Entwicklung und Evaluation eines Post Production Frameworks für Mental Ray in Autodesk Maya

Studiengang Medieninformatik

Masterarbeit

vorgelegt von

Peter Salziger
geb. in Leverkusen

durchgeführt an der
Technischen Hochschule Mittelhessen, Friedberg

Referent der Arbeit:	Prof. Dr. Cornelius Malerczyk
Korreferentin der Arbeit:	M. H. Edu. Sabine Langkamm

Friedberg, August 2012 - Januar 2013

Danksagung

Hiermit danke ich meinen Betreuern Prof. Dr. Cornelius Malerczyk und M. H. Edu. Sabine Langkamm für die Unterstützung während der Zeit meiner Masterarbeit. Die bei Ihnen besuchten Lehrveranstaltungen haben maßgeblich zu meiner Motivation beigetragen, meine Masterarbeit in dem Umfeld 3D und Post Production zu verfassen.

Des Weiteren möchte ich meiner Familie danken, welche mich in den vergangenen sechs Monaten bei meiner Arbeit begleitet und Korrektur gelesen hat. Mein Dank geht an meinen Vater Dr. Rolf Salziger, meine Mutter Barbara Salziger und meine Geschwister Carsten und Caroline Salziger.

Ich danke auch allen interessierten Personen, die sich Zeit für die Bearbeitung meines Fragebogens genommen haben und mir Motivation bei der Fortführung meiner Arbeit gegeben haben.

Abschließend gilt ein besonderer Dank an meine Freundin Medina Neumann, die zu jedem Zeitpunkt unterstützend an meiner Seite stand und mir Hoffnung und Kraft verliehen hat.

Selbstständigkeitserklärung

Ich erkläre, dass ich die eingereichte Masterarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die von mir angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Werken wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Friedberg, Januar 2013

Peter Salziger

Inhaltsverzeichnis

Abbildungsverzeichnis

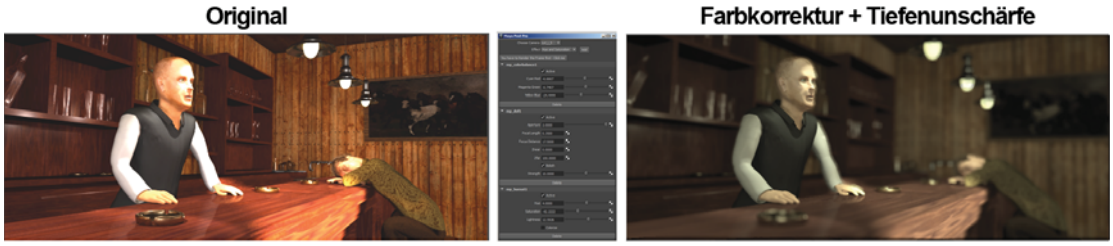
Diagrammverzeichnis

Kapitel 1

Einleitung

Dieses Kapitel ist der einleitende Teil dieser Arbeit und dient dazu, den Leser mit der Problematik des Themas vertraut zu machen. Hierbei wird besonders die Problemstellung und Zieldefinition bewusst verallgemeinert, um einen leichten Einstieg in die Thematik zu ermöglichen. Am Ende dieses Kapitels befindet sich eine Zusammenfassung der gesamten Arbeit.

1.1 Motivation



Post Production ist ein essentieller Bestandteil jeder professionellen Film- und Bildbearbeitung. In diesem finalen Abschnitt wird einem Werk - meist auf 2D-Basis - der letzte Feinschliff verliehen. Eine Nachkorrektur ist zeitlich gesehen sehr effizient, da die Bildalgorithmen z.B. zur Farbkorrektur in wenigen Millisekunden das Aussehen eines ganzen Bildes verändern können. Hierbei kann auf ein zeitaufwendiges erneutes Rendern verzichtet werden.

Heutzutage wird die Post Production jedoch strikt von der 3D Production getrennt. Für eine professionelle 3D-Animation bedarf es folglich den Erwerb von zwei Softwareprodukten (z.B. Autodesk Maya und Adobe After Effects). Die Überführung des Renderergebnisses aus der 3D Applikation in die Post Production wird meistens mittels unkomprimierten Bilddateien vorgenommen, welche je nach Länge der Animation viel Speicherplatz belegen. Eine repräsentative Vorstellung über die Animation erhält man schließlich erst nach der Ver-

arbeitung durch die zweite Post Production Applikation. Besonders für die Erstellung der sogenannten „dailies“ beinhaltet der zweite Arbeitsschritt einen erheblichen Mehraufwand. Unter den „dailies“ versteht man die Vorschau des aktuellen Standes einer Filmproduktion, welche täglich erstellt wird, und vom Regisseur meist am nächsten Tag betrachtet wird.

Die Fragestellung dieser Arbeit beschäftigt sich mit der Zusammenführung von 3D- und Post Production. Inwieweit lässt sich Post Production direkt in den Renderprozess integrieren und welche Vorteile könnte diese Methode mit sich bringen? Hierzu wird ein Prototyp entwickelt, welcher Aspekte der Post Production in Autodesk Maya integriert. Der 3D-Renderer Mental Ray wird um ein 2D-Post Production Framework erweitert. Mit diesem soll es möglich werden, die gängigen 2D-Filter der Post Production auf das Renderergebnis anzuwenden. Dabei sollen Veränderungen an den Filtereinstellung direkt auf dem finalen Rendering sichtbar gemacht werden, so wie man es von Post Production Software gewohnt ist.

Diese Entwicklung ermöglicht die Optimierung der Abläufe einer Filmproduktion. Je nach dem Bedarf verschiedener Interessengruppen könnten diese ganz oder zumindest teilweise auf eine zweite Post Production Software verzichten. Zudem kann das Framework dazu verwendet werden, schnelle Previews zu erzeugen.

1.2 Problemstellung

Eine Filmproduktion lässt sich von der technischen Seite her in zwei Bereiche aufteilen (siehe Abbildung 1.1). Als erster Schritt steht die Erzeugung von Bildmaterial im Vordergrund. Dies können klassische Realaufnahmen von einer analogen oder digitalen Filmkamera sein oder auch von einem 3D-Programm virtuell erzeugt werden. In einem zweiten Schritt, der Postproduktion, wird das zuvor erzeugte Bildmaterial verarbeitet und anschließend auf ein Zielmedium ausgegeben.

Zu den wesentlichen Arbeitsschritten der Postproduktion gehört zuerst die Optimierung des Ausgangsbildes. Hier werden nicht nur visuelle Effekte dem Bild hinzugefügt, sondern auch der farbliche Gesamteindruck des Filmes definiert. In einem nächsten Schritt werden die einzelnen Filmaufnahmen zu einem Gesamtwerk zugeschnitten und mit einer Tonspur unterlegt. Zuletzt wird der fertige Film für die Wiedergabe (Kino, DVD, etc...) vorbereitet.

Die Abbildung 1.1 macht ersichtlich, dass für die Erstellung eines 3D-Filmes mindestens drei Computerprogramme verwendet werden müssen, wenn man unterstellt, dass die Tonspur im Schnittprogramm erzeugt wird[1]. Zum Beispiel wird Autodesk Maya für die 3D-Bilderzeugung genutzt, Adobe After Effects für die Nachbearbeitung und der Avid Media Composer für den Filmschnitt[2].

[1] Bei heutigen Filmproduktionen werden für die Tonerzeugung meistens zusätzliche Programme verwendet

[2] Dies ist nur eine mögliche von vielen weiteren erhältlichen Softwarekombinationen

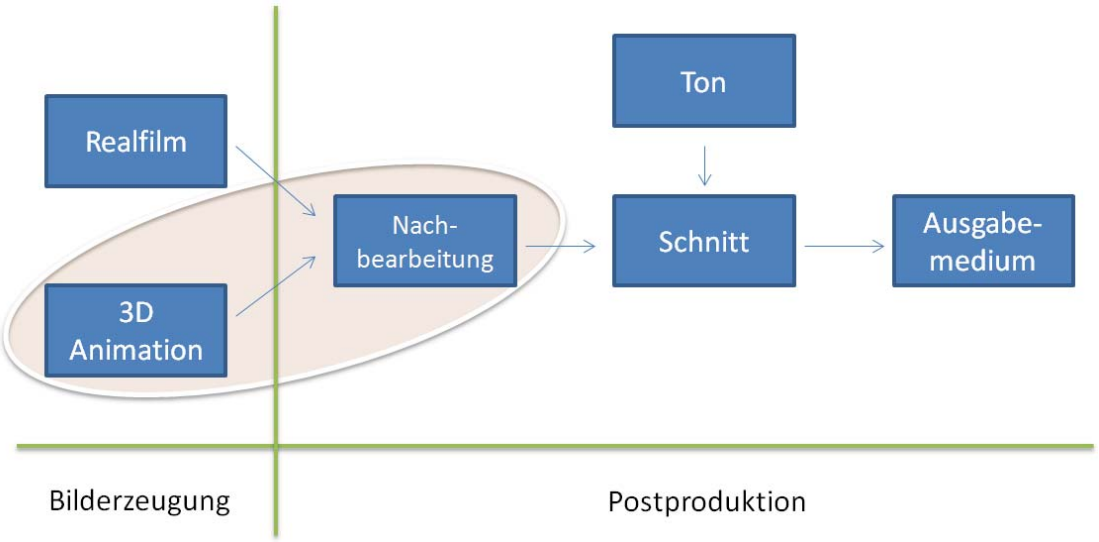


Abbildung 1.1: Abstrakte Ansicht auf die Verarbeitungskette einer Filmproduktion

Im Folgenden wird der Fokus auf die zwei in Abbildung 1.1 eingekreisten Elemente der Verarbeitungskette einer Filmproduktion gesetzt - dem Zusammenhang der Bilderzeugung einer 3D-Produktion und ihrer Nachbearbeitung. Weshalb werden momentan zwei unterschiedliche Softwareprodukte (Erzeugung / Postproduktion) eingesetzt, um das finale Aussehen eines Filmes zu erzeugen?

Die Bilderzeugung (Rendering[3]) einer 3D-Animation nimmt sehr viel Computerrechenzeit in Anspruch. Eine Sekunde Film besteht durchschnittlich aus 25 einzelnen Bildern und jedes dieser Bilder kann, je nach Ausmaß der Produktion, mehrere Stunden brauchen, um von einem Computer generiert zu werden[4]. Die Wahl der richtigen Farbgebung eines Filmes ist ein künstlerischer Prozess, welcher zumeist in Abstimmung mit dem Regisseur durchlaufen wird. Um das finale Aussehen eines Filmes zu bestimmen, sind oft viele Versuche bei der Feinjustage der Farbgebung von Nöten. Die Aufwände für die Produktion steigen enorm an, da für jede kleine Änderung an der Farbgebung das Bild erneut stundenlang berechnen werden muss. Ein intuitives Arbeiten wird erheblich erschwert. Aus diesem Grund werden diese Arbeitsschritte in ein Nachbearbeitungsprogramm verlagert, welches die einmalig erzeugten Bilder auf 2D-Basis weiterverarbeitet. Auf 2D-Basis lassen sich Änderungen an der Farbgebung in wenigen Millisekunden berechnen. Das Bild wird nicht mehr erneut auf 3D-Basis generiert.

Dieser Prozess (Bilderzeugung und anschließende Nachbearbeitung) ist seit den Anfängen der computerunterstützten Filmerzeugung durchgesetzter Standard. Schaut man sich die Arbeitsabläufe in dieser Verarbeitungskette genau an, stellt man fest, dass diese an eini-

[3] siehe Grundlagenkapitel, Seite 19

[4] Quelle: http://www.digitalartsonline.co.uk/features/creative-lifestyle/weta-rendering-lord-of-rings/, abgerufen am 30.11.2012

gen Stellen noch optimiert werden können. Bis man eine Vorschau auf das finale Ergebnis einer Sequenz erhalten kann, muss das soeben gerenderte Bildmaterial zuerst in die zweite Nachbearbeitungssoftware übertragen werden. Dort wird es dann gemäß den Vorgaben des Regisseurs angepasst. In der Regel gestaltet sich die Übertragung das Bildmaterials folgendermaßen: Da man beim Wechsel der Programme keine Qualitätsverluste in Kauf nehmen möchte, wird das Bildmaterial verlustfrei abgespeichert. Eine Sekunde Filmmaterial (25 Einzelbilder) werden in 25 einzelnen Bilddateien abgespeichert. Ein Bild nimmt bei einer Auflösung von 1280x720 Pixeln bereits über 2MB Festplattenspeicher in Anspruch. Eine Sekunde Film entspricht folglich 50MB an Speicher. Geht man von den Ansprüchen einer Kinoproduktion aus kommt man bei einer Auflösung von 4096x1714 Pixeln und 24 Bildern pro Sekunde auf 480MB[5] für eine Sekunde Film. Diese Summen von Speicherplatz werden ein zweites Mal benötigt, wenn das Filmmaterial aus dem Nachbearbeitungsprogramm in das Schnittprogramm - erneut verlustfrei - überführt wird.

Oft ist es für Entscheidungsprozesse erforderlich, möglichst schnell eine Vorabansicht (Preview) auf das spätere Werk zu erhalten, um sich Dinge, über die man spricht, besser vorstellen zu können. Die Auslagerung der Nachbearbeitung in ein zweites Programm verlangsamt den Prozess der Preview-Erstellung. Wenn sich in der 3D-Animation etwas ändert (z.B. Kameraeinstellung oder Inhalt), muss der Vorgang der Bildüberführung in das Nachbearbeitungsprogramm erneut durchgeführt werden. Die derzeitigen 3D-Programme haben keine Nachbearbeitungstools integriert und unterstellen allesamt den Erwerb einer Fremdsoftware zur weiteren Verarbeitung. Deshalb geht es in dieser Arbeit auch in einem ersten Schritt darum, ob die Integration technisch überhaupt realisiert werden kann. Die 3D-Renderingsysteme schaffen in der Regel keine oder nur sehr wenige Voraussetzungen, um Post Production betreiben zu können. Oftmals sind diese Systeme soweit isoliert, dass sie keinen Ansatzpunkt für Bildmanipulation auf 2D-Basis bereitstellen.

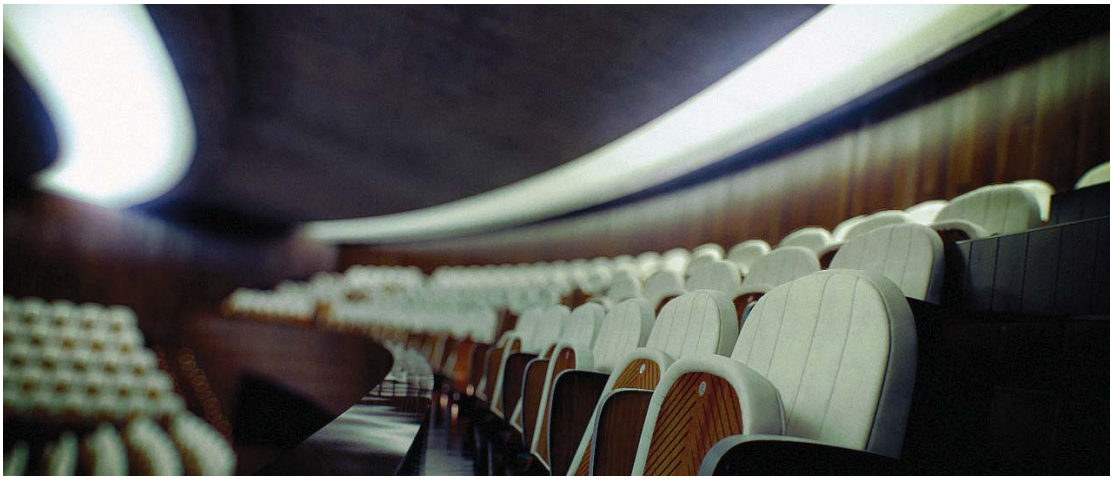


Abbildung 1.2: Das künstlerische Spiel mit einer geringen Tiefenschärfe, Autor: Alex Roman

[5] 480MB entsprechen ca. 74% der Größe einer handelsüblichen CD

Eine oft genutzte Verfahrensweise in Film- und Bildbearbeitung ist das künstlerische Spiel mit der geringen Tiefenschärfe, mit dessen Hilfe wichtige Elemente im Film von unwichtigen optisch getrennt werden (siehe Abbildung 1.2). Letztere werden zumeist unscharf außerhalb des Fokus abgebildet. Dies ist ein typischer visueller Effekt, welcher zumeist in der Nachbearbeitung auf 2D-Basis hinzugefügt wird. Die korrekte physikalische Berechnung des Effektes beim Rendern in der 3D-Umgebung verlangsamt den Renderprozess um ein Vielfaches.

Damit der Effekt in der Nachbearbeitung arbeiten kann, braucht er jedoch Tiefeninformationen über das Bild, welche nach dem Rendern, bzw. nach der Überführung, verloren gegangen sind, da man es ab diesem Zeitpunkt mit 2D-Bildern zu tun hat. Hier behilft man sich einem Trick, indem zu jedem Einzelbild ein zweites Bild abspeichert wird, welches Tiefeninformationen über das Ursprungsbild enthält. Ein sogenanntes Graustufenbild weist jedem Pixel eine Farbe zwischen Weiß und Schwarz zu. Je dunkler der Grauton eines Pixels ist, desto weiter befindet sich der korrespondierende Pixel im Originalbild von der Kamera entfernt. Auf diese Art und Weise ist es möglich, einem zweidimensionalen Bild eine dritte Dimension (Tiefe) zu verleihen.

Auch hier ist zu erkennen, dass sich der gesamte Arbeitsablauf verlangsamt, um diesen Effekt in der Nachbearbeitung hinzuzufügen. Die meisten aktuellen 3D-Programme bieten eine zeitsparende Tiefenschärfe-Simulation auf 2D-Basis nicht an. Hier sind während des Renderns noch alle 3D-Informationen vorhanden, aus welchen ein 2D-Algorithmus ohne Umwege seine Informationen beziehen könnte. Der Tiefenschärfe-Effekt ist nur einer von vielen weiteren Effekten, welche sich in der Nachbearbeitung auf 2D-Basis schneller berechnen lässt, jedoch noch 3D-Informationen für seine Kalkulation braucht.

Nicht immer werden 3D-Animationen einer Nachbearbeitung unterzogen. Im Ausbildungsbereich liegt der Fokus oft auf der reinen Erstellung von 3D-Animationen. Die 3D-Softwarepakete sind mittlerweile so vielfältig geworden, dass man alleine mit dem Erlernen dieser Programme Jahre verbringen kann. Oft wird der Schwerpunkt auf die Erstellung bzw. Animation von virtuellen Objekten gelegt. Meistens hat man dann keine Zeit mehr, sich in ein weiteres Nachbearbeitungsprogramm einzulernen oder es besteht auch gar kein Interesse daran. Nachbearbeitungsprogramme wie z.B. Adobe After Effects sind mittlerweile so vielfältig und komplex geworden, dass hierfür bereits eigene Lehrveranstaltungen existieren.

Hier zeigt sich ein interessanter Optimierungsansatz für 3D-Programme, welche komplett auf integrierte Nachbearbeitungstools verzichten. Mit z.B. der Funktionalität einer Nachbearbeitungsoption, wie z.B. der Farbkorrektur oder des zuvor erwähnte Tiefenschärfe-Effekts, könnten die Animationen der Auszubildenden sehr schnell größere visuelle Qualität annehmen, ohne eine aufwendige Nachbearbeitung nutzen zu müssen.

1.3 Zielsetzung

Für die verschiedenen Aspekte, welche in der vorigen Problemstellung erläutert wurden, sollen in dieser Arbeit Konzepte entwickelt und realisiert werden, um die Arbeitsabläufe während einer 3D-Filmproduktion zu optimieren. Es soll die Frage geklärt werden, inwiefern man auf die Auslagerung der Nachbearbeitung in eine Zweitsoftware verzichten kann. Dafür wird ein Prototyp entwickelt, welcher Aspekte der 2D-Nachbearbeitung direkt in die 3D-Software implementiert.

Das 3D-Programm Autodesk Maya stellt hierzu einen idealen Ansatzpunkt dar, da es dank einer großen Entwicklungsumgebung sehr einfach zu erweitern ist. Es gewährt den programmatischen Einfluss auf den Renderprozess. Dadurch kann man an den dreidimensionalen Renderprozess zweidimensionale Nachbearbeitungsprozesse anhängen und somit eine Nachbearbeitungsumgebung innerhalb des 3D-Programmes erschaffen.

Dieser Ansatz beherbergt viele Vorteile: Er bietet schnell einen realistischen Preview auf das Endprodukt, ohne aufwendige Überführung in ein Zweitprogramm. Weiterhin kann der notwendige Speicherbedarf drastisch reduziert werden, der bei einer Nutzung eines Zweitprogrammes anfallen würde. Effekte, wie die Tiefenschärfe, lassen sich intuitiv bedienen, da man sich mit der Rekonstruktion der Tiefeninformationen nicht mehr beschäftigen muss. Besonders der Fokus der Kamera lässt sich auf diesem Weg direkt im virtuellen Raum des 3D-Programmes platzieren. Generell stehen alle 3D-Informationen für etwaige 2D-Filter nativ in der Umgebung zur Verfügung.

Dennoch ist es ein Ziel dieser Arbeit, die Grundprinzipien der Nachbearbeitung nicht außer Acht zu lassen. Veränderungen an den Effekten (z.B. Farbänderungen) sollen sofort sichtbar werden, ohne das Bild auf 3D-Basis erneut rendern zu müssen.

In einem letzten Schritt wird evaluiert, ob die Konzepte bei dem betroffenen Anwenderkreis auf Akzeptanz trifft. Mittels Umfragen, gerichtet an verschiedene Interessengruppen, soll eine möglichst ausdifferenzierte Auswertung ermöglicht werden. Hierfür werden nicht nur Personen aus einem professionellem Umfeld befragt, dessen Interesse ein möglichst effizienter Arbeitsablauf ist, sondern auch Personen, welche sich in der Ausbildung befinden und Nachbearbeitung ein aus Zeitgründen eher nebenläufiges Thema ist. Als dritte Gruppe wird sich auch mit der Meinung von Hobbyentwicklern auseinandergesetzt. Welchen Vorteil sehen diese Personen in der Realisierung der Konzepte?

