Brandenburgische Technische Universität Cottbus
Institut für Informatik, Informations- und Medientechnik
Lehrstuhl Grafische Systeme

Studienarbeit

von Hagen Steidelmüller
Matrikel: 2020890

Erweiterung eines bestehenden Raytracers um Bidirektionales Raytracing und Metropolis-Tracing

Betreuer:
Prof. Dr. Winfried Kurth

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides Statt, dass ich die vorliegende Arbeit selbständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

Cottbus, 20. September 2008 _____

Unterschrift

Danksagung

Ich möchte mich bedanken

bei Herrn Prof. Dr. Winfried Kurth, Dr. Ole Kniemeyer und Reinhard Hemmerling für die Betreuung meiner Arbeit,

bei Ralf Kopsch für lange und anregenden Diskussionen über die verschiedenen Algorithmen,

bei Janine Lentzy für das Korrekturlesen, die konstruktive Kritik und das rege Interesse,

und nicht zuletzt bei meinen Eltern für ihr Vertrauen und den bedingungslosen Rückhalt, den ich immer wieder bei ihnen finden konnte.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

ray tracing engl.; Strahlverfolgung; ein auf Aussenden von Strahlen basierender Algorithmus der 3D-Computergrafik, um auf physikalisch möglichst korrekte Art Verschattungs- und Beleuchtungsszenarien zu berechnen.

Mit ihren Arbeiten in den frühen 60iger Jahren des letzten Jahrhunderts schufen Appel [App68], Goldstein und Nagel [GN71] die Grundlagen der modernen 3d-Computergrafik. Ihr damals vorgestellter Raytracing-Algorithmus bildete den Ausgangspunkt vieler weiterer Entwicklungen, die heute Kernelemente einer Milliarden schweren Grafikindustrie sind.
Der Algorithmus selbst bot genug Ausbaupotential, so dass im Laufe der Jahre viele einander ergänzende Verfahren entwickelt wurden, die mit den immer höher werdenden Anforderungen der Industrie schritthalten konnten.
Im Jahre 1986 veröffentlichte Kaija die so genannte Rendergleichung [Kaj86]. Das dort beschriebene Path Tracing (PT) war erstmals in der Lage eine globale Beleuchtungssimulation durchzuführen, d.h. alle Möglichkeiten der Lichtstrahlausbreitung in einer Szene korrekt zu berechnen.
1993 und 1994 veröffentlichten Lafortune & Willems [LW93] und Guibas & Veach [LW94] von einander unabhängig eine Erweiterung des PT: das sogenannte Bidirectional Path Tracing (BPT). 1997 erweiterten Veach & Guibas den Ansatz zum Metropolis Light Transport (MLT)[VG97].

Diese beiden Algorithmen - vielmehr ihre Implementierung und Eingliederung in das „Growth Grammar-related Interactive Modelling Platform (GroIMP)“ [Gro07] des Lehr-

stuhls „Grafische Systeme“ der BTU Cottbus stellen den inhaltlichen Schwerpunkt der Studienarbeit dar und werden im Folgenden näher beschrieben.

Auch wenn das PT die Rendergleichung prinzipiell lösen kann und damit in der Lage ist eine globale Beleuchtung zu simulieren, ist es doch einigen pragmatischen Einschränkungen unterworfen, die seinen Einsatz limitieren. Um beispielsweise komplexe Beleuchtungsszenarien erfolgreich zu rendern, in denen Kaustikeffekte zum Tragen kommen oder ein hoher Anteil indirekten Lichts vorherrscht, muss eine hohe Anzahl von stochastisch verteilten Strahlen generiert und durch die Szene gesandt werden, was die Gesamtberechnung nur wenig performant macht.
Wie Veach in seiner Dissertation [Vea97] zeigen konnte, sind sowohl BPT als auch MLT in solchen Fällen die bessere Wahl. Die Erweiterungen des PT liefern sogar bessere Ergebnisse bei vergleichbarer Rechenzeit.

Hauptziel der vorliegenden schriftlichen Arbeit ist es, dem interessierten Leser einen Einstieg in das BPT- und das MLT-Verfahren zu geben, ihm deren Funktionsprinzip zu erklären und die besonderen Eigenschaften sowie die Vor- und Nachteile zu veranschaulichen. Des Weiteren soll das Vorgehen bei der Implementierung des BPT und des MLT beschrieben, auf die dabei aufgetretenen Probleme und GroIMP-spezifischen Lösungen eingegangen und einen Ausgangspunkt für weiterführende Implementierungen geben werden. Letztlich soll der Leser mit den Renderern in GroIMP umgehen und die Arbeitsweise nachvollziehen können.

Die vorliegende Arbeit ist wie folgt gegliedert.
In Kapitel 2 wird vorbereitend ein kurzer Überblick über die 3D-Modellierungsplattform GroIMP gegeben.
Um genügend Vorwissen seitens des Lesers zu gewährleisten, wird in Kapitel 3 nochmals kurz auf das PT eingegangen, das ja die Basis sowohl für den BPT als auch den MLT bildet.
Daraufhin stellt Kapitel 4 das BPT-Verfahren genauer vor. Es werden dem Leser die Grundlagen des Algorithmusses, elementare Parameter, sowie der Umgang mit dem zugehörigen Renderer-Plugin in GroIMP nähergebracht.
Das gleiche gilt für Kapitel 5, welches sich mit dem MLT-Algoritmus befasst und dessen Renderergebnisse mit dem des BPT vergleicht.

Kapitel 2

GroIMP

Das nachfolgende Kapitel gibt einen kurzen Einblick in die Menüführung der Modellierungsplattform GroIMP. Dabei beschränkt es sich auf das Notwendigste, um dem Leser den Umgang mit den Raytracer-Plugins und das Rendern einer 3D-Szene in GroIMP zu ermöglichen. Weiterführende Informationen zu GroIMP können auf den Projekt-Webseiten [Gro07] gefunden werden.

2.1. GroIMP Übersicht

GroIMP ist ein Akronym für „Growth Grammar-related Interactive Modelling Platform“ [Gro07] und wird betreut und entwickelt am Lehrstuhl „Grafische Systeme“ der BTU Cottbus [Kur]. Wie Ralf Kopsch bereits in seiner Arbeit [Kop08] beschrieben hat, handelt es sich dabei um eine in Java programmierte 3D-Modellierungsplattform. Sie ist darauf spezialisiert, Modelle mithilfe von Wachstumsgrammatiken erzeugen und darstellen zu können. Im Rahmen dieser Studienarbeit wurde sie um verschiedene Render-Verfahren erweitert:

- Radiosity und
- Photon-Mapping[1],
- Bidirectional Path Tracing und

[1] durch Ralf Kopsch

- Metropolis Light Transport[2].

Nach Fertigstellung dieser Verfahren ist GroIMP nach Wissen des Autors das einzige OpenSource-Projekt, dass diese ausgefeilten Rendering-Algorithmen implementiert.

2.2. GroIMP Raytracing-Optionen

Das Editieren der GroIMP-Programmparametern, die das Verhalten der Anwendung steuern, ist relativ einfach gestaltet.
Über die Menüleiste des Hauptfensters («Panels → Preferences») gelangt der Nutzer zum Konfigurationsfenster der GroIMP-Anwendung. Wie in Abbildung 2.1 zu erkennen, lassen sich eine Vielzahl von Parametern bearbeiten - unter anderem auch die der eingebauten Software-Renderer, welche nachfolgend näher betrachtet werden sollen. Die linke Spalte des Konfigurationsfensters offenbart eine vertraute Baumdarstellung semantisch gruppierter Parametersammlungen. Unter dem Punkt «Renderer → Twilight» gelangt der Nutzer zu den eigentlichen Raytracing-Parametern der verschiedenen installierten Rendering-Verfahren.
Im oberen Teil der rechten Spalte des Konfigurationsfensters findet der Nutzer 4 verschiedene Gruppierungsboxen, die jeweils mit Eingabetextfeldern für die speziellen Parameter der 4 implementierten Verfahren versehen sind.
Die genaue Bedeutung der Parameter für BPT und MLT sowie eine Beschreibung des Wertespektrums findet der Leser in den Abschnitten 4.2 bzw. 5.2. Näheres zu den Einstellungen für Radiosity und Photon-Mapping kann man der Arbeit Ralf Kopschs [Kop08] entnehmen.

[2] durch Hagen Steidelmüller; Schwerpunkt dieser Arbeit

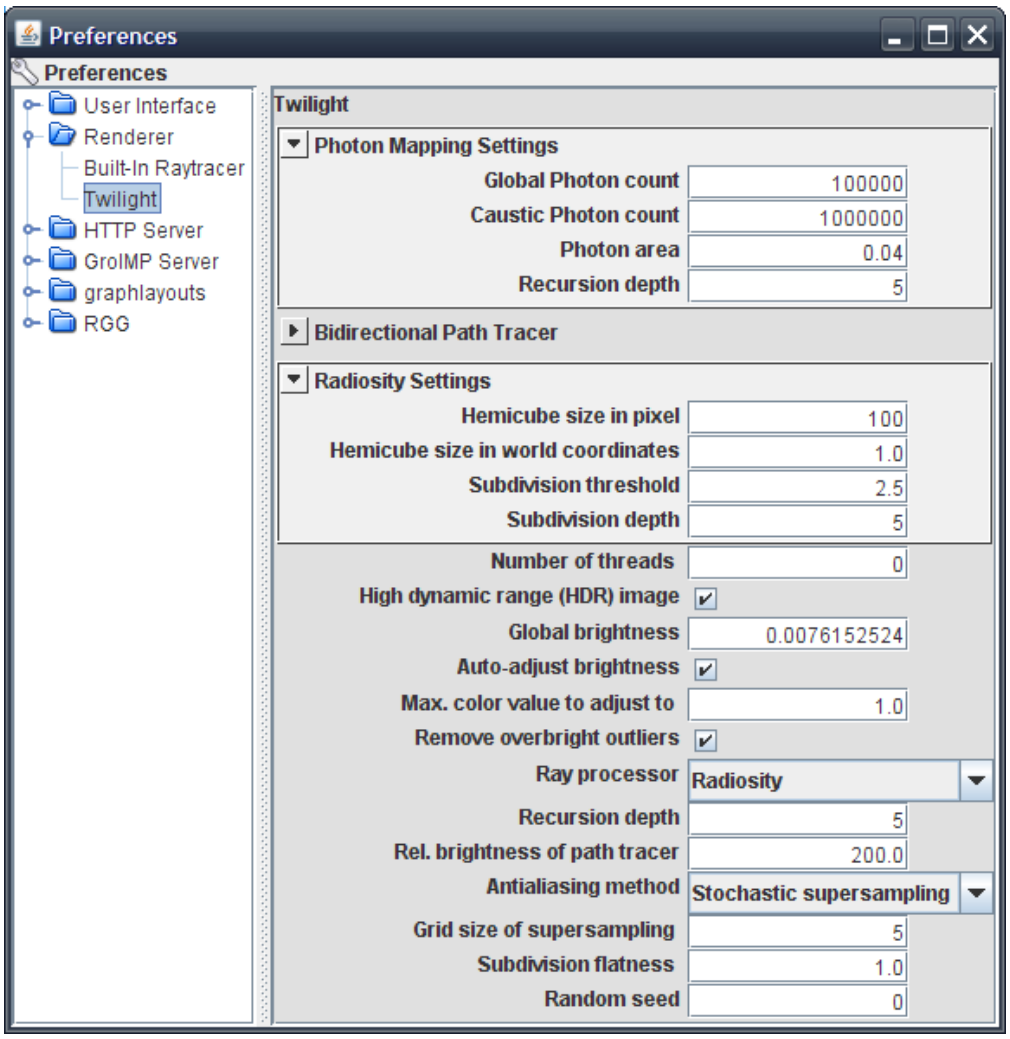


Abbildung 2.1.: Die GroIMP Raytracing-Optionen

Kapitel 3

Path Tracing

Schwerpunkt dieses Kapitels bildet das PT. Dem Leser sollen die wichtigsten Grundlagen dieses speziellen Raytracing-Verfahrens vermittelt werden, da es der Ausgangspunkt der in den späteren Kapiteln diskutierten BPT- und MLT-Verfahren ist.
Kajiya beschrieb dieses Verfahren bereits 1986 in seinem Wissenschaftsbeitrag *The rendering equation*[Kaj86]. Auf die von ihm erarbeiteten Grundlagen, geht der Abschnitt 3.1 genauer ein.
Der Abschnitt 3.2 beschreibt den prinzipiellen Ablauf des PT-Verfahrens.
Im letzten Abschnitt 3.4 werden die für die Steuerung des PT-Renderers verantwortlichen Parameter in GroIMP vorgestellt.

3.1. Die PT-Grundlagen

Bevor anschließend der Algorithmus genauer betrachtet wird, sollen in diesem Abschnitt nochmal kurz auf die Rendergleichung Kajiyas und die verwendete Monte-Carlo (MC)-Simulation eingegangen werden.

Die Rendergleichung

In seiner gleichnamigen Veröffentlichung [Kaj86] konnte Kajiya beweisen, dass alle bisherigen Rendering-Verfahren auf Variationen einer einzigen Energieflussgleichung zurück-

zuführen sind.

$$L(x, x') = g(x, x') \cdot \left(L_e(x, x') + \int_S b(x, x', x'') L(x', x'') dx'' \right) \tag{3.1}$$

ist eine heutzutage gebräuchliche Darstellung dieser Gleichung. Damit stellt Kajiya erstmals die bis dahin gebräuchlichen Rendering-Verfahren auf eine solide mathematische Basis. Die einzelnen Therme der Gleichung haben folgende Entsprechung:

- $L(x, x')$: Energie- oder Lichtfluss vom Szenen-Vertex x' zum Vertex x
- $g(x, x')$: Geometrietherm zwischen x und x'
- $L_e(x, x')$: Emissionstherm, der angibt wie viel Licht von x' nach x abgestrahlt wird, mit der Voraussetzung, dass x' eine Lichtquelle ist
- $b(x, x', x'')$: Streuungstherm, der angibt, wie viel Anteil des Lichts, das x' von x" herkommend erreicht, in Richtung x reflektiert wird.
- S : die Menge aller in der Szene enthaltenen Flächen

Da das Integrieren über alle Flächen der Szene nur wenig perfomant ist, wich Kajiya auf die Monte-Carlo-Simulation aus.

Die Monte-Carlo-Simulation

Laut [BSMM01] versteht man unter einer Simulation die Untersuchung eines Systems mit Hilfe eines (mathematisch meist weniger komplexen) Ersatzsystems. Spielen dabei Zufallsgrößen eine Rolle, wird von einer MC- oder zufallsbedingter Simulation gesprochen, in Anlehnung an die Spielbanken des monegassischen Stadtteils Monte Carlo.
Der Phasenraum der Problemdomäne wird dabei mit einem wahrscheinlichkeitsgewichteten Pfad, einer so genannten Markow-Kette, durchlaufen. Die Wichtungen der Pfadabschnitte entsprechen dabei den angenommenen Häufigkeiten innerhalb des Phasenraums (Importance Sampling).
Auf die Raytracing-Domäne übertragen bedeutet dies, dass nicht die Menge aller möglichen Pfade zu allen Oberflächen berechnet werden muss, sondern dass es genügt, einen einzigen Pfad, der dem statistischen Mittel entspricht, zu verfolgen.
Eine Reduktion des Pfadphasenraums hat jedoch eine Verfälschung des Resultatbildes zur Folge, da der zufällig generierte Strahl sich dem Integral der Rendergleichung nur nähert, diese aber nicht vollständig erfüllt. Diese Verfälschung äußert sich in Bildrauschen.

Der Effekt kann jedoch durch eine genügend hohe Anzahl an Zufallsstrahlen reduziert werden.

3.2. Der PT-Algorithmus

Abbildung 3.1 gibt einen schematischen Überblick des PT-Verfahrens.

