, we refer to the supplemental.

8.3.1.4. Uniformity

To ensure that densely tessellated regions are not underrepresented, we mix the clamped sampling probabilities with uniform probabilities through defensive importance sampling[73]. Uniform sampling effectively increases the chance of sampling dense regions, and we can see its effect in Figure 8.3. To

Algorithm 6: Histogram weight clamping

```
input  : Array $W$ filled with weights to clamp
         Stratum size $s$ and bin base $b$, count $b_c$ and offset $o$
output : Clamping weight
hist ← $[0 | i \in [0, b_c)]$;                                  // histogram
for $w \in W$ do                                                // binning of weights
    bin ← $\min(\max(o + \lfloor \log_b w \rfloor, 0), b_c - 1)$;
    hist[bin] ← hist[bin] + 1;
uSum ← 0;                                                       // unclamped sum
cSum ← $|W|$;                                                   // clamped sum
for $i$ ← 0 to $b_c - 1$ do                                     // bin search
    clamp ← $b^{i-o+1}$;
    if clamp/(uSum + clamp · cSum) ≥ $s$ then
        return clamp;
    uSum ← uSum + hist[$i$] · clamp;
    cSum ← cSum − hist[$i$];
return ∞;
```

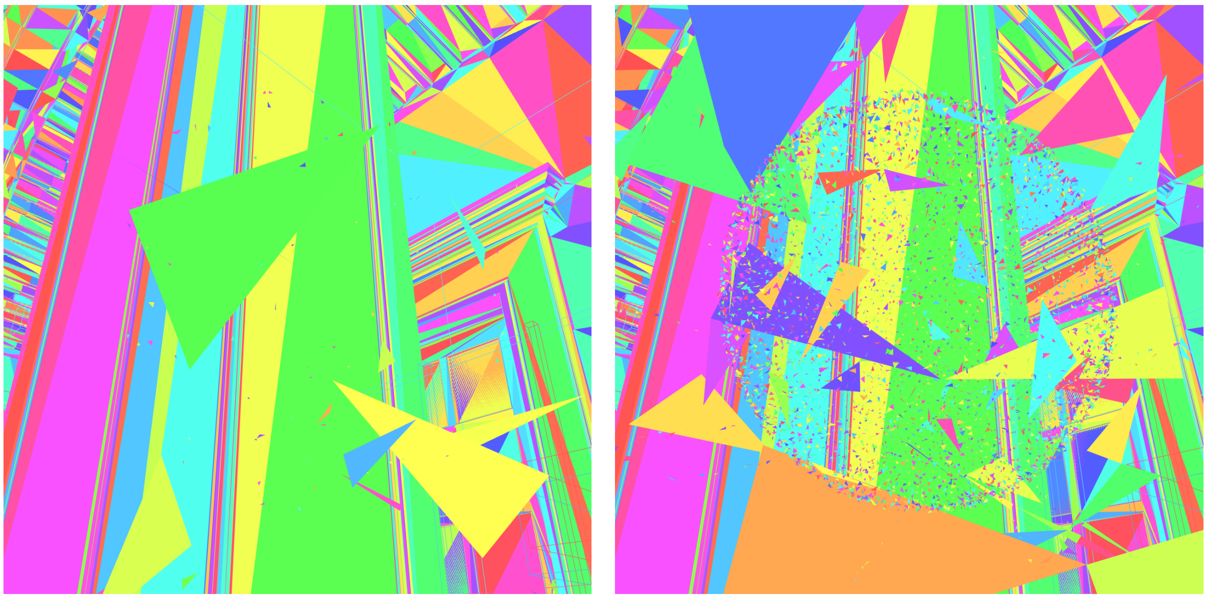


Figure 8.3.: Impact of uniformity on sampling of tiny dense regions in scene San Miguel. Uniformity ensures that tiny dense regions remain well-represented in the subset. Figure taken from [146].

retain the sampling guarantee of large primitives, we need to perform a minor change to our weight clamping procedure.

We define our mixture probability p_i^* as

$$p_i^* = u \cdot \frac{1}{N} + (1 - u) \cdot \frac{\min(w_i, c)}{\sum_j \min(w_j, c)}, \tag{8.3}$$

where $u \in [0,1]$ is the uniform fraction. Using this mixture directly with the clamped probabilities would destroy the guarantee of sampling large primitives. Accordingly, we require the probability derived from the clamping weight to be equal to the stratum size:

$$u \cdot \frac{1}{N} + (1-u) \cdot \frac{c}{\sum_j \min\left(w_j, c\right)} = s. \tag{8.4}$$

After rearranging, we are left with:

$$\frac{c}{\sum_j \min\left(w_j, c\right)} = \frac{s - u/N}{1-u}. \tag{8.5}$$

This equation differs from equation 8.2 only on the right side. We can extract it as an updated stratum size s':

$$s' = \frac{s - u/N}{1-u}. \tag{8.6}$$

s' can be used in place of s in the previous section when a uniform mixture is used when sampling. Intuitively, we increase the stratum size such that the resulting clamping weight reserves additional weight for large primitives. After applying the uniform mixture, this additional weight is then redistributed among all primitives, leaving large primitives with just enough weight to fully cover the actual stratum size s (therefore still being guaranteed to be sampled). The trade-off of increasingly adding uniformity is that, while densely tessellated regions are represented better, some large primitives start to lose the guarantee of being sampled, namely the ones whose weight is between the previous and new clamping weight. As such, the uniformity should not be set too high. We generally recommend a uniformity between 10% and 20%. For more details on the effect of adding uniformity we refer to the supplemental.

8.3.2. Primitives Insertion

The subset BVH so far only contains subset primitives. To complete the hierarchy, the remaining primitives need to be inserted. To this end, we need to associate each of the remaining primitives with a leaf note of the subset BVH, which we refer to as *clusters*. The insertion minimizes a cost model (section 8.3.2.1), and considers multiple leaf nodes for each primitive for insertion (section 8.3.2.2).

8.3.2.1. Cost Model

We use the surface area heuristic (SAH) as our cost model for guiding insertion decisions, as it is also used for split decisions in the interior builder. This cost model, however, is defined for the whole BVH and would require reevaluating the entire topology for every insertion candidate, which is prohibitively expensive. We follow the approach by [18], where instead of computing the overall cost, we only compute the change of the cost when inserting a primitive. The optimal insertion decision is invariant under this transformation, since it only applies a constant offset to all evaluated costs. In contrast to the normal SAH, only changed nodes with changed bounding boxes need to be considered. These nodes can only be the ancestors of the candidate leaf node.

We consider the flattened SAH metric [99]:

$$C(N) = \frac{1}{\mathrm{SA}(N)} \left[C_T \sum_{N_i} \mathrm{SA}(N_i) + C_I \sum_{N_l} \mathrm{SA}(N_l)|N_l| \right]. \tag{8.7}$$

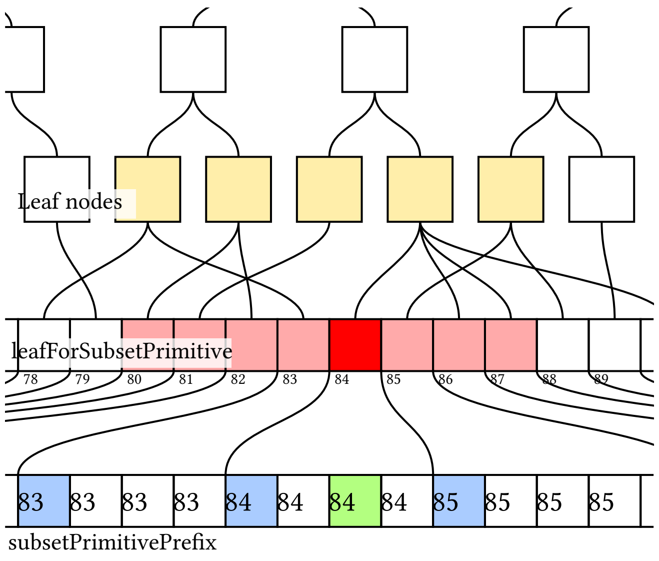


Figure 8.4.: Window construction of the pruning Morton window search. Each primitive possesses a reference (subset-PrimtivePrefix) into a compacted array of subset primitives (leafForSubsetPrimitive). This array contains references to the leaf node (*cluster*) each subset primitive belongs to. For each primitive we take its reference (i.e. green) and then generate a window around its previous subset primitive in the compacted array (red). Each leaf node pointed to in the window is considered for insertion (yellow). Figure taken from [146].

The division by the root bounding box area can be ignored since it is just a constant factor and therefore also doesn't affect the optimum. We then define the SAH increase of inserting primitive p into leaf node N_p by subtracting the cost of the previous BVH from the cost of the new BVH and canceling all cost terms that remain unchanged:

For two topologically equivalent BVHs with root nodes N' and N with differing bounding box dimensions and primitives, the SAH difference can be reduced to differences of bounding box areas and primitive counts:

$$\begin{aligned} I(N', N) &= C(N') - C(N) && (8.8)\\ &= C_T \sum_{N_i} \mathrm{SA}(N_i') + C_I \sum_{N_l} \mathrm{SA}(N_l')|N_l'| - C_T \sum_{N_i} \mathrm{SA}(N_i) - C_I \sum_{N_l} \mathrm{SA}(N_l)|N_l| && (8.9)\\ &= C_T \sum_{N_i} \left(\mathrm{SA}(N_i') - SA(N_i)\right) + C_I \sum_{N_l} \left(\mathrm{SA}(N_l')|N_l'| - \mathrm{SA}(N_l)|N_l|\right). && (8.10) \end{aligned}$$

In the case where we insert a primitive p into a leaf node N_l, most of these differences evaluate to zero apart from the given leaf node and its ancestors:

$$I(p, N_l) = C_T \sum_{N_p} \left(\mathrm{SA}(N_p') - SA(N_p)\right) + C_I \left(\mathrm{SA}(N_l')|N_l + 1| - \mathrm{SA}(N_l)|N_l|\right), \tag{8.11}$$

where N_p are the ancestors of the leaf node N_l and N_p' and N_l' are the bounding boxes after inserting the primitive.