Wichtig ist auch die Frage, ob auf Nachbearbeitungsprogramme komplett verzichtet werden kann oder ob es Sinn macht, nur spezielle 2D-Effekte in den Renderprozess zu integrieren. Welche Effekte wären dies dann? Generell stellt sich die Frage nach den meist benutzten Effekten. Welche Effekte müsste man anbieten, um einen produktiven Einsatz in der Industrie überhaupt realisieren zu können? Lässt sich die Entwicklung zum Generieren von Previews nutzen? Ziel dieser Arbeit ist es, eine Antwort auf diese Fragen zu erlangen. Dazu sollen zum einen quantitative Messwerte über mögliche Beschleunigungen der Ar-

beitsprozesse aufgezeigt werden und zum anderen qualitative Aussagen aus den Umfragen heraus gearbeitet werden.

1.4 Aufbau dieser Arbeit

Diese Arbeit ist in acht Bereiche untergliedert. Im ersten Kapitel wird der Leser an die Problematik herangeführt und mit ersten Lösungsansätzen und Zieldefinitionen vertraut gemacht. Auch befindet sich am Schluss des ersten Kapitels eine Zusammenfassung der gesamten Arbeit. Im zweiten Kapitel wird der derzeitige wissenschaftliche Standpunkt erläutert. Anhand der verwandten Arbeiten soll dem Leser die Problematik dieser Arbeit vertraut gemacht werden. Im dritten Kapitel wird der Leser, welcher mit der Thematik der 3D-Erstellung nicht vertraut ist, mit den grundlegenden Themen bekannt gemacht, um die Arbeit im Ganzen verstehen zu können. Kapitel 4 beschreibt die Konzepte dieser Arbeit, mit dessen folgender Implementierung die aufgezeigten Fragen beantwortet werden sollen. Kapitel 5 und 6 beschreibt den methodischen Teil dieser Arbeit. Hier wird auf die Programmierung des Post Production Frameworks im Detail eingegangen. Die Architektur wird aufgezeigt sowie die Implementierung der wichtigsten 2D-Filter. Kapitel 7 stellt die subjektiven und objektiven Messwerte vor und trägt die Ergebnisse zusammen. Kapitel 8 und 9 beinhalten eine Zusammenfassung der Arbeit und zeigen einen Ausblick auf zukünftige Weiterentwicklungen.

1.5 Zusammenfassung

In dieser Arbeit konnte ein Weg aufgezeigt werden, wie sich ein Aspekt der Post Production in Maya integrieren lässt. Dabei können 2D-Filter in den Renderprozess integriert und in Mayas Renderview visuell eingestellt werden. Der 3D-Renderer Mental Ray bietet hierzu erste Vorkehrungen an. In Kombination mit der Outputshader-Architektur von Mental Ray und einem selbst entwickelten 2D-Renderer erhält man Zugriff auf den Renderview von Maya. Damit erhält der Anwender eine sofortige Rückmeldung auf die Veränderung der Filtereinstellungen. Der 2D-Renderer greift dabei auf ein temporär abgespeichertes 3D-Rendering von Mental Ray zurück und wendet die Filter an. Mit diesem Konzept muss der komplette Rendervorgang nicht erneut angestoßen werden, wenn der Anwender die Filtereinstellungen ändert.

In dieser Arbeit wurden drei Post Production-Filter prototypisch implementiert. Zwei schnelle Farbkorrekturfilter sowie ein rechenintensiver Depth of Field-Filter wurden implementiert. Die grafische Benutzeroberfläche lehnt sich an das Aussehen von Adobe After Effects an.

Bei der Auswertung der Ergebnisse stellt sich heraus, dass die Anwender eine Zeitersparnis beim Arbeiten mit dem entwickelten Plugin erkennen. Dies konnten auch objektive

Messwerte belegen. In Abhängigkeit vom Grad der Professionalität sehen die Nutzer verschiedene Anwendungsbereiche. Personen, welche sich im Umfeld der Ausbildung befinden, können sich zu einem großen Teil vorstellen, das Plugin im vollen Umfang zu nutzen, wohingegen professionell agierende Personen den Nutzen des Plugins mehr im Bereich der Erzeugung von Previews sehen.

Kapitel 2

Verwandte Arbeiten

Diese Arbeit beschäftigt sich mit dem Thema der Integration von Post Production in eine 3D-Animationssoftware. Ziel dabei ist, den Renderprozess zu beschleunigen und dem Nutzer die Möglichkeit zu geben, seine Arbeit im Vorhinein zu previsualisieren. Arbeitsabläufe während der Erzeugung von 3D-Animationen sollen dabei optimiert werden.

In diesem Kapitel werden wissenschaftliche Arbeiten zu diesem Themenkomplex vorgestellt. Dabei fällt auf, dass viele Ideen aus dem Bereich der Spielebranche kommen. 3D-Spiele sind ein gutes Beispiel für zeitkritische Anwendungen, in welchen der Nutzer Echtzeit gerenderte 3D-Umgebungen präsentiert bekommen soll. Hierbei werden nach dem 3D-Rendering oftmals 2D Post Processing-Filter angehängt, um dem Spiel einen cineastischen Look zu verleihen. Die meisten Arbeiten in diesem Themengebiet beschäftigen sich mit dem Tiefenschärfe-Filter „Depth of Field“. Andere Post Processing-Anwendungsbereiche, wie z.B. die Farbkorrektur, erfahren in wissenschaftlichen Kreisen momentan weniger Aufmerksamkeit. Farbkorrektur-Algorithmen werde seit langem sehr effizient und schnell ausgeführt. Der Depth of Field-Effekt ist hingegen ein sehr rechenintensiver Filter. Daher wird versucht, ihn auch in zeitkritischen Anwendungen verfügbar zu machen.

Weitere Arbeiten beschäftigen sich mehr mit Gedanken, wie man eine schnellere Vorschau über die Produktionsinhalte bekommen kann, um kreative Prozesse früh genug anstoßen zu können. In dem zweiten Abschnitt dieses Kapitel, werden einige bereits auf dem Markt erhältliche Softwarelösungen vorgestellt, welche sich dem Thema Integration von Post Processing in den Renderprozess gewidmet haben.

2.1 Practical Post-Process Depth of Field

Earl Hammon [Ham07] entwickelte für die Firma Infinity Ward einen Post Processing-Algorithmus für das Spiel Call of Duty 4. Dabei handelt es sich um einen 2D Depth of Field-Filter (Dof), welcher an den 3D-Rendervorgang des Spiels angehängt wird. Er analysiert zuerst die Ausarbeitungen von Joe Demers [Dem04], welcher sich mit den verschiedenen

Implementierungen von Dof in Render-Engines beschäftigt hat. Beide kommen zu dem Ergebnis, dass eine schnelle Berechnung von Dof nur auf 2D-Basis möglich ist.

In der Realität ist der Grad der Tiefenschärfe von der Brennweite und der Größe der Blendenöffnung eines Objektives abhängig. Je größer die Brennweite und die Blendenöffnung in einer Kamera eingestellt ist, desto kürzer ist der Bereich in der Realität, der noch scharf abgebildet wird[1]. Die aktuellen Rendersysteme senden Lichtstrahlen aus einer unendlich kleinen Blendenöffnung (Perfect pinhole). Dadurch wird das gerenderte Bild in allen Bereichen perfekt scharf dargestellt. Um dem Bild nun einen realistischen Tiefenschärfe-Effekt hinzuzufügen, muss das Kameramodell der 3D-Rendersysteme erweitert werden. Bei der Simulation einer beliebigen Blendengröße müssen folglich mehrere Lichtstrahlen entlang der Fläche der Blende ausgesendet werden. Dies beansprucht eine enorme Verarbeitungszeit.

Bei der neuen Technik, die [Ham07] in seiner Arbeit vorstellt, werden die unscharfen Bereiche des Bildes nachträglich, basierend auf einer 2D-Tiefenmap, dem fertigen 3D-Rendering hinzugefügt. Die klassischen Ansätze zur Erzeugung von Dof, wie z.B. das Raytracing, sind für eine schnelle Verarbeitung zu langsam, wobei die Qualität der Unschärfe hier immer noch am Besten gesehen wird.

Bei 2D-Implementierungen existieren immer noch Probleme mit Situationen, in welchen Objekten nahe vor der Kamera im Unscharfen erscheinen. Dies sei aufgrund der schnellen Vearbeitungszeit jedoch hinnehmbar: „The measured performance cost was 1 to 1.5 milliseconds at 1024x768 on our tests with a Radeon X1900 and GeForce 7900 [Ham07].“

2.2 High quality previewing of shading and lighting for Killzone 3

Francesco Giordana [Gio11] hat eine Möglichkeit vorgestellt, mit welchen Mitteln seine Produktionsfirma die Arbeitsabläufe bei der Erstellung eines 3D-Spiels optimieren konnte. Um den Mitarbeitern während der Modellierung der Modelle eine realistische Vorschau zu geben, wie diese Modelle später im Spiel aussehen könnten, wurde die Spiele-Engine vollständig in Autodesk Maya integriert. Maya erlaubt dem Entwickler, die Renderpipeline des Viewports komplett auszutauschen. „Being a deferred renderer, we can preview any number of lights, with their respective real-time shadows, and also apply a full range of post-processing effects [Gio11].“

Auch hier sieht man, dass die Spielebranche Vorreiter im Bereich Post-Processing von 3D-Renderings geworden ist. Bei den klassischen 3D-Animationsprogrammen, wie z.B. bei Maya, findet man nur vereinzelt Ansätze dieser Überlegungen. [Gio11] hat im Grunde Post Processing in Maya integriert. Diese Techniken lassen sich jedoch nicht in einem normalen Maya-Rendervorgang nutzen, sondern stellen für die Echtzeitmodellierung ein Abbild der Spiele-Engine im Viewport dar.

[1] Eine ausführlichere Beschreibung zu dem Thema Tiefenschärfe befindet sich auf Seite 55

2.3 Techniken und Umgang mit Tiefenschärfe in 3D-Programmen

André Schaarschmidt [Sch08] beschäftigt sich in seiner Arbeit mit dem Umgang und der Previsualisierung von Tiefenschärfe-Effekten (Dof) in den den anerkannten 3D-Animationsprogrammen. Zu diesen zählt er 3ds Max und Maya.

In einer Umfrage bei drei größeren Produktionsfirmen (Pixomondo, 4k Animation und Bumbo Animation) stellt sich allerdings heraus, dass diese mit der Benutzerfreundlichkeit der Dof-Lösungen der Programme nicht zufrieden sein. „Die Unternehmen wünschen sich zur Verbesserung eine schnelle Vorschaufunktion, die Möglichkeit, im Darstellungsfenster den Fokuspunkt schnell kontrollieren zu können und eine Visualisierung der unterschiedlichen Schärfebereiche [Sch08]“.

Diesen Sachverhalt weiß er wie folgt zu erklären: „Die Kontrolle der Tiefenschärfe erfolgt nur über diese Werte und es gibt keine grafische Darstellung dafür, wo sich die Fokusebene befindet oder in welchem Schärfebereich ein Objekt liegt. Um festzustellen, ob sich die Fokusebene an der richtigen Stelle befindet und wie groß der Tiefenschärfebereich ist, müssen Testbilder gerendert werden [Sch08].“ Um eine grafische Visualisierung des Tiefenschärfebereichs zu ermöglichen, entwickelt er in seiner Arbeit ein Script für Maya, welches diesen Bereich grafisch im Viewport darstellt. Mit dieser Implementierung geht [Sch08] einen ersten Schritt in die richtige Richtung. Das Rendern von Testbildern aus dem alleinigen Grund, den richtigen Fokuspunkt zu finden, entfällt mit dieser Technik.

Das Setzen von Tiefenschärfe ist zumindest im Kinobereich jedoch ein sehr künstlerischer Prozess, welcher durch Ausprobieren und dem Spiel mit der Kamera geprägt ist. Unter dieser Annahme wird man sicherlich dennoch einige Testrenderings zu vollziehen haben, um den künstlerischen Prozess der Findung des finalen Looks abschließen zu können.

2.4 Translating 2D german expressionist woodcut artwork into 3D

Elona Muscha [Mus06] verwandelt in ihrer Arbeit Kunstwerke deutscher Expressionisten auf Basis des Holzschnitts in eine dreidimensionale Repräsentation. Dabei versucht sie die Stilelemente des Holzschnitts beizubehalten und gibt dieser Kunstrichtung einen neuwertigen Denkanstoß. In ihrer Arbeit verwendet [Mus06] Maya, um die Kunstwerke nachzumodellieren und den 3D-Renderer Mental Ray, um die Konturen im Rendering zu erzeugen. Dabei bedient sie sich dem Contour Outputshader aus Mental Ray, welcher nach dem 3D-Rendering in einem Post Processing-Schritt das eigentliche Bild mitsamt seinen Konturen erzeugt. Der Geometrie wird in einer Vorbereitungsphase ein Contourshader zugewiesen, welcher während des 3D-Renderings die Konturen der Objekte ermittelt. Die Sichtbarkeit oder auch Nichtsichtbarkeit der Konturen wird im Vorhinein durch eine Vielzahl von Parametern in Maya

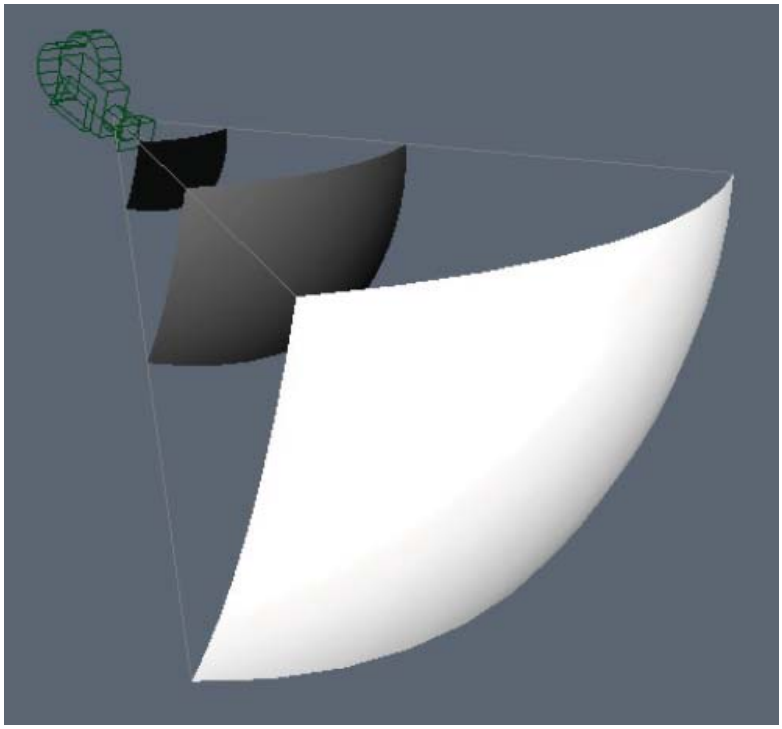


Abbildung 2.1: Visualisierung des Tiefenschärfebereichs durch ein Maya-Script, Abbildung entnommen aus [Sch08]

festgelegt. Die Informationen über die ermittelten Konturen werden nach dem 3D-Rendering in einer Datei gespeichert und dem Post Processing Contour Outputshader übergeben. Anhand dieser Datei wird dann das finale Bild auf 2D-Basis erstellt.

Diese wertvolle Datei mit den zwischengespeicherten Informationen über die Konturen wird nach Abschluss des Renderingvorgangs, bedingt durch die Architektur von Mental Ray, wieder verworfen. [Mus06] erwähnt diesen Sachverhalt nicht. Sie erklärt jedoch die Vielzahl der Filtereinstellungen, welche zwangsläufig mit einem erneuten kompletten Rendervorgang verbunden sind. Dabei wird der vorgeschaltete 3D-Rendervorgang unnötig wiederholt.

Mit dieser Arbeit wird ein kritischer Punkt dieser Vorgehensweise hervorgehoben. Positiver Weise ermöglicht Mental Ray durch seine „Production Shader Library“ (siehe [Men08]) die Beschleunigung des Renderprozesses durch Anhängen von 2D Processing-Filtern an das fertige 3D-Rendering. Er lässt den Nutzer aber dennoch immer komplett von Neuem rendern, anstatt auf ein fertiges Rendering aufzusetzen. Von diesem Standpunkt aus, man hat den einmalig notwendigen 3D-Renderprozess durchgeführt, wäre es vorteilhaft, die Anwendung der 2D-Filter in einen gesonderten Prozess auszulagern, so wie es klassische Post Production Anwendungen, wie z.B. After Effects heutzutage machen. Man könnte so die optimalen Einstellungen der Konturen sehr schnell ermitteln, ohne testweise neu rendern zu müssen.

Abbildung 2.2: Beispiel eines originalen Holzschnitts (links) mitsamt seiner 3D-generierten Repräsentation (rechts), Abbildung entnommen aus [Mus06]

2.5 Experiments in the Use of Game Technology for Pre-Visualization

Michael Nitsche [Nit08] stellt in seiner Arbeit Techniken vor, mit welchen sich Filmprojekte vor ihrer tatsächlichen Aufnahme previsualisieren lassen. Dabei werden Spiele-Engines verwendet, um z.B. Kamerafahrten im Vorhinein mit dem Regisseur und dem Cinematographer zu planen. „It allows a preparation of often very complicated shots that have to be clear not only to the director but also to the director of cinematography, the set designer, the lighting crew, and other members of the film team including the special effects and visual effects units. [Nit08]“ Dabei stellt er fest, dass Previsualierung heutzutage in vielen Filmproduktionen angewendet wird.

Für die Entwicklung dieser Masterarbeit sind die Aussagen von [Nit08] eine entscheidende Feststellung. Previsualisierung wird in der Filmindustrie gebraucht, um künstlerische Entscheidungen so weit wie möglich vorher treffen zu können. Auch wenn sie später verworfen werden und nicht im fertigen Film zu sehen sind, bieten sie für die künstlerischen Akteure einer Filmproduktion eine gute Vorschau über das Endprodukt und haben somit die Möglichkeit, Probleme frühzeitig zu erkennen und die gewonnenen Eindrücke in das Endresultat einfließen zu lassen. Wie bereits in der Zieldefinition dieser Arbeit beschrieben, dient die Integration eines Post Production Frameworks in eine 3D-Software nicht ausschließlich

dazu, den Renderprozess zu beschleunigen, sondern auch auf eine schnelle Art und Weise Eindrücke über ein mögliches finales Bild zu erhalten, auch wenn das previsualisierte Ergebnis im Endeffekt verworfen wird.

2.6 Verwandte Softwarelösungen

Heutzutage gibt es bereits Renderer und 3D-Animationspakete, welche Grundsätze der Post Production integriert haben. Zu diesen gehört der Renderer V-Ray, welcher auch für Maya verfügbar ist, sowie das kostenlose Animationspaket Blender.

2.6.1 V-Ray

V-Ray ist ein fotorealistischer Renderer der Firma Chaos Group, der im Jahre 2002 erstmalig veröffentlicht wurde. Ursprünglich für 3ds max entwickelt, kann der Renderer mittlerweile in viele weitere 3D-Animationspakete (Maya, Cinema 4D, Softimage,...) integriert werden. In Maya bietet V-Ray von Haus aus die Möglichkeit, das fertige Rendering einer 2D-Nachbearbeitung zu unterziehen. Dabei wird der Maya Renderview umgangen und ein V-Ray eigener Renderview (V-Ray Framebuffer, VFB) erzeugt, an welchen eine Palette an Helligkeitskorrekturwerkzeugen angedockt werden kann. Zu diesen Werkzeugen gehören Kontrolloptionen für die Belichtung, die Tonwerte und die Möglichkeit, die Graditionskurve des Bildes zu manipulieren. Diese Manipulationen beziehen sich auf die Helligkeitseigenschaften des Bildes und werden auf allen drei Farbkanälen gleichzeitig angewandt. Damit lässt sich ein Bild in seiner Helligkeit optimieren, eine echte Farbmanipulation ist jedoch nicht möglich.

Die Firma Chaos Group bietet in Ihrer Produktpalette jedoch noch ein weiteres Programm an, welches dazu genutzt werden kann, das fertig gerenderte Bild mit Hilfe von mehreren Werkzeugen umfassend zu optimieren. Der sogenannte Pdplayer ist eine eigenständige Software, welche als Betrachter von Bildsequenzen genutzt wird. Animationen werden meistens als Einzelbildsequenz heraus gerendert. Jeder Frame entspricht einer eigenen Bilddatei auf der Festplatte. Mit Hilfe dieses Players werden die Bilddateien in den Arbeitsspeicher geschrieben, um somit ein flüssiges Abspielen für Vorschauzwecke zu ermöglichen. Die Bilddateien müssen nicht erst in eine Filmdatei kodiert werden, um die Animation zu betrachten.

Neben der Eigenschaft als Betrachter von einzelnen Bilddateien wurden der Software viele Post Production Aspekte verliehen. Es existiert eine breite Palette an 2D-Filtern (Farbkorrekturfilter), sowie die Möglichkeit mehrere importierte Animationen in Ebenen zu gruppieren. Diese Ebenen lassen sich auch mit einander Verrechnen (Blending Modes), wie man es aus gängiger Post Production Software kennt. Aus V-Rays Framebuffer heraus lässt sich das fertig gerenderte Bild in den Pdplayer übertragen und dort weiterverarbeiten.