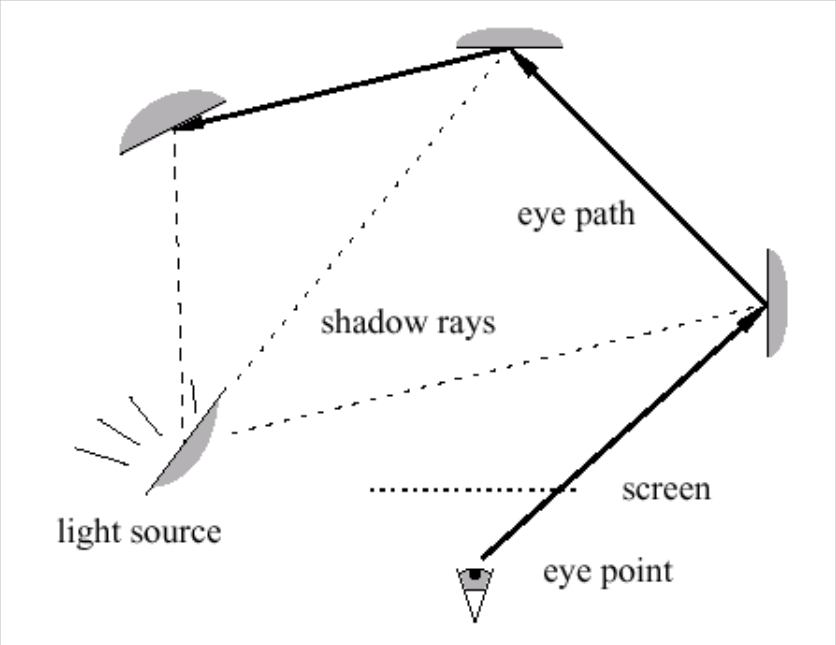


Abbildung 3.1.: Schematische Abbildung des Path-Tracing-Verfahrens nach [LW93]

Der Pseudocode 3.1 zeigt den prinzipiellen Ablauf des PT-Verfahrens:

```
methode SUPER_SAMPLING(Zeichenfläche c, Anzahl a){
    für alle y mit (y < Höhe von c) tue{
        für alle x mit (x < Breite von c) tue{
            für alle (i < a) tue{
                normalverteile Ausgangspunkt(x',y') innerhalb Pixel(x,y);
                Farbe f = PATH_TRACING(x',y');
                färbe Pixel(x,y) mit f;
            }
        }
```

```
10      }
11  }
12
13  [Farbe] funktion PATH_TRACING(x,y){
14      erzeuge Initialstrahl e ausgehend von der Kamera durch den Pixel(x,y);
15      Farbe f = VERFOLGE_STRAHL(e);
16      gib f zurück;
17  }
18
19  [Farbe] funktion VERFOLGE_STRAHL(Strahl e){
20      berechne nächsten Schnittpunkt in Vertex v des Strahls e in der Szene;
21
22      wenn ein v existiert{
23          wenn in v eine Lichtquelle liegt{
24              berechne Lichtquellenbeitrag b in Vertex v;
25          }
26          berechne Direktbeleuchtung d von allen Lichtquellen zu v;
27          ermittle Farbe f aus b und d
28
29          wenn maximale Pfadtiefe noch nicht errreicht{
30              berechne nächsten Strahl e' ausgehend von v;
31              Farbe f' = VERFOLGE_STRAHL(e');
32              füge f' zu f hinzu;
33          }
34          gib f zurück;
35      }
36      gib Leerfarbe zurück;
37  }
```

Listing 3.1: Pseudocode für das PT

Die einzelnen Methoden und der Zusammenhang zur Rendergleichung wird in den nachfolgenden Unterabschnitten beschrieben.

Die Methode SUPER_SAMPLING()

Sie ist der Ausgangspunkt des Algorithmus. Kernelemente sind die 3 Schleifen, in denen sie

1. über die gesamte Höhe der Zeichenfläche (gemessen in Pixel),
2. über die gesamte Breite der Zeichenfläche (gemessen in Pixel),

3. und innerhalb des durch die beiden äußeren Schleifen festgelegten Pixels mit einer vorher festgelegten Durchlaufzahl

die Kreuzungspunkte der späteren Strahlen mit der Zeichenfläche auf dieser verteilt. Eine hohe Durchlaufzahl in der 3. Schleife reduziert das Bildrauschen und sorgt für Antialiasing (Aa). Die entsprechenden GroIMP-Parameter sind im Abschnitt 3.3 zu finden. Der sich aus der Strahlverfolgung ergebene Farbwert $L(x, x')$ wird dann im Bild eingetragen. Die Methode implementiert damit eine Art Scanline-Verfahren, in der die Bildfläche zeilenweise abgetastet wird und die Farbwerte der Strahlverfolgung synchron zur Abtastung in das Zielbild übertragen werden.

Die Funktion PATH_TRACING ()

Diese Funktion hat lediglich die Aufgabe, den Initialstrahl zu erzeugen und durch Zeichenfläche an der vorgegebenen Stelle zu schicken (Zeile 14). Die weitere Strahlerzeugung und -verfolgung durch die Szene übernimmt die Funktion `VERFOLGE_STRAHL()` (Zeile 15). Da diese Strahlen von der Kamera bzw. vom Auge ausgehen, werden sie nachfolgend vereinfacht auch Augenstrahlen genannt. Der Weg, den sie durch die Szene einschlagen, wird dementsprechend als Augenpfad bezeichnet.

Die Funktion VERFOLGE_STRAHL()

Die Aufgaben dieser Funktion sind vielfältig. Zuerst wird der nächste Schnittpunkt (Vertex) des übergebenen Strahls in der Szene ermittelt (Zeile 20).
Von den Eigenschaften des Vertex hängt der weitere Verlauf ab. Ist er nicht existent (Zeile 22), hat der Strahl offensichtlich die Szene verlassen und der Algorithmus bricht ab (Zeile 36).
Handelt es sich bei dem getroffenen Objekt um eine Lichtquelle (Zeile 23), wird der Emissionstherm $L_e(x, x')$ berechnet.
In Zeile 26 wird mit Hilfe sogenannter Schattenstrahlen die Beiträge aus der Direktbeleuchtung durch alle Lichtquellen ermittelt. Dazu wird der jeweilige Geometrietherm $g(x, x')$ berechnet und ausgewertet.
Wenn die Maximallänge des Pfades bzw. die Maximaltiefe der Rekursion noch nicht erreicht ist (Zeile 29), wird der nächste im vom aktuellen Vertex ausgehende Strahl ermittelt. Hierbei wird der Streuungstherm $b(x, x', x'')$ mit einem Wert belegt, anschließend

erneut die Funktion `VERFOLGE_STRAHL()` aufgerufen und der neue Strahl übergeben. Dieser rekursive Abstieg entspricht dem Therm $L(x', x'')$ der Rendergleichung.

Die Implementierung des PT-Algorithmus unterlag vollständig der Verantwortung von Dr. Ole Kniemeyer und Michael Tauer. Sie wird daher nicht näher betrachtet.

3.3. Die PT-Parameter in GroIMP

In Abbildung 3.2 erkennt man einen Ausschnitt des GroIMP-Konfigurationsfensters.

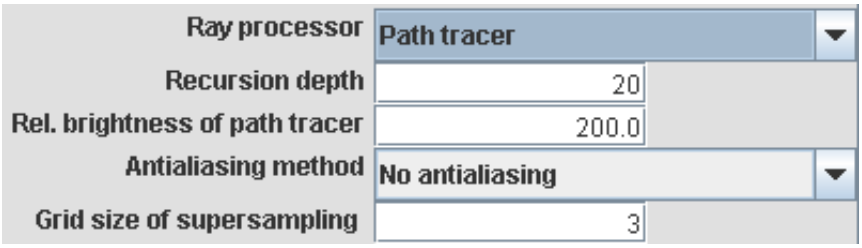


Abbildung 3.2.: Konfiguration des Path-Tracers

In der Drop-Down-Liste „Ray Processor“ lassen sich die verschiedenen Rendering-Verfahren anwählen (im Bild beispielhaft das PT)
Im Eingabefeld „Recursion depth“ wird die maximal Rekursionstiefe angegeben wie in Abschnitt 3.2 erwähnt.
In der Dorp-Down-Liste „Antialiasing method“ kann der Nutzer auswählen, ob das Bild mittels Super-Sampling geglättet wird, oder ob kein Aa vollzogen wird (siehe Abschnitt 3.2).
Über das Feld „Grid size of super sampling“ lässt sich die Anzahl der Sampling-Durchläufe steuern. Dabei muss beachtet werden, dass hierfür immer das Quadrat des Eingabewertes verwendet wird. Dies entspricht der Anzahl der Durchläufe der dritten Schleife (vgl. Pseudocode 3.1 Zeile 4).

3.4. Zusammenfassung

Dieses Kapitel hatte die Aufgabe, den Leser die Grundlagen des PT näher zubringen und ihn auf die kommenden Beschreibungen von BPT und MLT vorzubereiten.

Mit PT gelang es Kajiya das erste Mal, eine vollständige globale Simulation für alle denkbaren Beleuchtungsszenarien durchzuführen. Dieser Vielseitigkeit verdankt das PT seine herausragende Stellung unter den Rendering-Verfahren.
Jedoch wird diese prinzipielle Genauigkeit in der Praxis stark von den zur Verfügung stehenden Ressourcen an Zeit und Rechenleistung limitiert. Denn umso komplexer die Beleuchtungsverhältnisse werden (viel Kaustik, viel indirektes Licht), umso höher ist der Sampling-Aufwand, d.h. die Menge an Augenstrahlen, die benötigt wird, um das Bildrauschen zu kompensieren.
Modernere Verfahren, die teilweise auf PT aufsetzen, haben sich dahingehend als performanter erwiesen. Ein mögliche Verbesserung ist die Kombination des PT mit Photon-Mapping, auf das Ralf Kopsch in seinem Teil der Arbeit [Kop08] näher eingeht. Im Nachfolgenden sollen zwei weitere Vertreter dieser neuen Ansätze, BPT und MLT, vorgestellt werden.

Kapitel 4

Bidirectional Path Tracing

Dieses Kapitel beschäftigt sich mit dem BPT-Verfahren. Es wurde fast zeitgleich und unabhängig von einander sowohl durch Lafortune & Willems [LW93] als auch Guibas & Veach [LW94] entwickelt. Im Folgenden wird jedoch vorwiegend auf Veachs Ansatz eingegangen, da dessen Dissertation [Vea97] die Grundlage dieser Arbeit bildet.
Da BPT eine Erweiterung von PT (siehe Kapitel 3) ist, wird es ebenfalls in die Gruppe der MC-Verfahren eingeordnet. Motivation für dieses Verfahren ist nach [Vea97] die ungenügende Leistung des traditionellen PT in Szenen mit hohem indirekten Beleuchtungsanteil. BPT berücksichtigt hingegen auch reflektiertes Licht, wodurch Szenen realistischer ausgeleuchtet wirken. Verallgemeinernd kann man sagen, dass beim BPT zusätzlich zu den „Augenstrahlen“ des PT-Verfahrens auch Strahlen ausgehend von den Lichtquellen durch die Szene verfolgt werden und ihr Anteil an der Beleuchtung berücksichtigt wird.
Abschnitt 4.1 geht näher auf die algorithmischen Grundlagen des Verfahrens ein.
In Abschnitt 4.2 wird die Implementierung im GroIMP-Projekt erläutert. Im Anschluss daran wird in Abschnitt 4.3 ein Vergleich des implementierten BPT-Verfahrens mit Photon Mapping, PT und den entsprechenden Referenzimplementierungen Veachs vollzogen.
Abschnitt 4.4 stellt eine weitere Verbesserungsmöglichkeit vor.
Es folgt schließlich eine Zusammenfassung des Kapitel in Abschnitt 4.5.

4.1. Der BPT-Algorithmus

4.1.1. Übersicht

Ausgangspunkt des BPT ist der PT-Algorithmus mit direkter Beleuchtung (vgl. Kapitel 3). Dieser wird dahingehend erweitert, dass zusätzlich zu dem aus der Kamera austretenden Strahl auch Strahlen aus jeder Lichtquelle durch die Szene verfolgt werden. Diese Lichtstrahlen verhalten sich prinzipiell nicht anders als die Augenstrahlen, d.h. sie können von den Oberflächen absorbiert, gebrochen oder reflektiert werden. Abbildung 4.1 gibt einen schematischen Überblick über den BPT-Algorithmus.

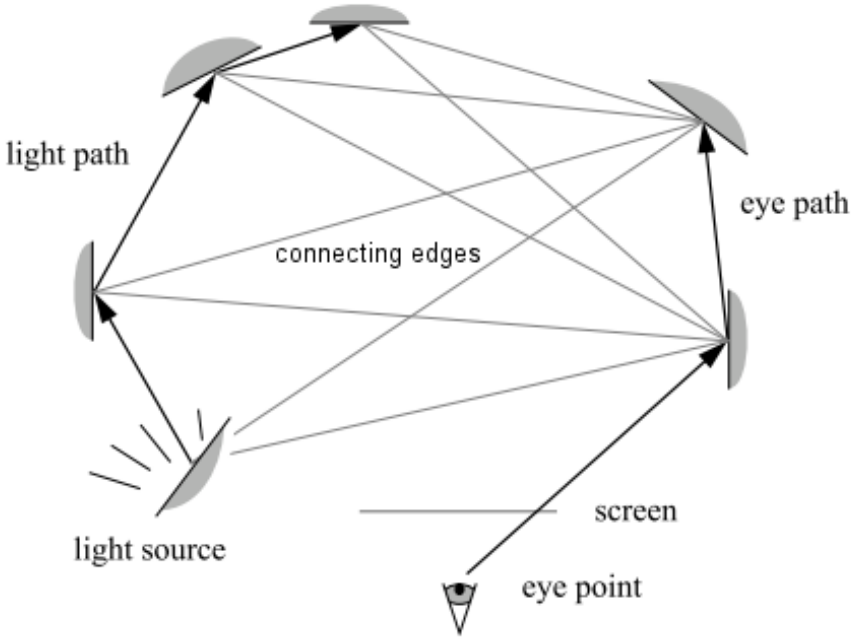


Abbildung 4.1.: Schematische Darstellung des BPT mit neuerzeugten Verbindungskanten nach [LW93]

Anders als beim PT, bei dem der Farbbeitrag des Augenpfads zum Pixel (x,y) synchron zur Strahlverfolgung durch die Szene ermittelt werden konnte, kann dies beim BPT nur nacheinander geschehen. So zergliedert sich der Algorithmus in zwei Teile,

1. der reinen Strahlverfolgung von Kamera und Lichtquelle aus und

2. der Pfad-Rekombination zur Ermittlung der gewichteten Farbbeiträge,

die in den folgenden Abschnitten näher erläutert werden.

4.1.2. Pfadverfolgung

Im nachfolgenden Pseudocode-Ausschnitt 4.1 zeigt die nun auf das BPT-Verfahren angepasste Funktion `VERFOLGE_STRAHL(Strahl e)` (vgl. mit Pseudocode 3.1).

```
1  funktion VERFOLGE_STRAHL(Pfad p, Strahl e){
2      berechne nächsten Schnittpunkt in Vertex v des Strahls e in der Szene;
3      wenn ein v existiert{
4          berechne benötigte Attribute a des aktuellen Pfades p bis zu v;
5          speichere e,v und a in p;
6
7          wenn maximale Pfadtiefe noch nicht errreicht{
8              berechne nächsten Strahl e' ausgehend von v;
9
10             VERFOLGE_STRAHL(p, e');
11         }
12
13     }
14 }
```

Listing 4.1: Pseudocode für BPT-Strahlverfolgung

Aufgrund der Trennung von Strahlverfolgung und Farbberechnung müssen nun alle notwendigen Ergebnisse wie bspw. der Geometrietherm (vgl. mit Kapitel 3), die während der Verfolgung auftreten, im Pfad zwischengespeichert werden.
Dementsprechend konnten auch die Zeilen für Emissionstherm und Direktbeleuchtung aus der Funktion entfernt werden.
Prinzipiell ist dabei das Verhalten von Licht- und Augenstrahlen gleich. Einzig das geänderte Vorzeichen des Richtungsvektors ist bei der Berechnung des Streutherms $b(x, x', x'')$ zu beachten.

4.1.3. Pfadrekombination

Der Pseudocode-Ausschnitt 4.2 zeigt die ebenfalls abgeänderte Funktion `PATH_TRACING(x,y)` (vgl. mit Pseudocode 3.1).

```
1  [Farbe] funktion BIDI_PATH_TRACING(x,y){
2
3      erzeuge Augenstrahl e ausgehend von der Kamera durch den Pixel(x,y);
4      lege leeren Pfad E für die Augenstrahlen an
5      verfolge_strahl(E,e);
6
7      für alle Lichtquellen q tue{
8          erzeuge Lichtstrahl l ausgehend von q;
9          lege leeren Pfad L für die Lichtstrahlen an
10         VERFOLGE_STRAHL(L,l);
11
12         für alle Längen s von L tue{
13             für alle Längen t von E tue{
14                 kombiniere L der Länge s mit E der Länge t zu
15                     Pfad C der Länge s+t;
16                 berechne kombinierten Lichtbeitrag b von C;
17                 berechne Pfadwichtung w von C;
18                 füge b*w zu Farbe f hinzu;
19             }
20         }
21     }
22     gib f zurück;
23 }
```

Listing 4.2: Pseudocode BPT-Pfadrekombination

Die Änderungen im Vergleich zum PT-Verfahren sind hier größer.
Zur Erzeugung und Verfolgung des Augenpfads (Zeile 3-5) gesellt sich nun noch eine Berechnungsschleife über alle Lichtquellen (Zeile 7). Für jede Lichtquelle wird nun ebenfalls ein Strahl erzeugt und durch die Szene verfolgt (Zeile 8-10), wie bereits in Abschnitt 4.1.2 beschrieben.
In den Zeilen 12 bis 20 werden nun alle Längenkombinationen des Augenpfades und des aktuellen Lichtpfades miteinander zu einem kombinierten, bidirektionalen Licht-Augen-Pfad verknüpft. Es wird weiterhin der Licht- oder auch Energiebeitrag dieses Pfades mit Hilfe der gespeicherten Pfadattribute ermittelt, entsprechend gewichtet und auf den Farbwert des Pixels(x,y) aufaddiert.
Abbildung 4.2 zeigt anhand eines Beispielpfades der Länge 2 die auftretenden Pfadkombinationsmöglichkeiten.