8.3.2.2. Pruning Morton Window Search

Evaluating the cost function for each primitive in every cluster would be too expensive in practice. Similar to [109], we leverage the already computed spatial ordering of primitives from the subset

Algorithm 7: Pruning Morton window search

```
input  : Primitive $p$ with index $i$ and window size $w$
output : Leaf node to insert primitive into
$m \leftarrow$ subsetPrimitivePrefix[$i$];
minCost $\leftarrow \infty$, minLeaf $\leftarrow \perp$;
for $j \in [m - w, m + w]$ do
    leaf $\leftarrow$ leafForSubsetPrimitive[$j$];
    if leaf = minLeaf then continue;
    cost $\leftarrow I_l(p, \text{leaf})$;                    // Eq. 8.11
    node $\leftarrow$ getParent(leaf);
    while node $\neq \perp \wedge$ cost < minCost do
        diffCost $\leftarrow I_i(p, \text{node})$;            // Eq. 8.11
        if diffCost = 0 then break;
        cost $\leftarrow$ cost + diffCost;
        node $\leftarrow$ getParent(node);
    if cost < minCost then
        minCost $\leftarrow$ cost;
        minLeaf $\leftarrow$ leaf;
return minLeaf;
```

sampling phase (Section 8.3.1) in order to perform a localized search, in our case of insertion candidates (Figure 8.4). The algorithm considers a fixed number of subset primitives and their clusters around a primitive. Since subset primitives are sparsely represented in the original array, we store the cluster pointers in a separate compacted array (leafForSubsetPrimitive). We use a prefix sum to perform this compaction (subsetPrimitivePrefix), which we also use to later index into this compacted array. Due to the sparsity of subset primitives, the search will cover a large spatial region even with small windows.

Algorithm 7 details the pruning search. After computing the window center using the prefix array, we compute the cost change for every subset primitive in the window by traversing the hierarchy upwards (eq. (8.7)). To skip as much computation as possible, we try to discard candidates as early as possible. Since all terms of the cost are positive, we can abort the upwards traversal as soon as the cost change exceed the cost of the best candidate found so far, since continuing traversal will only increase this cost further. Traversal can also stop if the cost increase of an inner node is zero, since its parents will then also not exhibit a cost increase (line 10). Finally, subset primitives are oftentimes part of the same cluster, and we skip reevaluating the same cost if we are testing the best candidate found so far again (line 5).

8.3.3. Algorithmic Complexity

In this section we analyze the theoretical complexity of our algorithm to estimate the speedup that we can expect depending on the parameterization. We will first approximate the complexity of our algorithm and then relate this to the complexity of top-down construction, which is $O(n \log n)$, to compute the actual speedup.

Our algorithm can be divided into the complexity of subset and cluster BVH construction and the additional overhead that we introduce. The subset BVH construction only operates on a subset of m primitives, resulting in a complexity of $O(m \log m)$. The cluster BVH construction operates on n

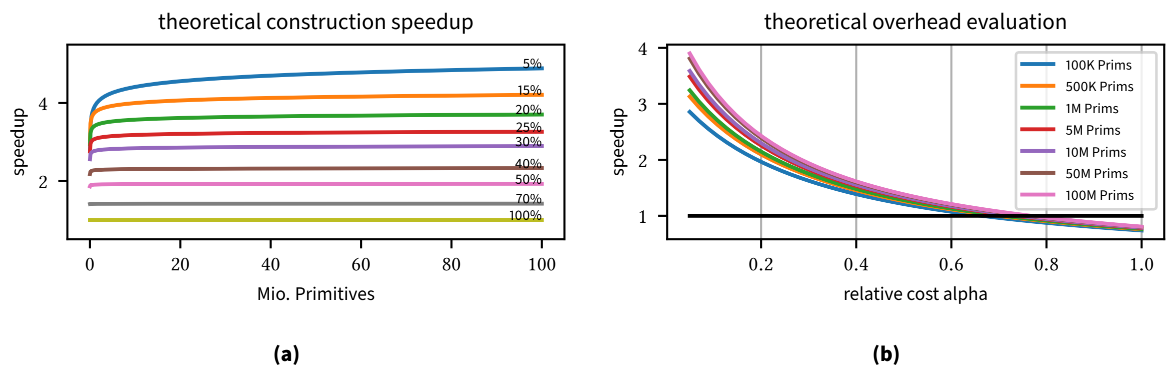


Figure 8.5.: Theoretical speedup to be expected from our algorithm compared to top-down construction. Disregarding our overhead (a), speedup increases with more primitives and a lower subset fraction. Speedup plateaus early for subset fractions larger than 20%. With our overhead (b), speedup depends on the relative cost compared to BVH construction and needs to be lower than ca. 70% to provide a meaningful speedup. Figure taken from [146].

primitives, but only for the remaining $\log n/m$ levels, resulting in a complexity of $O(n \log n/m)$. Our overhead is composed of the subset sampling and insertion stages. Subset sampling is linear in the number of primitive with a complexity of $O(n)$ and the insertion has logarithmic complexity for every primitive due to the upwards traversal of the subset BVH, resulting in a total complexity of $O(n \log m)$. The overall complexity of our overhead is therefore $O(n \log m)$. Finally, the complexity of all stages amounts to $O(m \log m + n \log n/m + n \log m)$.

We will first consider the possible speedup when ignoring our overhead. The complexity is given as $O(m \log m + n \log n/m)$ and the speedup is the fraction of the top-down approach to this complexity $O(n \log n/(m \log m + n \log n/m))$. Figure 8.5a shows the speedup depending on primitive count and different subset fractions. We can observe that lower subset fractions and higher primitive counts generally perform better.

When accounting for the overhead, the speedup has the form $O(n \log n/(m \log m + n \log n/m + n \log m))$. While the top-down stages can be compared reasonably in O-notation since they are always the same algorithm, the overhead is quite different. We therefore use a constant factor α to gauge relative cost between top-down construction and the overhead. The speedup then has the form $n \log n/(m \log m + n \log(n/m) + \alpha n \log m)$. Figure 8.5b shows the speedup for different primitive counts and relative costs at a subset fraction of 20% (which we deem the most reasonable based on our experimentation). Due to the overhead, a speedup is not always guaranteed. Beyond $\alpha = 0.7$, the overhead is too large, and the algorithm becomes overall slower.

8.4. Implementation

Terminology In the following, we will use the terminology from OpenCL to refer to the GPU programming model. A work group (CUDA: Thread block) is a set of threads that can directly synchronize with each other and also exchange data through fast but limited shared memory. Work groups are further subdivided into sub groups (CUDA: Warp). Threads inside a sub group are executed in lockstep and can directly exchange data through register permutations.

Framework We implemented our algorithm both as a GPU-builder based on oneAPI DPC++ (`https://www.oneapi.io/`) and a CPU-builder in PBRT [128]. The GPU-builder implements the presented method from Section 8.3, while the CPU builder was used to test various approaches that led to the final design (We refer to the supplemental for the analysis of various parameters of our builder on the CPU). In this section we focus on the implementation details of the GPU-builder.

Interior Builder We use a binned SAH builder in the style of Wald [157] as the interior top-down BVH builder. Each axis is considered for splitting using 16 bins each. We employ their horizontal and vertical parallelization technique with an additional middle phase to efficiently exploit the massive parallelism of the GPU at each BVH level. Since nodes are initially few, the horizontal phase is parallelized over primitives. For each iteration (seven in total), three kernels are launched for (1) initializing and (2) accumulating bins and (3) computing the splits and partitioning primitives. Only nodes with primitive counts larger than the average over all active nodes are considered, leading to a more even primitive distribution for the following phases. After the horizontal phase, we transition to the middle phase where active nodes with more than 1024 primitives are processed. For each iteration (twelve in total), a single kernel is launched where each work group processes a single node. The vertical phase then proceeds to compute the remaining subtrees of all nodes. This is performed with a single kernel launch. Each sub group in the kernel processes a single node.

For building cluster BVHs, we directly use vertical parallelization, since the number of clusters typically already far exceeds the number of concurrently executing sub groups. This is also the reason why uniformity (Section 8.3.1.4) must be added to the subset sampling. Otherwise, large clusters of highly tessellated geometry will introduce a strong load imbalance, hindering individual sub group processing capabilities and arbitrarily increasing the overall build time.

Subset Sampling We use single-precision floating point numbers for the CDF for performance reasons. While the available precision in large scenes is not enough to faithfully represent regions with low probability, we found that it did not affect the results much. However, rounding errors in the CDF do have an effect. The prefix scan routine which we employ performs the scan across an implicit hierarchy with three levels. First, a scan per sub group is performed, then a scan per work group, and finally a scan over all work groups. This approach exhibits good error properties, since it is comparable to a pairwise summation (but wider).

The random numbers used during the sampling stem from equidistant points with a distance of the stratum size s in $[0, 1]$ with a random initial offset. The sampling quality was not affected much compared to Sobol points. However, the highly coherent access during the bisection of the CDF led to a $5\times$ performance increase in the sampling dispatch. Additionally, the equidistant spacing decreases the chances of duplicates (caused by the approximate weight clamping) with the trade-off in a potentially smaller subset size.

Primitives Insertion We performed slight modifications to the pruning Morton window search (Algorithm 7) to better exploit the parallelism of the hardware. Instead of having each thread traverse its own window, we unify the windows of all threads in a sub group. All threads then proceed to step through this larger shared window in lockstep, which results in perfectly coherent memory access. Additionally, it seems beneficial to first evaluate the center of the window for each thread independently. While the memory access is relatively inefficient in that case, the found node is oftentimes already the optimum. As such, pruning may be triggered early when the window is traversed. With both optimizations, the performance of the search dispatch improved by 15%.

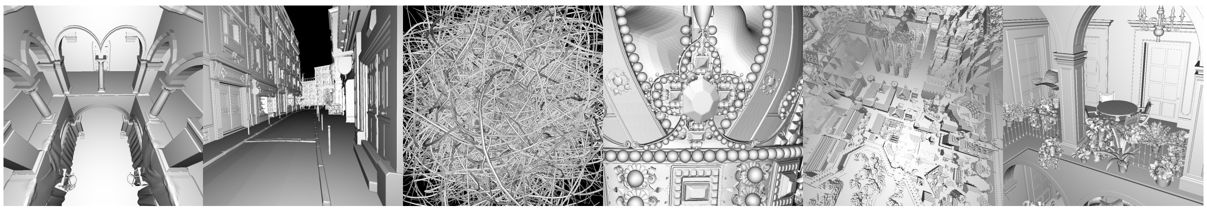


Figure 8.6.: Test scenes we used for the evaluation annotated with primitive count. Scenes were taken from McGuire [104]. Figure taken from [146].