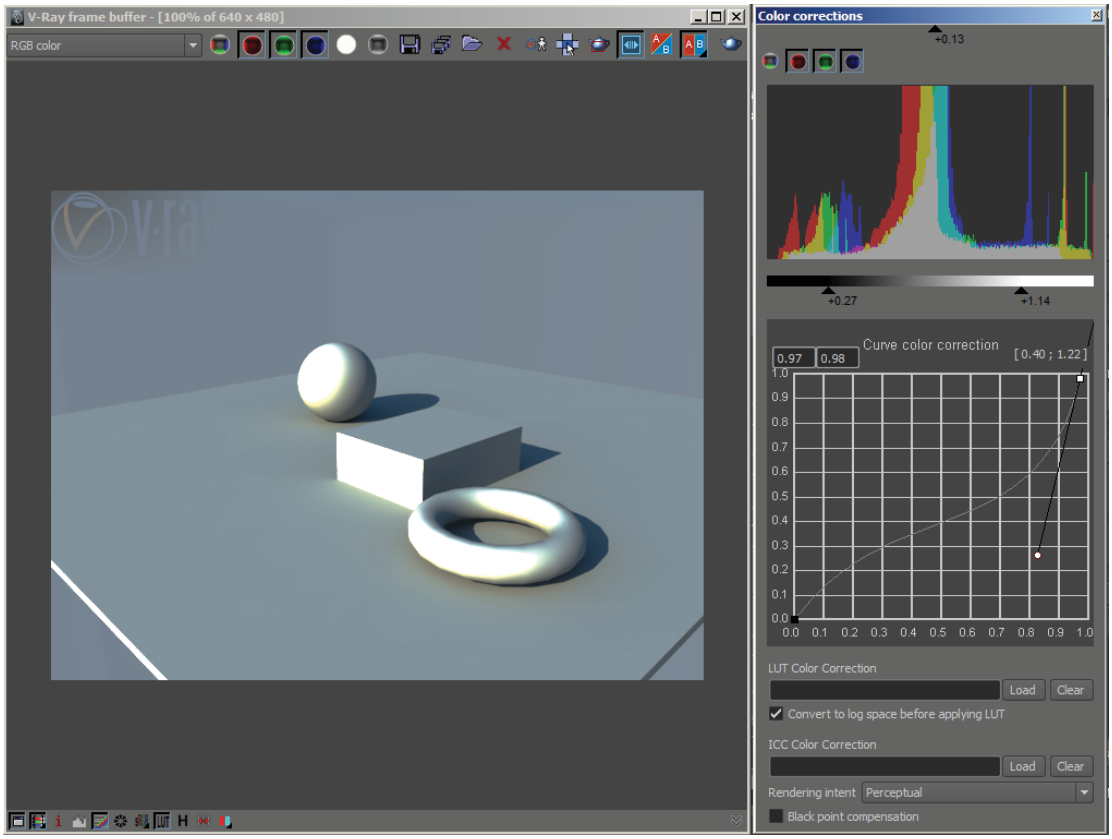


Abbildung 2.3: V-Ray Framebuffer mit Bildkorrekturoptionen

Ein Vorteil in dieser Kombination zeigt sich, wenn man in V-Ray verschiedene Renderebenen (z.B. die Separation von Farben und Schatten im Bild) aktiviert hat und das Ergebnis schließlich in den Pdplayer überträgt. Dabei werden die verschiedenen Ebenen sofort integriert und korrekt miteinander verrechnet, womit man dann die Vorzüge der Post Production, die Schatten separat vom Rest des Bildes in Ihrer Helligkeit zu optimieren, nutzen kann.

Ein Nachteil beider Lösungen, VFB und Pdplayer, zeigt sich beim Rendern von ganzen Animationen (Batch Rendering). Die zuvor eingestellten Bildoptimierungen werden nicht in den Renderprozess integriert. Nach dem Rendern einer ganzen Animation in viele Einzelbilder müssen diese entweder wieder im Pdplayer erneut optimiert und abgespeichert werden oder man überträgt das Rendering anschließend in eine vollwertige Post Production Software wie zum Beispiel Adobe After Effects. Beide Lösungen haben auch den zuvor angesprochenen Depth of Field-Effekt nicht integriert. Bei der Verwendung des (kostenpflichtigen) Pdplayers verlässt man zudem die Arbeitsumgebung von Maya. Der Nutzer muss sich mit einer weiteren grafischen Oberfläche vertraut machen. Diese wurde nicht in Anlehnung an die Oberfläche von Maya konzipiert, dies ist jedoch auch nicht als das Ziel einer eigenständigen Software anzusehen.

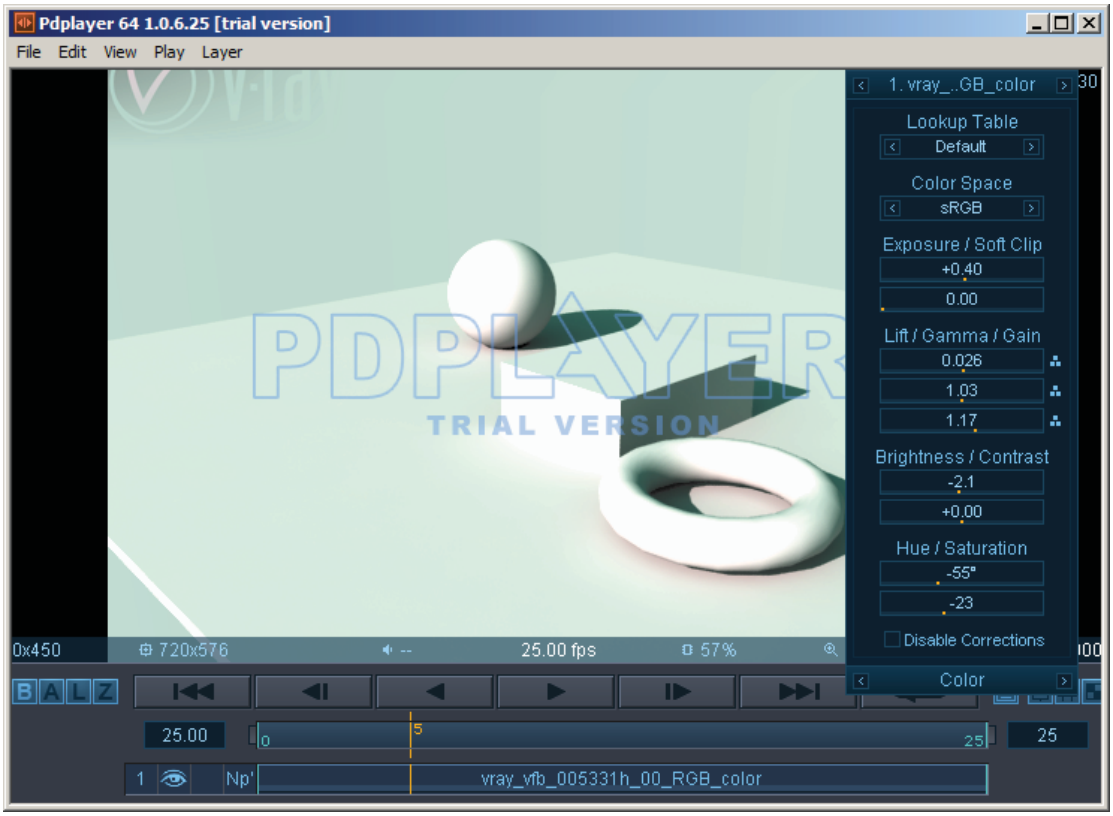


Abbildung 2.4: Post Production mit dem Pdplayer

2.6.2 Blender

Blender ist ein 3D-Programm, welches 1994 von der Firma NeoGeo entwickelt und im Jahr 2002 von der Blender Foundation als freie Software unter GPL-Lizenz veröffentlicht wurde. Seitdem wird es von einer großen Community weiterentwickelt und erfährt einen immer größere werdenden Funktionsumfang. Neben einer integrierten Spieleengine beinhaltet Blender auch einen Videoschnitteditor sowie eine Post Production-Umgebung. In dieser steht ein großer Funktionsumfang von 2D-Filtern zur Verfügung, welche bei Bedarf auch direkt in das Rendering integriert werden können.

Der Post Production-Editor von Blender ist Node-basiert. Generell unterscheiden sich Post Production-Programme in ihrem Erscheinungsbild zwischen den Node-basierten und den Ebenen-basierten. Ebenen-basiert ist z.B. Adobe After Effects und Node-basiert Nuke von The Foundry. Wie in Abbildung 2.5 (oberer Teil) zu sehen ist, nimmt jeder 2D-Filter die Form einer Node (Knotenpunkt) an. Jede Node hat Eingangs- und Ausgangsparameter. Als Eingangsparameter wird in der Regel das zu verarbeitende Bild gewählt und der Ausgangsparameter ist das durch den Filter verarbeitete Ergebnis, welches wiederum an den Eingang einer zweiten Filter-Node verbunden oder auch als Renderergebnis festgelegt werden kann.

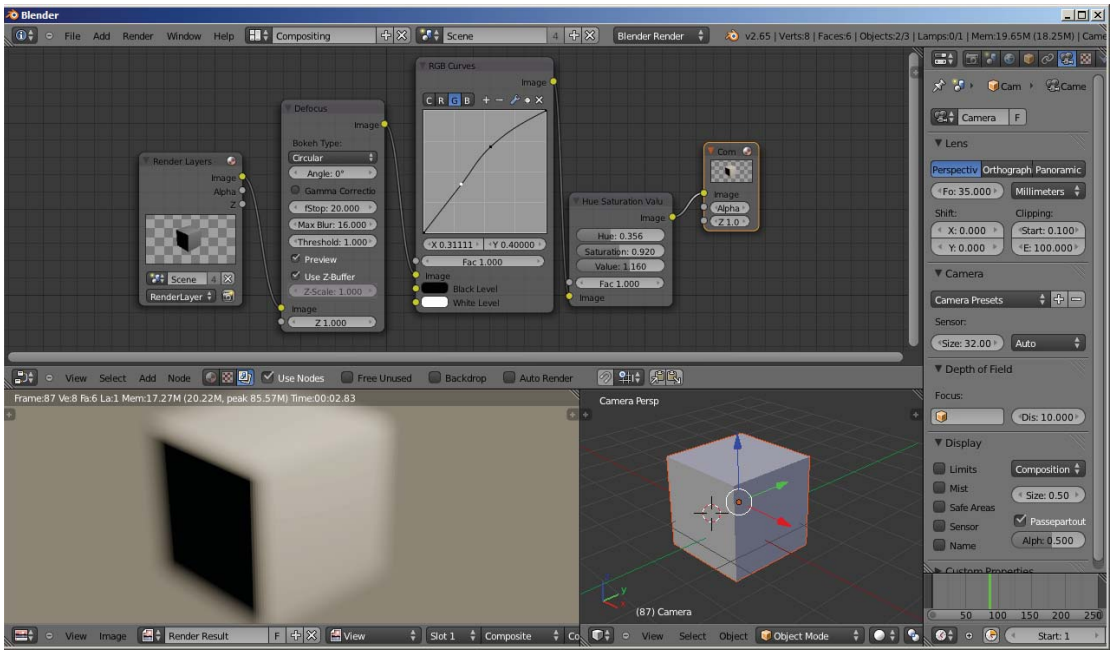


Abbildung 2.5: Post Production in Blender

Werden die Filter komplexer, so kann eine Node auch mehrere Ein- und Ausgangsparameter aufweisen, welche händisch miteinander verbunden werden müssen.

Diese System verleiht dem Nutzer eine große Flexibilität in seiner Arbeit, macht das Verschaffen eines Überblicks bei Verwendung vieler Nodes jedoch auch komplexer, da man sich weiterhin in einem großen Netz von verbundener Nodes zurechtfinden muss. Im Vergleich hierzu stehen die anzuwendenden Filter in Adobe After Effects in einer Liste untereinander, gemäß der Reihenfolge ihrer Abarbeitung. Zudem muss man sich hierbei nicht mehr um die Verbindung der einzelnen Filter kümmern. Node- und Ebenen-basierte Post Production Systeme sind beides oft verwendete und durchgesetzte Post Production-Paradigmen.

Kapitel 3

Grundlagen

In diesem Kapitel soll der Leser noch einmal mit dem Begriff des Renderings und den Gründen der langen Renderzeit vertraut gemacht werden. Des Weiteren werden hier noch einmal die in der Masterarbeit verwendeten Softwareprodukte Maya und Mental Ray vorgestellt. Da in den methodischen Kapiteln der Arbeit oft Mayas Hypershade angesprochen wird, erfolgt hier auch eine kurze Erläuterung.

3.1 Rendering

Unter dem Stichwort Rendering versteht man die computerunterstützte Generierung eines Bildes anhand eines zugrundeliegenden Datenmodells. Im Folgenden wird zwischen dem 3D-Rendering und dem 2D-Rendering unterschieden. Die Form des vorliegenden Datenmodells ist für die Unterscheidung wichtig. Liegt das Datenmodell in Form eines dreidimensionalen Raums vor, so wird von 3D-Rendering gesprochen. Entsprechend liegt dem 2D-Rendering ein zweidimensionales Datenmodell vor.

In diesem Kapitel soll dem Leser ein kurzer Eindruck vermittelt werden, aus welchen Gründen ein 3D-Rendering sehr lange brauchen kann, generiert zu werden. Im Vergleich dazu werden 2D-Renderings verhältnismäßig sehr schnell generiert. Da sich der Kern dieser Arbeit darauf stützt, dass 2D-Renderings wesentlich schneller abgeschlossen werden können als 3D-Renderings, werden einige Gründe hierfür nun dargelegt.

Die Pioniere [Kaj86] des 3D-Renderings haben im Jahr 1986 eine abstrakte allgemeingültige Formel entwickelt, aus welcher alle aktuellen Rendertechniken hergeleitet werden können. Rendertechniken wie das Raytracing (Strahlenverfolgung), gab es zwar schon vor

$$L_o(p,\omega_o) = L_e(p,\omega_o) + \int_\Omega \rho(p,\omega_i,\omega_o) L_i(p,\omega_i) \cos\theta \, d\omega_i$$

Veröffentlichung dieser Formel, sie bringt jedoch sämtliche Techniken auf einen Nenner. Die Formel beschreibt zusammengefasst, wie sich Lichtstrahlen in einem dreidimensionalem Raum an Oberflächen verhalten und prägte auch den Begriff der Energieerhaltung bei der Ausbreitung der Lichtstrahlen.

Die folgende Abbildung zeigt ein 3D-Rendering aus dem Jahr 1986, welches zwei Rendertechniken (Raytracing und Path Tracing) miteinander vereint. 3D-Rendering basiert zumeist

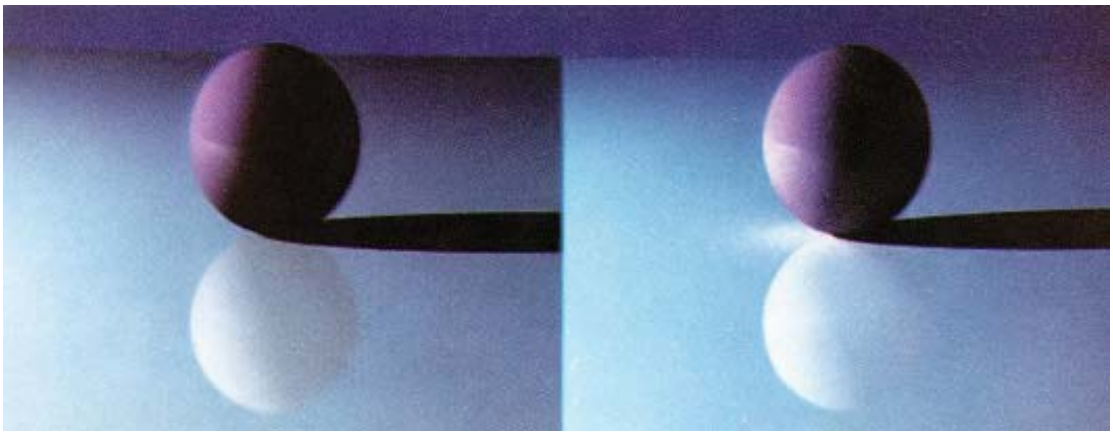


Abbildung 3.1: Das Zusammenspiel von Raytracing und Path Tracing, Abbildung entnommen aus [Kaj86]

auf dem (rekursiven) Aussenden von Lichtstrahlen in einen dreidimensionalen Raum. Dies ist einer der Punkte, der zu sehr rechenintensiven Prozessen führen kann.
Das 3D-Rendering des Künstlers Alex Roman (Abbildung 3.2) erzeugt ein Bild, das aus einem Datenmodell basierend auf der Verfolgung einer immens hohen Anzahl von Lichtstrahlen entstand. Daraus resultieren sehr hohe Rechenzeiten bzw. große notwendige Rechenressourcen.

3.2 Autodesk Maya

Autodesk Maya (im Folgenden Maya genannt) lässt sich als Industriestandard in der 3D-Filmproduktionsbranche bezeichnen. Hauptsächlich basierend auf der historisch sehr erfolgreichen Software PowerAnimator des ehemaligen Unternehmens Alias Systems Corporation wurden mit Hilfe dieses Modellierungs- und Animationstools bereits in früher Zeit 3D-Elemente für Filmproduktionen wie *The Abyss* (1989) und *Terminator 2* (1991) erstellt. PowerAnimator wurde bis zu dem Ende der 90er Jahre in vielzähligen Filmproduktionen verwendet. Als letztes großes Projekt wurde damit *Star Wars Episode I* (1999) realisiert. Im Jahre 1998 wurde PowerAnimator schließlich durch Maya schrittweise ersetzt bzw. weiterentwickelt. Zu diesem Zeitpunkt wurde Maya unter dem Firmennamen *Alias|Wavefront* vermarktet, nachdem Silicon Graphics im Jahre 1995 die beiden Firmen Alias und Wavefront

Abbildung 3.2: Fotorealistisches 3D-Rendering, Autor: Alex Roman

Technologies aufgekauft hatte. Im Jahre 2006 wurde die Firma von Autodesk übernommen, sie entwickelt Maya in seiner derzeitigen Form weiter.[1]

Maya stellt eine Vielzahl von Werkzeugen bereit, mit welchen der Nutzer 3D-Welten modellieren und animieren kann. Dazu gehören u.a. Modellierungtsools auf Basis von Polygonen, Nurbs, Curves und Surfaces. Effekte lassen sich mit einem eigenen Partikelsystem generieren, für Erstellung von Stoffen und Haaren gibt es wiederum eigene Werkzeuge. Die Oberflächen von Objekten lassen sich über eine Vielzahl von komplexen Shadernetzwerken darstellen.

Der Erfolg des Softwareproduktes Maya basiert maßgeblich auf seiner uneingeschränkten Erweiterbarkeit. Über eine C++-Programmierschnittstelle lassen sich neue Befehle hinzufügen, um den individuellen Anforderungen einer Produktion gerecht zu werden. Über die Skriptsprache MEL lässt sich die grafische Benutzeroberfläche erweitern. Mit MEL lassen sich zudem sämtliche Eigenschaften der 3D-Objekte programmatisch steuern und automatisieren.

Maya zeigt sich auch offen für die Verwendung von alternativen Rendersystemen. Die in diesem Kapitel genannten Filmprojekte wurden nicht mit dem Standardrenderer aus Maya generiert, sondern mit Pixars PhotoRealistic RenderMan, welcher separat erworben werden kann. Seit dem Jahr 2001 gehört der Renderer Mental Ray (siehe Kapitel 3.4) fest zum

[1] Weitere Referenzen im Anhang B.2, S.86

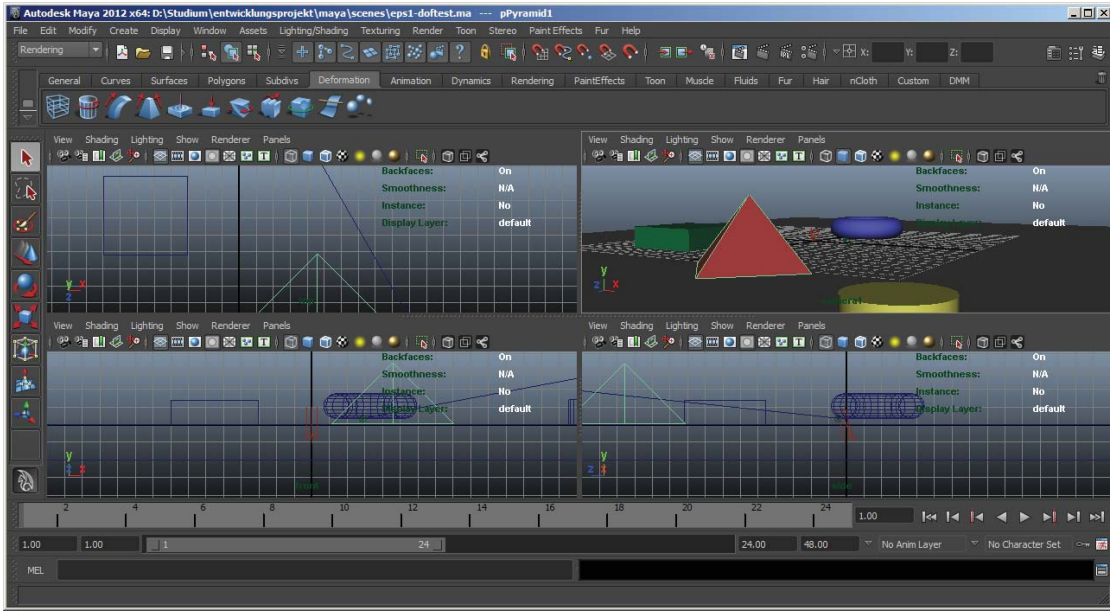


Abbildung 3.3: Die Benutzeroberfläche von Autodesk Maya

Produktumfang von Maya. Aus folgenden Gründen wurde Maya als Softwarebasis für diese Masterarbeit gewählt:

Industriestandard: Als durchgesetztes Softwareprodukt, kann die Entwicklung dieser Arbeit einem möglichst großen Nutzerkreis zugänglich gemacht werden.

Erweiterbarkeit: Die Erweiterbarkeit macht es überhaupt erst möglich, die Konzepte dieser Arbeit implementieren zu können.

Fehlende Post Production Aspekte: Im Vergleich zu anderen Softwareprodukten bietet Maya von Haus aus keine Möglichkeit, im Bereich der Post Production tätig zu werden. 3D-Software von anderen Herstellern sind schon weiter in diesen Bereich fort gedrungen.

Verwendung in der Lehre: Da Maya oft in der Lehre verwendet wird, erschließt sich hier eine weitere bedeutende Interessengruppe.

3.3 Hypershade

Mayas zugrunde liegendes Datenmodell, welches die 3D-Szene beschreibt, nennt sich *Dependency Graph*. Dabei stellt jedes Objekt ein Knotenpunkt (Node) im Rahmen eines großen Netz vieler Knotenpunkte dar. Jede Node weist mehrere Eigenschaften auf, welche über Ein- und Ausgänge (Slots) mit anderen Nodes in Verbindung treten kann.