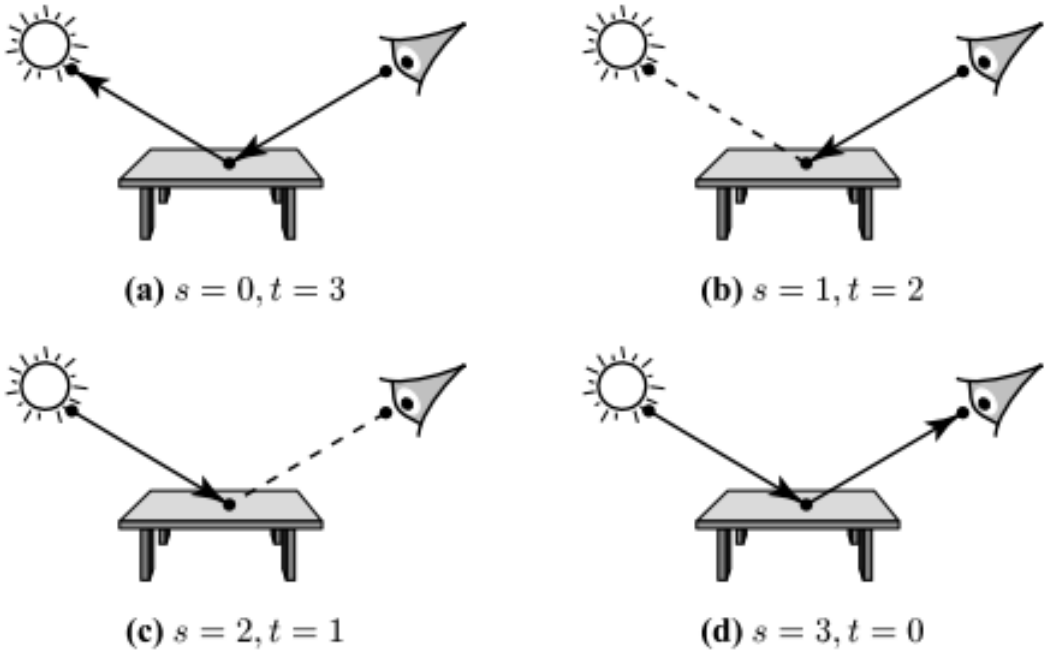


Abbildung 4.2.: Die 4 möglichen Pfadkombinationen für die Länge k=2 [Vea97, S. 299]. Sie können auch als Tracing-Spezialfälle betrachtet werden: **(a)** Path-Tracing ohne spezielle Handhabung von Lichtquellen; **(b)** Path-Tracing mit direkter Beleuchtung; **(c)** Photonen aus der Lichtquelle werden verfolgt und ihr Farbbeitrag im Bild gespeichert, immer wenn sie eine sichtbare Fläche treffen **(d)** Photonen aus der Lichtquelle werden verfolgt und ihr Farbbeitrag im Bild gespeichert, aber nur dann, wenn sie die Kamera treffen

4.1.4. Mathematische Grundlagen

Veach formuliert in seiner Dissertation [Vea97, S. 298-306] ein umfangreiches mathematisches Fundament. Dies soll an dieser Stelle nur in Auszügen aus Gründen der besseren Verständlichkeit präsentiert werden.
Ausgangspunkt seiner Überlegungen ist die Gleichung 4.1, die der Algorithmus lösen muss.

$$F = \sum_{s \geq 0} \sum_{t \geq 0} w_{s,t}(\bar{x}_{s,t}) \frac{f_j(\bar{x}_{s,t})}{p_{s,t}(\bar{x}_{s,t})} \tag{4.1}$$

Die einzelnen Therme der Gleichung haben dabei folgende Bedeutung:

- F : der Gesamtenergiebeitrag aus der Kombination des aktuellen Licht- und des Augenpfads
- $\bar{x}_{s,t}$: der rekombinierte Pfad der Länge s+t-1 mit $\bar{x}_{s,t} = y_0 \dots y_{s-1} z_{t-1} \dots z_0$ wobei y_i ein Vertex auf dem Lichtpfad und z_i ein Vertex auf dem Augenpfad ist
- $w_{s,t}(\bar{x}_{s,t})$: Wichtung des rekombinierten Pfads $\bar{x}_{s,t}$
- $f_j(\bar{x}_{s,t})$: der gemeinsame Streuungstherm des rekombinierten Pfads $\bar{x}_{s,t}$
- $p_{s,t}(\bar{x}_{s,t})$: Wahrscheinlichkeitsdichte des rekombinierten Pfads $\bar{x}_{s,t}$

Die Gleichung kann umgeschrieben werden zu

$$F = \sum_{s \geq 0} \sum_{t \geq 0} C_{s,t} \tag{4.2}$$

wobei $C_{s,t}$ dem gewichtet Energiebeitrag des Pfads $\bar{x}_{s,t}$ entspricht. Dies kann wiederum umformuliert werden zu

$$C_{s,t} = w_{s,t}(\bar{x}_{s,t}) C^*_{s,t} \tag{4.3}$$

mit $C^*_{s,t}$ als ungewichteter Energiebeitrag. Dieser lässt sich wie folgt ausdrücken:

$$C^*_{s,t} = \alpha^L_s c_{s,t} \alpha^E_t \tag{4.4}$$

Die Faktoren α^L_s und α^E_t hängen dabei ausschließlich vom Licht- bzw. Augenpfad ab und ergeben sich aus dem Quotienten von Streuungstherm und Wahrscheinlichkeitsdichte des jeweiligen Pfadabschnitts. $c_{s,t}$ bezieht sich auf die erzeugte Verbindungskante zwischen dem Licht- und Augenpfadabschnitt.
$c_{s,t}$ ergibt sich im allgemeinen Fall aus

$$c_{s,t} = f_s(y_{s-2} \to y_{s-1} \to z_{t-1}) G(y_{s-1} \leftrightarrow z_{t-1}) f_s(y_{s-1} \to z_{t-1} \to z_{t-2}) \tag{4.5}$$

mit $f_s(\dots)$ als den Streuungsthermen auf dem jeweiligen Pfadende und $G(y_{s-1} \leftrightarrow z_{t-1})$ als dem Geometrietherm dazwischen.
Bleibt nur noch der Gewichtungsfaktor $w_{s,t}$, der sich allgemein ausdrücken lässt mit

$$w_{s,t} = \frac{p^\beta_{s,t}}{\sum_i p^\beta_i} \tag{4.6}$$

Hierbei stellt p_s die Wahrscheinlichkeit da, mit der der gegebene Pfad erzeugt wurde, während $p_0 \dots p_{s-1}$ und $p_{s+1} \dots p_{s+t}$ den Wahrscheinlichkeiten entsprechen, wie der Pfad

noch hätte erzeugt werden können. Bei der Berechnung dieser Wahrscheinlichkeiten muss erneut auf die gespeicherten Geometrietherm- und Wahrscheinlichkeitsdichte-Werte zurückgegriffen werden.
β ist der heuristische Exponent, mit dessen Hilfe der Nutzer den Gewichtungsfaktor varieren kann. Bspw. ergibt sich für $\beta = 0$ ein $w_{s,t} = \frac{1}{s+t}$, während Veach $\beta = 2$ als „Power Heuristic“ bezeichnet. Nähere Informationen zu den Gleichungen und ausführliche Herleitungen können in Veachs Dissertation [Vea97, S. 298-306] gefunden werden.

4.2. Die BPT-Implementierung in GroIMP

Für die Umsetzung des BPT-Verfahrens sind nur relativ wenig Klassen verantwortlich, wie in Abbildung 4.3zu erkennen.

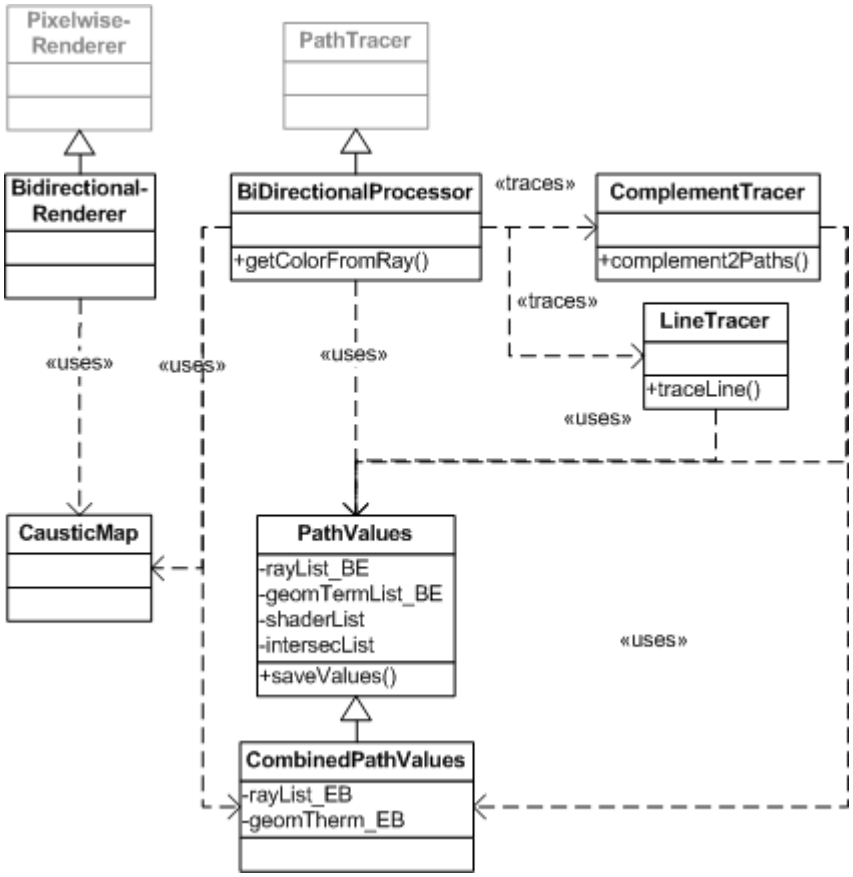


Abbildung 4.3.: Klassenarchitektur des BPT in GroIMP

In den nachfolgenden Abschnitten werden die einzelnen Klassen kurz besprochen.

Die Klasse BidirectionalProcessor

Die Klassen `BiDirectionalProcessor` befindet sich im Paketen `de.grogra.ray2.tracing` des RayTracing-Plugins und leitet sich direkt von der Klasse `PathTracer` ab, die das PT-Verfahren realisiert und von Dr. Ole Kniemeyer geschrieben wurde.
Die Instanzen des `BiDirectionalProcessor` übernehmen zur Programmlaufzeit die Abarbeitung des BPT-Algorithmusses. Der Aufbau der Klasse ist im Großen und Ganzen dem Pseudocode-Fragment 4.2 nachempfunden. Die Klasse überschreibt die

Methode `getColorFromRay()` des `PathTracer` dahingehend, dass nun zusätzlich auch Strahlen aus den Lichtquellen generiert und durch die Szene verfolgt werden. Die Verfolgung der Strahlen übernimmt eine Instanz der Klasse `LineTracer`, die in Abschnitt 4.2 beschrieben wird. Diese Aufgliederung wurde unternommen, um das Seperation of Concerns (SoC)-Prinzip zu gewährleisten und so die Klasse übersichtlich zu halten.
Die Hauptaufgabe der Klasse besteht in der Rekombination der über den `LineTracer` erhaltenen Pfade für Lichtquelle und Auge und der Berechnung des Wichtungsfaktors.

Die Klasse LineTracer

Die Klasse `LineTracer` liegt im Paket `de.grogra.ray2.tracing.modular`. Ihre Aufgabe besteht im Verfolgen des in der Funktion `traceLine()` übergebenen Strahls durch die Szene. Alle für spätere Berechnungen notwendigen Objekte, die während des Verfolgungsprozessen auftreten, werden in einer Instanz der Klasse `PathValues` abgespeichert. Diese Instanz steht nach Verfolgungsende dem Aufrufer (eine Instanz der Klasse `BiDirectionalProcessor`) zur Verfügung.

Die Klasse PathValues

Instanzen der Klasse `PathValues` dienen als Container-Objekte für die Zwischenergebnisse der Pfadverfolgung. Abgespeichert werden derzeit Instanzen der Klassen:

- `Environment`, da diese für die Shading-Funktionen benötigt werden,
- `Intersection`, um Sichtbarkeitstests durchführen zu können,
- `Scattering`, um auf Ihnen die Shading-Funktionen aufrufen zu können,
- `Line`, entsprechen den Strahlen zwischen den Vertexen des Pfades und speichern ihrerseits das Lichtspektrum, mit dem der Strahl ausgesandt wurde, und seine Ursprungs- als auch Richtungswahrscheinlichkeitsdichte
- `Lights`, Lichtquellen, die auf dem Weg getroffen werden

Des Weiteren beinhaltet die Klasse auch ein Boolean-Feld um eventuelle spekulare Reflexionen zu sichern. Dazu kommen 2 Integer, die die Pfadlänge und die Erzeuger-ID aufnehmen. Die Klasse selbst liegt im Paket `de.grogra.ray2.tracing.modular`.

Die Klasse CombinedPathValues

Im Verlauf der Arbeit hat es sich herausgestellt, dass eine eigene Datenstruktur für die kombinierten Licht- und Augenpfade von Vorteil ist, da sich in ihr bspw. einfacher indizieren lässt. Daher wurde die Klasse `CombinedPathValues` kreiert, ebenfalls zu finden im Paket `de.grogra.ray2.tracing.modular`. Sie leitet sich von `PathValues` ab und erweitert diese um 3 weitere Listen für `Line`, `Spectrum` und den Float-Werten der Geometrietherme. Ziel dieser Erweiterung war es, eine Datenstruktur für Pfade zu erhalten, die möglichst einfach von Lichtquelle in Richtung Auge („Beginn to End“) bzw. von Auge in Richtung Lichtquelle („End to Beginn“) durchlaufen und indiziert werden kann.

Die Klasse ComplementTracer

Bei der Studie der Berechnungsvorschrift für den Gewichtungsfaktor $w_{s,t}$ [Vea97, S. 306] wurde offensichtlich, dass es notwendig ist, Werte für die Wahrscheinlichkeitsdichten und Geometrietherme in beide Richtungen über den kompletten Pfad zu besitzen.
Diese können jedoch nicht bei der initialen Strahlverfolgung durch den `LineTracer` ermittelt werden. Erst danach stehen die `PathValues`-Objekte von Auge- und Lichtpfad zur Rekombination zur Verfügung.
Daher übernimmt eine Instanz der Klasse `ComplementTracer` die nachträgliche Berechnung der oben genannten Größen und verschmilzt die Pfade zu einem kombinierten Pfad (eine Instanz von `CombinedPathValues`) mit den jeweils gewünschten Anteilen.

Einer der Vorteile der zweiseitigen Strahlverfolgung des BPT besteht in der Möglichkeit Kaustikeffekte darzustellen. Pfade, die Kaustik hervorbringen, haben die Form $L(L|D|S)^*S^*DE$, wobei L für eine Lichtquelle, E für die Linse und D für diffuse bzw. S für spekulare Oberflächen stehen. Im BPT-Algorithmus tragen dazu vor allem die Pfade bei, bei denen $s > 1$ und $t = 1$ sind. Ausgehend vom letzten Vertex des Lichtpfads wird eine Kante zum einzigen Vertex des Augenpfades auf der Kameraoberfläche gezogen. Diese Kante durchkreuzt dabei die Bildschirmfläche in einem Pixel (x', y'), der in den meisten Fällen ungleich dem ursprünglichen Pixel ist, durch den der Augenpfad eigentlich führt. In der vorgegebenen Implementierung der Klasse `PixelwiseRenderer` werden jedoch keine Farbwerte für Pixel, die asynchron zum Verlauf des Scanline-Prozesses auftreten, abgespeichert. Sie muss deshalb für den BPT-Spezialfall erweitert werden.

Die Klasse CausticMap

Instanzen dieser Klasse beinhalten im wesentlichen ein zweidimensionales Array von der Größe des zu rendernden Bildes, in dem die Farbwerte des Spezialfalls $s > 1$ und $t = 1$ zwischengespeichert werden können. Jede Instanz der Klassen `BiDirectionalProcessor` bekommt dazu eine eigene lokale CausticMap zugestellt, die dann während der Ergebnissynchronisierung vom Renderer mit ausgelesen wird.