Memory Requirements Compared to the binned SAH builder, our stochastic builder additionally needs 32 bytes of memory for each of the N primitives (Morton codes, CDF, subset mask & prefix sum and insertion selection) and 44 bytes for each subset primitive (Compacted primitive bounding boxes, back-references to nodes as well as atomic counters for the insertion step). None of this memory is used when the cluster BVH build phase starts, so parts or even all the memory can be aliased with memory required by the binned SAH builder. For example, the preallocated node memory can be used, since only the smaller subset BVH occupies it by that point. In the scene Crown (Fig. 8.6), we measured a memory consumption (binary BVH construction only) of 411 MiB, compared to 131 MiB of the binned SAH builder. Other builders detailed in Section 8.5.1 require 177 MiB (LBVH), 242 MiB (PLOC++) and 196 MiB (ATRBVH). Note that all memory is allocated upfront in our implementation.

8.5. Evaluation

We evaluate our method in this section. We initially compare build performance to other GPU builder in section 8.5.1, then analyze variation of our approach due to stochastic sampling in section 8.5.2. Next, we show a performance break-down of the individual stages and relate this to top-down construction in section 8.5.3 and analyze how the theoretical complexity analysis translates to real-world performance section 8.5.4. Finally, we compare against a deterministic clustering scheme in section 8.5.5.

We use an Intel Alchemist A770 GPU [30] (32 Xe cores) that is powered by an Intel Core i5 9600K CPU clocked at 3.70GHz with 16GB of DDR4 RAM. We run Ubuntu 20.04 LTS Linux OS on an NVMe SSD.

All images are rendered at a resolution of 1024×1024. We differentiate between primary rays with one sample per pixel as a workload with high data coherence, as well as ambient occlusion with 64 samples per pixel as an incoherent workload. The scenes (fig. 8.6) range from geometrically simple with low primitive count (Crytek Sponza with 279K primitives) to high geometric complexity and primitive count (San Miguel with 7.9M primitives).

Unless otherwise noted, the subset size is set to a fraction of 20% of total primitive count and the uniform fraction to 10%.

8.5.1. Build Performance

We compare our method against existing build algorithms targeted at GPUs, namely LBVH [93], PLOC++ [12], ATRBVH [38] and binned SAH construction [157]. We base our LBVH and PLOC++ implementation on the work of Karras [85] and Benthin [12] respectively, while we ported the publicly available code for ATRBVH (`https://github.com/leonardo-domingues/atrbvh`) into oneAPI DPC++.

	Host build time (ms)	Device build time (ms)	SAH cost	Primitive throughput (MPrim/s)	Primary (GRay/s)	Ambient occlusion (GRay/s)	Host build time (ms)	Device build time (ms)	SAH cost	Primitive throughput (MPrim/s)	Primary (GRay/s)	Ambient occlusion (GRay/s)
	Crytek Sponza						Bistro					
LBVH	**2.25**	**1.49**	61.32	**63.21**	2.745	2.844	**9.61**	**6.11**	74.53	**151.76**	1.276	0.814
PLOC++	6.14	*2.28*	*47.47*	23.19	**3.479**	**3.187**	*14.22*	*7.45*	53.75	102.54	1.349	*0.882*
ATRBVH	*4.97*	3.79	51.47	*56.16*	*3.314*	2.987	20.63	19.25	62.2	*137.01*	1.302	0.813
Stochastic (ours)	10.47	7.14	**42.7**	13.6	3.124	*3.09*	20.0	16.34	**38.43**	72.91	**1.38**	**0.883**
Binned SAH	9.71	7.69	50.01	14.66	2.83	2.846	23.24	20.78	*39.75*	62.76	*1.367*	0.849
	Hairball						Crown					
LBVH	**7.0**	**5.96**	320.11	**205.83**	1.255	1.089	**10.58**	**9.45**	22.15	**231.0**	2.38	1.928
PLOC++	*12.24*	*6.51*	283.82	117.66	1.312	1.096	*18.18*	*11.03*	19.64	134.42	2.509	2.01
ATRBVH	19.45	18.19	301.96	*148.04*	1.305	1.11	30.9	29.61	19.69	*157.56*	2.522	2.001
Stochastic (ours)	18.03	14.44	*184.52*	79.85	*1.362*	*1.184*	25.37	21.3	**14.82**	96.35	*2.635*	*2.102*
Binned SAH	23.75	21.61	**182.3**	60.63	**1.365**	**1.194**	34.78	32.16	*15.39*	70.27	**2.651**	**2.113**
	Rungholt						San Miguel					
LBVH	**13.41**	*12.33*	130.12	**250.01**	2.423	3.218	**18.08**	**16.92**	64.35	**250.19**	1.789	1.106
PLOC++	*15.35*	**10.58**	90.4	*218.44*	2.739	3.489	*26.74*	*17.97*	41.44	*169.13*	**2.401**	**1.559**
ATRBVH	42.08	40.77	84.95	159.34	*2.939*	*3.813*	56.23	54.81	47.17	140.14	2.299	1.395
Stochastic (ours)	30.08	25.84	*69.97*	111.45	2.86	3.652	40.28	36.26	**38.81**	112.29	*2.369*	*1.466*
Binned SAH	42.69	40.26	**62.74**	78.52	**2.967**	**3.964**	73.47	70.93	*40.25*	61.56	2.302	1.459

Table 8.1.: Comparison of our stochastic builder with existing GPU build algorithms on Intel Alchemist A770 GPU (32 Xe cores) using Ubuntu 20.04 Linux. Figure taken from [146].

The implementation of the binned SAH builder is identical to the interior builder we use for the subset BVH (Section 8.4).

We measure ray tracing performance using hardware traversal. For the binned SAH and our stochastic builder, we stop the construction process of the BVH as soon as there are eight or fewer primitives in a node. LBVH, PLOC++ and ATRBVH construct hierarchies with one primitive per leaf. All implementations build binary BVHs, that are then converted to the hardware format on GPU. As the expected input format is quads, primitives are converted prior to construction, thus roughly halving the effective number. Host time includes time spent on device, dispatches and synchronizations on CPU. Conversion times from binary BVH (BVH2) to hardware specific BVH format (HW BVH) are not included. For a recent study on the traversal performance and SAH cost impact of different BVH formats after conversion from BVH2, we refer to [108].

We summarize our findings in Table 8.1. Compared to the binned SAH builder, we improved build times by 1.33× on average (1.47× on device time). We generally observe that for large scenes the build time reduction is more significant. With San Miguel (7.9M primitives), we reach an improvement of 1.82× (1.96× on device), while only on Crytek Sponza (279K primitives) we see a small increase in the host build time (7%). The additional dispatches and synchronization time we introduce are not amortized in that case. The stochastic builder is competitive with ATRBVH in all cases involving more than a few hundred thousand primitives. At the same time, while not reaching similar build time competitiveness, it considerably reduces the well-known gap from top-down builders to faster ones like LBVH or PLOC++. In terms of SAH quality, our stochastic builder is able to maintain a comparable or even lower cost than the binned SAH builder. As a result, both builders are consistently the lowest in our tests. Although this consistency does not translate linearly to the final rendering performance due to the hardware format conversion, our stochastic builder still remains the best or second-to-best and within ±1% from the more expensive binned SAH in all the scenes, except Rungholt.

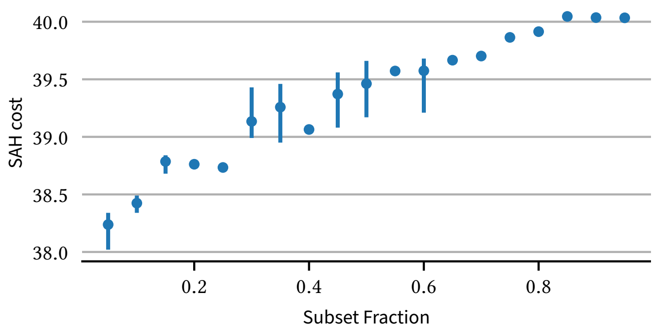


Figure 8.7.: Variance of SAH of computed hierarchy in scene San Miguel for multiple subset fraction in range between 5% and 95%. Each data point is computed using 50 random seeds. Variance is mostly negligible with some instances possessing higher variance. Figure taken from [146].

8.5.2. Variance Analysis

As our subset sampling approach is stochastic, this poses the question on how reliable the quality of the final hierarchy is between independent runs. We evaluate this aspect by executing our algorithm multiple times with different seeds for a given scene. Figure 8.7 shows variance for the San Miguel scene for different subset fractions. Each data point comprises 50 independent runs. SAH cost decreases as the subset fraction is reduced while the individual data points are mostly stable. Some data points posses increased variance, although it is much less expressed than the dependence on the subset fraction. Other scenes possess different trends, although variance remains similarly low. We refer to the supplemental document for an analysis considering the other scenes.

8.5.3. Build Time Breakdown

Our stochastic build consists of many passes contributing to the total running time. Figure 8.9 shows a breakdown of the individual passes at a subset fraction of 20% and 50% in the San Miguel and Rungholt scenes. We also show a breakdown of the Binned SAH builder. While the added overhead is non-negligible, it is more than amortized by the time reduction of the horizontal, middle and vertical phases that exist in both algorithms at 20% subset fraction (for a description of these passes, see Section 8.4, *Interior Builder*). At 50%, we can see a significant increase in the middle phase execution time. In general, the efficiency of subset BVH construction does not scale linearly with a reduction in the other phases or with the SAH cost, suggesting that a carefully chosen small subset of geometry is able to achieve good performance compared to a more sizable one at a fraction of the cost.

We additionally present a primitive throughput analysis on the individual levels of the BVH for our stochastic builder and the binned SAH builder in the scene San Miguel (Figure 8.8). We collect histograms per dispatch on how many primitives are processed for each level of the BVH and then proportionally distribute the execution time of the dispatch to each level. The timings are more fine-grained for the first levels, since those are processed by multiple dispatches. We found that the primitive throughput in the first levels, where construction occurs exclusively based on the subset, differs by a factor of 0.4× compared to the binned SAH builder.