Eine Sicht auf den Dependency Graph erlaubt u.a. das Tool mit dem Namen Hypergraph (siehe Abbildung 3.4). Über dieses Tool lassen sich Rückschlüsse ziehen, wie mehrere Objekte miteinander in Beziehung stehen. Eine Node kann z.B. ein 3D-Objekt in der Szene

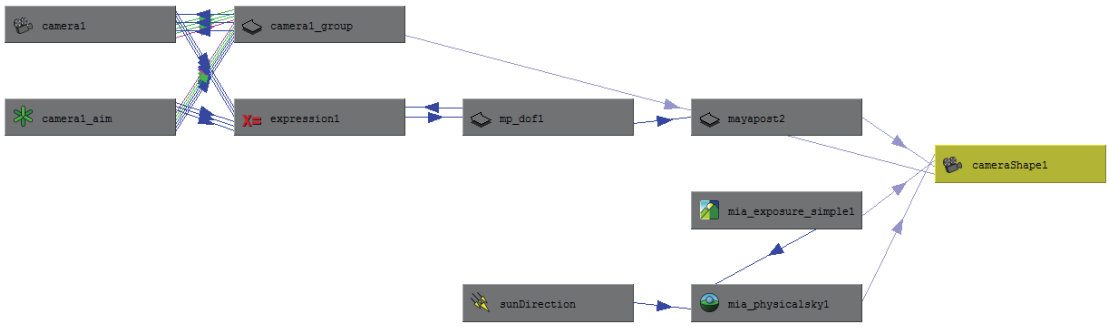


Abbildung 3.4: Mayas Hypergraph

sein, eine Kamera oder auch einfach nur ein Algorithmus zur Berechnung von Oberflächeneigenschaften (Shader).

Am Beispiel von Abbildung 3.5 sieht man zwei Nodes. Eine Node mit dem Namen *sunDirection*, welche die Richtung des Sonnenscheins in der 3D-Szene anhand von parallelen Vektoren visualisiert sowie eine Node, welchen einen Renderalgorithmus zur Berechnung eines physikalisch korrekten Himmels mitsamt einer Sonne darstellt. Diese Node braucht Informationen darüber, in welche Richtung die Sonne zu scheinen hat. Über eine Verbindung mit der *sunDirection*-Node erhält sie diese Information.

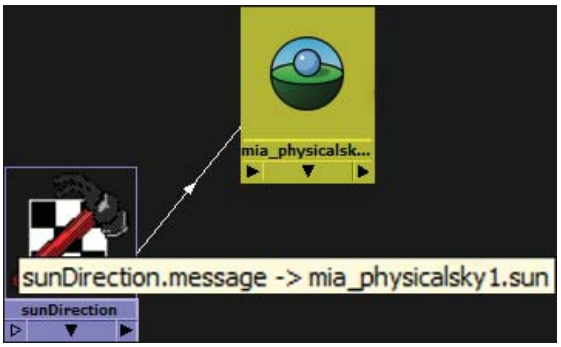


Abbildung 3.5: Verbindung zweier Slots im Dependency Graph

Ein weiteres Werkzeug, mit welchem sich der Dependency Graph überblicken lässt, ist der Hypershade (siehe Abbildung 3.6). Er beschränkt seine Anzeige jedoch auf die für das Rendering wichtigen Nodes. Dazu gehören hauptsächlich die Materialshader, Lichtobjekte und Kameras. Er bietet zudem die Möglichkeit, neue Nodes zu erstellen und daraufhin miteinander zu verbinden.

Die Maya C++-API sowie die Skriptsprache Mel enthalten beide die Möglichkeit, pro-

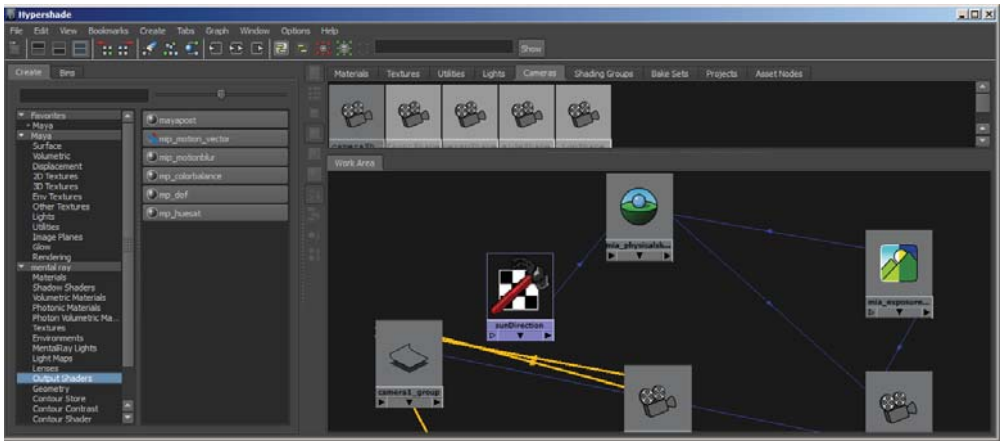


Abbildung 3.6: Maya Hypershade

grammatisch auf den Dependency Graph zuzugreifen. Somit ist es möglich, automatisiert Verbindungen zwischen zwei Slots zu erstellen und durch Iteration über den Graph die Abhängigkeiten der Objekte und ihre Eigenschaften auszulesen und zu manipulieren.

Der Hypershade ist während der Entwicklung dieser Arbeit ein wichtiges Werkzeug, um die Korrektheit der Programmierung zu beurteilen, bei welcher Nodes und Verbindungen automatisiert erzeugt werden.

3.4 Mental Ray

Mental Ray ist ein weiterer 3D-Render, welcher seit 1986 von der Mental Images GmbH entwickelt wurde und mittlerweile von Nvidia seit 2007 weiterentwickelt wird. Fester Bestandteil in Maya ist Mental Ray seit 2001. Entwickelt wurde Mental Ray als eigenständiger Renderer mit eigenem Dateiformat (Mi-Format). Dadurch ist er mittlerweile in weitere gängige 3D-Applikationen (3ds max, Softimage) eingebunden worden.

Rendert man in Maya ein Bild oder eine Animation, wird die Maya-Szene zuerst durch ein Maya-Plugin (mayatomr) in das Mental Ray eigene Format umgewandelt und dann verarbeitet.

Zu den unterstützten Rendertechnologien gehören globale und indirekte Beleuchtungsmodelle, Kaustiken, Final Gather sowie auch das klassische Raytracing. Auch Mental Ray besitzt den Vorteil, dass er sich durch eine gut dokumentierte C-API erweitern lässt. Hauptsächlich werden hierbei neue Materialshader programmiert, welche den Bedürfnissen des Nutzers entsprechen. Mental Ray wird oft in der Filmindustrie verwendet[2] und hat dadurch einen hohen Bekanntheitsgrad erreicht.

[2] Filmreferenz Mental Ray B.1, S.85

Abbildung 3.7: Beispiel Rendering, Urheber www.deltatracing.com

Kapitel 4

Konzeptentwicklung

Dieses Kapitel stellt die Konzepte vor, mit welchen sich die Fragen dieser Arbeit beantworten lassen. Vorrangig steht hier die Auswahl und Implementierung bestimmter Elemente der Post Production im Vordergrund, welche prototypisch in eine 3D-Applikation integriert werden. Durch diese Integration wird den Anwendern später ein Werkzeug zur Verfügung gestellt, mit dessen Hilfe es ihnen leicht fallen soll, einen Fragebogen auszufüllen.

4.1 Auswahl eines Post Production-Szenarios

Post Production-Software besteht immer aus einer Zusammensetzung verschiedener Tools, welche einem erlauben, sein Werk visuell zu optimieren. Dazu gehören unter anderem folgende Merkmale:

2D-Filter erlauben dem Anwender das Bild direkt zu manipulieren. Farbkorrektur, Schärfe- und Weichzeichner sind nur einige Beispiele, welche oft verwendet werden.

Ebenen tragen dazu bei, Bildteile von einander zu trennen und auch räumlich übereinander zu positionieren.

Animation ist ein Mittel, sämtliche Objekte und deren Eigenschaften über die Zeit hinweg zu verändern.

Text-Werkzeuge bieten die Möglichkeit, Text in den verschiedensten Ausprägungen auf dem Video zu platzieren.

Motion Tracking analysiert die Bewegung einzelner ausgewählter Motive in einem Video. Anhand dieser Bewegungsinformationen lassen sich z.B. künstliche Bildelemente in ein reales Video integrieren und erwecken den Anschein, als wären diese Elemente in der Realität schon gegeben.

Zeichnen lässt sich auch direkt auf dem Videomaterial. Sei es zu Retuschezwecken oder zur Erstellung von eigenen gemalten Bildelementen.

In dieser Arbeit wird das Hauptaugenmerk auf die Verwendung von 2D-Effekt-Filtern gelegt. Die Anwendung von 2D-Filtern ist ein naheliegendes Szenario, welches in jeder Produktion durchlaufen wird oder zumindest am Beispiel der Farbkorrektur durchlaufen werden sollte.

> *„The colorist working in film and video is the individual responsible for breathing life into characters, bringing a mood into a scene, and making the final product polished and professional-looking.“* [Hur10, Book Description]

Des Weiteren lässt sich ein weiterer, dieses mal rechenintensiverer 2D-Filter prototypisch implementieren, der sogenannte Tiefenschärfe-Filter (Depth of Field). Dieser trägt auch maßgebend zu einem professionellen Look dazu und wird häufig in Produktionen verwendet.

> *„Depth of field is frequently used in photography and cinematography to direct the viewer's attention within the scene, and to give a better sense of depth within a scene.“* [Dem04, S.375]

Es wird nun im Folgenden ein Szenario (Anwendungsfall 2D-Effekt-Filter) aus einer Post Production Software analysiert und versucht auf ein 3D-Programm abzubilden. Die Wahl der Softwareprodukte sieht folgendermaßen aus: Als 3D-Software wird Maya gewählt und als Post Production-Software wird Adobe After Effects als Beispiel herangezogen. Laut [HP10] sind dies zwei Programme, die sehr oft in Produktionen zusammen verwendet werden und perfekt miteinander harmonieren.

In der 2D-Filter Umgebung (Effects Workspace) von Adobe After Effects sind zunächst vier Bereiche im User Interface für die Anwendung der Filter wichtig. Die linke Bildhälfte

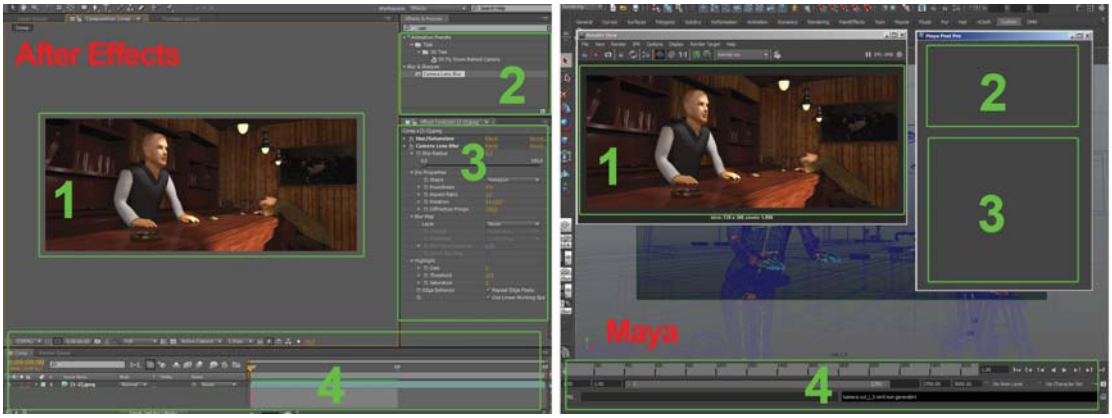


Abbildung 4.1: Merkmale von Post Production

von Abbildung 4.1 ist ein Screenshot aus After Effects. Bereich 1 stellt das zu bearbeitende Bild dar und aktualisiert sich automatisch, wenn der Benutzer 2D-Filter hinzufügt oder Änderungen an diesen vornimmt. Bereich 2 stellt eine Auswahlliste der verfügbaren Filter bereit. Hier lassen sich auch Filter dem derzeitigen Bild zuordnen und anwenden. Bereich 3 stellt die zurzeit ausgewählten Filter mitsamt ihren Einstellungsmöglichkeiten dar. Bereich 4 ist die sogenannte Zeitleiste. Mit ihr lässt sich das Video in seiner zeitlichen Dimension darstellen. Zusätzlich ist sie eine Hilfe, wenn zeitliche Änderungen an den Filtern vorgenommen werden. Man stelle sich hierbei ein scharfes Bild vor, welches innerhalb von zehn Sekunden seine Schärfe verliert. Mit der Zeitleiste lässt sich dieser Übergang einstellen und kontrollieren.

Diese 4 Elemente sollen nun in Maya übertragen werden. Die rechte Bildhälfte von Abbildung 4.1 auf Seite 28 ist ein grafischer Prototyp aus Maya, der die zuvor genannten 4 Elemente aus After Effects simuliert. Zu sehen ist hier in Bereich 1 der Render View aus Maya, welcher das Ergebnis eines Renderprozesses anzeigt. Dieser muss im Folgenden so umfunktioniert werden, dass er die Änderungen von 2D-Filtern darstellt, ohne zuvor den langwierigen 3D-Renderprozess zu durchlaufen (siehe Kapitel 4.2). Die Idee von Bereich 2 (Filterauswahl) und 3 (Filterbearbeitung) können direkt aus After Effects übernommen und in Maya implementiert werden. Mayas Zeitleiste aus Bereich 4 bedarf keiner weiteren Änderungen, da ihr dieselbe Grundidee wie die Zeitleiste aus After Effects zugrunde liegt.

Ein weiteres wichtiges Merkmal bei der Anwendung von 2D-Filtern besteht in der Möglichkeit, die Filtereinstellungen anhand von Keyframes über die Zeit verändern zu können. In der linken Bildhälfte von Abbildung 4.2 sieht man die Effekteinstellungen aus After Effects. Mit dem rot markierten Uhrensymbol lassen sich zu bestimmten Videozeiten (welche man mit der Zeitleiste auswählt) Effekteinstellungen speichern. Setzt man zu einem Zeitpunkt A das

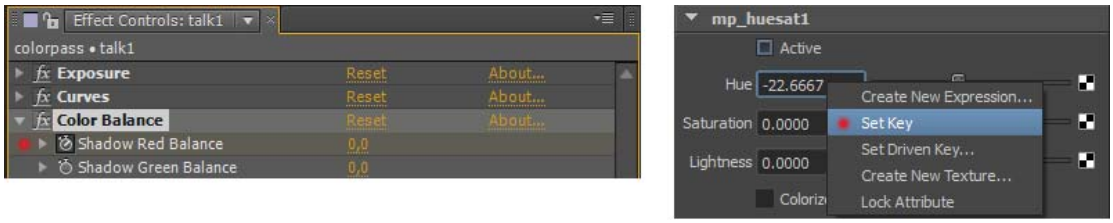


Abbildung 4.2: Keyframes in After Effects und Maya

Filterattribut „Red Balance“ auf 0 und zu einem Zeitpunkt B dasselbe Attribut auf 10, so errechnet der Filter für jedes einzelne Videobild zwischen Zeitpunkt A und B den korrekten Wert des Attributs, um einen fließenden Übergang von 0 auf 10 in dieser Zeitspanne zu ermöglichen. Diese Keyframe-Funktionalität lässt sich in Maya bei programmierten Anwendungen auch implementieren (vgl. Abbildung 4.2, rechte Bildhälfte).

Um auch dem erfahrenen Benutzer einen leichten Einstieg zu ermöglichen, wurde die grafischen Oberfläche von After Effects als Beispiel herangezogen. *„Indem er auf bekanntes Wissen zurückgreifen kann, um die Eigenschaft und das Verhalten des Systems zu begreifen,*

kann er sich die Analogie zunutze machen, um sich in der Anwendungswelt leicht und schnell zurecht zu finden.“ [Kri04, S.112]

4.2 Konzept zur Realisierung der Vorschaufunktion

Dieser Abschnitt stellt ein Konzept vor, mit welchem sich Änderungen an den 2D-Filtern direkt in der Renderausgabe (Renderview) von Maya anzeigen lassen. Abbildung 4.1 (S. 28) zeigt in der rechten Bildhälfte im Bereich 1 den Renderview von Maya. Eine elementare Eigenschaft der Post Production besteht in der Anwendung der 2D-Filter auf bereits abgeschlossene 3D-Renderings. Im zweidimensionalem Raum können 2D-Filter, wie zum Beispiel Farbkorrekturfilter, innerhalb von wenigen Millisekunden angwendet werden. Der Nutzer erhält sofortige Rückmeldung über seine zuvor angepassten Einstellungen. Die lange Wartezeit auf ein 3D-Rendering entfällt. 3D- und 2D-Rendering werden voneinander getrennt.

Diese Trennung muss nun auch in Maya vollzogen werden. Dafür bietet der 3D-Renderer Mental Ray bereits eine Vorkehrung. Zu seiner Shader-Sammlung gehört bereits der sogenannte *Outputshader*, welcher 2D-Verarbeitung an seinen 3D-Renderprozess anhängt. Dieser Vorgang wird in Abbildung 4.3 anhand des roten Pfades dargestellt. Nachdem der Nutzer den Rendervorgang startet, sei es ein Einzelbild-Rendering oder ein automatisiertes Rendering einer ganzen Animation (Batch Rendering), wird auf dieses nach dem lang andauernden 3D-Rendering ein 2D-Filter angewendet und schließlich ausgegeben (Renderausgabe). Im Fall des Batch Renderings wird das Ergebnis in eine Datei gespeichert und im Fall des Einzelbild-Renderings, aus Maya heraus gestartet, im Renderview grafisch angezeigt.

Wenn der Nutzer beim Einstellen der 2D-Filter seine Veränderungen sofort sehen möchte, muss der langwierige Prozess des 3D-Renderings (Mental Ray) umgangen werden. Folglich muss Mental Ray umgangen werden. Jedoch besitzen in Maya nur Renderer, wie z.B. Mental Ray, die Möglichkeit, ihre Ergebnisse in den Renderview zu platzieren. Deshalb muss ein eigenständiger Renderer programmiert werden, der nur 2D-Verarbeitung durchführt und sich an Mayas Konventionen hält, um sein Ergebnis im Renderview darstellen zu können. Dieser Renderer übernimmt dann auch die Verwaltung und Reihenfolge der vom Nutzer hinzugefügten 2D-Filter.

Wie zuvor erwähnt, operieren Post Production Programme auf bereits fertig gerenderten 3D-Bildern. Diese Bilder bzw. dieses Bild muss dem 2D-Renderer in Maya auch zur Verfügung gestellt werden. Er erhält es durch ein zuvor einmalig ausgeführtes Rendern durch Mental Ray. Dieser speichert das fertige Bild, bevor die 2D-Filter angewendet werden, auf der Festplatte in einem temporären Ordner ab. Der 2D-Renderer bedient sich vor seiner Ausführung dieser Bilddatei, wendet die 2D-Filter an und stellt das Resultat im Renderview dar. Dieser Ablauf ist in Abbildung 4.3 anhand der grünen Linie dargestellt.

Zuletzt bleibt noch die Frage zu klären, wann und wie der 2D-Renderer gestartet wird. Er wird immer dann gestartet, wenn der Nutzer sein Rendering mit Hilfe von 2D-Filtern

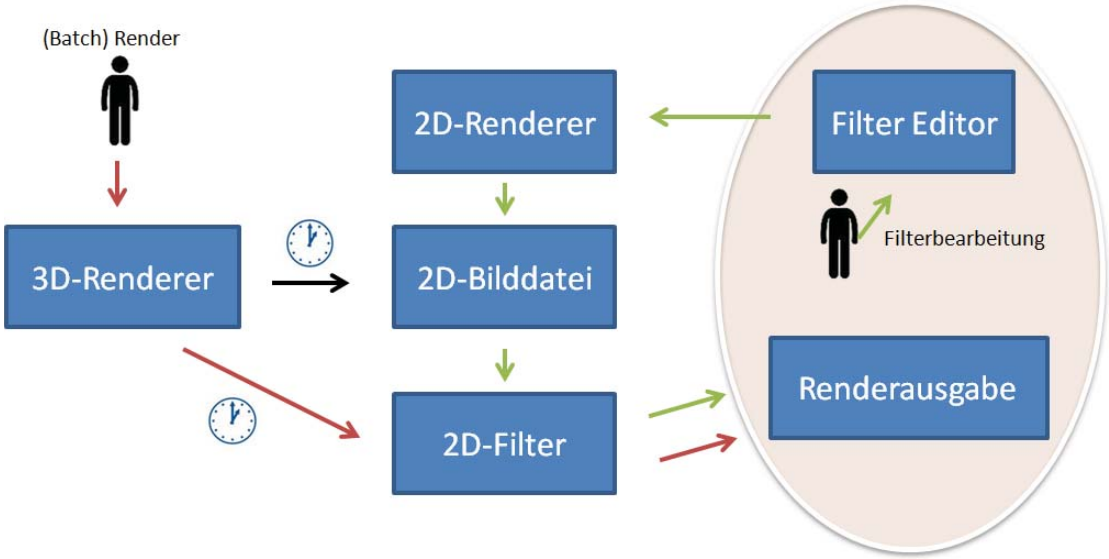


Abbildung 4.3: Renderpipeline

verbessern möchte. Zum einem wenn er Filter hinzufügt und zum anderen, wenn er Änderungen an diesen vornimmt. Beispiel: Fügt der Nutzer einen Farbkorrekturfilter hinzu und möchte zum Beispiel den Rotanteil des Bildes verstärken, erhält er hierzu einen Schieberegler, welcher die Rotverstärkung widerspiegelt. Zieht er diesen Regler auf die rechte Seite, so wird bei jeder Veränderung des Reglers der 2D-Renderer aufgerufen und das neue Bild im Renderview dargestellt. Der Aufruf des 2D-Renderers geschieht versteckt. Er muss keinen Renderknopf bewusst drücken.

Um dem Nutzer ein intuitives Arbeiten zu ermöglichen bleibt ihm die Architektur des 2D-Renderers verborgen. Er braucht auch nicht zu wissen, dass es einen 2D-Renderer gibt. Er soll sich auf seine Arbeit konzentrieren können und alleinig mit dem Filtereditor und dem Renderview arbeiten. Sein sichtbares Umfeld ist in dem Kreis in Abbildung 4.3 auf Seite 31 dargestellt.

Dieses Konzept wurde in einem Entwicklungsprojekt in Form eines *proof of concept* programmatisch implementiert. Damit konnte die Funktionalität des hier vorgestellten Konzepts bewiesen werden.