Die Klasse BidirectionalRenderer

Die Klasse `BiDirectionalRenderer` erbt von `PixelwiseRenderer` und erweitert diese hauptsächlich um Funktionalitäten im Zusammenhang mit der Speicherung von Kaustikinformationen. Dazu wird das Renderer-Objekt ebenfalls mit einer CausticMap ausgestattet, in die die Zwischenergebnisse aus den BPT-CausticMaps während der Synchronisierung zwischengespeichert werden. Nach Abschluss des gesamten Rendervorgangs werden die Farbwerte aus der globalen CausticMap in das Bild eingetragen.

Die BPT-Parameter in GroIMP

Die Steuerung der BPT-Komponente in GroIMP geschieht über 3 Parameter, die im Renderer-Konfigurationsfenster gefunden werden können, wie in Abschnitt 2.2 bereits beschrieben.
In Abbildung 4.4 ist die zum BPT-Renderer gehörende Paramtergruppierungsbox zu sehen.

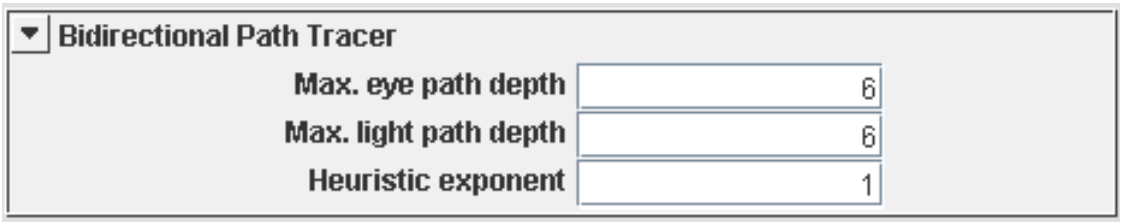


Abbildung 4.4.: Gruppierungsbox der BPT-Parameter

Über die Parameter «Max. eye path depth» und «Max. light path depth» lassen sich dabei die maximale Anzahl an Vertexen auf dem Augen- bzw. dem Lichtpfad festlegen.

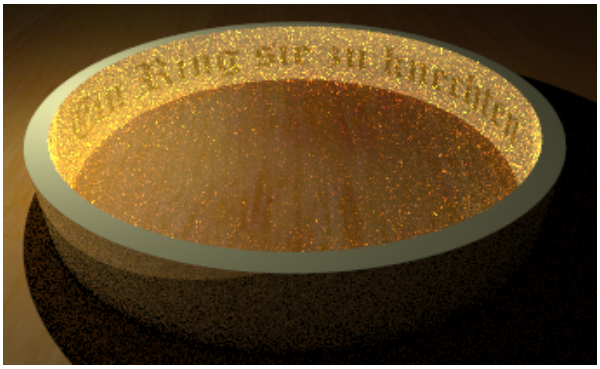
Durch eine Lichtpfadlänge von 1 kann die Arbeitsweise eines normale PT simuliert werden. Andersrum werden mit einer Augenpfadlänge von 1 ausschließlich Kaustikereignisse dargestellt. Werte kleiner als 1 sind nicht erlaubt.
Der Paramter «Heuristic exponent» steuert die Berechnung des Wichtungsfaktors $w_{s,t}$, wie in Abschnitt 4.1.4 beschrieben.

4.3. Der BPT-Algorithmus im Vergleich

Nachfolgend soll anhand eines Vergleichs von Rendering-Ergebnissen die Qualitäten und Stärken des BPT-Verfahrens herausgestellt werden im Verhältnis zum PT und dem Photonen-Mapping-Verfahren, welches Ralf Kopsch in seinem Teil der Studienarbeit umgesetzt hat [Kop08].
Alle Bilder wurden in etwa gleicher Rechenzeit gerendert. Die genauen Einstellungen entnehme der Leser bitte der Tabelle A.1 im Anhang.

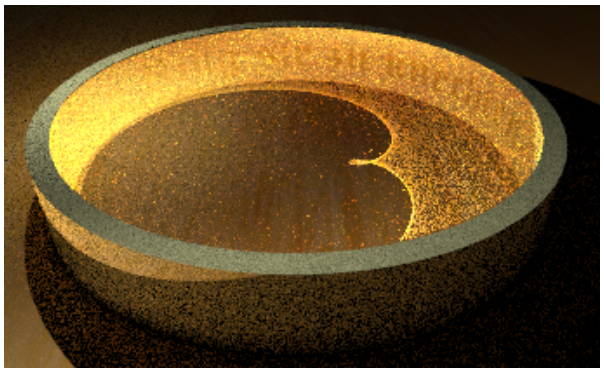
Kaustik

In Abbildung 4.5 wird die bereits aus der Arbeit von Ralf Kopsch bekannte Goldringszene gezeigt, jeweils gerendert mit dem PathTracer (a), Photon Mapping (b) und dem BPT-Raytracer (c).



(a) PathTracer mit 25 Samples pro Pixel

(b) Poton Mapping mit 500000 Kaustikphotonen



(c) BPT mit 16 Samples pro Pixel

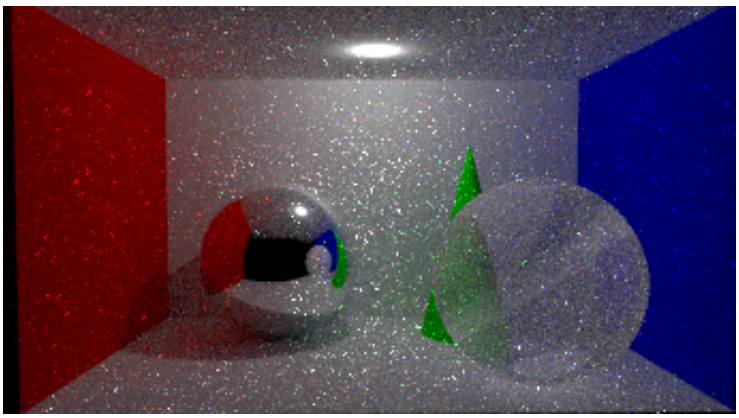
Abbildung 4.5.: Vergleich der Verfahren in der Darstellung von Kaustik

Das Pathtracing-Bild 4.5 (a) ist gekennzeichnet durch relativ starkes Rauschen und der Abwesenheit von Kaustikeffekten. Während jedoch dem Rauschen durch eine erhöhte Sampling-Rate entgegengewirkt werden kann, hilft dies nur marginal bei der Darstellung von Kaustik. Während der Strahlverfolgung müsste dafür die Lichtquelle[1] direkt getroffen werden, was nur selten der Fall ist.
Das Photon Mapping in Bild 4.5 (b) kann hingegen mit Kaustik aufwarten. Auch scheint die Bildqualität entsprechend, was sich in seiner Rauscharmut und der Brillianz der Spiegelungen im Ringinneren äußert. Bei genauer Betrachtung weist das Verfahren jedoch Nachteile hinsichtlich der Szenenausleuchtung auf, zu erkennen daran, dass der Schatten hinter dem Ring vollkommen schwarz ist; die Rückseite des Rings offensichtlich nicht diffus beleuchtet wurde. Die Kaustik selbst ist Algorithmus bedingt unscharf und großflächig.
Bild 4.5 (c) wurde mit dem BPT-Verfahren gerendert. Analog zum PT-Bild kann auch hier eine relativ starke Verrauschung festgestellt werden, was aus der nahen Verwandtschaft der Verfahren herrührt und ebenfalls durch erhöhte Sampling-Raten behoben werden kann. Jedoch weist das Bild eine klar erkennbare Kaustik auf, die anders als beim Photon Mapping präzise und scharf dargestellt wird. Zusätzlich wird auch noch die Ringrückseite diffus beleuchtet.

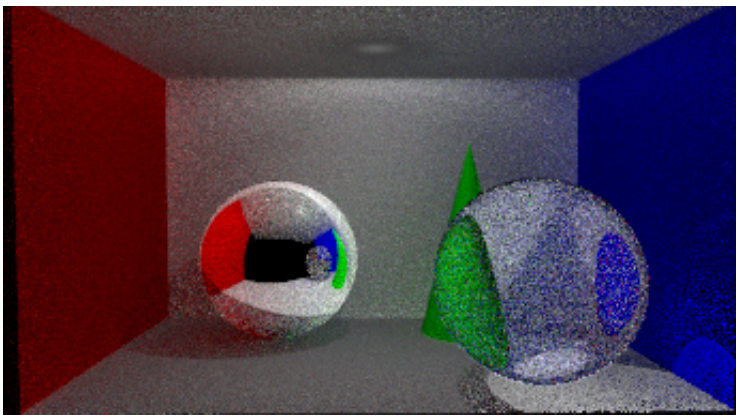
Verschattung

Abbildung 4.6 (a) und (b) zeigen eine Cornellbox mit überwiegend direkter Beleuchtung. Die Szene beinhaltet verschattete Bereiche und spekular transmittierende, spekular reflektierende und diffus reflektierende Objekte. Es ist ein deutlicher Qualitätsunterschied zwischen dem mit dem PT-Verfahren erzeugten Bild 4.6 (a) und dem BPT-Resultat 4.6 (b) zu erkennen.
Letzteres ist bei gleicher Berechnungszeit weniger verrauscht, besitzt Kaustik und einen realistisch weicheren Halbschatten der Spiegelkugel. Die Glaskugel wird zu dem klarer dargestellt und wirft einen Schatten, der im PathTracing-Bild fehlt.

[1] oder eine stark strahlende Fläche in deren unmittelbarer Nähe

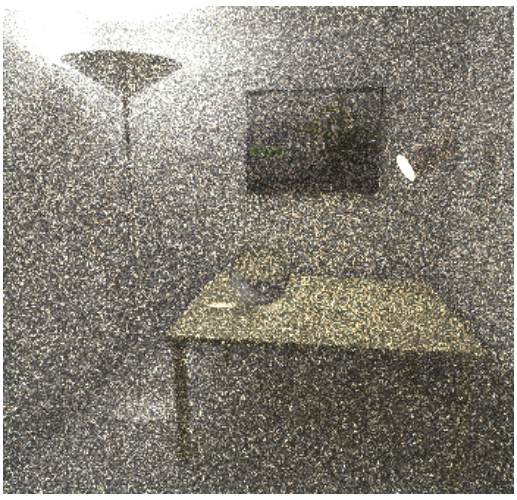


(a) PathTracer mit 25 Samples pro Pixel

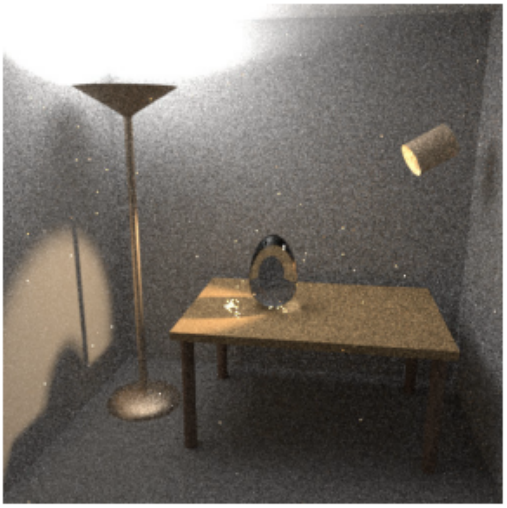


(b) BPT mit 16 Samples pro Pixel

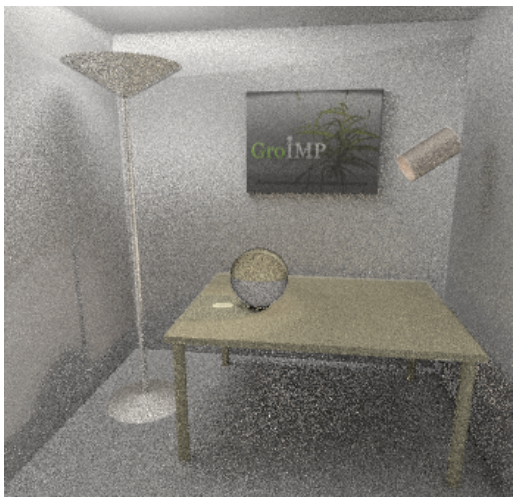
Abbildung 4.6.: Vergleich des PT- und des BPT-Verfahrens in der Darstellung von Schatten



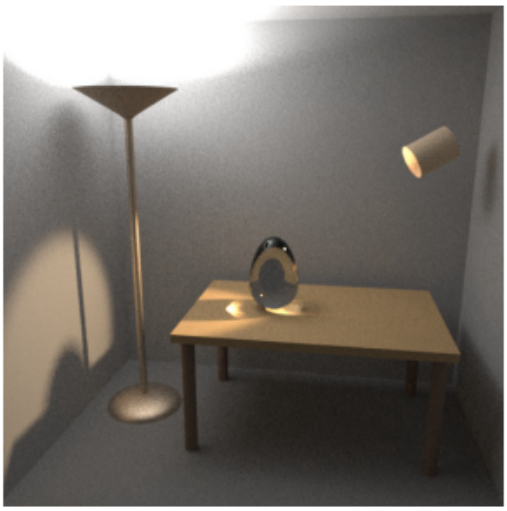
(a) PT aus GroIMP mit 64 Samples pro Pix.



(b) PT nach Veach mit 56 Samples pro Pix.



(c) BPT aus GroIMP 25 Samples pro Pix.



(d) BPT nach Veach 25 Samples pro Pix.

Abbildung 4.7.: Vergleich der GroIMP-Implementierungen des PT- und des BPT-Verfahrens mit der Referenzimplementierung von Eric Veach aus [Vea97][S.323] in der Darstellung einer komplexen Szene mit viel indirektem Licht

Indirekte Beleuchtung

In den Abbildungen 4.7 (a), (b), (c) und (d) wird eine komplexe Szene dargestellt: Die Wände und der Tisch reflektieren diffus, die Stehlampe spekular mit diffusem Anteil, die Glaskugel transmitiert und bricht das Licht. Die Beleuchtung selbst ist zu einem hohen Anteil indirekt und geschieht vorwiegend über die Raumdecke.
Die Abbildungen 4.7 (a) und (c) zeigen dabei eine GroIMP erstellte Szene, die der aus der Arbeit von Veach – Abbildungen 4.7 (b) und (d) – nachempfunden wurde, um eine größtmögliche Vergleichbarkeit zu gewährleisten. Weiterhin wurde versucht die Bilder mit der in etwa gleichen Sampling-Rate zu rendern.
Deutlich sind die Qualitätsunterschiede zwischen den PathTracing- und den BPT-Bildern zu erkennen. Diese sind weniger verrauscht, klarer ausgeleuchtet und weisen scharfe Kaustik auf.
Der Vergleich mit den Referenzbildern aus der Dissertation [Vea97] zeigt jedoch auch einen deutlichen Qualitätsunterschied zwischen den Implementierungen in GroIMP und denen von Veach. Der Grund ist vermutlich darin zu finden, dass Veach einige zusätzliche Optimierungen[2] eingebaut hat, die in GroIMP fehlen, sich aber nachhaltig auf Qualität und Berechnungsperformance auswirken. Generell lässt sich jedoch feststellen, dass das Bildresultat der GroIMP-Implementierung des BPT-Verfahrens in etwa im gleichen Verhältnis zu dem des GroIMP-PathTracer steht wie das in der Referenzimplementierung bei Veach der Fall ist.

4.4. Ausblick

Das BPT-Verfahren besitzt gegenüber PT bei gleichen Einstellungen einen erhöhten Berechnungsaufwand, da es zusätzlich zu der Strahlverfolgung aus der Kamera auch eine aus jeder Lichtquelle vollzieht und diese beiden Pfade über alle Längenmöglichkeiten miteinander kombiniert. Neben der reinen Strahlverfolgung durch die Szene schlägt dabei vor allem die erhöhte Anzahl an Sichtbarkeitstests, die über Schnittpunktberechnungen stattfinden, zu Buche.
Veach beschreibt in seiner Arbeit eine Möglichkeit, diese Sichtbarkeitstests effektiv zu reduzieren [Vea97][S. 317-321].

[2] bspw. Importance Sampling und Otimized Russian Roulett (vgl. [Vea97][S.317-320])

Effizienzsoptimiertes Russisches Roulett

Nach Veach kann der Farbwert eines Pixels als die Summe der gewichteten Beiträge der einzelnen Pfadkombinationen aufgefasst werden:

$$F = C_1 + ... + C_n, \tag{4.7}$$

Dabei ist festzustellen, dass viele Pfade nur einen geringen Beitrag leisten, aber im Verhältnis dazu hohe Kosten durch zusätzliche Sichtbarkeitstests verursachen. Veach schläg nun eine Art Russisches Roulette bei der Auswahl der Pfadbeiträge vor. Die Beiträge haben dann die Form:

$$C_i = \left\{ \begin{array}{l} (1/q_i)v_i t_i \; mit \; Wahrscheinlichkeit \; q_i \\ 0, \; sonst \end{array} \right\} \tag{4.8}$$

v_i ist darin der Sichtbarkeitsfaktor (entweder 0 oder 1), und t_i der aus der Verfolgung ermittelte vorläufige Beitragswert. Die Wahrscheinlichkeit q_i ergibt sich aus

$$q_i = min(1, t_i/\delta), \tag{4.9}$$

wobei δ einen festgesetzten Schwellwert darstellt, der abhängig ist von der durchschnittlichen Varianz des Bildes σ_0 und der durchschnittlichen Anzahl n_0 an erzeugten Strahlen (sowohl für den Licht als auch den Augenpfad) pro Pixel (Samplekosten).

$$\delta = \sqrt{\sigma^2/n_0} \tag{4.10}$$

Pfade, die diesen Schwellwert überschreiten, tragen auf jeden Fall zum Farbwert des Pixels bei, alle anderen weniger bedeutenden werden über das Verfahren zufällig angenommen oder abgelehnt. Veach räumt ein, dass es schwierig ist die Varianz und die Samplekosten im Vornherein zu ermitteln. In seiner Referenzimplemntierung verwendet er dafür den laufenden Durchschnitt der letzten N_0 Tracing-Operationen. Generell stellt das effizienzoptimierte Russische Roulett eher eine heuristische Näherung als ein genaues mathematisches Verfahren dar, mit dessen Hilfe jedoch die Anzahl der Schnittpunktsberechnungen drastisch gesenkt werden kann.