This discrepancy is explained by the scaling behavior of the horizontally-parallelized build phases. We suspect that the constant overhead due to bin accumulation and evaluation becomes a major factor with lower primitive counts. Reducing this overhead is therefore an interesting area for future work.

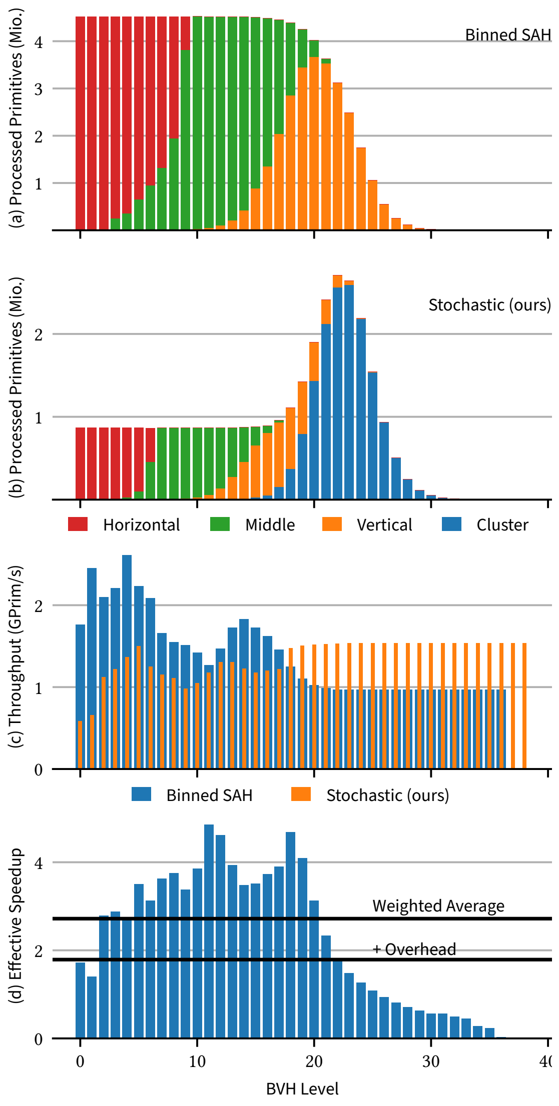


Figure 8.8.: Per-level comparison of our stochastic builder (20% data) with the binned SAH builder in scene San Miguel. For the first few levels, our builder accesses fewer primitives (a, b). Although the effective throughput in the first levels is considerably lower (c), the *effective speedup* (d) due to the reduced primitive set is still 2-4× in the stochastic BVH construction (first ~20 levels). On average, the speedup is around 2.7×, or 1.9× when including our additional overhead. Note that the topologies are not identical, hence the additional levels of our builder. Figure taken from [146].

8.5.4. Algorithmic Evaluation

As seen in the algorithmic complexity section 8.3.3, when the interior builders have similar performance the theoretical speedup can be up to 3-4×, depending on the costs introduced by the primitives reordering, subset sampling and primitives insertion passes, which we will reference as *overhead*. In Figure 8.8 we can assess how well the model predicts the final outcome by taking the San Miguel scene as an example. The first two plots compare the performance per level of each building stage: our stochastic builder operates on 20% of the binned SAH data, thus showing a lower amount of processed primitives, until the Stochastic BVH construction is done ((b), red, green and orange). The cluster BVHs build (blue) shows instead a higher count in a short burst due to the final construction phase involving

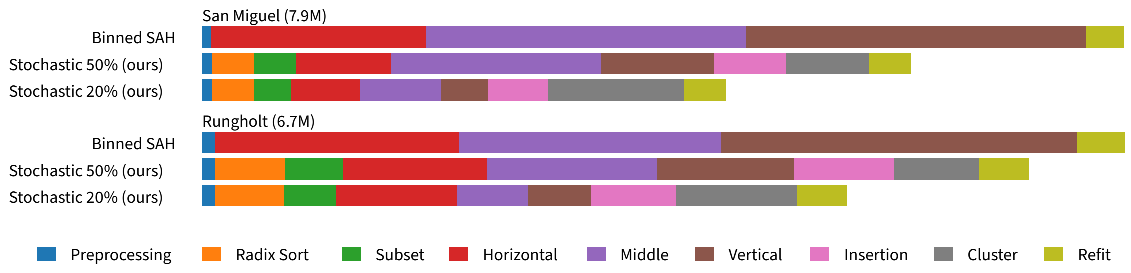


Figure 8.9.: Relative timing breakdown of our stochastic builder and the binned SAH builder in the San Miguel and Rungholt scenes. Our additional overhead at a subset fraction of 20% amounts to 27% and 32%, respectively, while we observe a total build time reduction of 57% and 70%. With increasing subset size the middle phase shows the highest increase in time. For a description of the horizontal, middle and vertical pass we refer to Section 8.4. Figure taken from [146].

all the primitives. What becomes apparent in the throughput plot is that our assumption over similar performances in BVH construction does not hold in this case. Our stochastic BVH is, in fact, utilizing a bit more than half of the bandwidth while processing a fifth of the data. The cluster BVHs build step instead is showing 1.5× the throughput of the binned BVH. Note how our initial approximation of the final BVH creates a few more levels. This brings down our initial projections to a 2.71× speedup over the construction time alone ((d), *weighted average* bar), and consequently influence the final outcome to a 1.89× ((d), *overhead* bar). By setting α to 0.19 (our measured relative overhead), the expected gain given by our model evaluates at 2.1×. The reason behind this behavior resides in the ability of the chosen builder to saturate the GPU: we can see its effect in (c), where the binned SAH dominates the first half of the construction, only to switch places when the cluster BVHs phase takes place. This highlights how important it is to increase the efficiency of the Stochastic BVH construction and the direct speedup gain that can be obtained, without counting the obvious benefit of a lower overhead.

8.5.5. Comparison with a Deterministic Clustering

The clustering of our stochastic approach is controlled by the subset BVH construction and therefore optimized according SAH. It is also conceivable to pregenerate a clustering of primitives a priori and independently construct the subset and cluster BVHs in a second step. We demonstrate this approach using HLBVH[126] that generates the first levels of the hierarchy using an efficient LBVH scheme. We form clusters by considering a fixed bit-prefix that primitives posses according to the space-filling Morton curve.

Figure 8.10 shows the resulting cost metrics and build times for the crown scene using different bit prefix sizes that parametrize the clustering. We can observe that our clustering consistently outperforms this approach. We think that this is mostly due to the rigid clustering that emerges from clustering a priori. For one, small, highly tessellated regions are not given an individual cluster (*teapot in a stadium* problem) and cluster boundaries often cut through regions densely filled with primitives. Instead, clusters should cut along empty regions to isolate dense regions from each other.

8.6. Discussion

Use Cases and Limitations Our approach generally scales better with larger scenes, as we have seen in the algorithmic analysis in section 8.3.3, but we also have a strong dependency on constant factors

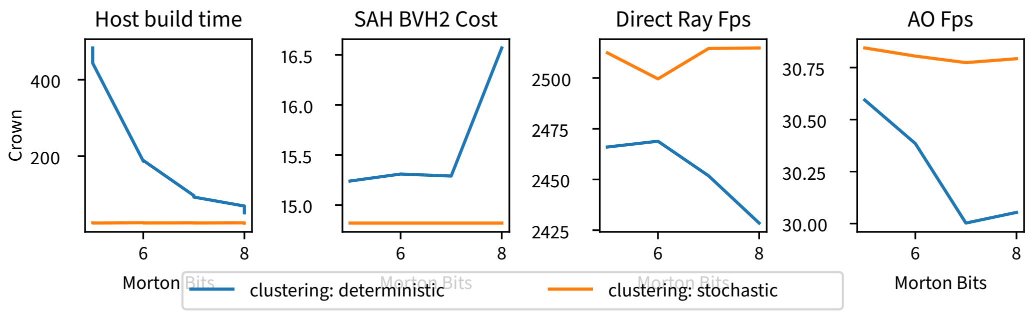


Figure 8.10.: Performance evaluation of a deterministic clustering approach relying on LBVH [92] to generate clustering against our stochastic clustering approach relying on top-down construction over subset primitives. Build times are very sensitive to the number of prefix bits (since the number of clusters increases with the number of bits) while costs remain higher than our construction method. Figure taken from [146].

of the overhead that we introduce. This is the prime reason why our approach exhibits lower speedups compared to the theoretical estimate: Our top-down implementation is not optimized for efficiently constructing BVHs from small primitive sets, succumbing to overheads introduced by kernel dispatches. Despite this, we can still achieve meaningful speedups of up to 1.8× in larger scenes. We generally recommend a subset fraction of 20% with a uniformity of 10%. The latter is mostly relevant to eliminate pathological cases where individual clusters would otherwise possess too many primitives and cause load imbalance issues.

Relation to Binning Binning [157] distinguishes itself from our approach in that reducing the bin count only accelerates computation of the split, but binning and partitioning still require access to all primitives. Our approach accelerates all parts of construction in the first levels, since the primitive set is smaller. Nevertheless, both approaches are orthogonal, and their combination retains their respective strengths, which is why we use a binned builder as the interior builder.

Applicability to other Builders Our algorithm is directly applicable to top-down construction methods like the binned builder we used, but also sweep SAH. Since the computational overhead per level is roughly the same ($O(n \log n)$), the use of subsets in the first levels gives a noticeable performance improvement. The strength of our stochastic method is to accelerate $O(n \log n)$ algorithms such as top-down construction, which generally yields higher quality trees than $O(n)$ construction methods. Other build algorithms like PLOC which are closer to $O(n)$ do not posses this property. Due to the decreasing set size in each iteration, the top of the tree is already fast to compute. Thus applying our approach to PLOC would give a negligible return if any at all.

Comparison with other Top-Down Build Algorithms Other methods have been presented in the past that attempt to accelerate top-down construction [49, 50, 69]. These methods, however, are tailored to CPU-hardware, and it is not straightforward to execute these efficiently on GPU hardware. In the future, it would be a worthwhile avenue to compare our method against these builders.

8.7. Conclusion

Through stochastic sampling, we have presented a novel method to accelerate BVH construction with a speedup of up to 1.8×, while largely retaining quality of top-down BVH construction. We believe that our method provides a new perspective on the problem of efficiently constructing BVHs and opens new possibilities that leverage random sampling techniques in this context in the future.