Kapitel 5

Implementierung des Post Production Frameworks

Dieses Kapitel beschäftigt sich mit der programmatischen Implementierung der im letzten Kapitel vorgestellten Konzepte. Das erste Unterkapitel gibt ein Überblick über die Gesamtarchitektur. In den weiteren Unterkapiteln werden die einzelnen Bausteine der Architektur detailliert erklärt und es wird auch aufgezeigt, weshalb die vorgestellten Techniken zur Lösung des Problems beigetragen haben.

5.1 Die Architektur im Überblick

Die Architektur besteht zusammengefasst aus drei Komponenten, welche mit Maya interagieren und in Form von Plugins ihre Anwendung finden. In Abbildung 5.1 erkennt man als erste Komponente einen Mental Ray Outputshader (Bereich 1), welcher nach dem 3D-Renderprozess wirkt und die 2D-Filter beinhaltet. Des Weiteren übernimmt er die Verwaltung der 2D-Filter. Hier ist hinterlegt, in welcher Reihenfolge die Filter im Renderingprozess abgearbeitet werden. Der Outputshader speichert vor dieser Verarbeitung auch das unangetastete 3D-Rendering in einer Tiff-Bilddatei unkomprimiert ab.

Diese Tiff-Datei verwendet dann der im Konzept genannte 2D-Renderer, um den Nutzer bei der Bearbeitung seines Bildes ein schnelles Resultat zu liefern, ohne den 3D-Rendervorgang erneut in Gang zu setzen. Der 2D-Renderer hat in der Abbildung (Bereich 2) und folgend auch in der Entwicklung den Namen PostPro erhalten. Bevor er implizit vom Nutzer durch Veränderung der Filtereinstellungen gestartet wird, liest er die 2D-Bilddatei, sammelt aus Maya heraus die vom Nutzer gewünschte Filterreihenfolge inklusive der Filtereinstellungen, wendet die Filter mit den Algorithmen des Mental Ray Outputshaders an und platziert das 2D-Renderergebnis in Mayas Renderview.

Als dritte Komponente (Bereich 3) befindet sich auf der Abbildung die Benutzeroberfläche (im folgenden GUI genannt), welche den Effekteditor widerspiegelt. Im Effekteditor

kann der Nutzer einer bestimmten Kamera die 2D-Filter zuweisen und diese konfigurieren. Zeitgleich aktualisiert sich fortlaufend der Renderview.

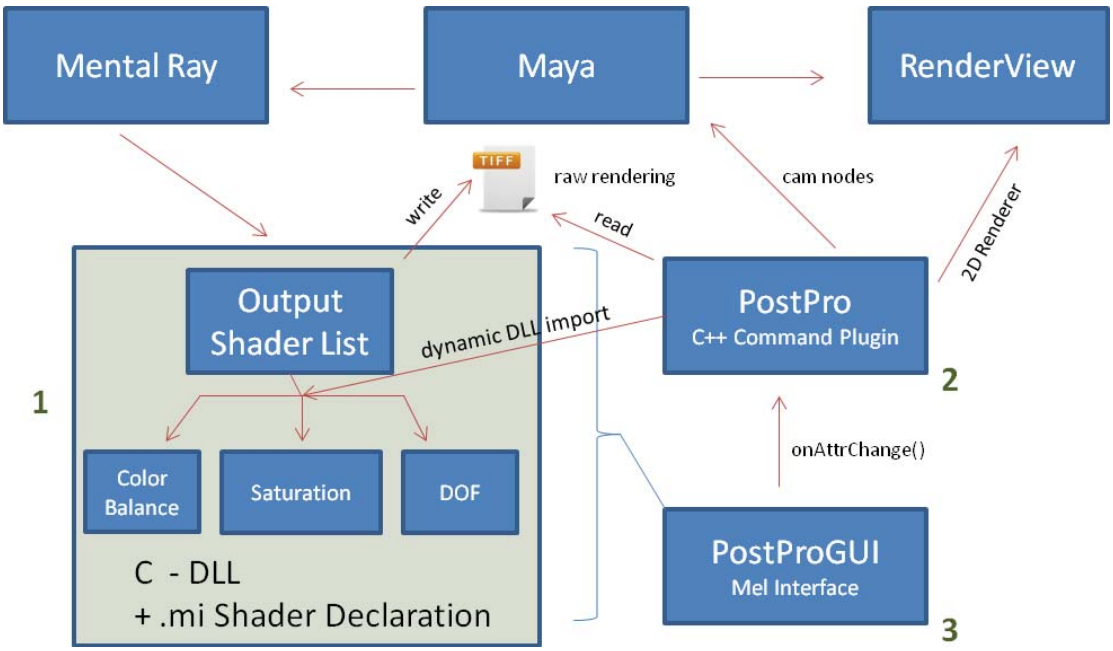


Abbildung 5.1: Ein Überblick über die Architektur

5.2 Outputshader

Der erste Schritt in der Entwicklung der Post Production Umgebung ist die Integration von 2D-Filtern in den regulären Renderprozess. Unter einem regulären Renderprozess ist das Batch Rendering zu verstehen oder auch der vom User ausgeführte einmalige Rendervorgang, welcher das Resultat im Renderview in Maya ausgibt. Die Funktionalität und Integration des 2D-Renderers wird in einem weiteren Unterkapitel behandelt.

Mental Ray stellt für die 2D-Verarbeitung die sogenannten Outputshader zur Verfügung. Wie in Abbildung 5.2 zu sehen ist, wird ein Outputshader nach dem 3D-Rendering Vorgang auf 2D-Basis angewandt. Neben den gerenderten Bildinformationen bekommt ein Outputs-

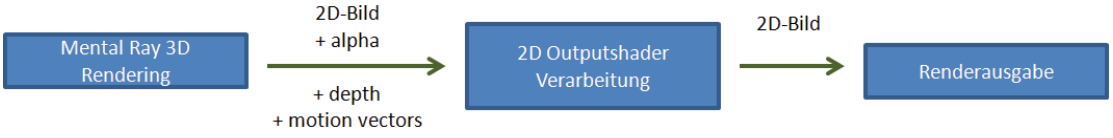


Abbildung 5.2: Mental Ray Renderpipline - Outputshader

hader weitere nützliche Informationen zur Verfügung gestellt. Zu diesen gehört u.a. eine 2D-Tiefenmap, welche jedem Pixel einen Tiefenwert in der 3D-Szenerie zuweist. Mit dieser Information lassen sich, trotz eines zweidimensionalen Bildes, Rückschlüsse auf die ursprünglichen 3D-Koordinaten eines Pixels ziehen. Dies ist z.B. für den Tiefenschärfe-Filter von Bedeutung. Des Weiteren stehen dem Outputshader Bewegungsvektoren jedes Pixels zur Verfügung, um einen realistischen Bewegungsunschärfe-Effekt dem Bild hinzufügen zu können.

Mental Ray stellt dem Nutzer nach einmaliger Aktivierung seine „Production Shader Library“ zur Nutzung bereit. Diese beinhaltet einige Outputshader, u.a. den zuvor angesprochenen Motionblur-Effekt. Mental Images hat hier den Geschwindigkeitsvorteil in der 2D-Verarbeitung erkannt und beschreibt seinen Filter passend mit den Worten: „The major advantage of this method is rendering speed. Scene or shader complexity has no impact.“ [Men08, S.5])

Es muss nun ein Weg gefunden werden, wie sich mehrere Outputshader beliebiger Reihenfolge an den 3D-Rendervorgang anhängen lassen. Auf den ersten Blick bietet sich hier der Outputshader-Dialog in den Kameraeinstellungen von Maya an (siehe Abbildung 5.3). Jedoch führt die Benutzung des Outputshader-Slots, selbst wenn man die originalen Men-

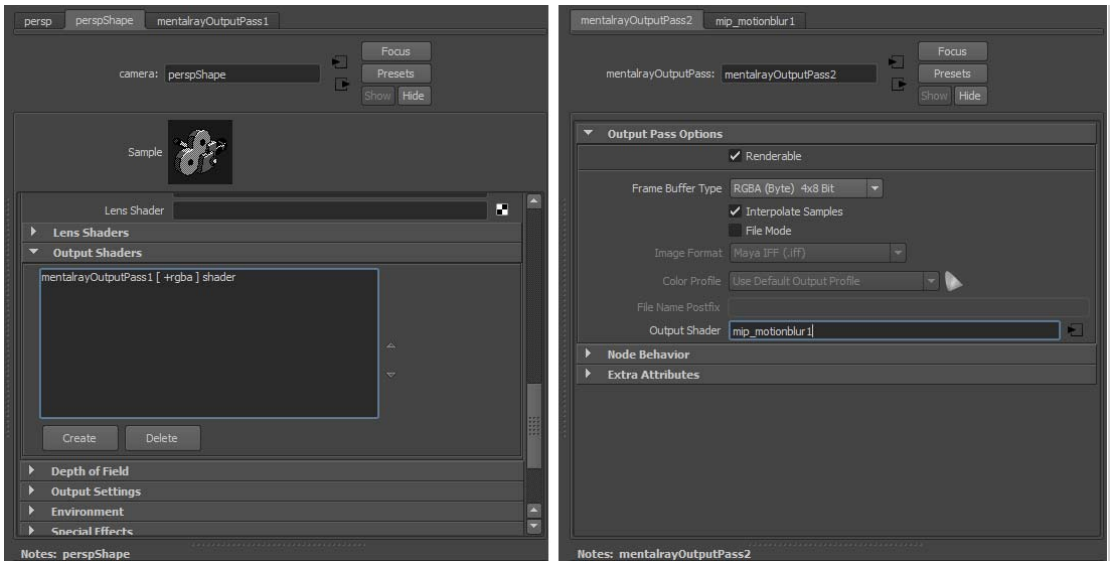


Abbildung 5.3: Konfiguration von Outputshadern

tal Ray Outputshader verwendet, beim Rendern zu einem sofortigen Absturz der gesamten Applikation. Den Grund für diesen Absturz findet man in der Mi-Datei. Mental Ray ist von Haus aus ein eigenständiger Renderer, welcher sein eigenes Dateiformat verwendet (.mi). Rendert man in Maya ein Bild heraus, wird die Maya-Szene zuerst in das Mental Ray Format umgewandelt und dann dem Renderer übergeben. Am Beispiel von Abbildung 5.3 wurde hier ein Motionblur-Filter mit dem Namen „mip_motioblur_1“ der Kamera hinzugefügt. In der mi-Datei findet man in den Kamerainformationen zwar den Verweis „output =

mip_motionblur1“, es fehlt jedoch die Deklaration des Shaders „mip_motionblur1“, welche hätte wie folgt aussehen müssen:

```
shader "mip_motionblur1"
  "mip_motionblur" (
    "shutter" 0.5,
    "shutter_falloff" 2.,
    "blur_environment" off,
    "calculation_gamma" 2.2,
    "pixel_threshold" 1.,
    "background_depth" 10000.,
    "depth_weighting" off,
    "blur_fb" "",
    "depth_fb" "",
    "motion_fb" "",
    "use_coverage" off
    )
```

Die Kamera verweist auf einen nicht vorhandenen Shader. Dies ist kein Fehler von Mental Ray, sondern ein Programmierfehler von Maya, welcher sich nun durch alle Softwareversionen hindurch gezogen hat und bei der Konvertierung der Szene in das Mental Ray Format die Deklaration der Outputshader auslässt. Autodesk hat zu diesem Problem noch keine Stellung bezogen, in den einschlägigen Internetforen wurde es jedoch bereits diskutiert.

Es existiert eine weitere Möglichkeit, *einen* Outputshader mit einer Kamera zu verknüpfen. Die Variante, welche in diversen Anleitungen für die „Production Shader Library“ angeführt wird, sieht eine manuelle Verknüpfung der Kamera mit dem Outputshader im Hypershade vor (siehe linke Bildhälfte Abbildung 5.4). Daraufhin wird in den Kameraeinstellungen ein verstecktes Menü „Legacy Output Shaders“ aktiviert - dieses ist in Abbildung 5.4 kenntlich gemacht. Mit dieser Technik lassen sich Outputshader fehlerfrei anwenden. Es

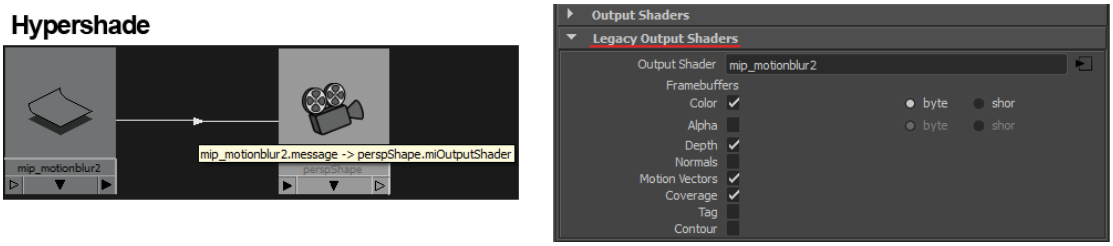


Abbildung 5.4: Hypershade - Legacy Output Shader

lässt sich hiermit jedoch nur *ein* Outputshader an den Renderprozess anhängen. Diese Entwicklung benötigt die Verwendung einer beliebigen Anzahl von Outputshadern sowie auch eine Möglichkeit die Reihenfolge der Abarbeitung festzulegen. Deshalb wurde ein Pseudo-Outputshader entwickelt, dessen einzige Aufgabe es ist, in sich mehrere Outputshader in einer Liste aufzunehmen und der Reihenfolge entsprechend abzuarbeiten. Wichtig hierbei ist, dass sich der neue Outputshader an die Konventionen von Maya und Mental Ray hält, um mit ihm im Hypershade als Node-basiertes Objekt arbeiten zu können.

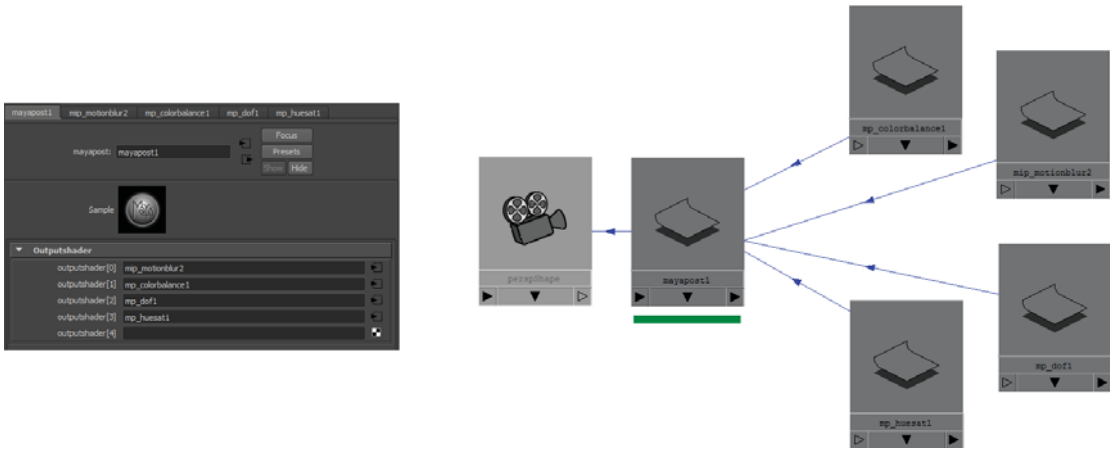


Abbildung 5.5: Hypershade - Outputshader - Verzweigung

Die programmatische Implementierung des im folgenden genannten Verzweigungsshaders wird im Anhang A.1 auf Seite 81 näher beschrieben. In Abbildung 5.5 sieht man auf der linken Seite die Eigenschaften des Verzweigungsshaders und die Möglichkeit, in ihn mehrere Outputshader aufzunehmen. Auf der rechten Bildhälfte sieht man das Node-basierte Netz aus Mayas Hypershade. Der Verzweigungsshader mit dem Namen „mayapost1“ wird nun mit der Kamera-Node verknüpft. Aus der Sicht der Kamera-Node wird nur *ein* Outputshader im Rendervorgang verarbeitet. Tatsächlich umhüllt der Verzweigungsshader eine beliebige Anzahl von Outputshadern. Damit wäre der Grundstein für die Weiterentwicklung des Post Production Frameworks gelegt.

5.3 Der 2D-Renderer

Wie bereits in der Konzeptentwicklung auf Seite 30 erwähnt, ist der 2D-Renderer für die schnelle Aktualisierung des Renderviews beim Ändern der 2D-Filtereinstellungen durch den Nutzer zuständig. Dabei bedient er sich einer einmalig zuvor temporär abgespeicherten 3D-Rendering-Ausgabe. Diese liegt unkomprimiert in einer Tiff-Datei vor. Nach Anwendung der 2D-Filter wird das Resultat im Renderview angezeigt.

Der 2D-Renderer muss durch einen Maya-Befehl aufrufbar sein. Die grafische Nutzeroberfläche, der Effekteditor, wird diesen Befehl später implizit aus einem Mel-Script heraus aufrufen. Um solch einen Befehl zu implementieren, wird man die C++-API von Maya (vgl. [Gou03, S. 308]) genutzt. Der Befehl wurde in dieser Entwicklung „Mayapost“ bezeichnet und erwartet genau einen Parameter „c“, welcher den Namen der zu rendernden Kamera beinhaltet. Der Befehl „Mayapost -c camera1“ verarbeitet dann die 2D-Filter, welche der Kamera mit dem Namen „camera1“ zugeordnet sind, und gibt das Resultat im Renderview aus.

Wie zuvor erwähnt, setzt der 2D-Renderer auf eine zuvor abgespeicherte 3D-Rendering-Ausgabe auf. Dieses soll nicht vom Nutzer selbst abgespeichert, sondern automatisiert abgespeichert werden. Das erfolgt zu einem Zeitpunkt, zu welchem noch keine 2D-Filter angewendet wurden. Dieser Zeitpunkt befindet sich im Verzweigungsshader, bevor die Liste der Outputshader durchlaufen wird. In Abbildung 5.1 (S. 34) wird dieser Sachverhalt durch den Pfeil „write Tiff„“ dargestellt. Zum Abspeichern wird eine Funktion aus der Mental Ray-API verwendet[1]. Hiermit wird nun auch begründet, weshalb es notwendig ist, vor der Arbeit mit den 2D-Filtern das Bild einmal vollständig gerendert zu haben. Erst nach diesem Zeitpunkt liegt die Tiff-Datei gespeichert auf dem Computer vor und kann vom 2D-Renderer verwendet werden. Ein erneutes vollständiges Rendern ist erst dann wieder notwendig, wenn der Nutzer entweder die zu bearbeitende Kamera wechselst, oder sich der grafische Inhalt der Kamera ändert. Dies kann durch die Bewegung der Kamera oder durch die Wahl eines anderen Zeitpunktes (Timeline - Objekte verändern durch Animation ihre Position) der Fall sein.

Die Tiff-Datei wird nun im temporären Ordner des Benutzerverzeichnisses von Windows abgelegt und danach vom 2D-Renderer in einem ersten Schritt in den Arbeitsspeicher eingelesen. Programmatisch liegt es in C++ als ein Struct-Array mit folgender Definition vor:

```
struct customImage{
    float r;
    float g;
    float b;
    float a;
};
```

In Form eines Arrays lassen sich später die 2D-Filter effizient durch Iteration über eine Schleife verarbeiten. Jedes Element in dem Array entspricht einem Pixel, welcher wiederum seine Farb- und Transparenzwerte RGBA kennt.

Im Folgenden muss dem 2D-Renderer vorgegeben werden, welche 2D-Filter, mitsamt ihren Parametern, in welcher Reihenfolge angewendet werden müssen. Voraussetzung ist hierbei der manuelle Aufbau des Outputshader-Netzwerk im Hypershade. Der automatisierte Aufbau durch den Effekteditor wird im nächsten Unterkapitel beschrieben. Unter dem händischen Aufbau des Outputshader-Netzwerkes ist zu verstehen, dass man den Verzweigungsshader im Hypershade mit der zu rendernden Kamera verbindet und die 2D-Filter-Outputshader mit dem Verzweigungsshader verbindet (vgl. Abbildung 5.6, S. 39). Mit Hilfe der Funktionen, welche durch die Maya-API gegeben sind, lässt sich durch den Kamera-Parameter herausfinden, welcher konkrete Verzweigungsshader mit ihr verbunden ist. Im Hypershade lassen sich die Verbindungen genauer analysieren. In Abbildung 5.6 sieht man im rot markierten Bereich 1 über welche beiden „Slots“ die Kamera-Node mit der Verzweigungsshader-Node verbunden ist. Der Slot „miOutputShader“ der Kamera ist

[1] mi_img_image_write() siehe: http://docs.autodesk.com/MENTALRAY/2012/ENU/mentalray3.9Help/files/integration/node44.html, Datum der Abfrage: 10.12.2012

mit dem Slot „message“ des Verzweigungsshaders (mayapost1) verbunden. Wird der Slot „miOutputShader“ mittels der Maya-API abgefragt, erhält man als Ergebniswert die passende Verzweigungsshader-Node zurück. Diese Node besitzt einen Slot mit dem Namen „outputshader[...]“. Dieser Slot ist ein Array, und beherbergt eine variable Anzahl von Slots, welche alle auf einen 2D-Filter-Outputshader verweisen. Über den Index des Arrays (Zählweise beginnt ab 0) ist die gewünschte Verarbeitungsreihenfolge der 2D-Filter festlegbar. Über diese Slots lassen sich auch die entsprechenden 2D-Filter-Nodes auslesen. Darüber werden alle Parameter, welche für einen 2D-Filter gewählt wurden, verfügbar.