4.5. Zusammenfassung

Das in diesem Kapitel vorgestellte BPT-Verfahren verbessert die Ergebnisse bei der Bildsynthese von Szenen mit schwieriger Beleuchtung (Kaustik, viel indirektes Licht)

deutlich bei ungefähr gleichem Zeitbedarf wie PT.
Anders als beim Photon-Mapping werden dabei jedoch die Gesetzte der geometrischen Optik eingehalten und somit prinzipiell physikalisch korrektere Bilder erzielt. Das macht das BPT zu einem unverzichtbaren Mitglied der Raytracing-Familie.

Kapitel 5

Metropolis Light Transport

Drei Jahre nach der Veröffentlichung des BPT-Verfahrens [LW94] fügte Veach der MC-Tracing-Familie ein neues Mitglied hinzu, das MLT-Verfahren.
Dank seines radikalen neuen Ansatzes, in dem er BPT mit dem Metropolisalgorithmus aus der Stochastik kombinierte, gelang Veach ein nochmals deutlicher Qualitätsgewinn bei der Bildsynthese komplexer Beleuchtungsszenarien im Vergleich zum BPT oder PT.
Das MLT-Verfahren bildet daher das inhaltliche Kernstück des nachfolgenden Kapitels.
Der Abschnitt 5.1 befasst sich mit dem prinzipiellen Ablauf und den Eigenschaften des MLT-Verfahren.
Die Umsetzung des MLT-Algorithmus im GroIMP-Projekt beschreibt der Abschnitt 5.2.
In Abschnitt 5.3 werden die Rendering-Resultate mit denen des BPT-Verfahren anhand zweier Beispielszenen verglichen.
Der Abschnitt 5.4 gibt einen kleinen Ausblick auf Verbesserungsmöglichkeiten und aktuelle Weiterentwicklungen.
In Abschnitt 5.5 werden die Inhalte dieses Kapitel noch einmal rekapituliert.

5.1. Der MLT-Algorithmus

Das MLT-Verfahren dreht sich um die Prämisse, dass ein einmal gefundener energiereicher Pfad von Lichtquelle zu Auge, nicht „weggeworfen“ werden sollte. Und das sich vermutlich in seiner Nähe noch weitere solcher Pfade befinden.

Die nachfolgenden Abschnitte befassen sich damit, was einen guten Pfad charakterisiert und wie er durch Änderungen am Verlauf des Ausgangspfads gefunden werden kann.

5.1.1. Übersicht

Die Herangehensweise des MLT steht denen der anderen MC-Verfahren konträr gegenüber. Während bspw. beim PT der Phasenraum aller Pfade quasi blind mit einer großen Zahl an Strahlen durchlöchert wird und sich die Pixel-Farbwerte als arithmetisches Mittel aus einzelnen Pfadfarbwert ergeben, wird beim MLT der gleiche Phasenraum nur lokal durchsucht: ist einmal ein guter – d.h. ein Energie reicher – Pfad gefunden, kann seine unmittelbare Nachbarschaft, in der sich vermutlich weitere gute Pfade befinden, durch kleine Änderungen am Pfadverlauf exploriert werden. Diese Änderungen umfassen dabei Maßnahmen wie das Verschieben einzelner Vertexe oder das Hinzufügen oder Löschen von Pfadkanten. Dabei werden bevorzugt Änderungen akzeptiert, die einen neuen Pfad mit möglichst großem Energiebeitrag zur Folge haben. Durch geschickte Anwendung der Änderungstechniken, im weiteren Verlauf „Mutationen“ genannt, kann so ebenfalls der gesamte Pfad-Phasenraum durchforstet werden.
Die aus MLT resultierenden Bilder erweisen sich folglich gegenüber vergleichbaren Rendering-Verfahren als sehr rauscharm, da sie mit Energie optimierten Pfaden gebildet wurden.

Der prinzipielle Ablauf des MLT-Algorithmus ist denkbar einfach, Pseudocode 5.1 fasst ihn zusammen.

```
1  [Bild] funktion Metropolis_Light_Transport(){
2
3      finde geeigneten Initialpfad x
4      erzeuge leeres Bild b
5
6      für i = 1 bis N tue{
7
8          bestimme Mutationsstrategie m per Zufallsauswahl;
9          Pfad y = MUTIERE_PFAD(m,x)
10         ermittle Akzeptanzwahrscheinlichkeit a für die Änderung von x -> y
               ;
11         erzeuge normalverteile Zufallszahl z;
12         wenn  z < a {
13             ersetze x durch y;
14         }
15         SPEICHER_BILDPIXEL(b,x);
16     }
```

```
17      gib b zurück;
18  }
```

Listing 5.1: Pseudocode für MLT-Strahlverfolgung

Es beginnt in Zeile 3 mit der Suche nach einem geeigneten bidirektionalen Initialpfad (siehe. Abschnitt 5.1.2) der vom Auge zu einer Lichtquelle der Szene führt. Ist der Pfad gefunden, kann das Bild initialisiert werden, in das später die Farbwerte der Pfade eingetragen werden.
Nun folgt in Zeile 6 eine Schleife über die unbestimmte Länge N. Veach geht in seiner Dissertation [Vea97] nicht näher auf die Größe ein, doch durch einfache Überlegungen kann schnell festgestellt werden, aus welchen Faktoren N besteht:

- die Anzahl der Pixel des Bildes, denn der Initialpfad muss mindestens so oft erfolgreich mutiert werden, dass jeder Pixel einmal durchlaufen wurde, um schwarze Pixel zu vermeiden,
- die Anzahl der vorhandenen Strategien nach denen die Mutation vollzogen wird, denn nicht jede Strategie verschiebt den Pfad über die Zeichenfläche, sowohl
- die vom Benutzer gewünschte durchschnittliche Anzahl an Mutationen pro Pixel.

Innerhalb der Schleife wird nun in Zeile 8 zufällig eine der vorhandenen Mutationsstrategien ausgewählt, die gleich darauf den übergebenen Pfad nach gewissen Regeln abändert (siehe Abschnitt 5.1.4).
In Zeile 10 wird die Wahrscheinlichkeit berechnet, mit der die Änderungen am Ursprungspfad akzeptiert werden.
Diese Wahrscheinlichkeit wird einer normalverteilten Zufallszahl gegenübergestellt (Zeile 12). Bei Akzeptanz wird der alte Ursprungspfad durch den mutierten Pfad ersetzt. Der Pfad durchkreuzt nun im Idealfall einen neuen Pixel in der Bildfläche. An diese Stelle wird nun der Farbwert, der sich aus dem Farbbeitrag des Pfades ergibt, im Bild eingetragen(Zeile 15).

5.1.2. Der Initialpfad

Die Wahl des richtigen Initialpfads ist von entscheidender Bedeutung für den weiteren Verlauf. Prinzipiell ist jeder Augenpfad durch einen beliebigen Punkt auf der Bildfläche zum Initialpfad geeignet, so dass mit einem einzigen ausgekommen werden könnte. Jedoch existieren in vielen Szenen, die gerendert werden sollen, Augenpfade, die nicht

mit den vorhanden Lichtpfaden zu einem vollständigen bidirektionalen Pfad verschweißt werden können (wenn bspw. die Vertexe der Pfade nicht sichtbar untereinander sind). Der Energieflusstherm $L(x, x')$ der Rendergleichung (vgl. Abschnitt 3.1) ist für diese Pfade 0.
Um die Chancen auf die Auswahl eines solchen ungeeigneten Therms zu verringern, empfiehlt es sich, den Algorithmus mit einer Initialpfadzahl größer 1 laufen zu lassen.
In seiner Dissertation [Vea97, Seite 337-342] diskutiert Veach weitere Möglichkeiten um diese Startverzerrung zu beheben.

5.1.3. Die Akzeptanzwahrscheinlichkeit

Das MLT-Verfahren basiert auf dem *Metropolis Sampling Algorithmus* [?]Met53). In seiner Arbeit stützt sich Veach dabei auf eine etwas modernere Betrachtungsweise von Kalos & Whitstock [KW86]. Sie beschreiben das Problem mit Hilfe einer Markov-Kette. Das ist eine Kette von Zuständen der Form $\overline{X}_0, \overline{X}_1 \ldots$ wo $\overline{X}_i$ ausschließlich von $\overline{X}_{i-1}$ abhängt und proportional zu einer Funktion $f : \Omega \to \Re_+$ verteilt ist unabhängig vom Initialzustand der Kette. In Veachs Raytracing-Umsetzung des MLT-Algorithmuses entspricht $\overline{X}_i$ einem Pfad innerhalb des Pfadraums.
Ausgehend von $\overline{x} = \overline{X}_{i-1}$ beschreibt $T(\overline{x} \to \overline{y})$ die Wahrscheinlichkeitsdichte mit der aus einem möglichen $\overline{X}'_i = \overline{y}$ der nächste $\overline{X}_i$ wird.
Mit Hilfe der Aktzeptanzwahrscheinlichkeit $a(\overline{x} \to \overline{y})$ wird $\overline{X}_{i-1}$ nun in $\overline{X}_i$ überführt oder bleibt gleich. Ziel der Umformungen ist dabei ein Gleichgewicht oder auch Equilibrium zu erhalten, bekannt unter dem Namen *Detailed Balance* mit der Form:

$$f(\overline{x})T(\overline{x} \to \overline{y})a(\overline{x} \to \overline{y}) = f(\overline{y})T(\overline{y} \to \overline{x})a(\overline{y} \to \overline{x}) \tag{5.1}$$

Um aus einem Ungleichgewicht so schnell wie möglich wieder ins Equilibrium zu kommen, sollten daher $a(\overline{x} \to \overline{y})$ und $a(\overline{y} \to \overline{x})$ so groß wie möglich werden. Dies im Hinterkopf lässt sich die *Detailed Balance* umstellen zu:

$$a(\overline{x} \to \overline{y}) = min\left\{1, \frac{f(\overline{y})T(\overline{y} \to \overline{x})}{f(\overline{x})T(\overline{x} \to \overline{y})}\right\} \tag{5.2}$$

Interpretiert in der Domäne des Raytracings bedeutet das: eine Mutation eines Pfades $\overline{x}$ zum Pfad $\overline{y}$ wird dann wahrscheinlicher, wenn das Verhältnis des Farbwerts $f(\overline{y})$ des mutierten Pfads zur Wahrscheinlichkeitsdichte der Mutation von $\overline{x}$ nach $\overline{y}$ größer ist als das der reversen Mutation von $\overline{y}$ nach $\overline{x}$.

5.1.4. Die Mutationsstrategien

In seiner Dissertation [Vea97, Seite 343-356] beschreibt Veach verschiedene Strategien, um die gegebenen Pfade zu mutieren.
Diese Strategien müssen dabei folgende Eigenschaften erfüllen:

- *Hohe Akzeptanzwahrscheinlichkeit*: Werden Mutationen zu häufig abgelehnt, führt das zu langen änderungslosen und daher unnützen Schleifendurchläufen. Die daraus resultierende schlechte Abdeckung der Bildfläche stellt sich als starkes Rauschen dar.
- *Große Änderungen*: Selbst bei hoher Akzeptanzwahrscheinlichkeit führen Mutationen mit nur geringen Änderungen ebenfalls zu einer schlecht abgedeckten Bildfläche.
- *Ergodizität*: Die Mutationen dürfen in ihren Mitteln nicht zu stark limitiert sein. Damit soll gewährleistet werden, dass immer der gesamte Pfad-Phasenraum erforscht werden kann unabhängig vom Initialpfad, so dass es nicht dazu kommt, dass Pfade in bestimmten Ecken der Szene steckenbleiben (Sackgasse) und weitere Mutationen nicht mehr akzeptiert werden.
- *Änderungen auf der Bildfläche*: Um eine möglichst gute Abdeckung der Bildfläche zu gewährleisten, sollten die Mutationen auch Änderungen der ersten Kante des Augenpfads - von Veach mit «lens edge» bezeichnet - umfassen.
- *Gleichverteilung*: Die Strategien sollten dafür sorgen, dass die Anzahl der Pfade, die einen Beitrag zu einem Pixelfarbwert liefern, für alle Pixel der Bildfläche ungefähr gleich ist.
- *Geringe Kosten*: Die Mutationen sollten möglichst wenig Berechnungsaufwand und Ressourcenverbrauch mit sich bringen.

Nachfolgend werden vier Mutationsstrategien vorgestellt, die Veach in seiner Dissertation [Vea97, Seite 345-355] beschrieben hat und die in dieser Arbeit praktisch umgesetzt wurden.

Die Bidirektionale Strategie

Die elementarste aller Strategien stellt die *Bidirektionale Strategie* dar. Die Mutation besteht dabei im wesentlichen aus dem Löschen eines Teilpfads $x_l \dots x_m$ des Ausgangspfads $x_0 \dots x_k$ und dem Hinzufügen eines neu verfolgten Weges, wie Abbildung 5.1 beispielhaft zeigt. Dazu werden mittels der Warscheinlichkeitsdichtefunktionen $p_d[l, m]$ der Anfangs- und Endindex l und m des zu löschenden Abschnitts ermittelt. Der Löschpfad besitzt dabei die Länge $k_d = m - l$ und $m - l - 1$ Indexe mit $0 \leq l < m \leq k$. $pd[l, m]$ ist so angelegt, dass das Löschen kürzerer Pfade wahrscheinlicher ist, da dies weniger Änderung bewirkt und somit die Akzeptanzwarscheinlichkeit zunimmt. Veach schlägt in seiner Arbeit dazu folgende Werte zu:
$p_d[2] = 0,5$, $p_d[1] = 0,25$ und $p_d[k_a] = 2^{-k_d}$ *mit* $k_d \geq 3$.
Durch das Löschen zwischen l und m zerfällt der Ausgangspfad in zwei Teilstücke $x^0..x_l$ (Lichtpfad) und $x_m \dots x_k$ (Augenpfad), die prinzipiell auch leer sein könnten[1]. Nachdem festgelegt wurde, zwischen welchen Vertexen gelöscht werden soll, muss nun auch noch die Länge des neu hinzuzufügenden Teilpfades bestimmt werden. Dazu werden mit Hilfe einer zweiten Wahrscheinlichkeitsdichtefunktion $p_a[l', m']$ die Anzahl der Vertexe l' und m' festgelegt, um die der Lichtpfad bzw. der Augenpfad erweitert werden soll. Die Länge des neuen Teilpfads ergibt sich aus $k_a = l' + m'$. Wieder wird darauf geachtet, dass die Änderungen zum Ausgangspfad nicht zu gravierend sind, um die Akzeptanz zu erhöhen. Veach gibt dazu folgende Parameter an:
$p_a[k_d] = 0,5$, $p_a[k_d \pm 1] = 0,15$ und $p_a[k_d \pm j] = 0.2(2^{-j})$ *mit* $j \geq 2$.
Der Licht- und der Augenpfad wird nun mittels der gängigen Raytracing-Operationen um die festgelegten Vertexe erweitert und zum neuen nun mutierten Pfad rekombiniert. Schlägt eine der Operationen fehl[2], wird die Mutation umgehend abgelehnt. Ansonsten wird die Akzeptanzwahrscheinlichkeit des neuen Pfades berechnet.
Dies geschieht wie folgt. Nach Formel 5.2 ergibt sich Akzeptanzwahrschienlichkeit aus:

$$a(\overline{x} \to \overline{y}) = min\left\{1, \tfrac{f(\overline{y})T(\overline{y} \to \overline{x})}{f(\overline{x})T(\overline{x} \to \overline{y})}\right\}$$

Diese kann man zusammenfassen zu:

$$a(\overline{x} \to \overline{y}) = min\left\{1, \frac{Q(\overline{y} \to \overline{x})}{Q(\overline{x} \to \overline{y})}\right\} \tag{5.3}$$

[1] Die Wahrscheinlichkeit dem Gesamtenpfad zu löschen ist gering aber gegeben, womit das Ergodizitätskriterium erfüllt ist.