Future avenues include tailoring the interior builders to the new circumstances (for example, smaller primitive counts in the subset BVH stage) and also applying our method to refitting. Applying other random sequences like blue noise could also prove as a fruitful avenue. Furthermore, our method could potentially also see application on faster build algorithms like PLOC for additional efficiency improvements.

9. Conclusion

In this thesis, we have explored the use of the Monte Carlo path tracing for real-time rendering. We developed data-driven methods in the context of path guiding and radiance caching to improve image quality and accelerated preprocessing of ray tracing acceleration structures.

We demonstrated the feasibility of guiding methods in a real-time context and explored three avenues in particular: (1) For direct illumination, we developed a caching scheme that solely learns visibility of lights (section 5.4). This can be sufficient if the remaining terms can be approximated accurately enough. (2) We developed a compressed representation of incoming radiance based on directional quadtrees that can be efficiently sampled and can provide a meaningful variance reduction (section 5.5). (3) For dynamic scenarios, an implicit mixture model defined by a Markov chain process can adapt fast to changing illumination (chapter 6) and is independent of explicit information about light sources.

While different representations of radiance caches can have a large impact on image quality, it is also relevant when the cache is actually accessed. We developed termination strategies (chapter 7) that allow to explicitly control the image quality to deliver guarantees for following processing stages like denoising.

Construction of bounding volume hierarchies remains an expensive preprocessing step that needs to be executed repeatedly in dynamic scenarios. By accelerating top-down construction with a representative subset of scene primitives, larger scenes can be used in interactive contexts.

In summary, our methods allow more complex scenes to be introduced in interactive rendering. For one, complexity of illumination can be increased through guiding and caching methods and geometric complexity can be increased through fast construction of used acceleration structures.

Bibliography

[1] T. Aila, T. Karras, and S. Laine. “On quality metrics of bounding volume hierarchies”. In: *Proceedings of the 5th High-Performance Graphics Conference*. HPG ’13. ACM, 2013. DOI: 10.1145/2492045.2492056.

[2] P. Andersson, J. Nilsson, P. Shirley, and T. Akenine-Möller. “Visualizing Errors in Rendered High Dynamic Range Images”. In: *Eurographics 2021 - Short Papers* (2021). DOI: 10.2312/EGS.20211015.

[3] J. Arvo. “Applications of irradiance tensors to the simulation of non-Lambertian phenomena”. In: *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques - SIGGRAPH ’95*. SIGGRAPH ’95. ACM Press, 1995. DOI: 10.1145/218380.218467.

[4] J. Arvo. “Stratified sampling of spherical triangles”. In: *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques - SIGGRAPH ’95*. SIGGRAPH ’95. ACM Press, 1995. DOI: 10.1145/218380.218500.

[5] J. Arvo and D. Kirk. “Particle transport and image synthesis”. In: *Proceedings of the 17th annual conference on Computer graphics and interactive techniques*. SIGGRAPH90. ACM, 1990. DOI: 10.1145/97879.97886.

[6] P. Bagchi and I. Guttman. “Theoretical considerations of the multivariate von Mises-Fisher distribution”. In: *Journal of Applied Statistics* 15.2 (1988). DOI: 10.1080/02664768800000022.

[7] A. Banerjee, I. S. Dhillon, J. Ghosh, and S. Sra. “Clustering on the Unit Hypersphere using von Mises-Fisher Distributions”. In: *J. Mach. Learn. Res.* 6 (2005). URL: https://jmlr.org/papers/v6/banerjee05a.html.

[8] M. Bangert, P. Hennig, and U. Oelfke. “Using an Infinite Von Mises-Fisher Mixture Model to Cluster Treatment Beam Directions in External Radiation Therapy”. In: *2010 Ninth International Conference on Machine Learning and Applications*. IEEE, 2010. DOI: 10.1109/icmla.2010.114.

[9] T. Bashford-Rogers, L. P. Santos, D. Marnerides, and K. Debattista. “Ensemble Metropolis Light Transport”. In: *ACM Transactions on Graphics* 41.1 (2021). DOI: 10.1145/3472294.

[10] T. Bashford-Rogers, K. Debattista, and A. Chalmers. “A Significance Cache for Accelerating Global Illumination”. In: *Computer Graphics Forum* 31.6 (2012). DOI: 10.1111/j.1467-8659.2012.02099.x.

[11] P. Bekaert, P. Slusallek, R. Cools, V. Havran, and H.-P. Seidel. *A custom designed density estimatorfor light transport*. Tech. rep. 2003.

[12] C. Benthin, R. Drabinski, L. Tessari, and A. Dittebrandt. “PLOC++: Parallel Locally-Ordered Clustering for Bounding Volume Hierarchy Construction Revisited”. In: *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 5.3 (2022). DOI: 10.1145/3543867.

[13] J. L. Bentley. “Multidimensional binary search trees used for associative searching”. In: *Communications of the ACM* 18.9 (1975). DOI: 10.1145/361002.361007.

[14] N. Binder, S. Fricke, and A. Keller. “Massively Parallel Path Space Filtering”. In: *Monte Carlo and Quasi-Monte Carlo Methods*. Springer International Publishing, 2022. DOI: 10.1007/978-3-030-98319-2_7.

[15] B. Bitterli. *Rendering resources.* 2016. URL: https://benedikt-bitterli.me/resources/.

[16] B. Bitterli, C. Wyman, M. Pharr, P. Shirley, A. Lefohn, and W. Jarosz. "Spatiotemporal reservoir resampling for real-time ray tracing with dynamic direct lighting". In: *ACM Transactions on Graphics* 39.4 (2020). DOI: 10.1145/3386569.3392481.

[17] J. Bittner and D. Meister. "T-SAH: Animation Optimized Bounding Volume Hierarchies". In: *Computer Graphics Forum* 34.2 (2015). DOI: 10.1111/cgf.12581.

[18] J. Bittner, M. Hapala, and V. Havran. "Fast Insertion-Based Optimization of Bounding Volume Hierarchies". In: *Computer Graphics Forum* 32.1 (2013). DOI: 10.1111/cgf.12000.

[19] J. Bittner, M. Hapala, and V. Havran. "Incremental BVH construction for ray tracing". In: *Computers & Graphics* 47 (2015). DOI: 10.1016/j.cag.2014.12.001.

[20] G. Boissé. "WORLD-SPACE SPATIOTEMPORAL RESERVOIR REUSE FOR RAY-TRACED GLOBAL ILLUMINATION". In: *SIGGRAPH Asia 2021 Technical Communications.* SA '21. ACM, 2021. DOI: 10.1145/3478512.3488613.

[21] J. Boksansky, P. Jukarainen, and C. Wyman. "Rendering Many Lights with Grid-Based Reservoirs". In: *Ray Tracing Gems II.* Apress, 2021. DOI: 10.1007/978-1-4842-7185-8_23.

[22] M. R. Bolin and G. W. Meyer. "An Error Metric for Monte Carlo Ray Tracing". In: *Rendering Techniques '97.* Springer Vienna, 1997. DOI: 10.1007/978-3-7091-6858-5_6.

[23] T. E. Booth. "Unbiased Monte Carlo Estimation of the Reciprocal of an Integral". In: *Nuclear Science and Engineering* 156.3 (2007). DOI: 10.13182/nse07-a2707.

[24] O. Cappé, A. Guillin, J. M. Marin, and C. P. Robert. "Population Monte Carlo". In: *Journal of Computational and Graphical Statistics* 13.4 (2004). DOI: 10.1198/106186004x12803.

[25] C. R. A. Chaitanya, A. S. Kaplanyan, C. Schied, M. Salvi, A. Lefohn, D. Nowrouzezahrai, and T. Aila. "Interactive reconstruction of Monte Carlo image sequences using a recurrent denoising autoencoder". In: *ACM Transactions on Graphics* 36.4 (2017). DOI: 10.1145/3072959.3073601.

[26] M. T. CHAO. "A general purpose unequal probability sampling plan". In: *Biometrika* 69.3 (1982). DOI: 10.1093/biomet/69.3.653.

[27] J. H. Clark. "Hierarchical geometric models for visible surface algorithms". In: *Communications of the ACM* 19.10 (1976). DOI: 10.1145/360349.360354.

[28] A. Conty Estevez and C. Kulla. "Importance Sampling of Many Lights with Adaptive Tree Splitting". In: *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 1.2 (2018). DOI: 10.1145/3233305.

[29] R. L. Cook and K. E. Torrance. "A Reflectance Model for Computer Graphics". In: *ACM Transactions on Graphics* 1.1 (1982). DOI: 10.1145/357290.357293.

[30] I. Corporation. *Intel Arc A-Series Graphics.* 2022. URL: https://ark.intel.com/content/www/us/en/ark/products/series/227957/intel-arc-a-series-graphics.html.

[31] C. Crassin, F. Neyret, M. Sainz, S. Green, and E. Eisemann. "Interactive indirect illumination using voxel cone tracing: a preview". In: *Symposium on Interactive 3D Graphics and Games.* I3D '11. ACM, 2011. DOI: 10.1145/1944745.1944787.

[32] K. Dahm and A. Keller. "Learning light transport the reinforced way". In: *ACM SIGGRAPH 2017 Talks.* SIGGRAPH '17. ACM, 2017. DOI: 10.1145/3084363.3085032.

[33] X. Deng, M. Hašan, N. Carr, Z. Xu, and S. Marschner. "Path graphs: iterative path space filtering". In: *ACM Transactions on Graphics* 40.6 (2021). DOI: 10.1145/3478513.3480547.

[34] M. Derevyannykh. “Real-Time Path-Guiding Based on Parametric Mixture Models”. In: (2022). DOI: 10.2312/EGS.20221024.

[35] A. Dittebrandt, J. Hanika, and C. Dachsbacher. “Temporal Sample Reuse for Next Event Estimation and Path Guiding for Real-Time Path Tracing”. In: *Eurographics Symposium on Rendering - DL-only Track* (2020). DOI: 10.2312/SR.20201135.

[36] A. Dittebrandt, V. Schüßler, J. Hanika, S. Herholz, and C. Dachsbacher. “Markov Chain Mixture Models for Real-Time Direct Illumination”. In: *Computer Graphics Forum* 42.4 (2023). DOI: 10.1111/cgf.14881.