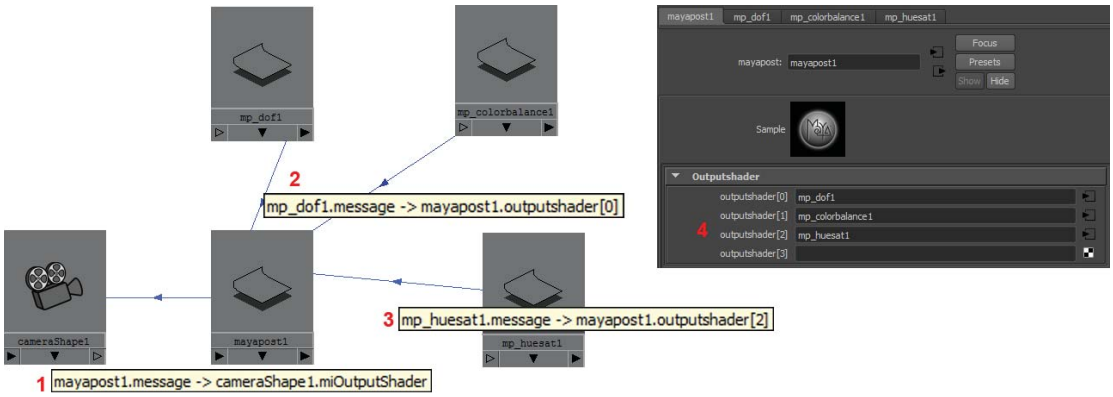


Abbildung 5.6: Outputshader verbunden im Hypershade

Zu diesem Zeitpunkt kennt der 2D-Renderer die zu bearbeitende Kamera, die Filter welche in einer gewählten Reihenfolge angewendet werden sollen und die vom Nutzer bestimmten Filterparameter. Zudem hat er das zu bearbeitende 3D-Rendering als 2D-Array im Arbeitsspeicher liegen. Dies sind die Voraussetzungen dafür, den 2D-Rendervorgang starten zu können.

Mit den nun zusammengetragenen Informationen kann der 2D-Renderer sein zwischengespeichertes Bild den 2D-Filteralgorithmen übergeben. Diese Algorithmen befinden sich jedoch in den Mental Ray Outputshadern in einer anderen kompilierten DLL-Datei. Diese werden von Mental Ray im Renderprozess aufgerufen, stehen jedoch noch in keiner Beziehung zum 2D-Renderer. Diese Beziehung wird über einen „Dynamic Link Library“-Import (DLL) hergestellt. Der 2D-Renderer erhält somit eine programmatische Referenz auf dieselben 2D-Filter-Funktionen, welche Mental Ray in seinem Renderingvorgang nutzt.

Es entsteht hierbei jedoch ein Problem mit den Konventionen der Funktionsparameter. Der 2D-Renderer benötigt als Übergabeparameter zuerst das originale 2D-Bild. Dieses wird im Folgenden im 2D-Filter verarbeitet. Die Auflösung des Bildes wird weiterhin mit den Werten der Bildbreite und Bildhöhe übergeben. Die Information, um welchen 2D-Filter es sich handelt mitsamt seinen Filterparametern, muss zuletzt auch übergeben werden. Die gewünschte Funktion mit den notwendigen Parametern sieht wie folgt aus:

```
//2D-Renderer (benötigte Parameter)
extern "C" DLLEXPORT void mp_colorbalance_impl(customImage *fb_color, int
    x_res, int y_res, struct mp_colorbalance *paras)
```

„fb_color“ steht hierbei für das 2D-Bild, „x_res“ und „y_res“ (Resolution) für die Breite und Höhe des Bildes und „paras“ beinhaltet die Filterparameter zu dem dazugehörigen Filter. Der Funktionsaufbau der tatsächlichen Mental Ray-Filterimplementierung sieht wie folgt aus:

```
//Mental Ray Funktion
extern "C" DLLEXPORT miBoolean mp_colorbalance(void *result, miState *
    state, struct mp_colorbalance *paras)
```

„Result“ ist ein Parameter, welcher „true“ oder „false“ zurückgibt, je nachdem ob der Algorithmus erfolgreich durchlaufen ist oder nicht. „State“ ist eine globale Variable, welche Informationen zum Renderprozess beinhaltet und „paras“ spiegelt auch hier die Informationen zu den Filtereinstellungen wider. Bei diesen beiden Funktionen passen die nötigen Übergabeparameter (Ausnahme: „paras“) nicht miteinander überein. In diesem Zustand kann der 2D-Renderer das 2D-Bild nicht verarbeiten.

Wieso wird in einem Mental Ray-Outputshader das zu verarbeitende Bild nicht als Funktionsparameter übergeben und wie erhält der 2D-Algorithmus Zugriff auf das Bild? Der Zugriff auf die einzelnen Pixel des Bildes (Framebuffer) erfolgt bei der Programmierung eines Outputshaders immer über API-Funktionen, welche während des Renderings durch Mental Ray aufrufbar sind. Ein Mental Ray-Outputshader erhält auch nie direkten Zugriff auf das gesamte Bild, sondern immer nur auf die Farbwerte eines bestimmten Pixels aus den Bildkoordinaten (X,Y). Im Folgenden Quellcode wird in Zeile 1 der Framebuffer geöffnet und eine Referenz auf diesen der Variable *fb_color* übergeben.

```
1 miImg_image *fb_color = mi_output_image_open(state, miRC_IMAGE_RGBA);
2
3 miColor color;
4 mi_img_get_color(fb_color, &color, x, y);
5 mi_img_put_color(fb_color, &color, x, y);
6
7 mi_output_image_close(state, miRC_IMAGE_RGBA);
```

Der Typenname miImg_image lässt zwar drauf schließen, dass sich in dieser Variable ein voller Zugriff auf die Pixel des Framebuffers verbirgt, dies ist jedoch nicht der Fall. Einen Beweis liefert die Betrachtung der originalen Typendefinition (siehe Anhang A.2, Seite 82). Mit dem Befehl aus Zeile 4 lassen sich die RGBA Werte eines Pixels (Koordinaten x,y) in der Variablen *color* speichern und in Zeile 5 in den Framebuffer zurück schreiben.

Da der 2D-Renderer Mental Ray selbst nicht aufruft und damit Mental Ray nicht aktiv ist, würden bei einem versuchten Durchlauf des Codes die API-Funktionen keine Werte zurückgeben. Um dieses Problem zu lösen, wird der Aufbau der Mental Ray-Outputshader-DLL umstrukturiert. Dies wird auf eine Art und Weise geschehen, mit welcher Mental Ray

und der 2D-Renderer unabhängig von einander auf die passenden Filterunktionen zugreifen können. Der eigentliche Algorithmus des 2D-Filters wird aus der Mental Ray-Funktion herausgelöst und in eine eigene Funktion ausgelagert, auf welche der 2D-Renderer zugreifen kann. Dabei werden die Variablen in der ursprünglichen Mental Ray-Funktion so vorbereitet, dass sie mit der neuen Funktion verwendet werden können. Abbildung 5.7 verdeutlicht diesen Prozess. Im Folgenden sind zwei Fälle zu unterscheiden: Die grün gezeichneten Elemente in

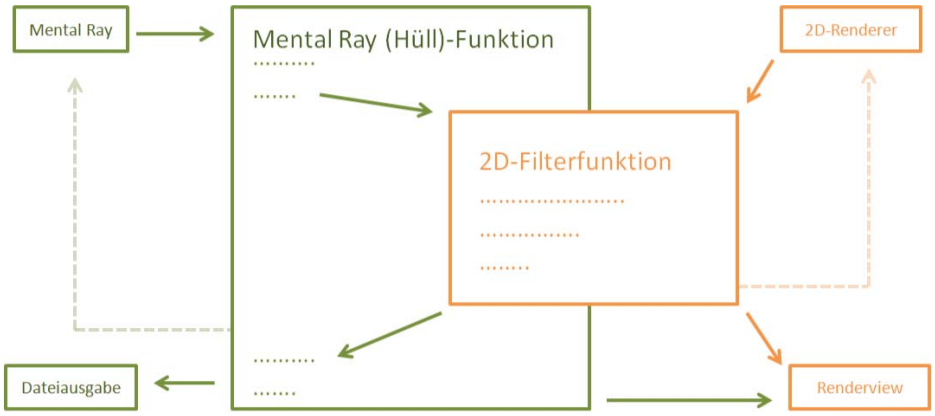


Abbildung 5.7: Hüllfunktion

Abbildung 5.7 beschreiben den regulären 3D-Renderprozess in Mental Ray mit abschließender Verarbeitung der 2D-Filter. Nach dem Abschluss des 3D-Rendervorgangs ruft Mental Ray den ersten 2D-Filter auf. Dabei erfolgt der Funktionsaufruf nach den Konventionen der Mental Ray-API.

```
1 //Mental Ray Funktion
extern "C" DLLEXPORT miBoolean mp_colorbalance(void *result, miState *
    state, struct mp_colorbalance *paras)
```

Die Funktion selbst führt nun jedoch nicht mehr den Filteralgorithmus aus, sondern bereitet ihre Variablen für einen zweiten internen Funktionsaufruf vor, den eigentlichen Filteralgorithmus.

```
extern "C" DLLEXPORT void mp_colorbalance_impl(customImage *fb_color,int
    x_res,int y_res, struct mp_colorbalance *paras)
```

Nach Durchlaufen dieser Funktion und erneutem Rücksprung in die Mental Ray-Funktion wird in einem letzten Schritt das fertige Bild wieder zurück in den Mental Ray-Framebuffer geschrieben. Zu diesem Zeitpunkt ist das Rendering entweder abgeschlossen, das Resultat wird in einer Datei gespeichert, oder im Renderview angezeigt, oder wird an Mental Ray zurückgegeben, um einen zweiten angehängten 2D-Filter zu durchlaufen. Dieser zweite Durchlauf wird durch die grüne punktierte Linie dargestellt.

Im Fall des 2D-Renderers , dargestellt durch die orange gezeichneten Elemente in Abbildung 5.7, wird die eigentliche Funktion des 2D-Filters direkt aufgerufen. Hierdurch wird die

zu diesem Zeitpunkt sowieso nicht aktive Mental Ray-API umgangen. Nach Verarbeitung des Bildes durch den Filter, wird das Ergebnis auch hier entweder im Renderview dargestellt oder dem 2D-Renderer zurück übergeben, um das Bild erneut durch einen weiteren 2D-Filter verarbeiten zu lassen.

Wie zuvor bereits angesprochen, wird innerhalb der Mental Ray-Funktion die eigentliche Filterimplementierungsfunktion aufgerufen. Dabei muss müssen zuerst die Variablen für den erneuten Funktionsaufruf vorbereitet werden. Die Filterfunktion erwartet als Parameter das Bild, dessen Dimension und die Parameter des Filters.

```
extern "C" DLLEXPORT void mp_colorbalance_impl(customImage *fb_color, int
    x_res, int y_res, struct mp_colorbalance *paras)
```

Da auf das gesamte Bild (Framebuffer) nicht direkt programmatisch zugegriffen werden kann, sondern nur auf die einzelnen Pixel, wird in einem ersten Schritt eine vollständige Bildvariable eigenständig zusammengesetzt. Dazu wird das zuvor definierte Struct-Array

```
struct customImage{
    float r;
    float g;
    float b;
    float a;
  };
```

durch Iteration über jeden Pixel des Framebuffers gefüllt (siehe Anhang A.3,S. 82). Diese Funtkion konvertiert letztendlich den Framebuffer in ein eigenes Bildformat („custom image“ genannt). In dieser Form kann es der Filterfunktion übergeben werden. Die Dimensionen des Bildes lassen sich über die state-Variable der Mental Ray-API auslesen.

```
state->camera->x_resolution //Breite
state->camera->y_resolution //Höhe
```

Wie in Abbildung 5.8 zu sehen ist, wird das Bild nach dem Durchlauf der Filterfunktion wieder zurück konvertiert und in den Framebuffer von Mental Ray geschrieben. Zu dem jetzigen Zeitpunkt hat der 2D-Renderer das Bild durch eine beliebige Anzahl von 2D-Filtern verarbeiten lassen. In seinem Speicher hält er das fertige Bild für die Darstellung im Renderview bereit. Auch wurde zuvor sicher gestellt, dass durch die Umstrukturierung der Outputshader der traditionelle 3D-Rendervorgang durch Mental Ray nicht gestört wird, in dem sich die Outputshader weiterhin an die API-Konventionen halten. Der 2D-Renderer erhält durch die Maya-API Zugriff auf den Renderview. Um diesen mit einem Bild zu beschreiben bedarf es den Aufruf von lediglich drei API-Funktionen:

```
MRenderView::startRender(width, height, true, false);
MRenderView::updatePixels(0, width-1, 0, height-1, (RV_PIXEL *)ci_color);
MRenderView::endRender();
```

Mit der Funktion „startRender“ wird dem Renderview signalisiert, dass nun ein neues Bild verarbeitet wird. „UpdatePixels“ kopiert das fertige Bild, in Form des bereits erwähnten

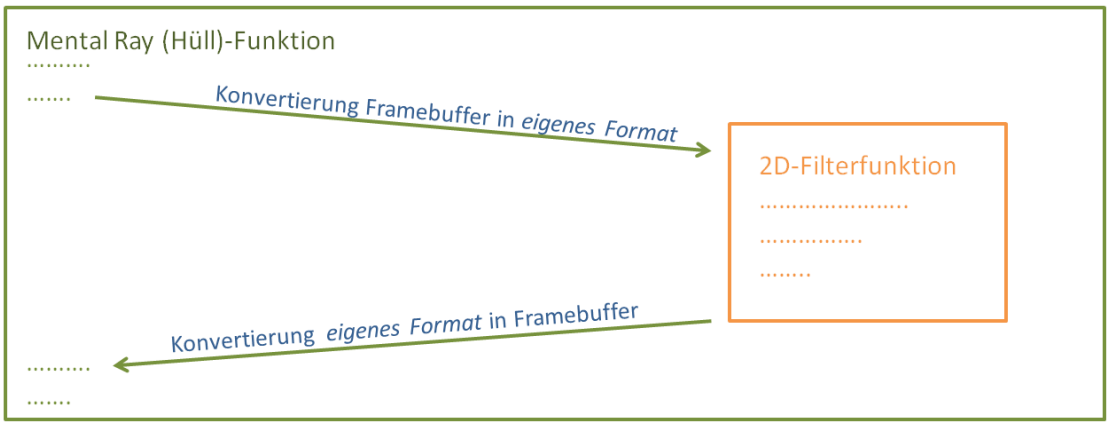


Abbildung 5.8: Variablenübergabe

Struct-Arrays, in den Renderview. Mit der Funktion „endRender“ wird dieser Prozess abgeschlossen und Maya für weitere Nutzerinteraktion wieder freigegeben. In der Regel vergeht bei einem 3D-Rendering eine lange Zeit, deshalb sieht die Maya-API die Funktionen „startRender“ und „endRender“ vor. Bei dieser Implementierung wird der Rendermechanismus nur dazu verwendet, um ein bereits fertiges 2D-Rendering im Renderview anzuzeigen.

Mit diesem letzten Schritt ist die Verarbeitungskette des 2D-Renderers abgeschlossen. Nimmt man die Berechnungsdauer der Filteralgorithmen aus der Betrachtung heraus, so verbraucht der 2D-Renderer während seiner Ausführung eine fast nicht messbare Zeit im Millisekundenbereich. Dies ist wichtig für eine spontane Reaktion des Renderviews auf die Veränderung der Filtereinstellungen durch den Nutzer.

5.4 Die grafische Benutzeroberfläche

Bis zu diesem Punkt wurde die Integration von mehreren 2D-Filter Outputshadern in den Renderprozess von Mental Ray, sowie die unabhängige Verarbeitung dieser durch den 2D-Renderer erklärt. Damit wäre die Entwicklung dieser Arbeit schon anwendbar. Jedoch müsste der Nutzer sein Shadernetzwerk im Hypershade selber zusammenbauen und einen Mel-Befehl ausführen um das Ergebnis seiner Filteranwendung im Renderview zu sehen. Dieser Vorgang soll im Folgenden von einer grafischen Benutzeroberfläche übernommen werden, welche sich in ihrer visuellen Gestaltung an die Filterverarbeitung von Adobe After Effects (vgl. Kapitel Konzeptentwicklung, S. 28) anlehnt.

Maya stellt hierfür eine Skriptsprache mit dem Namen „Maya Embedded Language“ (MEL) zur Verfügung, mit welcher sich nicht nur grafische Oberflächen entwickeln lassen. Mit ihr lassen sich zudem alle Aktionen, welche der Nutzer mit der Tastatur und Maus in Maya ausführt, programmatisch abbilden und automatisieren. Die Oberfläche verfolgt primär drei Ziele (vgl. Abbildung 5.9, S.44):

1. Der Nutzer kann die Kamera auswählen, auf dessen Rendering die 2D-Filter angewendet werden sollen. Nach Auswahl der Kamera wird der Verzweigungsshader automatisch an diese im Hypershade angehängt.

2. In einer Dropdownliste erhält der Nutzer die verfügbaren Filter, welcher er seiner ausgewählten Kamera zuordnen kann. Nach dem Hinzufügen ordnet sich der Filter in der Filterliste ein. Die Verbindung mit dem Verzweigungsshader wird automatisch aufgebaut.

3. Jeder Filter lässt sich konfigurieren. Während dem Anpassen der Filterparameter erhält der Nutzer durch einen impliziten Aufruf des 2D-Renderers sofortige visuelle Rückmeldung im Renderview. Des Weiteren lassen sich die Filter innerhalb der Filterliste in ihrer Anwendungsreihenfolge per „Drag and Drop“ sortieren.

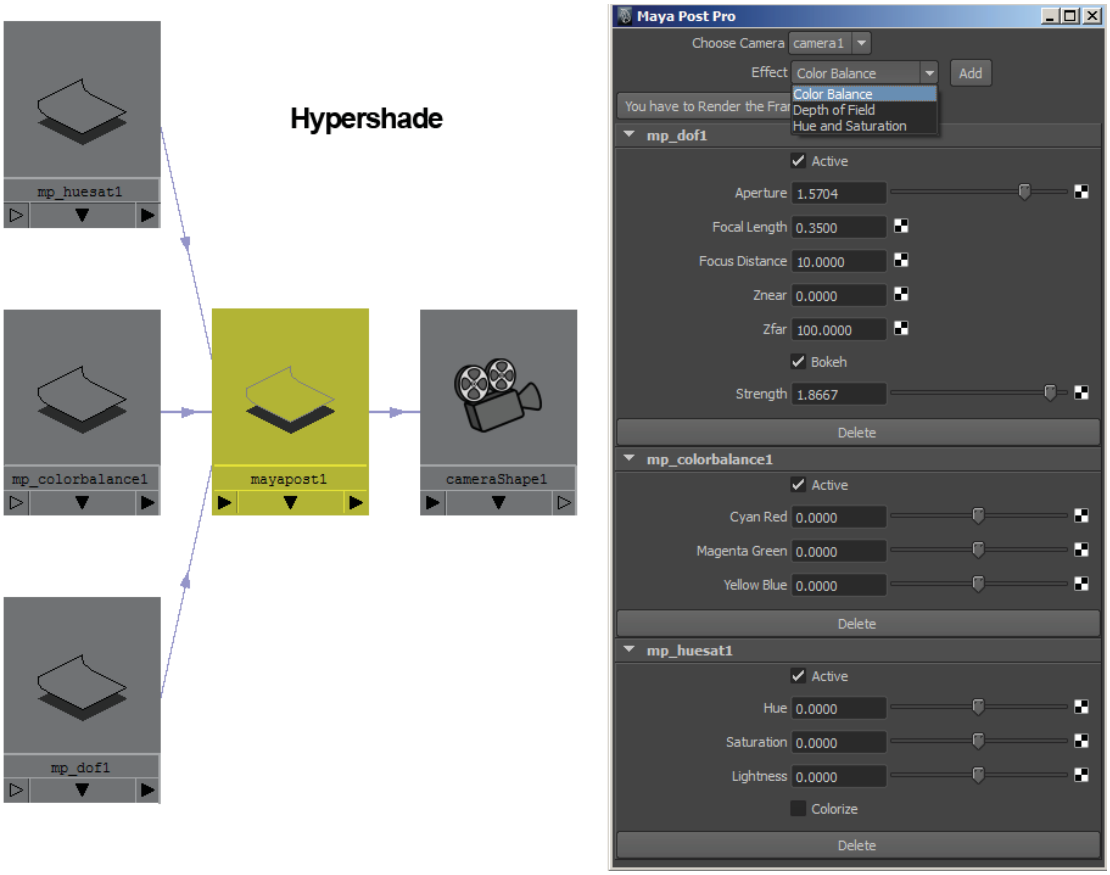


Abbildung 5.9: Verwaltung der Shadernodes durch die Mel GUI

Die Nutzeroberfläche befindet sich in einem eigenen Fenster, welches sich innerhalb der Maya-Oberfläche frei verschieben lässt. Sie besteht hauptsächlich aus zwei Auswahllisten und einer Stapelansicht der anzuwendenden Effekte. Mit folgendem Mel-Script, welches per Klick aus der Maya-Toolbar gestartet wird, wird die Oberfläche generiert.

```
if (`window -exists postpro`) deleteUI postpro;
window -title "Maya Post Pro" -width 300 postpro;
string $choseCamLayout = `setParent -q`;
string $effectlayout =`columnLayout $choseCamLayout`;
optionMenuGrp -label "Choose Camera" -changeCommand "updateFilterList();"
    camList;
rowLayout -numberOfColumns 2;
  columnLayout;
    optionMenuGrp -label "Effect" selectEffectList;
      menuItem -label "Color Balance";
      menuItem -label "Depth of Field";
      menuItem -label "Hue and Saturation";
      setParent..;
    columnLayout;
      button -label " Add " -c "createEffect(); updateFilterList();";
      setParent ..;
  setParent..;
rowLayout -numberOfColumns 1;
  columnLayout;
    button -label " Re-Render " -c "renderWindowRender redoPreviousRender
        renderView;" -ebg false stopButton;
    setParent ..;
setParent..;
listCams($choseCamLayout);
rowColumnLayout effectList;
showWindow postpro;
```

Die eigentliche Funktionalität der Oberfläche wird in Prozeduren ausgelagert. Eine Prozedur, wie im Folgenden zu sehen, ist z.B. dafür zuständig, die Dropdownliste mit den Namen der aktuell verfügbaren Kameras zu füllen. Aus dieser Liste wählt der Nutzer schließlich eine zu bearbeitende Kamera aus.

```
global proc listCams(string $choseCamLayout){

  string $list[] = `ls -cameras`;
  string $listcamparent[] =`listRelatives -p $list`;
    for ($eachcam in $listcamparent)
      {
        menuItem -label $eachcam -parent ($choseCamLayout + "|camList|
            OptionMenu") ;
      }
  }
```