[2] bspw. weil nicht die geforderte Anzahl an Vertexen erreicht wurde oder weil das eine Pfadende vom anderen aus nicht sichtbar ist

Für den Therm $Q(\overline{x} \rightarrow \overline{y})$ gilt dann:

$$Q(\overline{x} \rightarrow \overline{y}) = p_d[l, m] \sum_{i=1}^{ka} \frac{p_a[i-1, ka-i}{C_i^{bd}} \tag{5.4}$$

$C_i^{bd} \equiv C_{l+i,(k'+1)-(l+i)}^{bd}$ entspricht hier dem Therm $C_{s,t}^*$ bekannt aus Formel 4.4, also dem ungewichteten Beitrag des Pfads, ermittelt über die i. Verbindungskante zwischen Licht- und Augenpfad.
Wurde die Akzeptanzwahrscheinlichkeit erfolgreich berechnet, kann der mutierte Pfad in den Auswahlvorgang gestellt werden (vgl. Pseudo-Code Zeile 12-14).

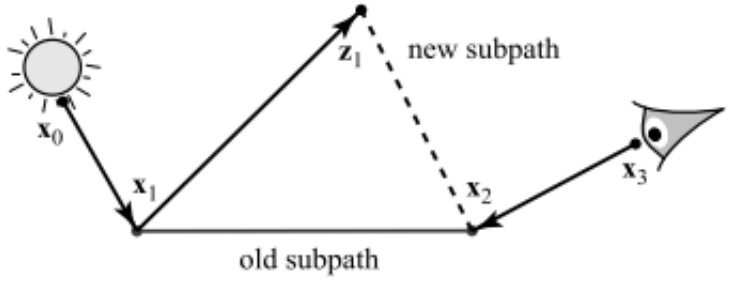


Abbildung 5.1.: Bidirektionale Strategie: Pfadmutation durch Löschen alter und Hinzufügen neuer Kanten und Vertexe, aus [Vea97][S. 348]

Die Lens-Edge-Stratifikation

Die *Bidirektionale Strategie* sorgt für die großen Änderungen innerhalb des Pfades, jedoch werden die *Augenkanten* davon nur relativ selten betroffen sein. Die Folge ist starkes Bildrauschen. Eine weitere Strategie wird benötigt, die die mutierten Pfade ausgeglichen über die Bildfläche verteilt und so das Rauschen minimiert.
Dies geschieht, indem die zu verändernden Vertexe nicht wie in der *Bidirektionale Strategie* frei gewählt sondern speziell auf die erste Kante (Augenkante) des Augensubpfads ausgerichtet werden.
Dieser hat die Form $(L|D)S^*E$. Er wird vom Ausgangspfads entfernt und durch einen neuen ersetzt. Dieser wird dadurch gewonnen, dass ausgehend von der Linse ein Strahl durch einen neuen, zufällig gewählten Pixel gesandt und durch die Szene verfolgt wird[3].

[3] Mit beliebig vielen spekularen Reflektionen und Transmissionen bis auf eine diffuse Oberfläche getroffen wird, so dass der Pfad die beschriebene Form erhält.

In der Implementierung wird dazu eine Liste an noch nicht ausreichend abgedeckten Pixeln verwaltet.
Die Mutation gilt dann als erfolgreich, wenn die geforderte Länge des Augenpfads erreicht worden ist, das spekulare Verhalten innerhalb der Vertexe sich nicht geändert hat[4] und der neue Augenpfad sich erfolgreich mit dem alten Teil rekombinieren ließ. Die Berechnung der Akzeptanzwahrscheinlichkeit geschieht analog zur *Bidirektionale Strategie*.

Die Lens-Perturbation

Sowohl die *Bidirektionale Strategie* als auch die *Lens-Edge-Stratifikation* erkunden mit relativ großen Änderungen den Pfadraum. Das kann dazu führen, dass ihre Mutationen in Szenen mit komplexeren Beleuchtungssituationen wie bspw. Kaustik oder schwieriger Sichtbarkeit häufig verweigert werden. Veach [Vea97, Seite 350-352] entwickelt dafür die sogenannten Perturbationstrategien, die geringfügig die Richtung des Strahlverlaufs verändern und so den Pfad nur sehr lokal mutieren, was zu einer erhöhten Akzeptanzwahrscheinlichkeit führt.
Die *Lens-Perturbation* hat die Aufgabe, den Augensubpfad zu perturbieren, wie in Abbildung 5.2 dargestellt. Dies geschieht analog zur *Lens-Edge-Stratifikation* nur wird diesmal kein zufälliger Pixel aus der Bildfläche bestimmt, sondern einer aus der unmittelbaren Umgebung des Pixels, durch den der Ursprungspfad führt. In der Implementierung wird darauf geachtet, dass sich die Perturbation innerhalb von fünf Prozent der Bildschirmfläche um den Ursprungspixel befindet.

[4] bspw. in dem sich durch die Änderung ein Vertex von einer spekularen Oberfläche auf eine diffuse verschoben hat

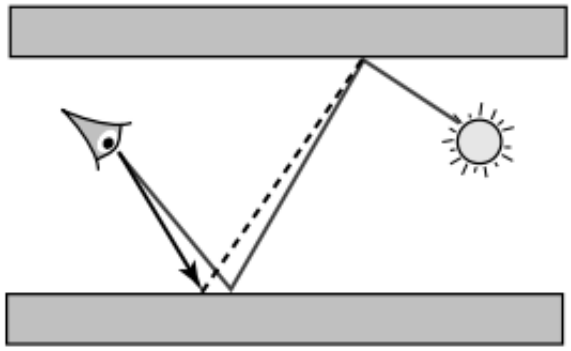


Abbildung 5.2.: Lens-Perturbation: Veränderung des Strahlverlaufs durch geringfügige Richtungsänderung der Augenkante, aus [Vea97][S.351]

Die Kaustik-Perturbation

Neben der Perturbation des Augenpfads kann auch eine Veränderung des Lichtpfads von Nutzen sein, um bspw. Kaustikeffekte stärker zum Tragen kommen zu lassen. Der Lichtpfad hat dabei die Form $(L|D)S^*DE$. Ausgehend von einem Vertex in der Szene wird die Richtung des Lichtstrahls geringfügig verändert[5] und neu durch die Szene verfolgt. Abbildung 5.3 verdeutlicht die Kaustik-Perturbation.

[5] so dass er am Ende die Bildfläche wieder innerhalb der fünf Prozent um den Ursprungspixel durchkreuzt

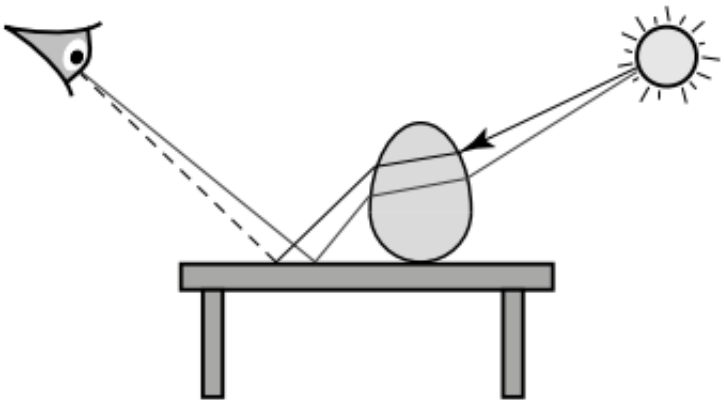


Abbildung 5.3.: Kaustik-Perturbation: Veränderung des Strahlverlaufs durch geringfügige Richtungsänderung des Lichtstrahls, aus [Vea97][S.353]

5.2. Die MLT-Implementierung in GromIMP

Nachfolgend soll dem Leser ein Einblick in die konkrete Implementierung des MLT-Verfahrens in GroIMP gegeben werden. Abbildung 5.4 zeigt dazu die umgesetzte Klassenarchitektur.

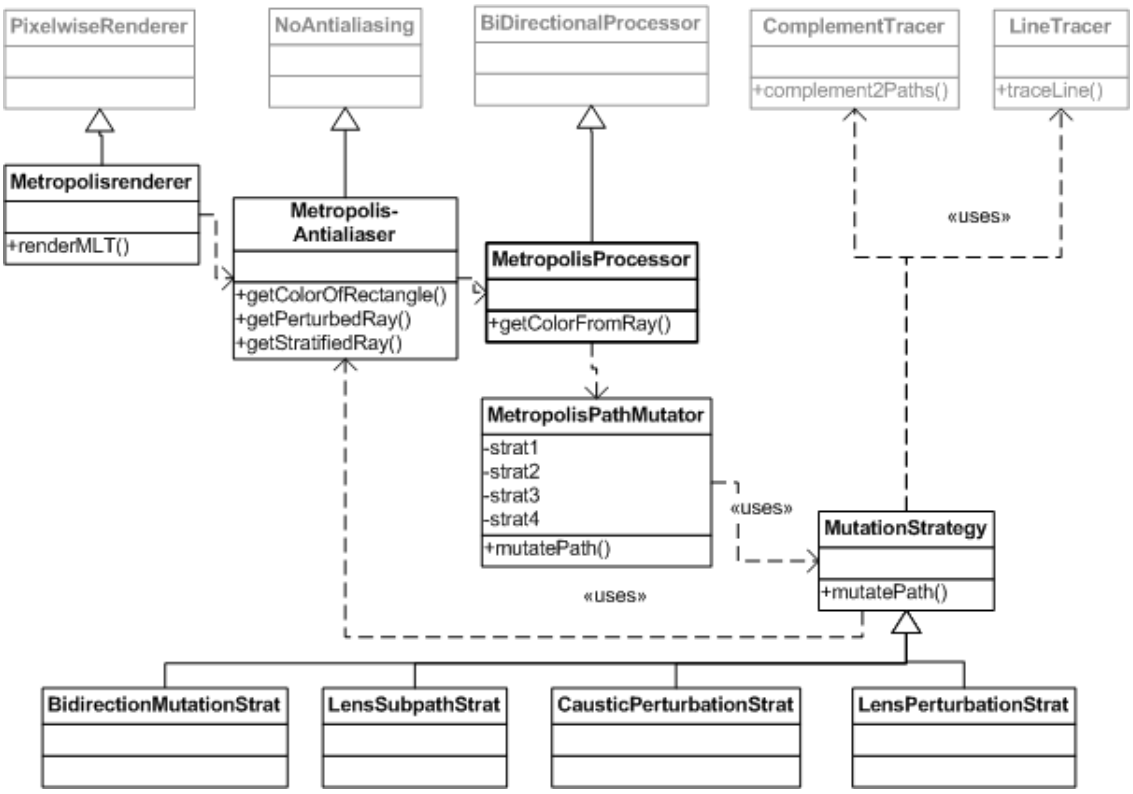


Abbildung 5.4.: Klassenarchitektur des MLT in GroIMP

Die Klasse MetropolisProcessor

Die Klasse `MetropolisProcessor` implementiert dem im Pseudocode-Ausschnitt 5.1 vorgestellten MLT-Algorithmus. Die Klasse selbst liegt im Paket `de.grogra.ray2` und leitet sich von `BiDirectionalProcessor` ab, um deren Methoden und Attribute wiederverwenden zu können. Gestartet wird der Algorithmus über den Aufruf der Funktion `getColorFromRay()`. Die Auswahl des Initialpfades gestaltet sich dabei wie folgt: Für den vom Aufrufer (einer Instanz vom `MetropolisAntialiaser`) übergebenen Augenstrahl wird mittels der `BiDirectionalProcessor`-Methode `getColorFromRay()` eine optimale Augen-Licht-Pfadkombination ermittelt, die maximale Spektrum-Werte besitzt.
Ist ein entsprechender Initialpfad gefunden, kann in die MLT-Phase eingetreten werden. Dabei wird dieser in einer Schleife an eine Instanz der Klasse

`MetropolisPathMutator` übergeben, die eine der beschriebenen Mutationsstrategie auf diesem ausführt. Im Anschluss wird mittels der errechneten Aktzeptanzwahrscheinlichkeit festgestellt, ob der mutierte Pfad übernommen wird – d.h. den Ursprungspfad ersetzt – oder nicht. Schließlich wird der Farbwert des Pfads zu späteren Übertragung ins Zielbild zwischengespeichert. Die Anzahl der Schleifendurchläufe hängt dabei von der Anzahl der bisher „mutierten“ Pixel ab.

Die Klasse MetropolisPathMutator

Die Klasse `MetropolisPathMutator` ist zu finden im Paket `de.grogra.ray2.metropolis`. Sie verwaltet die zu Verfügung stehenden Mutationsstrategien und steuert deren Auswahl. Der Methode `mutatePath()` wird der Ausgangspfad übergeben. Daraufhin wird zufällig eine Strategie ausgewählt und der mutierte Resultatpfad sowie die errechnete Aktzeptanzwahrscheinlichkeit zurückgereicht.

Die Klasse MutationStrategy

`MutationStrategy` ist eine abstrakte Klasse im Paket `de.grogra.ray2.metropolis.strategy` und Basis aller nachfolgenden Mutationsstrategie-Klassen. Sie versammelt einige in allen Strategien benötigte Funktionalität wie die Strahlverfolgung, die Berechnung der Akzeptanzwahrscheinlichkeit oder die Berechnung der zu ändernen Pfadvertexe.

Die Klassen BidirectionalMutationStrat, LensSubpathStrat, LensPerturbationStrat und CausticPerturbationStrat

Die Klassen `BidirectionalMutationStrat`, `LensSubpathStrat`, `LensPerturbationStrat` und `CausticPerturbationStrat` im Paket `de.grogra.ray2.metropolis.strategy` erben von `MutationStrategy` und vollziehen die vier beschriebenen Mutationen. Darüberhinaus sammeln sie statistische Informationen zur Durchführung der Mutation.

Neben der Implementierung der reinen MLT-Algorithmusklassen wurden auch Änderungen am Antialiaser und Renderer notwendig. Ursache dafür ist ein Paradigmenwechsel

in der Abtastung des Bildes. Das ursprünglich implementierte Scanline-Verfahren, das das Bild zeilenweise abläuft und für jeden Pixel ein Strahlbündel in die Szene aussendet, ist gut geeignet für die meisten in GroIMP enthaltenen Raytracing-Verfahren[6]. Der Mutationsansatz des MLT-Verfahren, der die Pfade zufällig über die Bildfläche verschiebt, verhindert jedoch eine zeilenweise Abtastung.
Darüber hinaus werden auch keine Strahlbündel durch einen Pixel mehr erzeugt, so dass der Antialiaser seine ursprüngliche Aufgabe verliert. Daraus folgt, dass sowohl der Renderer als auch der Antialiser in Teilen neu implementiert werden mussten.

Die Klasse MetropolisAntiAliaser

Die urpsrüngliche Antialiaser-Implementierung war für die stochastische Verteilung der Strahlen in einem festgelegten Pixel zuständig, was jedoch im MLT-Verfahren obsolet geworden ist. Die Instanzen der neuen Klasse `Metropolisantialiaser` im Paket `de.grogra.ray2.antialiasing` übernehmen nun die Strahlerzeugung von der Kamera aus. Die Klasse bietet dazu entsprechend der unterschiedlichen Mutationsstrategien, die die Augenkante bedienen, verschiedene Erzeugungsfunktionen.

Die Klasse MetropolisRenderer

Die Klasse `MetropolisRenderer` im Paket `de.grogra.ray2.tracing` erbt von der Klasse `PixelwiseRenderer`. Zu deren Aufgabe gehört es, mittels eines Scanline-Verfahrens die Bildfläche abzutasten, den Antialiaser mit den Pixelkoordinaten für den zu erzeugenden Strahl zu versorgen und die akkumulierten Farbwerte des Pixels aus der Strahlverfolgung in das Farbfeld des Zielbildes einzutragen. Um die Abarbeitung parallsieren zu können, wird dazu jedem Arbeitsthread ein anderer Bildbereich übertragen.
Diese Vorgehensweise ist ungegeignet für den MLT-Algorithmus, weshalb die entsprechenden Methoden des Pixelwiserenderer im Metropolisrenderer überschrieben werden mussten. In der Methode `renderMLT()` werden in einer Schleife über die Anzahl der Initialpfade zufällige Pixel auf der Bildschirmoberfläche ausgewählt. Dies werden an eine Instanz des `Metropolisantialiaser` weitergereicht, der die dazugehörigen Strahlen erzeugt, aus denen dann im `MetropolisProcessor` die Initialpfade erzeugt werden, die die Grundlage der Mutationen bilden. Das Ganze geschieht in einer weiteren Schleife, in der die Anzahl der bereits „mutierten“ Pixel mit den Zielvorgaben verglichen werden.

[6] Aber bereits für das *Bidirectionale Path Tracing* musste es erweitert werden (vgl. Abschnitt 4).