[37] A. Dodik, M. Papas, C. Öztireli, and T. Müller. “Path Guiding Using Spatio-Directional Mixture Models”. In: *Computer Graphics Forum* 41.1 (2021). DOI: 10.1111/cgf.14428.

[38] L. R. Domingues and H. Pedrini. “Bounding volume hierarchy optimization through agglomerative treelet restructuring”. In: *Proceedings of the 7th Conference on High-Performance Graphics*. HPG ’15. ACM, 2015. DOI: 10.1145/2790060.2790065.

[39] M. Donikian, B. Walter, K. Bala, S. Fernandez, and D. Greenberg. “Accurate direct illumination using iterative adaptive sampling”. In: *IEEE Transactions on Visualization and Computer Graphics* 12.3 (2006). DOI: 10.1109/tvcg.2006.41.

[40] M. Droske, J. Hanika, J. Vorba, A. Weidlich, and M. Sabbadin. “Path tracing in Production: The Path of Water”. In: *ACM SIGGRAPH 2023 Courses*. SIGGRAPH ’23. ACM, 2023. DOI: 10.1145/3587423.3595519.

[41] J. Dupuy, J.-C. Iehl, and P. Poulin. “Quadtrees on the GPU”. In: *GPU Pro 5*. A K Peters/CRC Press, 2014. ISBN: 9781482208634.

[42] E. Eisemann, ed. *Real-time shadows*. CRC Press, 2012. ISBN: 9781466538665.

[43] V. Elvira and E. Chouzenoux. “Optimized Population Monte Carlo”. In: *IEEE Transactions on Signal Processing* 70 (2022). DOI: 10.1109/tsp.2022.3172619.

[44] V. Elvira, L. Martino, D. Luengo, and M. F. Bugallo. “Population Monte Carlo schemes with reduced path degeneracy”. In: *2017 IEEE 7th International Workshop on Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP)*. IEEE, 2017. DOI: 10.1109/camsap.2017.8313090.

[45] T. Engelhardt and C. Dachsbacher. “Octahedron Environment Maps”. In: *International Symposium on Vision, Modeling, and Visualization*. 2008. URL: https://api.semanticscholar.org/CorpusID:5195042.

[46] L. Fascione, J. Hanika, D. Heckenberg, C. Kulla, M. Droske, and J. Schwarzhaupt. “Path tracing in production: part 1: modern path tracing”. In: *ACM SIGGRAPH 2019 Courses*. SIGGRAPH ’19. ACM, 2019. DOI: 10.1145/3305366.3328079.

[47] J. D. Foley, A. V. Dam, and S. K. Feiner. *Computer Graphics. Principles and Practice*. 2nd ed. Addison-Wesley Publishing Company, 1995. ISBN: 0201121107.

[48] H. Fuchs, Z. M. Kedem, and B. F. Naylor. “On visible surface generation by a priori tree structures”. In: *ACM SIGGRAPH Computer Graphics* 14.3 (1980). DOI: 10.1145/965105.807481.

[49] P. Ganestam, R. Barringer, M. Doggett, and T. Akenine-Möller. “Bonsai: Rapid Bounding Volume Hierarchy Generation using Mini Trees”. In: *Journal of Computer Graphics Techniques (JCGT)* 4.3 (2015). ISSN: 2331-7418.

[50] P. Ganestam and M. Doggett. “SAH guided spatial split partitioning for fast BVH construction”. In: *Computer Graphics Forum* 35.2 (2016). DOI: 10.1111/cgf.12831.

[51] K. Garanzha, J. Pantaleoni, and D. McAllister. “Simpler and faster HLBVH with work queues”. In: *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics*. HPG ’11. ACM, 2011. DOI: `10.1145/2018323.2018333`.

[52] H.-O. Georgii. *Stochastik: Einführung in die Wahrscheinlichkeitstheorie und Statistik*. DE GRUYTER, 2015. DOI: `10.1515/9783110359701`.

[53] J. Goldsmith and J. Salmon. “Automatic Creation of Object Hierarchies for Ray Tracing”. In: *IEEE Computer Graphics and Applications* 7.5 (1987). DOI: `10.1109/mcg.1987.276983`.

[54] A. Gruson, R. West, and T. Hachisuka. “Stratified Markov Chain Monte Carlo Light Transport”. In: *Computer Graphics Forum* 39.2 (2020). DOI: `10.1111/cgf.13935`.

[55] Y. Gu, Y. He, K. Fatahalian, and G. Blelloch. “Efficient BVH construction via approximate agglomerative clustering”. In: *Proceedings of the 5th High-Performance Graphics Conference*. HPG ’13. ACM, 2013. DOI: `10.1145/2492045.2492054`.

[56] H. Haario, E. Saksman, and J. Tamminen. “An Adaptive Metropolis Algorithm”. In: *Bernoulli* 7.2 (2001). DOI: `10.2307/3318737`.

[57] T. Hachisuka, W. Jarosz, R. P. Weistroffer, K. Dale, G. Humphreys, M. Zwicker, and H. W. Jensen. “Multidimensional adaptive sampling and reconstruction for ray tracing”. In: *ACM Transactions on Graphics* 27.3 (2008). DOI: `10.1145/1360612.1360632`.

[58] T. Hachisuka and H. W. Jensen. “Parallel progressive photon mapping on GPUs”. In: *ACM SIGGRAPH ASIA 2010 Sketches*. SA ’10. ACM, 2010. DOI: `10.1145/1899950.1900004`.

[59] S. Hadadan, S. Chen, and M. Zwicker. “Neural radiosity”. In: *ACM Transactions on Graphics* 40.6 (2021). DOI: `10.1145/3478513.3480569`.

[60] H. Halen, A. Brinck, K. Hayward, and X. Bei. *Global Illumination Based on Surfels*. Presentation. 2021.

[61] J. Hanika. *Lecture on photorealistic image synthesis*. Karlsruhe Institute of Technology. 2024.

[62] J. Hanika, M. Droske, and L. Fascione. “Manifold Next Event Estimation”. In: *Computer Graphics Forum* 34.4 (2015). DOI: `10.1111/cgf.12681`.

[63] J. Hanika, L. Tessari, and C. Dachsbacher. “Fast temporal reprojection without motion vectors”. In: *Journal of Computer Graphics Techniques (JCGT)* 10.3 (2021). ISSN: 2331-7418.

[64] D. Hart, M. Pharr, T. Müller, W. Lopes, M. McGuire, and P. Shirley. “Practical Product Sampling by Fitting and Composing Warps”. In: *Computer Graphics Forum* 39.4 (2020). DOI: `10.1111/cgf.14060`.

[65] D. Hart, P. Dutré, and D. P. Greenberg. “Direct illumination with lazy visibility evaluation”. In: *Proceedings of the 26th annual conference on Computer graphics and interactive techniques - SIGGRAPH ’99*. SIGGRAPH ’99. ACM Press, 1999. DOI: `10.1145/311535.311551`.

[66] P. S. Heckbert. “Adaptive radiosity textures for bidirectional ray tracing”. In: *ACM SIGGRAPH Computer Graphics* 24.4 (1990). DOI: `10.1145/97880.97895`.

[67] E. Heitz. “Understanding the Masking-Shadowing Function in Microfacet-Based BRDFs”. In: *Journal of Computer Graphics Techniques (JCGT)* 3.2 (2014). ISSN: 2331-7418.

[68] E. Heitz, J. Dupuy, S. Hill, and D. Neubelt. “Real-time polygonal-light shading with linearly transformed cosines”. In: *ACM Transactions on Graphics* 35.4 (2016). DOI: `10.1145/2897824.2925895`.

[69] J. Hendrich, D. Meister, and J. Bittner. “Parallel BVH Construction using Progressive Hierarchical Refinement”. In: *Computer Graphics Forum* 36.2 (2017). DOI: `10.1111/cgf.13143`.

[70] S. Herholz and A. Dittebrandt. *Intel® Open Path Guiding Library.* 2022.

[71] S. Herholz, O. Elek, J. Vorba, H. Lensch, and J. Křivánek. "Product Importance Sampling for Light Transport Path Guiding". In: *Computer Graphics Forum* 35.4 (2016). DOI: `10.1111/cgf.12950`.

[72] S. Herholz, Y. Zhao, O. Elek, D. Nowrouzezahrai, H. P. A. Lensch, and J. Křivánek. "Volume Path Guiding Based on Zero-Variance Random Walk Theory". In: *ACM Transactions on Graphics* 38.3 (2019). DOI: `10.1145/3230635`.

[73] T. Hesterberg. "Weighted Average Importance Sampling and Defensive Mixture Distributions". In: *Technometrics* 37.2 (1995). DOI: `10.1080/00401706.1995.10484303`.

[74] H. Hey and W. Purgathofer. "Importance sampling with hemispherical particle footprints". In: *Proceedings of the 18th Spring Conference on Computer Graphics.* PCK50. ACM, 2002. DOI: `10.1145/584458.584476`.

[75] G. Jacobson. "Space-efficient static trees and graphs". In: *30th Annual Symposium on Foundations of Computer Science.* IEEE, 1989. DOI: `10.1109/sfcs.1989.63533`.

[76] H. W. Jensen. "Global Illumination using Photon Maps". In: *Rendering Techniques '96.* Springer Vienna, 1996. DOI: `10.1007/978-3-7091-7484-5_3`.

[77] H. W. Jensen. "Importance Driven Path Tracing using the Photon Map". In: *Rendering Techniques '95.* Springer Vienna, 1995. DOI: `10.1007/978-3-7091-9430-0_31`.

[78] J. L. W. V. Jensen. "Sur les fonctions convexes et les inégalités entre les valeurs moyennes". In: *Acta Mathematica* 30.0 (1906). DOI: `10.1007/bf02418571`.

[79] J. T. Kajiya. "The rendering equation". In: *ACM SIGGRAPH Computer Graphics* 20.4 (1986). DOI: `10.1145/15886.15902`.

[80] S. Kallweit et al. *The Falcor Rendering Framework.* 2022.

[81] L. Kandlbinder, A. Dittebrandt, A. Schipek, and C. Dachsbacher. "Optimizing Path Termination for Radiance Caching Through Explicit Variance Trading". In: *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 7.3 (2024). DOI: `10.1145/3675381`.

[82] A. Kaplanyan and C. Dachsbacher. "Cascaded light propagation volumes for real-time indirect illumination". In: *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games.* I3D '10. ACM, 2010. DOI: `10.1145/1730804.1730821`.