Der Mel Befehl „ls -cameras“ in Zeile 3 erzeugt ein Array aus den Namen der verfügbaren Kameras und der Befehl „menuItem -label $eachcam -parent ...“ in Zeile 7 fügt die einzelnen Kameranamen der Dropdownliste hinzu. Die Auswahlliste der 2D-Filter wird nicht automatisch aufgebaut, sondern manuell im Mel-Sript angegeben. Dieser Prozess ließe sich zwar automatisieren, man hätte jedoch keinen Einfluss auf die Namensgebung der einzelnen Filter. Ein Filter (Outputshader) muss bestimmte Namenskonventionen verfolgen, z.B. darf er keine Leerzeichen beinhalten. Der „Depth of Field“-Effekt würde bei einer automatisier-

ten Verarbeitung den Namen der Shader-Node „mp_dof“ annehmen. Deshalb wurden die Filternamen der Effektauswahl mit eigenen verständlichen Namen versehen, welche später im Mel-Skript zu einer passenden Shader-Node übersetzt werden.

```
1  if ($effectName == "Color Balance")
     $newEffect ="mp_colorbalance";
3  if ($effectName == "Depth of Field")
     $newEffect ="mp_dof";
5  if ($effectName == "Hue and Saturation")
     $newEffect ="mp_huesat";
```

Die Liste mit den anzuwendenden Filtern mitsamt ihrer Einstellungen wird dynamisch über eine weitere Prozedur erstellt. Dazu wird der zu der Kamera passende Verzweigungsshader ermittelt und sein Array der Outputshader ausgelesen, welches auf die einzelnen 2D-Filter verweist. In Abbildung 5.10 in der linken Bildhälfte ist der Attributeditor eines Outputshaders zu sehen. Diese Ansicht der Filterparameter soll im Folgenden auf die selbe Art und Weise im Effekteditor angezeigt werden (siehe rechte Bildhälfte in Abbildung 5.10). Hierbei ist zu beachten, dass Änderungen in der Benutzeroberfläche gleichzeitig auch im Attributeditor der Hypershade-Node angewendet werden. Mental Ray bezieht seine Daten aus den Attributen einer Shader-Node und nicht aus der Oberfläche des Effekteditors. Dieser stellt lediglich eine Ansicht auf den Attributeditor dar. Im folgenden Mel-Skript (S.46) ist zu erkennen,

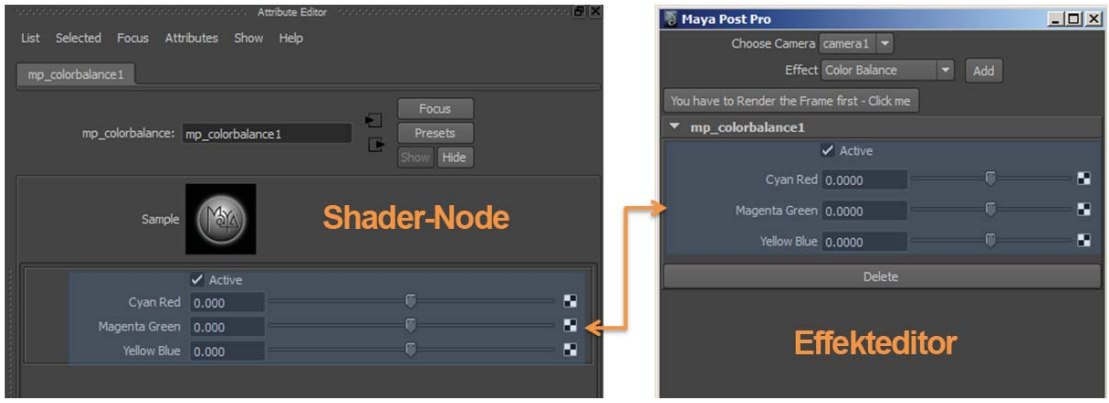


Abbildung 5.10: Filterparameter

wie diese Voraussetzung realisiert werden kann. Mit dem Befehl „attrControlGrp“ in Zeile 10 kann ein Attribut einer beliebigen Shader-Node in eine eigene Oberfläche „verlinkt“ werden. Dies bedeutet, dass die Kontrollelemente (z.B. Schieberegler und Zahlenwerte) nicht selbstständig in Mel erzeugt werden müssen, sondern automatisiert dargestellt werden. Ein weiterer Vorteil dieser Technik ist, dass Änderungen im Effekteditor durch die „Verlinkung“ automatisch in der Shader-Node im Hypershade übernommen werden. Dadurch dass in einem Attributeditor einer beliebigen Node jeder Wert mit Keyframes und Expressions versehen werden kann, erhält man nun auch im Effekteditor diese essenzielle Möglichkeit ohne weitere Programmierung zu betreiben.

```
for($shaderName in $shaderNames){
setParent effectList;
string $lstA, $lstAttributes[];
$lstAttributes=`listAttr -k $shaderName`;
frameLayout -label $shaderName -parent effectList -w 400 -cl true -cll
    true
        -borderStyle "out" -dragCallback dragEffect -dropCallback
            dropEffect;

for ($lstA in $lstAttributes){
  string $arg = $shaderName+"."+$lstA;
  attrControlGrp -a $arg -cc "update()";
}
button -label "Delete" -c ("removePPEffect(\""+$shaderName+"\");
    updateFilterList()");
}
```

Im oben dargestellten Mel-Skript ist im Weiteren zu erkennen, wie die Filterliste des Effekteditors durch Iteration (Zeile 1) über die an den Verzweigungsshader angehängten Filter aufgebaut wird. Dazu wird jeder Outputshader (Filter) zuerst in ein „frameLayout“ eingebettet. In solch einem Layout lässt sich jeder Filter im Editor ein- und ausklappen, um den Nutzer die Möglichkeit zu geben, bereits fertig eingestellte Filter grafisch zu minimieren. Innerhalb eines jeden Framelayouts wird zudem ein „Delete“-Button platziert, um den entsprechenden Filter löschen zu können.

Beim Löschen eines Filters, wird der dazugehörige Outputshader zuerst im Hypershade gelöscht. Daraufhin wird die Filterliste im Effekteditor gelöscht, um sie sofort danach erneut wieder korrekt aufzubauen. Dies geschieht optisch gesehen so schnell, dass in der Wahrnehmung des Nutzers lediglich der gelöschte Filter aus der Liste verschwunden ist. Derselbe Vorgang wird beim Hinzufügen eines neuen Filters durchlaufen. Zuerst wird der Outputshader (Filter) im Hypershade erzeugt und an den Verzweigungsshader angehängt. Dann wird die Filterliste im Effekteditor gelöscht und wieder von Neuem aufgebaut.

Die letzte Eigenschaft des Effekteditors, welche auch aus After Effects übernommen wird, ist die Möglichkeit, die Filter in ihrer Reihenfolge und Abarbeitung neu zu ordnen. Dazu bietet Mel zwei Funktionen, welche auf vordefinierte Ereignisse (Drag and Drop) reagieren. Zieht der Nutzer mit der mittleren Maustaste einen Filter über einen zweiten Filter, so lassen sich programmatisch die zwei betroffenen „frameLayouts“ ermitteln, in welchen die Filter eingebettet sind. Über die Namen der „frameLayouts“ lässt sich auf die Filter (Outputshader-Nodes) zurückschließen und die jeweilige Verbindung zu ihrem Verzweigungsshader austauschen. Nach erneutem Löschen und Neuaufbauen der Filterliste, ist der Vorgang abgeschlossen, die Reihenfolge der Filter vertauscht.

Mit diesem letzten Schritt sind alle Vorgänge in der Implementierung des Post Production Frameworks erklärt worden. In dem nächsten Kapitel wird die Implementierung der verwendeten 2D-Filter beschrieben.

Kapitel 6

Implementierung der 2D-Filter

In dem vorherigen Kapitel wurde gezeigt, wie sich die 2D-Filterbearbeitung in Maya integrieren lässt. Neben der Integration in den gewöhnlichen Renderprozess von Mental Ray, wurde weiterhin die Architektur des direkten Bearbeitens im Renderview vorgestellt. Dieses Kapitel widmet sich nun der Implementierung und Algorithmik der 2D-Filter. Um die Idee der Integration von Post Production in Maya beispielhaft umsetzen zu können, werden im Folgenden drei Filter prototypisch implementiert. Zu diesen gehören zwei Farbkorrekturfilter, welche in der Post Production weite Verbreitung gefunden haben und schnell zu verarbeiten sind, sowie einen rechenintensiven Tiefenschärfe-Filter, welcher Objekte außerhalb der Fokusebene der Filmkamera unscharf abbildet.

6.1 Color Balance

Der Color Balance-Effekt gehört in die Rubrik Farbkorrektur und ist mit diesem Namen auch in den Softwareprodukten von Adobe (After Effects und Photoshop) wiederzufinden. Die Implementierung lehnt sich an die Softwareprodukte an. In Abbildung 6.1 ist die Wirkungsweise dargestellt. Mit diesem Effekt lässt sich die Farbstimmung eines Bildes mit Hilfe von drei Parametern verändern. Eine kühle bläuliche Szenerie lässt sich in eine sonnige warme Stimmung verwandeln. Dieser Filter ist einfach gehalten. Mit drei Schiebereglern lassen sich die drei Farbkanäle (RGB) eines Bildes jeweils verstärken oder abschwächen. Verstärkt man z.B. den Rotkanal, erhält das Bild einen roten farbigen Stich. Vermindert man die Werte dieses Kanals, erhält das Bild eine Farbgebung, welche in Richtung Cyan geht.

Der „Cyan-Red“-Parameter des Filters kann Werte zwischen -100 und 100 annehmen. Der Wert 100 würde den Rotanteil jedes Pixels auf die volle Stufe setzen. In der 8-Bit Darstellung würde dies dem Wert 255 entsprechen. Mental Ray verarbeitet Farben intern in der arithmetischen Notation. Die Werte reichen hier von 0.0 bis zu 1.0, jedoch mit einer höheren Genauigkeit gegeben durch die Verwendung von 32-Bit-Gleitkommazahlen (float). Der Parameterwert 100 des Filters setzt den Rotanteil des Bildes schließlich auf 1.0. Für die

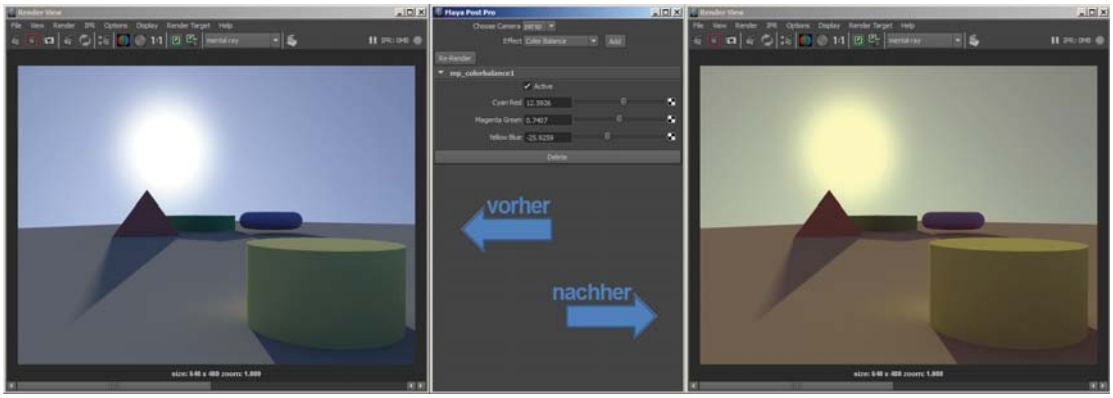


Abbildung 6.1: Color Balance - Vorher - Nachher

Darstellung im Renderview werden diese Werte jedoch wieder auf 0..255 linear abgebildet[1].

Im Folgenden Quellcode ist die Verarbeitung des Farbkanals am Beispiel des Rotanteils eines Bildes zu sehen. Zeile 1-3 bereiten die Iteration über jeden Pixels des Bildes vor. „fb_color[index].r“ spiegelt den Wert des Rotanteils wider, welcher durch den Filter verändert wird. „paras->red“ ist der vom Nutzer eingegebene Parameter (-100 bis 100).

```
1  unsigned int sum_pixels = x_res * y_res;

3  for( unsigned int index = 0; index < sum_pixels; index++ ) {
     if(paras->red > 0) //Verstärkung
5        fb_color[index].r = fb_color[index].r + ((1 - fb_color[index].r) *
             paras->red / 100);

7    if(paras->red < 0) //Abschwächung
         fb_color[index].r = fb_color[index].r + (fb_color[index].r * paras->
             red / 100);
9
           ...
11       }
```

Die Zeilen 4 und 7 stellen eine Fallunterscheidung dar. Ist der Parameter größer als der Wert 0, so handelt es sich um eine Verstärkung, im umgekehrten Fall um eine gewünschte Abschwächung des Rotanteils. In den Zeilen 5 und 8 wird dann schließlich der neue Rotanteil des Pixels durch eine lineare Skalierung errechnet.

Die Verarbeitung der Grün- und Blauanteile des Bildes verläuft analog zu der eben aufgezeigten Implementierung. Hiermit wäre nun ein erster Filter implementiert, welcher durch den einfachen Codeteil sehr effizient und zugleich wirksam in seiner Anwendung ist.

[1] Diese Umwandlung ist verlustbehaftet, da auf gerade Zahlen zwischen 0 und 255 abgerundet wird

6.2 Hue & Saturation

Während der Farbkorrektur eines Bildes möchte man zumeist nicht nur Farben an sich beeinflussen können, sondern auch die Helligkeit und Farbsättigung eines Bildes bestimmen. Dazu bietet sich ein weiterer weit verbreiteter Filter an, der „Hue and Saturation"-Effekt. Dieser ist auch aus programmatischer Sicht schnell und effizient zu implementieren und bietet einen großen Nutzen bei der Farbkorrektur. Die Benennung „Hue and Saturation"

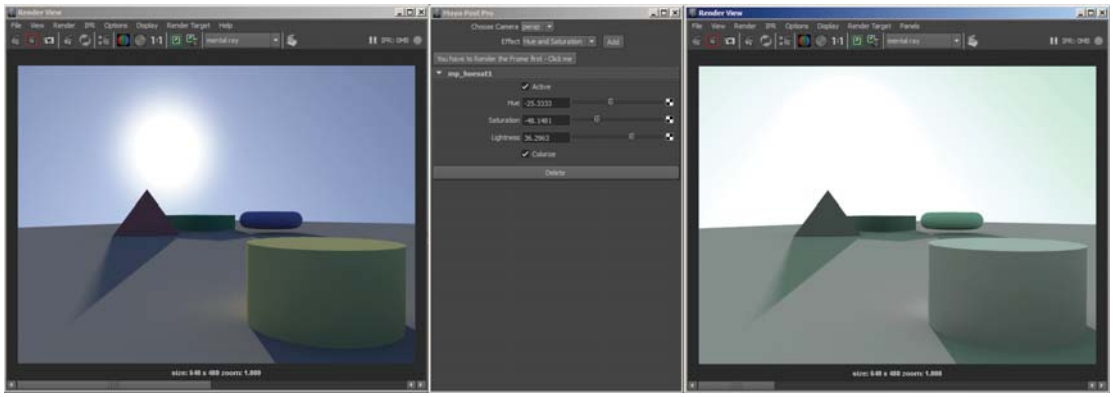


Abbildung 6.2: Hue and Saturation - Vorher - Nachher

ist an den HSI-Farbraum[2] (Hue, Saturation, Intensity) angelehnt. Im Gegensatz zu dem RGB-Farbraum, entspricht dieser mehr der menschlichen Wahrnehmung von Farbe. Der Mensch beschreibt eine Farbe zumeist an drei Anhaltspunken: Die Farbigkeit an sich (Hue), ihre Sättigung (Saturation) und die wahrgenommene Helligkeit (Intensity) (vgl. [GW01, S. 295]). Diese drei Parameter werden durch den Filter verändert. Damit dies möglich ist, muss der RGB-Wert jedes Pixels zuerst vom RGB-Farbraum in den HSI-Farbraum konvertiert werden. In diesem Farbraum kann z.B. die Helligkeit des Bildes durch Verringerung des Intensity-Parameters heruntergesetzt werden. Nach der Bearbeitung wird das Bild wieder vom HSI-Farbraum in den RGB-Farbraum zurückkonvertiert (vgl. [GW01, S. 301]).

Abbildung 6.2 zeigt ein Beispiel der Farbveränderungen durch den „Hue and Saturation"-Effekt. Zuerst lässt sich erkennen, dass die Sättigung der Farben vermindert wurde. Als nächster Schritt wurde die Gesamthelligkeit erhöht, welches sich auch an der stärkeren Sonnenintensität erkennen lässt. In einem dritten Schritt wurde die Farbstimmung verändert. Hierbei wurde das Bild komplett neu eingefärbt (Colorize-Option). Die Hue-Werte eines Pixels werden nicht relativ zur ihrem alten Wert verändert, sondern mit einem absoluten Wert neu gesetzt.

[2]Neben dem HSI-Farbraum, gibt es weitere ähnliche Farbräume mit den Namen HSV,HSB und HSL. Bis auf die unterschiedlichen Definitionen der Weiß- und Graupunkte sind diese Farbräume als gleichwertig anzusehen.

6.3 Depth of Field - Tiefenschärfe

Mit Hilfe des Tiefenschärfe-Effekts (Depth of Field, im Folgenden Dof genannt) kann man Objekte außerhalb des Fokus der Kamera unscharf erscheinen lassen. Dies ist im Kinobereich und in der Fotografie ein oft angewendetes Stilmittel, um den Blick des Zuschauers bewusst zu lenken (vgl. [Dem04, S.375]). Dieser Effekt lässt sich mit Mental Ray generieren, verlängert die Renderzeit jedoch um einen großen Faktor[3], da Mental Ray den Dof-Effekt auf 3D-Basis berechnet. Um dieses Problem der langen zusätzlichen Renderzeit zu umgehen, haben bereits viele Post Production Applikationen 2D-Filter zum Berechnen der Tiefenschärfe implementiert. Auf diese Art und Weise liegt die Anwendungsdauer des Effektes nur noch im Sekundenbereich.

Der 3D-Artist Alex Roman[4] hat durch seine Werke schon unlängst bewiesen, dass Dof keinesfalls mehr nur im 3D-Rendering gute Ergebnisse erzielen kann, sondern auch mit Hilfe von 2D-Filtern beeindruckende Ergebnisse (vgl. Abbildung 6.3) liefert. Die Dof-Effekte hat er mit einem Post Production-Plugin für After Effects erstellt[5]. Wie der Name Tiefenschärfe

Abbildung 6.3: Beispiel Tiefenschärfe, Autor: Alex Roman

schon andeutet, sind die unscharfen Bereiche in einem Bild abhängig von der Distanz eines Objektes zu einer gedachten fokussierten Ebene im *Raum*. Ein gerendertes 2D-Bild hat jedoch seine *Räumlichkeit* verloren. Diese Information über die ursprüngliche Tiefe eines Objekts muss jedoch zur Anwendung des Filter in der Post Production rekonstruiert werden.

[3] siehe Kapitel Ergebnisse, S.70
[4] Alex Roman, http://www.thirdseventh.com/
[5] Lenscare (Frischluft) - http://www.frischluft.com/gallery/lenscare.php

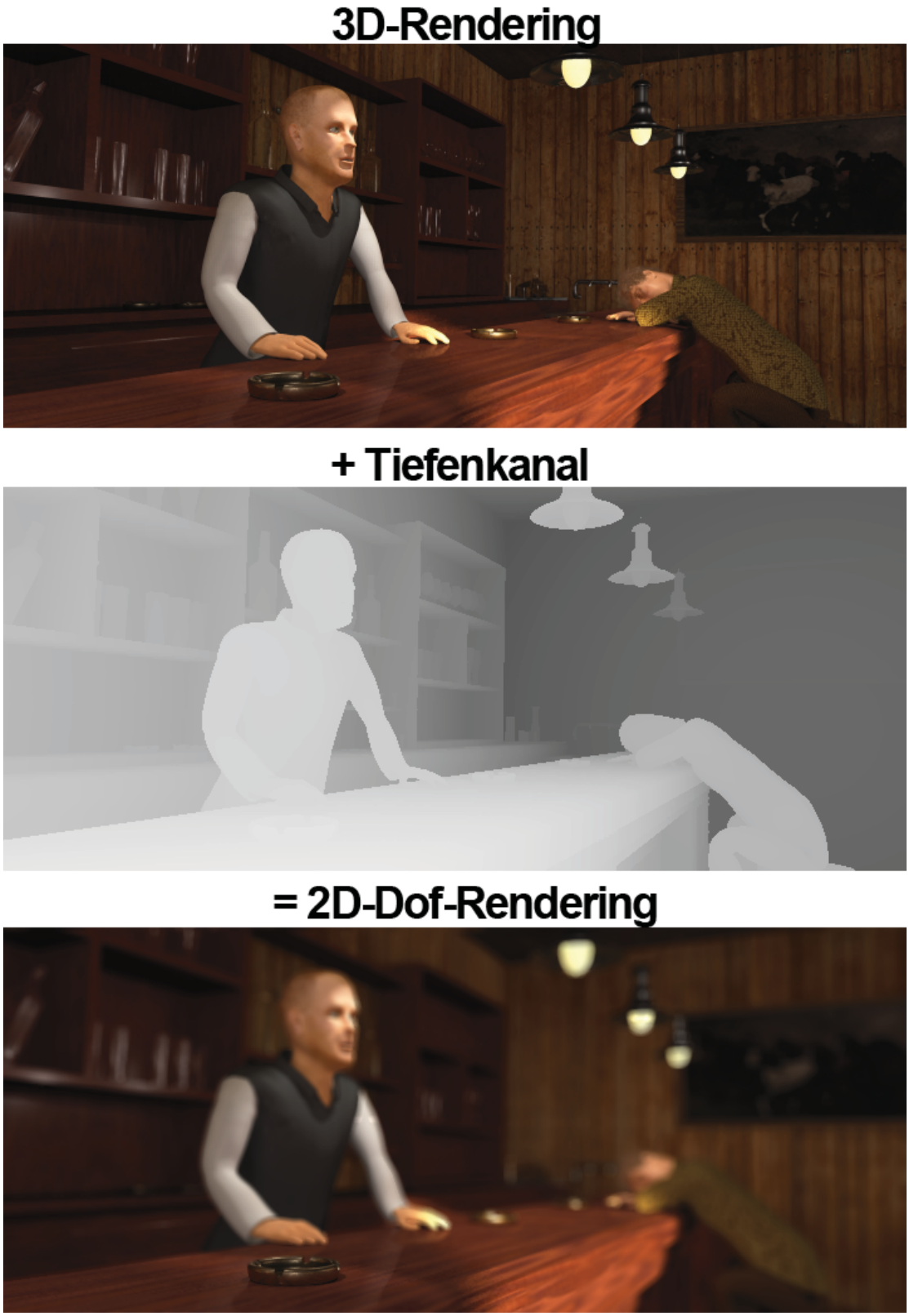


Abbildung 6.4: Dof - berechnet aus einem 3D-Rendering und einem Tiefenkanal. Der Fokus ist auf den Aschenbecher im Vordergrund gesetzt.