Die Parallelisierung des Prozesses findet auf Grundlage dieser beiden Werte statt: Jedem Arbeitsthread wird eine Zielvorgabe an zu verändernden Pixeln gemacht, solange bis die Gesamtvorgabe, die sich aus der Bildschirmfläche und dem vom Benutzer eingestellten Zielwert ergibt, erreicht ist.
Neben `renderMLT()` musste auch die Methode `merge()` überarbeitet werden. Diese wird jetzt nicht mehr wie ursprünglich durch den Renderer selbst aufgerufen, sondern durch die Instanzen des Metropolisprozessors. Die Farbwerte werden dabei nun nicht final an ihre Position im Bildfeld eingetragen sondern über die Bearbeitungszeit hinweg akkumuliert.

Die MLT-Parameter in GroIMP

Die Steuerung des MLT-Raytracers in GroIMP erfolgt über eine eigene Parameterbox in den Raytracing-Optionen, wie Abbildung 5.5 sie zeigt.
Der Nutzer hat die Möglichkeit per Checkbox die zu verwendenden Mutationsstrategien auszuwählen. Des Weiteren kann er die Anzahl der Initialpfade im Feld «Count of seed paths» bestimmen. Der entscheidenste Parameter für die Qualität des Endbildes ist das Feld «Count of mutations per pixel» über das die durchschnittliche Zielanzahl der Mutationen pro Bildpixel festgelegt wird. Dabei muss beachtet werden, das der Renderer versucht, diesen Durchschnittswert für alle Pixel des Bildes zu erreichen. Bei Szenen, die großteilig leer sind (bspw. ein frei im Raum schwebendes Objekt ohne umgebene Elemente), werden die mit einem Objekt hinterlegten Pixel entsprechend häufiger vom Algorithmus frequentiert.
Da im Vorfeld auf den Bidirectionalen Raytracer zurückgegriffen wird, bestimmen dessen Parameter für die Länge der Pfade auch die des MLT-Raytracers.

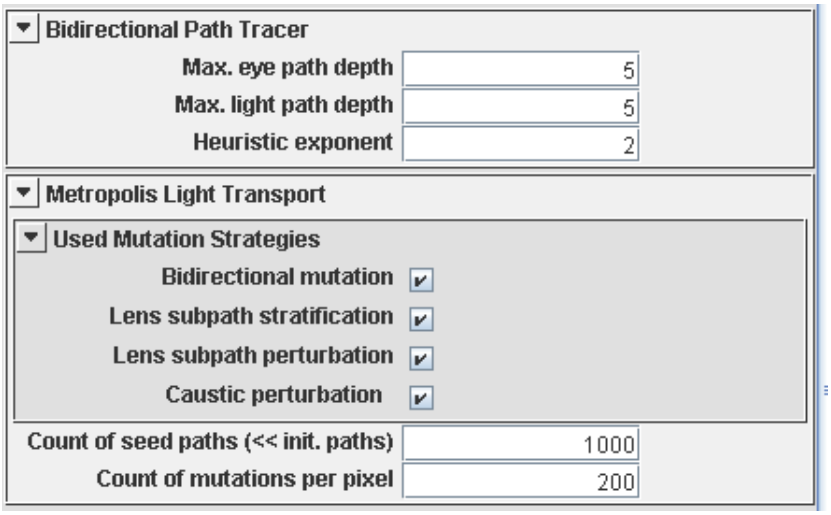


Abbildung 5.5.: Die MLT-Parameter in GroIMP

5.3. Der MLT-Algorithmus im Vergleich

Die nachfolgenden Bilder wurden sowohl mit dem implementierten BPT-Verfahren als auch dem MLT-Algorithmus in GroIMP gerendert. Sie dienen als Grundlage, um die Vor- und Nachteile des MLT-Verfahrens gegenüber BPT herauszuarbeiten. Alle Bilder wurden in etwa gleicher Rechenzeit gerendert. Die genauen Einstellungen entnehme der Leser bitte der Tabelle A.2 im Anhang.

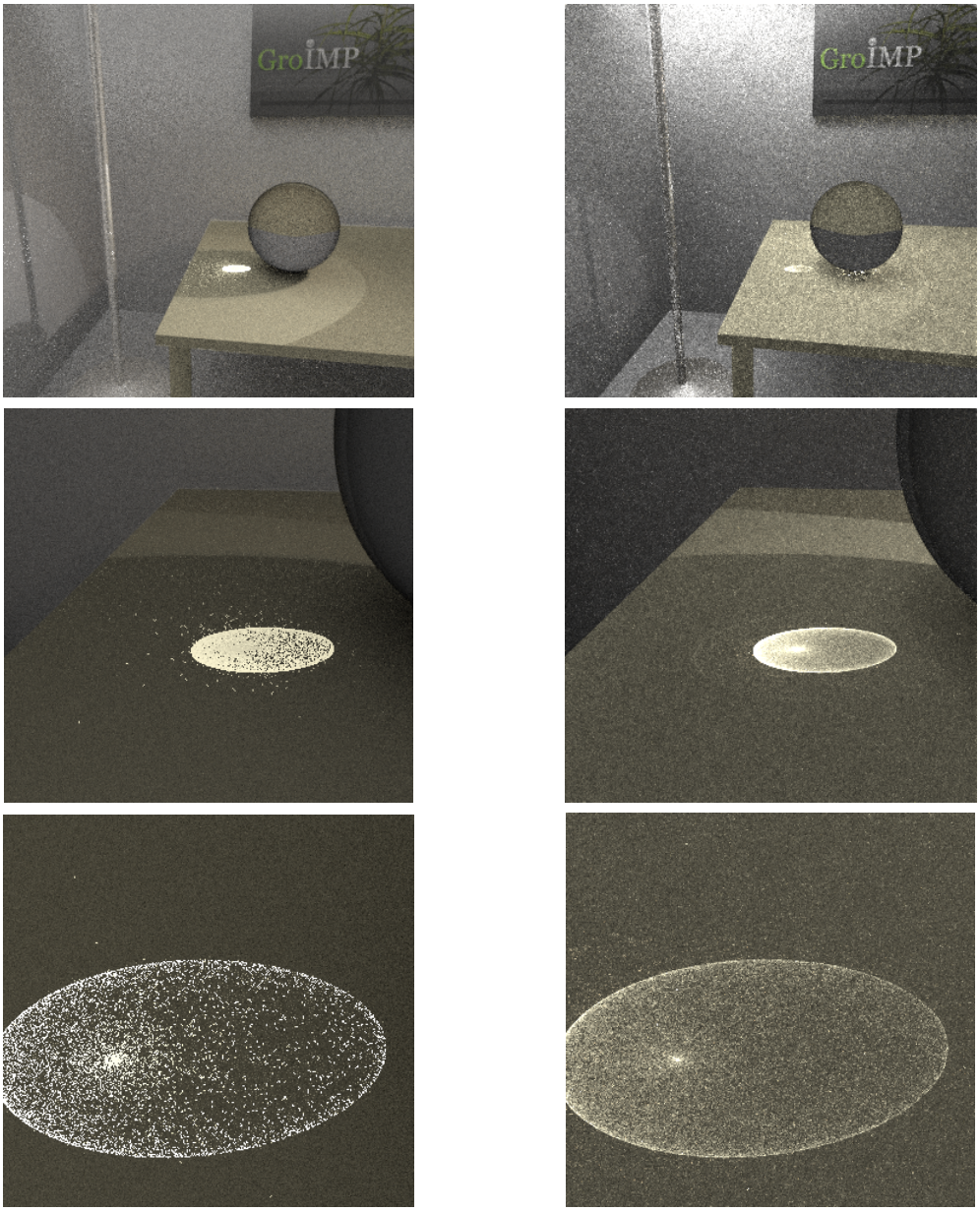
Kaustik

Die Abbildungsreihen 5.6 (a) und (b) zeigen die bereits aus Abschnitt 4 bekannten Tischszene. Diese wurde sowohl mit BPT, Spalte (a), als auch MLT, Spalte (b), in verschiedenen Zoomstufen gerendert, um die Unterschiede in der Darstellung von Kaustik hervorzuheben.
Gut zu erkennen ist, wie sich die Qualität der Kaustikdarstellung in den BPT-Bildern mindert, da die Anzahl der Lichtstrahlen, die durch die Glaskugel fallen und zur Kaustik beitragen, konstant ist und nicht durch die Auflösung des Bildbereichs tangiert wird. Durch das Zoomen treten die von der Kaustik betroffenen Pixel deutlich hervor, wodurch diese verrauscht wirkt.

Das MLT-Verfahren reagiert auf das Zoomen in dem Sinne, dass nun verstärkt Pfade, die durch die Glaskugel führen, erfolgreich mutiert werden. Das führt dazu, das sich die Qualität der Kaustikdarstellung der Zoomstufe anpasst und so auf gleich bleibend hohen Niveau liegt.

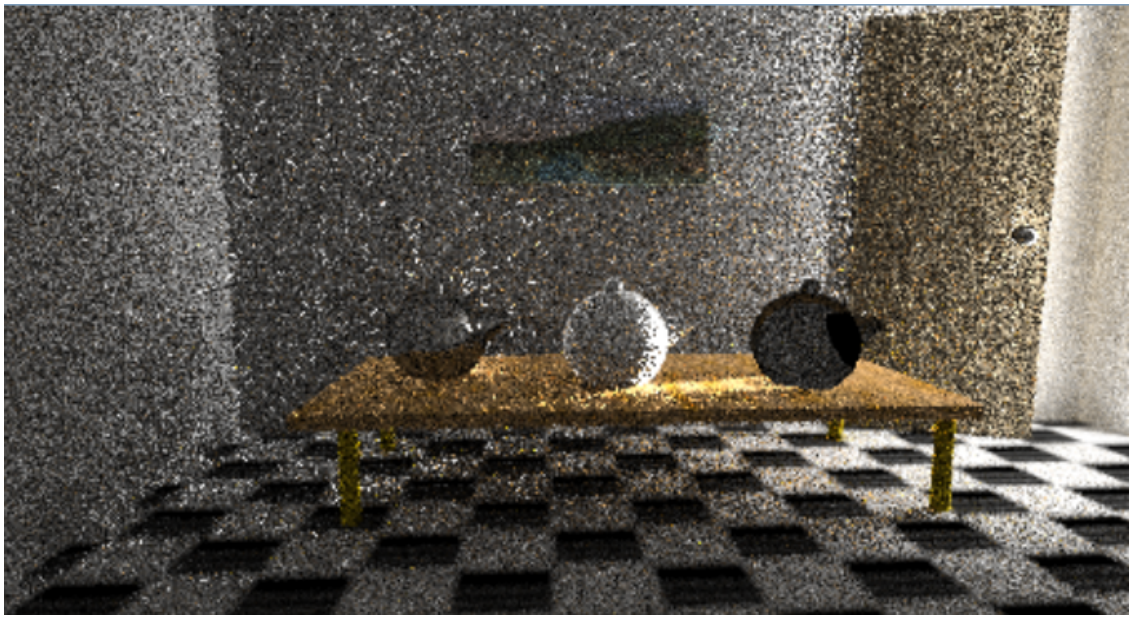
Indirekte Beleuchtung

Seine vollen Möglichkeiten entfaltet der MLT-Algorithmus in Szenen mit extrem schwierigen Beleuchtungsverhältnissen wie bspw. einem sehr hohen Anteil an indirektem Licht. Abbildungen 5.7(a) und (b) zeigen eine solche Szene, in dem die Beleuchtung ausschließlich über einen schmalen Türspalt erfolgt. Der Raum selbst ist nach allen Seiten hin geschlossen, enthält keine Lichtquellen, dafür aber spekulare, diffuse und texturierte Objekte.
Das BPT-Verfahren versagt in dieser Szene, da nur Licht- oder Augenpfade die durch den Spalt gelangen erfolgreich kombiniert werden können. Dies trifft jedoch nur auf einen geringen Anteil aller ausgesandten Strahlen zu. Der Rest scheitert hauptsächlich an der eingeschränkten Sichtbarkeit. Das führt zu einer schlechten und unregelmäßigen Ausleuchtung, die sich in starkem Rauschen äußert.
Das MLT-Verfahren umgeht dieses Problem, in dem es als Grundlage für seine Mutationen ausschließlich die erfolgreichen Pfade nutzt, die von der Lichtquelle durch den Türspalt in die Kamera führen. Mittels der beschriebenen Mutationen kann nun die Szene optimal ausgeleuchtet werden, was die resultierende Bildqualität stark verbessert. Details werden sichtbar, wie etwa der Inhalt des Bildes an der Wand oder die Textur von Tisch und Tür. Die Kanten werden schärfer dargestellt und an der linken Wand sind die leichten Halbschatten des Tischs und der Kannen hervorgerufen durch die stark beleuchtete rechte Wand zu erkennen.

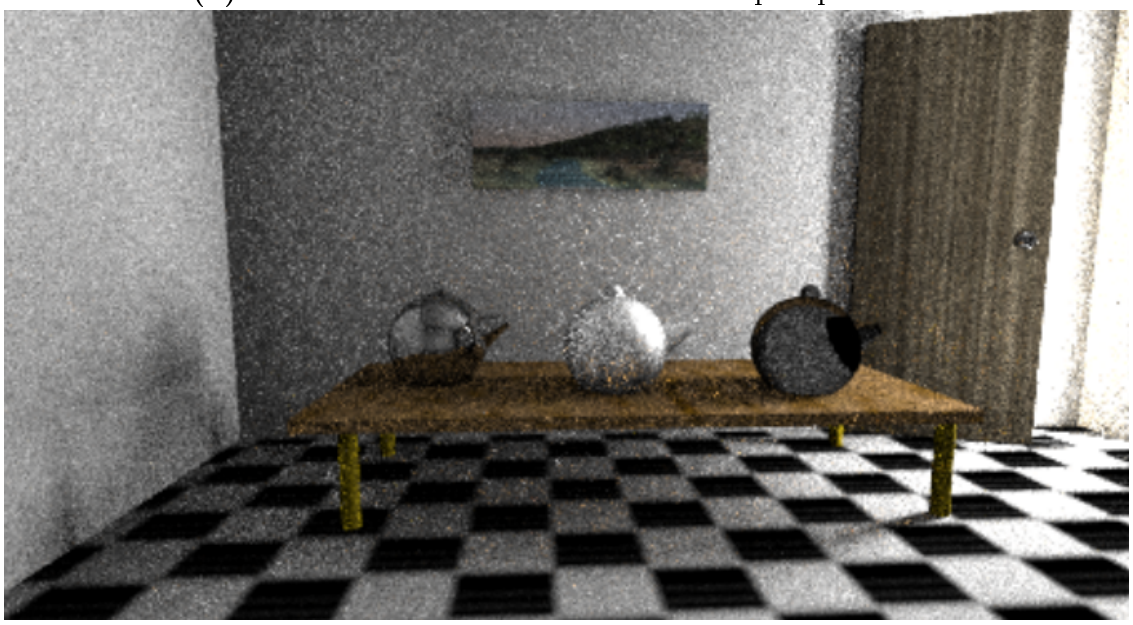


(a) BPT mit 25 Samples pro Pixel **(b)** MLT mit 350-250 Mutationen pro Pixel

Abbildung 5.6.: Vergleich der Genauigkeit von Kaustikdarstellungen zwischen BPT und MLT in verschiedenen Zoomstufen



(a) Bidirectional PathTracer mit 64 Samples pro Pixel



(b) Metropolis Light Transport mit 250 Mutationen pro Pixel

Abbildung 5.7.: Vergleich des BPT- und des MLT-Verfahrens in einer komplexen Szene mit ausschließlich indirektem Licht

5.4. Ausblick

Die vorliegende Implementierung umfasst die Umsetzung des MLT-Algorithmusses inklusive der vier gängigen Mutationstsrategien, wie weiter oben beschrieben. Zudem wurde auch die Erwartungswertverbesserung[7] nach Veach [Vea97][S.357] umgesetzt.
In dessen Dissertation werden weitere Verbesserungsmöglichkeiten vorgeschlagen, die hier kurz wiedergegeben werden sollen.

Multichain-Mutationsstrategie

Veach stellt eine weitere Mutationsstrategie vor, die auf Pfade der Form $L(D|S|L)^*DS^+DS^+E$ spezialisiert ist, wie sie bspw. eine Szene mit einem Wasser gefüllten Becken, auf dessen Boden Kaustik durch die Wellenbrechung entsteht, enthalten könnte, dargestellt in Abbildung 5.8. Zur Abarbeitung dieser Mutation schlägt Veach vor, zuerst den Augensubpfad DS^+E wie gewohnt zu perturbieren an dann für jede weitere Segment DS^+D den beginnenden Richtungsvektor analog zur Caustic-Perturbationstrategie zu verändern.