[83] A. S. Kaplanyan and C. Dachsbacher. "Adaptive progressive photon mapping". In: *ACM Transactions on Graphics* 32.2 (2013). DOI: `10.1145/2451236.2451242`.

[84] O. Karlík, M. Šik, P. Vévoda, T. Skřivan, and J. Křivánek. "MIS compensation: optimizing sampling techniques in multiple importance sampling". In: *ACM Transactions on Graphics* 38.6 (2019). DOI: `10.1145/3355089.3356565`.

[85] T. Karras. *Maximizing Parallelism in the Construction of BVHs, Octrees, and k-d Trees.* en. 2012. DOI: `10.2312/EGGH/HPG12/033-037`.

[86] T. Karras and T. Aila. "Fast parallel construction of high-quality bounding volume hierarchies". In: *Proceedings of the 5th High-Performance Graphics Conference.* HPG '13. ACM, 2013. DOI: `10.1145/2492045.2492055`.

[87] A. Keller, K. Dahm, and N. Binder. "Path Space Filtering". In: *Monte Carlo and Quasi-Monte Carlo Methods.* Springer International Publishing, 2016. DOI: `10.1007/978-3-319-33507-0_21`.

[88] A. Keller, C. Wächter, M. Raab, D. Seibert, D. van Antwerpen, J. Korndörfer, and L. Kettner. "The iray light transport simulation and rendering system". In: *ACM SIGGRAPH 2017 Talks.* SIGGRAPH '17. ACM, 2017. DOI: `10.1145/3084363.3085050`.

[89] M. Kettunen, E. D'Eon, J. Pantaleoni, and J. Novák. "An unbiased ray-marching transmittance estimator". In: *ACM Transactions on Graphics* 40.4 (2021). DOI: 10.1145/3450626.3459937.

[90] J. Krivanek, P. Gautron, S. Pattanaik, and K. Bouatouch. "Radiance Caching for Efficient Global Illumination Computation". In: *IEEE Transactions on Visualization and Computer Graphics* 11.5 (2005). DOI: 10.1109/tvcg.2005.83.

[91] E. P. Lafortune and Y. D. Willems. "A 5D Tree to Reduce the Variance of Monte Carlo Ray Tracing". In: *Rendering Techniques '95*. Springer Vienna, 1995. DOI: 10.1007/978-3-7091-9430-0_2.

[92] C. Lauterbach, M. Garland, S. Sengupta, D. Luebke, and D. Manocha. "Fast BVH Construction on GPUs". In: *Computer Graphics Forum* 28.2 (2009). DOI: 10.1111/j.1467-8659.2009.01377.x.

[93] C. Lauterbach, S.-e. Yoon, D. Manocha, and D. Tuft. "RT-DEFORM: Interactive Ray Tracing of Dynamic Scenes using BVHs". In: *2006 IEEE Symposium on Interactive Ray Tracing*. IEEE, 2006. DOI: 10.1109/rt.2006.280213.

[94] A. J. Lee. *U-Statistics*. 2019. DOI: 10.1201/9780203734520.

[95] D. Lin, M. Kettunen, B. Bitterli, J. Pantaleoni, C. Yuksel, and C. Wyman. "Generalized resampled importance sampling: foundations of ReSTIR". In: *ACM Transactions on Graphics* 41.4 (2022). DOI: 10.1145/3528223.3530158.

[96] D. Lin, C. Wyman, and C. Yuksel. "Fast volume rendering with spatiotemporal reservoir resampling". In: *ACM Transactions on Graphics* 40.6 (2021). DOI: 10.1145/3478513.3480499.

[97] Y. Liu, K. Xu, and L.-Q. Yan. "Adaptive BRDF-Oriented Multiple Importance Sampling of Many Lights". In: *Computer Graphics Forum* 38.4 (2019). DOI: 10.1111/cgf.13776.

[98] A. Lumberyard. *Amazon Lumberyard Bistro, Open Research Content Archive (ORCA)*. 2017.

[99] J. D. MacDonald and K. S. Booth. "Heuristics for ray tracing using space subdivision". In: *The Visual Computer* 6.3 (1990). DOI: 10.1007/bf01911006.

[100] M. Mara, M. McGuire, B. Bitterli, and W. Jarosz. "An efficient denoising algorithm for global illumination". In: *Proceedings of High Performance Graphics*. HPG '17. ACM, 2017. DOI: 10.1145/3105762.3105774.

[101] J. Marco, A. Jarabo, W. Jarosz, and D. Gutierrez. "Second-Order Occlusion-Aware Volumetric Radiance Caching". In: *ACM Transactions on Graphics* 37.2 (2018). DOI: 10.1145/3185225.

[102] E. Marinari and G. Parisi. "Simulated Tempering: A New Monte Carlo Scheme". In: *Europhysics Letters (EPL)* 19.6 (1992). DOI: 10.1209/0295-5075/19/6/002.

[103] L. Martino, V. Elvira, D. Luengo, and J. Corander. "Layered adaptive importance sampling". In: *Statistics and Computing* 27.3 (2016). DOI: 10.1007/s11222-016-9642-5.

[104] M. McGuire. *Computer Graphics Archive*. 2017. URL: https://casual-effects.com/data.

[105] D. Meagher. "Geometric modeling using octree encoding". In: *Computer Graphics and Image Processing* 19.2 (1982). DOI: 10.1016/0146-664x(82)90104-6.

[106] D. Meister and J. Bittner. "Parallel Reinsertion for Bounding Volume Hierarchy Optimization". In: *Computer Graphics Forum* 37.2 (2018). DOI: 10.1111/cgf.13376.

[107] D. Meister and J. Bittner. "Parallel BVH construction using k-means clustering". In: *The Visual Computer* 32.6–8 (2016). DOI: 10.1007/s00371-016-1241-0.

[108] D. Meister and J. Bittner. "Performance Comparison of Bounding Volume Hierarchies for GPU Ray Tracing". In: *Journal of Computer Graphics Techniques (JCGT)* 11.4 (2022). ISSN: 2331-7418.

[109] D. Meister and J. Bittner. “Parallel Locally-Ordered Clustering for Bounding Volume Hierarchy Construction”. In: *IEEE Transactions on Visualization and Computer Graphics* 24.3 (2018). DOI: `10.1109/tvcg.2017.2669983`.

[110] D. Meister, S. Ogaki, C. Benthin, M. J. Doyle, M. Guthe, and J. Bittner. “A Survey on Bounding Volume Hierarchies for Ray Tracing”. In: *Computer Graphics Forum* 40.2 (2021). DOI: `10.1111/cgf.142662`.

[111] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. “Equation of State Calculations by Fast Computing Machines”. In: *The Journal of Chemical Physics* 21.6 (1953). DOI: `10.1063/1.1699114`.

[112] D. P. Mitchell. “Generating antialiased images at low sampling densities”. In: *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*. SIGGRAPH ’87. ACM, 1987. DOI: `10.1145/37401.37410`.

[113] P. Moreau and P. Clarberg. “Importance Sampling of Many Lights on the GPU”. In: *Ray Tracing Gems*. Apress, 2019. DOI: `10.1007/978-1-4842-4427-2_18`.

[114] P. Moreau, M. Pharr, and P. Clarberg. “Dynamic Many-Light Sampling for Real-Time Ray Tracing”. In: *High-Performance Graphics - Short Papers* (2019). DOI: `10.2312/HPG.20191191`.

[115] T. Müller, A. Evans, C. Schied, and A. Keller. “Instant neural graphics primitives with a multiresolution hash encoding”. In: *ACM Transactions on Graphics* 41.4 (2022). DOI: `10.1145/3528223.3530127`.

[116] T. Müller, M. Gross, and J. Novák. “Practical Path Guiding for Efficient Light-Transport Simulation”. In: *Computer Graphics Forum* 36.4 (2017). DOI: `10.1111/cgf.13227`.

[117] T. Müller, F. Rousselle, J. Novák, and A. Keller. “Real-time neural radiance caching for path tracing”. In: *ACM Transactions on Graphics* 40.4 (2021). DOI: `10.1145/3450626.3459812`.

[118] J. Munkberg, J. Hasselgren, P. Clarberg, M. Andersson, and T. Akenine-Möller. “Texture space caching and reconstruction for ray tracing”. In: *ACM Transactions on Graphics* 35.6 (2016). DOI: `10.1145/2980179.2982407`.

[119] K. Ng and B. Trifonov. “Automatic bounding volume hierarchy generation using stochastic search methods”. In: *Mini-workshop on stochastic search algorithms*. 2003.

[120] NVIDIA. *NVIDIA Turing GPU Architecture*. 2018.

[121] J. O’Rourke. “Finding minimal enclosing boxes”. In: *International Journal of Computer & Information Sciences* 14.3 (1985). DOI: `10.1007/bf00991005`.

[122] O. Olsson, M. Billeter, and U. Assarsson. *Clustered Deferred and Forward Shading*. en. 2012. DOI: `10.2312/EGGH/HPG12/087-096`.

[123] O. Olsson, E. Sintorn, V. Kämpe, M. Billeter, and U. Assarsson. “Efficient virtual shadow maps for many lights”. In: *Proceedings of the 18th meeting of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. I3D ’14. ACM, 2014. DOI: `10.1145/2556700.2556701`.

[124] Y. Ouyang, S. Liu, M. Kettunen, M. Pharr, and J. Pantaleoni. “ReSTIR GI: Path Resampling for Real-Time Path Tracing”. In: *Computer Graphics Forum* 40.8 (2021). DOI: `10.1111/cgf.14378`.

[125] J. Pantaleoni. “Online Path Sampling Control with Progressive Spatio-temporal Filtering”. In: *SN Computer Science* 1.5 (2020). DOI: `10.1007/s42979-020-00291-z`.

[126] J. Pantaleoni and D. Luebke. *HLBVH: Hierarchical LBVH Construction for Real-Time Ray Tracing of Dynamic Geometry*. en. 2010. DOI: `10.2312/EGGH/HPG10/087-095`.

[127] C. Peters. “BRDF importance sampling for polygonal lights”. In: *ACM Transactions on Graphics* 40.4 (2021). DOI: 10.1145/3450626.3459672.

[128] M. Pharr. *Physically based rendering. From theory to implementation.* Ed. by W. Jakob and G. Humphreys. Fourth edition. The MIT Press, 2023. ISBN: 9780262048026.

[129] A. Rath, P. Grittmann, S. Herholz, P. Vévoda, P. Slusallek, and J. Křivánek. “Variance-aware path guiding”. In: *ACM Transactions on Graphics* 39.4 (2020). DOI: 10.1145/3386569.3392441.

[130] A. Rath, P. Grittmann, S. Herholz, P. Weier, and P. Slusallek. “EARS: efficiency-aware russian roulette and splitting”. In: *ACM Transactions on Graphics* 41.4 (2022). DOI: 10.1145/3528223.3530168.

[131] F. Reibold, J. Hanika, A. Jung, and C. Dachsbacher. “Selective guided sampling with complete light transport paths”. In: *ACM Transactions on Graphics* 37.6 (2018). DOI: 10.1145/3272127.3275030.

[132] T. Ritschel, C. Dachsbacher, T. Grosch, and J. Kautz. “The State of the Art in Interactive Global Illumination”. In: *Computer Graphics Forum* 31.1 (2012). DOI: 10.1111/j.1467-8659.2012.02093.x.

[133] D. Rudolf and B. Sprungk. “On a Metropolis–Hastings importance sampling estimator”. In: *Electronic Journal of Statistics* 14.1 (2020). DOI: 10.1214/20-ejs1680.

[134] L. Ruppert, S. Herholz, and H. P. A. Lensch. “Robust fitting of parallax-aware mixtures for path guiding”. In: *ACM Transactions on Graphics* 39.4 (2020). DOI: 10.1145/3386569.3392421.

[135] R. Sawhney, D. Lin, M. Kettunen, B. Bitterli, R. Ramamoorthi, C. Wyman, and M. Pharr. “Decorrelating ReSTIR Samplers via MCMC Mutations”. In: *ACM Transactions on Graphics* 43.1 (2024). DOI: 10.1145/3629166.

[136] C. Schied, A. Kaplanyan, C. Wyman, A. Patney, C. R. A. Chaitanya, J. Burgess, S. Liu, C. Dachsbacher, A. Lefohn, and M. Salvi. “Spatiotemporal variance-guided filtering: real-time reconstruction for path-traced global illumination”. In: *Proceedings of High Performance Graphics.* HPG ’17. ACM, 2017. DOI: 10.1145/3105762.3105770.

[137] V. Schüßler, J. Hanika, A. Jung, and C. Dachsbacher. “Path Guiding with Vertex Triplet Distributions”. In: *Computer Graphics Forum* 41.4 (2022). DOI: 10.1111/cgf.14582.

[138] I. Schuster and I. Klebanov. “Markov Chain Importance Sampling—A Highly Efficient Estimator for MCMC”. In: *Journal of Computational and Graphical Statistics* 30.2 (2021). DOI: 10.1080/10618600.2020.1826953.

[139] J. Schwarzhaupt, H. W. Jensen, and W. Jarosz. “Practical Hessian-based error control for irradiance caching”. In: *ACM Transactions on Graphics* 31.6 (2012). DOI: 10.1145/2366145.2366212.

[140] P. Shirley, C. Wang, and K. Zimmerman. “Monte Carlo techniques for direct lighting calculations”. In: *ACM Transactions on Graphics* 15.1 (1996). DOI: 10.1145/226150.226151.

[141] I. Sobol’. “On the distribution of points in a cube and the approximate evaluation of integrals”. In: *USSR Computational Mathematics and Mathematical Physics* 7.4 (1967). DOI: 10.1016/0041-5553(67)90144-9.

[142] M. Stark, P. Shirley, and M. Ashikhmin. “Generation of Stratified Samples for B-Spline Pixel Filtering”. In: *Journal of Graphics Tools* 10.1 (2005). DOI: 10.1080/2151237x.2005.10129186.

[143] R. H. Swendsen and J.-S. Wang. “Replica Monte Carlo Simulation of Spin-Glasses”. In: *Physical Review Letters* 57.21 (1986). DOI: 10.1103/physrevlett.57.2607.

[144] J. Talbot, D. Cline, and P. Egbert. *Importance Resampling for Global Illumination.* en. 2005. DOI: 10.2312/EGWR/EGSR05/139-146.

[145] S. J. Teller. “Visibility Computations in Densely Occluded Polyhedral Environments”. PhD thesis. EECS Department, University of California, Berkeley, 1992. URL: `http://www2.eecs.berkeley.edu/Pubs/TechRpts/1992/6250.html`.

[146] L. Tessari, A. Dittebrandt, M. J. Doyle, and C. Benthin. “Stochastic Subsets for BVH Construction”. In: *Computer Graphics Forum* 42.2 (2023). DOI: `10.1111/cgf.14759`.

[147] W. C. Thibault and B. F. Naylor. “Set operations on polyhedra using binary space partitioning trees”. In: *ACM SIGGRAPH Computer Graphics* 21.4 (1987). DOI: `10.1145/37402.37421`.

[148] M. M. Thomas, G. Liktor, C. Peters, S. Kim, K. Vaidyanathan, and A. G. Forbes. “Temporally Stable Real-Time Joint Neural Denoising and Supersampling”. In: *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 5.3 (2022). DOI: `10.1145/3543870`.

[149] T. S. Trowbridge and K. P. Reitz. “Average irregularity representation of a rough surface for ray reflection”. In: *Journal of the Optical Society of America* 65.5 (1975). DOI: `10.1364/josa.65.000531`.

[150] E. Veach. “Robust monte carlo methods for light transport simulation”. PhD thesis. 1998. ISBN: 0591907801.

[151] P. Vévoda, I. Kondapaneni, and J. Křivánek. “Bayesian online regression for adaptive direct illumination sampling”. In: *ACM Transactions on Graphics* 37.4 (2018). DOI: `10.1145/3197517.3201340`.

[152] P. Vévoda and J. Křivánek. “Adaptive direct illumination sampling”. In: *SIGGRAPH ASIA 2016 Posters*. SA ’16. ACM, 2016. DOI: `10.1145/3005274.3005283`.

[153] M. Vinkler, J. Bittner, and V. Havran. “Extended Morton codes for high performance bounding volume hierarchy construction”. In: *Proceedings of High Performance Graphics*. HPG ’17. ACM, 2017. DOI: `10.1145/3105762.3105782`.

[154] J. Vorba, J. Hanika, S. Herholz, T. Müller, J. Křivánek, and A. Keller. “Path guiding in production”. In: *ACM SIGGRAPH 2019 Courses*. SIGGRAPH ’19. ACM, 2019. DOI: `10.1145/3305366.3328091`.

[155] J. Vorba, O. Karlík, M. Šik, T. Ritschel, and J. Křivánek. “On-line learning of parametric mixture models for light transport simulation”. In: *ACM Transactions on Graphics* 33.4 (2014). DOI: `10.1145/2601097.2601203`.

[156] J. Vorba and J. Křivánek. “Adjoint-driven Russian roulette and splitting in light transport simulation”. In: *ACM Transactions on Graphics* 35.4 (2016). DOI: `10.1145/2897824.2925912`.

[157] I. Wald. “On fast Construction of SAH-based Bounding Volume Hierarchies”. In: *2007 IEEE Symposium on Interactive Ray Tracing*. IEEE, 2007. DOI: `10.1109/rt.2007.4342588`.

[158] A. J. Walker. “An Efficient Method for Generating Discrete Random Variables with General Distributions”. In: *ACM Transactions on Mathematical Software* 3.3 (1977). DOI: `10.1145/355744.355749`.

[159] B. Walter, K. Bala, M. Kulkarni, and K. Pingali. “Fast agglomerative clustering for rendering”. In: *2008 IEEE Symposium on Interactive Ray Tracing*. IEEE, 2008. DOI: `10.1109/rt.2008.4634626`.

[160] B. Walter, S. Fernandez, A. Arbree, K. Bala, M. Donikian, and D. P. Greenberg. “Lightcuts: a scalable approach to illumination”. In: *ACM SIGGRAPH 2005 Papers*. SIGGRAPH05. ACM, 2005. DOI: `10.1145/1186822.1073318`.

[161] B. Walter, S. R. Marschner, H. Li, and K. E. Torrance. *Microfacet Models for Refraction through Rough Surfaces*. en. 2007. DOI: `10.2312/EGWR/EGSR07/195-206`.

[162] G. J. Ward. “Adaptive Shadow Testing for Ray Tracing”. In: *Photorealistic Rendering in Computer Graphics*. Springer Berlin Heidelberg, 1994. DOI: `10.1007/978-3-642-57963-9_2`.

[163] G. J. Ward, F. M. Rubinstein, and R. D. Clear. “A ray tracing solution for diffuse interreflection”. In: *ACM SIGGRAPH Computer Graphics* 22.4 (1988). DOI: `10.1145/378456.378490`.

[164] R. West, I. Georgiev, A. Gruson, and T. Hachisuka. “Continuous multiple importance sampling”. In: *ACM Transactions on Graphics* 39.4 (2020). DOI: `10.1145/3386569.3392436`.

[165] A. Wilkie, S. Nawaz, M. Droske, A. Weidlich, and J. Hanika. “Hero Wavelength Spectral Sampling”. In: *Computer Graphics Forum* 33.4 (2014). DOI: `10.1111/cgf.12419`.

[166] D. Wright, K. Narkowicz, and P. Kelly. *Real-time Global Illumination in Unreal Engine 5*. Presentation. 2022.

[167] H. Ylitie, T. Karras, and S. Laine. “Efficient incoherent ray traversal on GPUs through compressed wide BVHs”. In: *Proceedings of High Performance Graphics*. HPG ’17. ACM, 2017. DOI: `10.1145/3105762.3105773`.

[168] C. Yuksel. “Stochastic Lightcuts”. In: *High-Performance Graphics - Short Papers* (2019). DOI: `10.2312/HPG.20191192`.

[169] T. Zeltner, I. Georgiev, and W. Jakob. “Specular manifold sampling for rendering high-frequency caustics and glints”. In: *ACM Transactions on Graphics* 39.4 (2020). DOI: `10.1145/3386569.3392408`.

[170] Q. Zheng and M. Zwicker. “Learning to Importance Sample in Primary Sample Space”. In: *Computer Graphics Forum* 38.2 (2019). DOI: `10.1111/cgf.13628`.