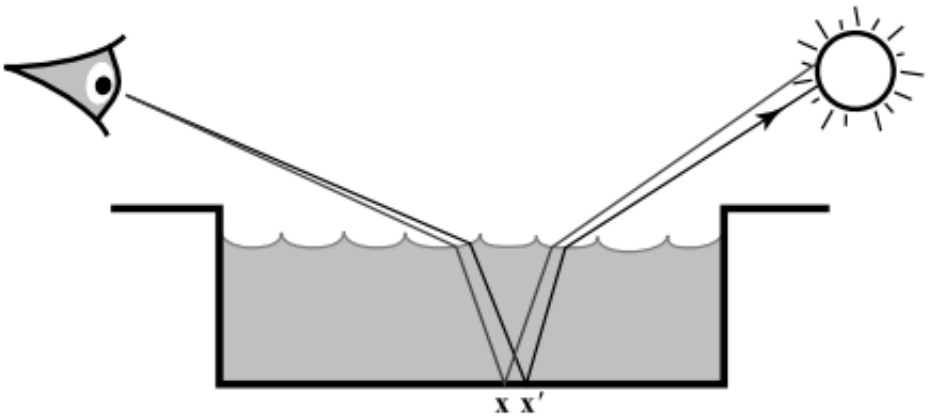


Abbildung 5.8.: Multichain-Strategie angewendet auf einen Pfad durch ein Wasserbecken, aus [Vea97][S.353]

[7] engl. **expected values**

Direkte Beleuchtung

In seiner Referenzimplementierung nutzt Veach immer, wo es möglich ist, die Techniken der Direkten Beleuchtung des Standard-Raytracings. Laut Veach erzielen die Standardtechniken dabei bessere Ergebnisse bei geringeren Kosten. Zurückzuführen sei das darauf, dass im MLT-Algorithmus die Bildfläche nicht immer gleichmäßig durch Strahlen abgedeckt wird. Daher ist es ihm zu Folge günstig die Berechnung der Direkten Beleuchtung immer dann auszuführen, wenn eine Lens-Edge-Stratifikation-Strategie eine neue Augenkanten erzeugt. Die Pfade mit direkter Beleuchtung $L(D|S)E$ sollten von der Erzeugung im MLT-Algorithmus ausgeschlossen sein.

Zweistufige Bildgenerierung

Der MLT-Algorithmus fokussiert auf die Mutation energiereicher Pfade. Dies führt dazu, dass er sich verstärkt in den helleren Bereichen aufhält, diese später also genauer und weniger verrauscht dargestellt werden als die dunkleren Bereiche. Zur Lösung dieses Problems schlägt Veach vor, den Algorithmus zweistufig laufen zu lassen.
In der ersten Phase wird ein Bild mit einer relativ geringen Auflösung gerendert. Dessen Farbwerte werden in der zweiten Stufe für die weiteren Berechnungen herangezogen. Mathematische betrachtet wird $f(\overline{x})$ aus Formel 5.1 umgestellt zu

$$f'(\overline{x}) = f(\overline{x})/I_0(\overline{x}) \tag{5.5}$$

Mit $I_0(\overline{x})$ ist dabei der Farbwert aus dem Testbild der ersten Stufe gemeint, durch den der Pfad $\overline{x}$ führt. Der Einsatz dieses zweistufigen Renderings führt nach Veach dazu, dass nun in den hellen Bereichen des Bildes weniger Samples mit höheren Werten und in den dunkleren mehr Samples mit geringen Werten erzeugt werden. Dies begünstigt eine bessere Verteilung der Pfade über die Bildfläche, was sich in weniger Rauschen niederschlägt.

Importance Sampling

Um die Aktzeptanzwahrscheinlichkeit der durchgeführten Mutationen zu erhöhen und damit den Gesamtberechnungsaufwand zu verringern, führt Veach eine Form des *Importance Samplings* ein, d.h. es wird versucht bei der Auswahl des zu mutierenden Teilpfades

von vornherein diejenigen zu wählen, die in der sich anschließenden Berechnung der Aktzeptanzwahrscheinlichkeit die höheren Werte erhalten.
Dazu betrachtet Veach die Formeln zu Berechnung der Wahrscheinlichkeit genauer. Dabei ist festzustellen, dass bereits vor Ausführung der eigentlichen Mutation einige Therme der Gleichung feststehen. Lediglich die Therme der Wahrscheinlichkeit sind unbekannt, da sie von den Indexen des zu mutierenden Teilpfades abhängen. Setzt man sie auf 1 so erhält man für $Q(\overline{y} \rightarrow \overline{x})$:

$$Q(\overline{y} \rightarrow \overline{x}) = \sum_{i=1}^{kd} \frac{1}{C_i^{bd}} \tag{5.6}$$

Dieser Wert ist niedrig, wenn die Beiträge des Subpfads des ursprünglichen Pfads $\overline{x}$ groß sind, und groß, wenn sie nur kleine Werte aufweisen. Er kann also als Gewichtungsfaktor bei der Auswahl des Anfangs- und Endindexes des zu löschenden Subpfads eingesetzt werden.

Für weiterführende Erklärungen oder eine genaue mathematische Herleitung wird auf die Dissertation Eric Veachs [Vea97][S.356-359] verwiesen.

5.5. Zusammenfassung

Das MLT-Verfahren nach Veach überträgt der Metropolis Sampling Algorithmus in die Domäne der Bildgenerierung. Anders als PT, Photon Mapping oder BPT exploriert es den Raum aller möglichen Pfade von Lichtquelle zu Kamera in einer Szene sehr lokal. Einmal gefundene energiereiche Pfade werden mutiert, d.h. geringfügig abgewandelt, um weitere gute Pfade in der Näher zu finden. Die Mutationen sollten dabei möglichst große Änderungen am Pfad vornehmen, die jedoch eine hohe Akzeptanzwahrscheinlichkeit besitzen. Weiterhin sollten sie die Pfade möglichst gleichmäßig über der Bildschirmfläche verteilen. Außerdem muss immer gewährleistet sein, einen Sackgassenpfad zu verlassen. Veach beschreibt in seiner Arbeit eine Reihe an Mutation, die das leisten und auf bestimmte Rendersituationen hin optimiert sind:

- Die Bidirektionale Strategie vollzieht die großen Änderungen am Pfad in dem zufällig zwei Vertexe ausgewählt werden, zwischen den der Subpfad gelöscht und durch einen neu-verfolgten ersetzt wird.
- Die Lens-Edge-Startifiaktionsstrategie verteilt die Pfade möglichst gleichmäßig über der Zeichenfläche

- Die Lens-Perturbationsstrategie verschiebt den Augenstrahl geringfügig auf der Bildschirmfläche, um dessen nähere Umgebung zu erforschen.
- Die Kaustik-Perturbationsstrategie ist auf mögliche Kaustikpfade aboniert.

Die Implementierung in GroIMP gestaltete sich nicht ganz einfach, da dass Scanline-Konzept des PixelwiseRenderer verworfen werden musste, um den zufälligen Strahlerzeugungsmuster des MLT-Verfahrens gerecht zu werden.
Die Qualität der Renderergebnissen von Szenen mit „normaler“ Beleuchtung, d.h. Beleuchtung die sich aus direkten und indirekten Anteilen zusammensetzt, ist das MLT-Verfahren mit den anderen MC-Verfahren durchaus vergleichbar. Seine volle Überlegenheit kann es allerdings bei Szenen mit komplexerer Beleuchtung (viel Kaustik, sehr viel indirekter Beleuchtung, geringe Sichtbarkeit zwischen Kamera und Lichtquelle) ausspielen.

Glossar

Antialiasing (Aa)

Kantenglättungsverfahren

Bidirectional Path Tracing (BPT)

Path-Tracing-Verfahren, bei dem sowohl von der Kamera als auch von allen Lichtquellen der Szene Strahlenpfade erzeugt und miteinander verknüpft werden

Growth Grammar-related Interactive Modelling Platform (GroIMP)

3D-Modellierungsplattform des Lehrstuhls Grafishe Systeme der BTU Cottbus, spezialisiert auf Wachstumsgrammatiken

Kaustik

durch Strahlbündelung erzeugte Stellen hoher Lichtintensität

Metropolis Light Transport (MLT)

Path-Tracing-Verfahren, bei dem auf der Bildfläche zufällig verteilte Initialpfade solange nach speziellen Regeln mutiert werden, bis alle Bildpixel abgedeckt sind und die gewünschte Genauigkeit und Mutationsrate erreicht ist.

Monte-Carlo (MC)

Stochastisches Verfahren, um mittels Zufallszahlen komplexe mathematische Probleme zu lösen

Path Tracing (PT)

Ray-Tracing-Verfahren, mit dessen Hilfe eine globale Beleuchtungssimulation durchgeführt werden kann

Seperation of Concerns (SoC)

Grundlegendes Architekturprinzip in der Software-Technik, das besagt, dass ein Programm so zu zergliedern ist, dass die einzelnen Teile möglichst wenig funktional überlappen

Kapitel 6

Literaturverzeichnis

[App68] Appel, Arthur: Some Techniques for Shading Machine Renderings of Solids. In: *Spring Joint Computer Conference 1968*. Arlington, USA : AFIPS Press, 1968, S. Seiten 37–45

[BSMM01] Bronstein, I.N. ; Semendjajew, K.A. ; Musiol, G. ; Mühlig, H.: *Tachenbuch der Mathematik*. Frankfurt(Main), Germany : Verlag Harri Deutsch, 2001. – ISBN 3–8171–2005–2

[GN71] Goldstein, Robert ; Nagel, Roger: *3-D Visual Simulation. Simulation 16*. 1971

[Gro07] GroIMP: *Internetseite von GroIMP*. URL: http://www.grogra.de/software/groimp, Stand: 03. August 2007

[Kaj86] Kajiya, James: The Rendering Equation. In: *Proceedings of ACM SIGGRAPH Computer Graphics 20 (Aug. 1986)*, 1986. – ISBN ISSN 0097–8930, S. Seiten 143–150

[Kop08] Kopsch, Ralf: *Erweiterung eines bestehenden Raytracers um Photon Mapping und Radiosity*. URL: http://www-gs.informatik.tu-cottbus.de/index.htm, 2008

[Kur] Kurth, Prof. Dr. W.: *Lehrstuhl Graphische Systeme, BTU Cottbus*. URL: http://www-gs.informatik.tu-cottbus.de/index.htm,

[KW86] Kalos, Malvin H. ; Whitlock, Paula A.: *Monte Carlo methods. Vol. 1: basics*. New York, NY, USA : Wiley-Interscience, 1986. – ISBN 0–471–89839–2

[LW93] Lafortune, Eric ; Willems, Yves: Bi-Directional Path Tracing. In: *Proceedings of Computer graphics 1993*, 1993, S. Seiten 145–153

[LW94] Lafortune, Eric ; Willems, Yves: Bidirectional Estimators for Light Transport. In: *Eurographics Rendering Workshop 1994 Proceedings*, 1994, S. Seiten 147–162

[MRR+53] Metropolis, Nicholas ; Rosenbluth, Arianna W. ; Rosenbluth, Marshall N. ; Teller, Augusta H. ; Teller, Edward: Equation of State Calculations by Fast Computing Machines. In: *The Journal of Chemical Physics* 21 (1953), Nr. 6, 1087–1092. http://dx.doi.org/http://dx.doi.org/10.1063/1.1699114. – DOI http://dx.doi.org/10.1063/1.1699114

[RS03] Richter, Marcus ; Schäfe, Dörte: *Advanced Raytracing*. Seminar 3D-Grafik, Humboldt Universität Berlin, 2003

[Vea97] Veach, Eric: *Robust Monte Carlo Methods for Light Transport Simulation*, Stanford University, Diss., 1997

[VG97] Veach, Eric ; Guibas, Leonidas: Metropolis Light Transport. In: *Proceedings SIGGRAPH 1997*. New York, NY, USA : ACM Press, 1997, S. Seiten 65–76

Anhang A

Renderparameter

Tabelle A.1.: Übersicht der Rendering-Parameter der Abbildungen des BPT-Kapitels

Abbildung	**Dimension**	**Renderingparameter**
Tabelle 4.5 (a)	340x210	Path Tracing: • recursion depth: 10 • super sampling: 5
4.5 (b)	340x210	Photon Mapping: • recursion depth: 7 • super sampling: 5 • caustic phton count: 500000 • photon area: 0.04

Abbildung	Dimension	Renderingparameter
4.5 (c)	340x210	Bidirectional Path Tracing: • max. eye & light path depth: 5 • Heuristic exponent: 2 • super sampling: 4
4.6 (a)	340x190	Path Tracing: • recursion depth: 10 • super sampling: 5
4.6 (b)	340x190	Bidirectional Path Tracing: • max. eye & light path depth: 5 • Heuristic exponent: 2 • super sampling: 4
4.7 (a)	350x350	Path Tracing: • recursion depth: 10 • super sampling: 8
4.7 (c)	350x350	Bidirectional Path Tracing: • max. eye & light path depth: 5 • Heuristic exponent: 2 • super sampling: 5

Tabelle A.2.: Übersicht der Rendering-Parameter zu den Abbildung des MLT-Kapitels

Abbildung	**Dimension**	**Renderingparameter**
5.6 (a)	alle 350x350	Bidirectional Path Tracing: • max. eye & light path depth: 4 • Heuristic exponent: 2 • super sampling: 5
5.6 (b)	alle 350x350	Metropolis Light Transport: • max. eye & light path depth: 5 • Heuristic exponent: 2 • mutation strategies: alle • Count of seed paths: 1000 • Count of mutation per pixel: 350 - 250
5.7(a)	450x250	Bidirectional Path Tracing • max. eye & light path depth: 5 • Heuristic exponent: 2 • super sampling: 5
5.7(b)	450x250	Metropolis Light Transport: • max. eye & light path depth: 5 • Heuristic exponent: 2 • mutation strategies: alle • max. light path depth: 5 • Count of seed paths: 1000 • Count of mutation per pixel: 250

Anhang B

Aufgabenstellung

Für die am Lehrstuhl Grafische Systeme in Entwicklung befindliche, in Java implementierte 3D-Plattform GroIMP wird derzeit ein Raytracer-System entwickelt, das neben den grundlegenden Bausteinen Szenegraph,Schnittpunktberechnung und Tochterstrahlerzeugung auch die Steuerungslogik für einen klassischen rekursiven Raytracer enthält. Das System selbst ist dabei unabhängig von GroIMP, die Integration erfolgt über die Implementation der vom System bereitgestellten Schnittstelle.

Für dieses System sollen die folgenden Erweiterungen entwickelt werden:

Stochastisches Raytracing ist ein Verfahren zur stochastischen Integration der Rendering-Gleichung, die den Lichttransport in einer Szene korrekt und vollständig (im Sinne der geometrischen Optik) beschreibt. Es soll ein Algorithmus zum bidirektionalen Raytracing entwickelt werden, das von Eric Veach in seiner Dissertation beschrieben wurde. Dieses ist konzeptionell einfach:
Es werden von den Lichtquellen und der Kameralinse aus unabhängig zufällige Pfade erzeugt und deren Vertizes miteinander verbunden, um vollständige Pfade von Quellen zu Senken zu generieren. Die Mathematik muss aber mit Sorgfalt umgesetzt werden. Es kann dabei jedoch auf Methoden zurückgegriffen werden, die für Oberflächen und Lichtquellen bereits physikalisch korrekte, stochastische Verteilungen erzeugen.

Der Metropolis-Algorithmus ist ebenfalls ein stochastisches, von Eric Veach entwickeltes Verfahren. Über das bidirektionale Raytracing werden hier einige geeignete Startpfade

von Quellen zu Senken erzeugt, die vom Metropolis-Algorithmus nach bestimmten Regeln und mit bestimmten Akzeptanz-Wahrscheinlichkeiten mutiert werden. Die Beiträge solcher Pfade werden in den entsprechenden Bildpixeln aufsummiert.

Das Photon-Mapping entspricht eher dem klassischen Raytracing. In einer ersten Phase werden Photonen von den Lichtquellen ausgesandt und ihre Auftrefforte aufgezeichnet. Hieraus ergeben sich Helligkeiten für die getroffenen Flächen, diese werden als ambiente Licht-Terme in der zweiten Phase genutzt, die mit klassischem Raytracing arbeitet.

Das Radiosity-Verfahren ist geeignet, um die Ausleuchtung von Szenen mit rein diffusen Wechselwirkungen zu berechnen. Es basiert nicht auf einzelnen Strahlen, sondern auf der wechselseitigen Sichtbarkeit der elementaren Flächen,in die die Szene aufgeteilt wird. Diese Sichtbarkeit wird in der Formfaktorbestimmung ermittelt. Das Verfahren soll implementiert und mit Raytracing kombiniert werden.

Wie das Basis-System sollen auch die Erweiterungen nur über Schnittstellen an GroIMP angebunden werden, so dass das gesamte Rendering-System unabhängig von GroIMP genutzt werden kann. Über Import-Filter für gängige 3D-Formate könnte man so ein Kommandozeilen-Programm zum hochwertigen Rendern dreidimensionaler Szenen erstellen, das eines der ersten auf erweiterten stochastischen Verfahren beruhenden Systeme wäre und sogar das erste im Open-Source-Bereich.

Anhand einiger präsentierbarer Beispiel-Szenen sollen die Verfahren verglichen werden. Die Ergebnisse dieses Vergleichs sollen auf der Netzseite www.grogra.de verfügbar gemacht werden. Hierzu sollen auch Verknüpfungen in Wikipedia erstellt werden.