Dazu verwendet man einen Tiefenkanal. Er besteht aus einem gleich großen zweidimensionalem Graustufenbild, mit dessen Hilfe jedem Pixel des 3D-Renderings seine ursprüngliche Tiefe im dreidimensionalen Raum rekonstruiert werden kann (siehe Abbildung 6.4, S.53). Helle Bildbereiche befanden sich in der ursprünglichen 3D-Szene im Vordergrund und je dunkler die Bildbereiche werden, desto tiefer befanden sie sich ursprünglich im Raum.

Diese Technik hat zur Folge, dass man für die Verarbeitung in After Effects zwei getrennte Bilder erzeugen muss. Diese werden auf zwei übereinander liegenden Ebenen platziert. Dem Filter muss weiter die korrekte Ebene des Tiefenkanals zugeordnet werden, bevor er auf das 3D-Rendering angewendet werden kann. Integriert man Dof direkt in das Mental Ray-Rendering, müssen diese Schritte nicht durchlaufen werden. Man wird dann jedoch wieder mit der Problematik der langen Renderzeit konfrontiert. Dies ist auch der Grund, weshalb dieser wichtige Filter in das Post Production Framework implementiert werden soll. Man erhält von beiden Varianten die Vorteile. Die schnelle Renderzeit aus After Effects und die sofortige Verarbeitung durch Mental Ray.

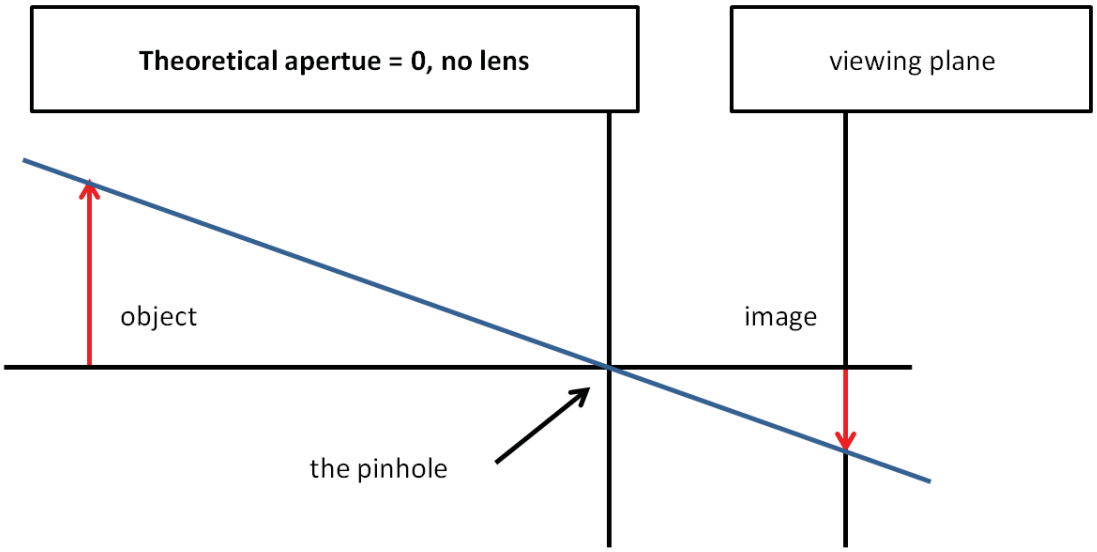


Abbildung 6.5: Das Modell der Lochkamera (the perfekt pinhole camera), Abbildung entnommen aus [Yu04]

Computer generierte Grafiken verfügen in der Regel über keine Dof-Simulation. Dies liegt daran, dass viele 3D-Renderer das Model der Lochkamera (siehe Abbildung 6.5) verwenden, welches kein klassisches Objektiv für die Abbildung verwendet, sondern ein unendlich kleines Loch (vgl. [YM04]). Alle Abbildung werden scharf dargestellt. Um nun einen realistischen Tiefenschärfe-Effekt zu erstellen, muss man auch ein realistisches Kameramodel verwenden, welches ein Objektiv mit einer variablen Blende beinhaltet.

Abbildung 6.6 auf Seite 55 verdeutlicht, wie Unschärfe durch ein Objektiv entsteht. Die punktierte grüne Linie verweist auf eine Ebene im Raum, welche scharf auf der Filmebene (Kamerasensor) durch das Objektiv abgebildet wird. Der grüne Baum befindet sich

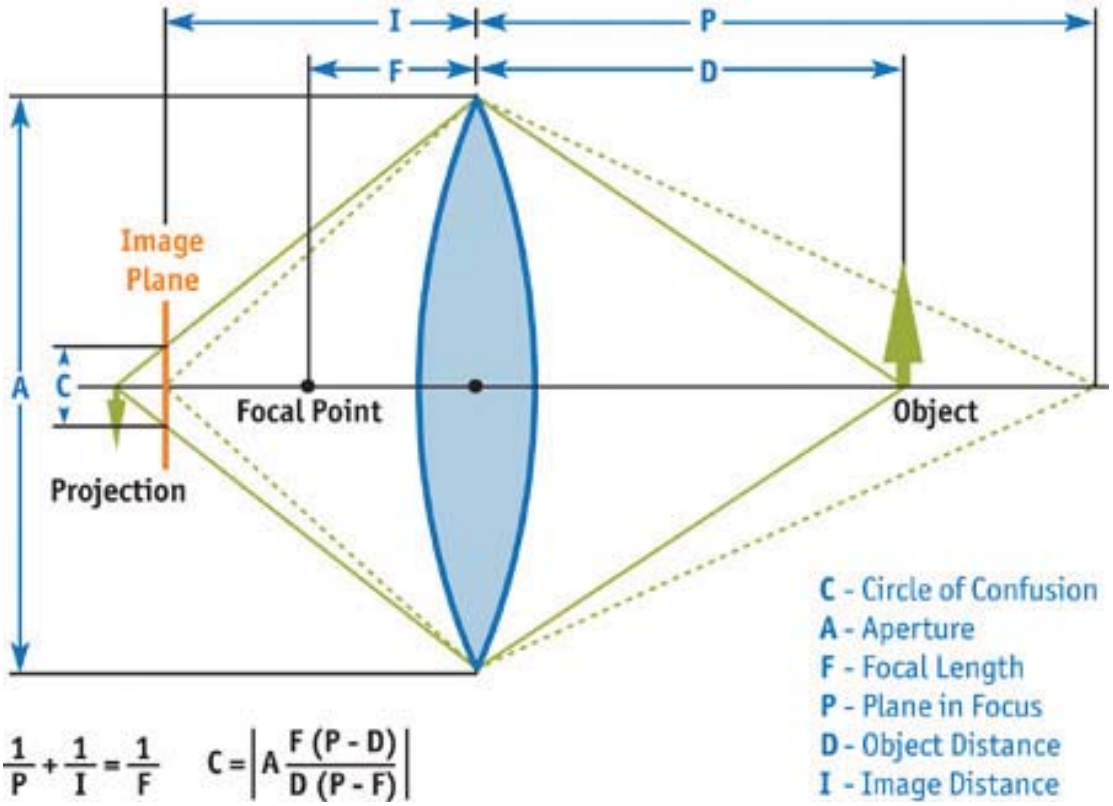


Abbildung 6.6: Der Zerstreuungskreis (Circle of Confusion), Abbildung entnommen aus [Dem04]

jedoch in der Realität vor dieser Ebene, wodurch der gedachte Scharfpunkt virtuell hinter der Filmebene liegt. Auf die Filmebene trifft jedoch ein Lichtbündel in Form eines Kreises (Zerstreuungskreis, im Folgenden CoC genannt) auf. Die Abbildung ist somit unscharf oder „zerstreut“. Der Durchmesser dieses Kreises lässt sich durch die in Abbildung 6.6 genannte Formel (C) ausrechnen. Mit ihr kann man jedem Pixel eines zweidimensionalen Bildes einen gedachten CoC zuordnen. Während der 2D-Verarbeitung des Dof-Filters kann man auf alle genannten Variablen zurückgreifen. Den Abstand zu der gedachten scharfen Ebene (P) legt der Nutzer in Maya fest. Den Abstand eines jeden Pixels zu seinem ursprünglichen räumlichen Abstand zum Objektiv (D) erhält man über den Tiefenkanal. Die Größe der Blende (A) legt der Nutzer auch in den Filtereinstellungen fest. Je größer dieser Wert ist, desto weiter ist die Blende geöffnet und desto unschärfer werden die unscharfen Bereiche im Bild. Die Brennweite (F) und der Abstand der Filmebene zum Objektiv (I) lassen sich aus der Maya-Kamera ablesen.

Abbildung 6.7 verdeutlicht, wie ein Zerstreuungskreis auf einer Filmebene aussieht. Dabei ist die Filmebene letztendlich das aus Pixeln aufgebaute Computerbild. Auf der linken Seite der Abbildung wurde ein Objekt im Raum, welches sich auf der Fokusebene befand, scharf abgebildet. Eine scharfe Abbildung umfasst genau einen Pixel. Dasselbe Objekt befindet sich

nun auf der rechten Bildhälfte in einem unscharfen Bereich, entweder vor oder hinter der Fokusebene. Seine Abbildung ist nun unscharf und gestaltet sich auf der Filmebene in Form eines Unschärfekreises. Kreise können durch Pixel nicht perfekt abgebildet werden, deswegen wählt man die Pixel aus, welche sich (fast) vollständig in einem Kreis befinden. Die unscharfe Abbildung umfasst nun fünf Pixel. Der einst scharfe Pixel verteilt seine Intensität nun auch auf vier benachbarten Pixel. Zusammen also fünf Pixel.

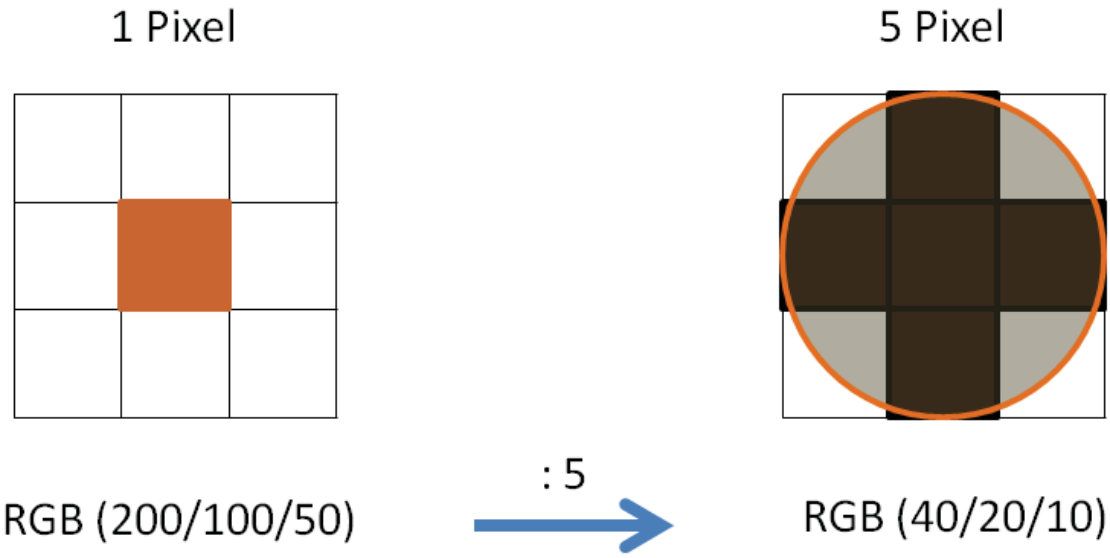


Abbildung 6.7: Verteilung des Lichts innerhalb des CoCs - *Uniform light distribution*

[Yu04] schlägt einfachheitshalber als Lichtverteilungsfunktion eine lineare Verteilung (Uniform light distribution) vor. Dies bedeutet, dass alle 5 Pixel denselben Wert erhalten. In der Realität verringert sich die Intensität zum Rand des Kreises hin. Entscheidend ist jedoch, dass das Bild nicht an Intensität verliert, sonst würde es dunkler wirken. Die Summe aller einzelnen RGB-Werte darf sich im Verlauf der Verarbeitung nicht verringern. Jeder einzelne Pixel erhält damit ein Fünftel der ursprünglichen Intensität. Ein Pixel eines fertig verarbeiteten Bildes stellt sich folglich aus allen Intensitätsverteilungen seiner Nachbarpixel zusammen. Durch Aufsummieren dieser neuen Nachbarwerte erhält der Pixel *ungefähr* seinen alten Helligkeitswert zurück. Das Bild verliert nicht an Helligkeit, sondern in gewissen Bildbereichen an Schärfe.

Durchläuft man diesen Prozess der Intensitätsverteilung über jeden Pixel des Bildes, mitsamt seinem dazugehörigenden CoC, so erhält man das Resultat des 2D-Dof-Filters. [Yu04] macht in seinen Ausführungen einen Vorschlag für die programmatische Implementierung. Dieser in eine abstrakte Form gefasste Vorschlag wird nun im Folgenden gezeigt und erläutert. Auf diesem Vorschlag basiert auch die konkrete Implementierung des Filters dieser Arbeit.

```
1  For each scan line,
     For each pixel (x, y) on the scan line
3      PixelDepth = db[x, y]
       CoC diameter = abs( Aperture-Coefficient *
5        ( ScreenPos*(1.0/FocalLen - 1.0/PixelDepth) - 1 ) )
       UniformDistribution(CoC diameter, x, y, fb1, fb2)
7
9
   Function UniformDistribution( c, x, y, frameBuffer1, frameBuffer2 )
11   r = c/2
     a = pi * r^2
13   intensity = frameBuffer1[x, y]/a
     for ( row <- -r to r )
15     for ( col <- -r to r )
         if ( sqrt( row^2 * col^2 ) < r )
17         frameBuffer2[row, col] += intensity
```

Der Code in Zeile 1 und 2 sorgen dafür, dass die nun folgende Verarbeitung über jeden Pixels eines Bildes durchgeführt wird. Zu jedem Pixel wird in Zeile 4 und 5 sein virtueller CoC-Durchmesser auf der Filmebene berechnet. Mit diesen gesammelten Informationen wird die Funktion *UniformDistribution* aufgerufen, welche die Verteilung der Intensität auf die benachbarten Pixel durchführt. Bei dieser Methode werden die neuen Intensitäten der Pixel in ein neues, noch leeres Bild (*frameBuffer2*) geschrieben. Das originale Bild, welches das Renderergebnis aus Mental Ray widerspiegelt, wurde *frameBuffer1* bezeichnet. Dieses dient nur noch dazu, die originalen Intensitäten der scharf abgebildeten Pixel auszulesen, um sie dann in *frameBuffer2* zu verteilen.

Die Funktion *UniformDistribution* beginnt in Zeile 11 zuerst damit, aus dem Kreisdurchmesser einen Radius zu berechnen. Dieser wird später benötigt, um die Pixel innerhalb eines CoCs ausfindig zu machen. In Zeile 12 wird errechnet, wieviele Pixel sich in dem dazugehören CoC befinden. Der Variablenname *a* steht hierbei für die Anzahl (amount) der Pixels im CoC. Die neue Intensität der einzelnen zu verteilenden Pixel wird in Zeile 13 errechnet, indem die Intensität des Originalpixels durch die Anzahl der betroffenden Pixel geteilt wird (vgl. 6.7). In Zeile 14 und 15 werden die Pixel im Kreis ermittelt und in Zeile 15 werden die neuen Intensitäten auf die Pixel des *frameBuffer2* aufsummiert. In einem letzten Schritt wird der *frameBuffer2* in den *frameBuffer1* umkopiert und dem Renderer das fertige Bild wieder übergeben.

Bei starken unscharfen Bereichen gibt es allerdings einen Schwarzverlauf im Bereich des Randes, wie auf Abbildung 6.8 zu sehen ist. Dies liegt an der Zusammensetzung der Pixel im Randbereich. Ein Pixel setzt sich wie zuvor erklärt aus der Summe der Intensitätsverteilungen seiner benachbarten Pixel zusammen. Im Randbereich haben die Pixel jedoch keine benachbarten Pixel mehr. Das Bild hört hier auf. In der Summe der Intensitäten fehlen einige Summanden. Je größer der CoC an einer bestimmten Stelle ist, desto stärker wird der Rand. Stellt man sich bei einem Randpixel einen CoC-Wert von 10 vor, so fehlen ihm die

Summanden der benachbarten 10 Pixel. In der Summe hat dies eine dunklere Abbildung zur Folge.

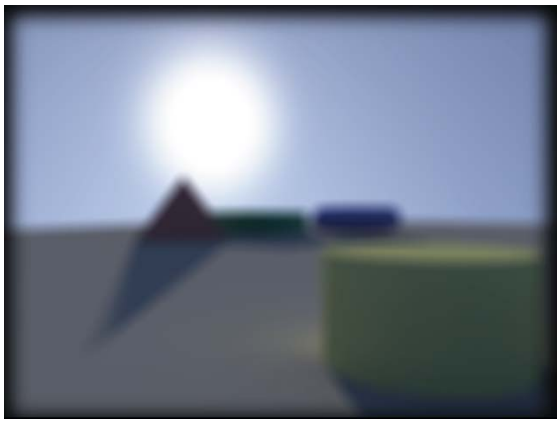


Abbildung 6.8: Problematik Schwarzverlauf bei einem Dof-Rendering

Um dieses Problem zu lösen, erzeugt man vor dem Vearbeitungsvorgang ein virtuell größeres Bild, welches die Pixeldimensionen des Bildes ausweitet. Dies geschieht durch Pixelverdopplung (edge repeat) der Randbereiche. In Abbildung 6.9 sieht man die Simulation der Pixelverdopplung. Der gesamte Verarbeitungsprozess wird nun mit einem größeren Bild

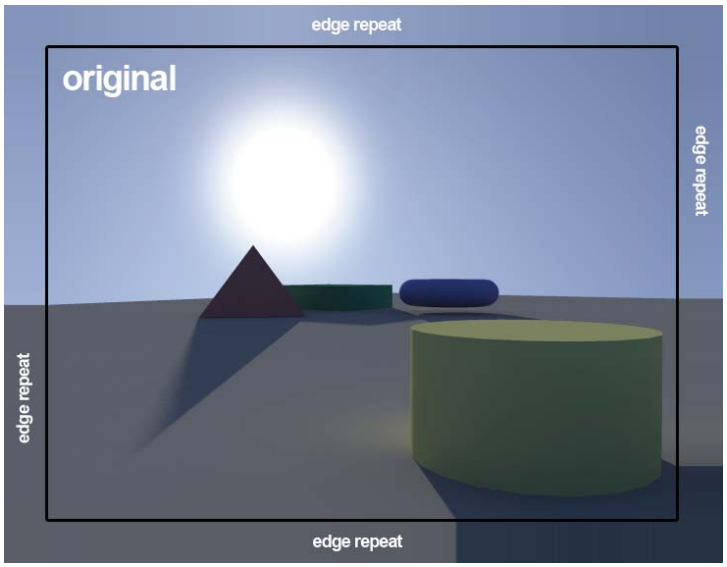


Abbildung 6.9: Edge repeat bei einem Dof-Rendering

durchgeführt. Dieses wird intern an seinen Rändern auch wieder Schwarzverläufe aufweisen, diese ragen jedoch nicht mehr in das eigentliche Ausgangsbild hinein. Nachdem das Bild fertig verarbeitet ist, wird das Bild entsprechend der ursprünglichen Größe aus dem *frameBuffer2* herausgeschnitten und in den *frameBuffer1* übertragen.

Zu diesem Zeitpunkt ist das visuelle Resultat des Filters zufriedenstellend. Die Verarbeitungsdauer liegt bei einem Bild in 640x480 Auflösung jedoch noch im zweistelligen Sekundenbereich[6]. Dies liegt an den vielen Iterationsschritten, die der Algorithmus durchlaufen muss. Nimmt man an, dass durchschnittlich jeder Pixel einen CoC-Wert von 4 aufweist, so werden bei einem Bild mit der Auflösung von 640x480 c.a. 7,6 Millionen Iterationsschritte verarbeitet[7]. Inhalt dieser Schritte ist z.b. eine für den Prozessor umständlich zu rechnende Wurzelberechnung, sowie Quadratberechnung, Addition und Multiplikation. Folgende Maßnahmen wurden unternommen, um die Verarbeitungszeit zu minimieren:

Lookup-Tables Sämtliche CoC- und Wurzelwerte werden einmalig berechnet und in einem Array (Lookup-Table) abgespeichert. Während der Iterationsschleife werden die Wurzelwerte nicht erneut berechnet, sondern aus der Lookup-Table übernommen, was die Rechenintensität deutlich verringert.

Multithreading Das zu verarbeitende Bild wird in 4 gleichgroße Teile aufgeteilt und mit Hilfe von Threads parallel berechnet. Dabei hat jeder Thread seinen eigenen Framebuffer. Nachdem alle Threads fertig sind, werden die vier Framebuffer zusammenaddiert.

Single instruction, multiple data (SIMD) Hierbei werden die Befehlssätze aktueller Prozessoren verwendet, um Pixelwerte schneller mit einander zu verrechnen. Mit der Befehlssatzerweiterung SSE2 können bis zu vier Verarbeitungsschritte in einem Prozessorzyklus verarbeitet werden. Addiert man zwei RGBA-Werte miteinander, geschieht dies normalerweise durch Addition der einzelnen Komponenten R,G,B und A. Hierfür wären vier Prozessorzyklen von Nöten. Mit Hilfe von SSE2-Instruktionen können zwei RGBA-Werte in einem Prozessorzyklus miteinander addiert werden, indem ein RGBA-Wert als vierdimensionaler Vektor aufgefasst wird. Die Instruktionen der SSE-Familie funktionieren einen Prozessor zu einem Vektorprozessor um.

Mit diesen Optimierungsansätzen konnte die Verarbeitungsdauer unter gleich bleibenden Bedingungen von 22 Sekunden auf 0,3 Sekunden reduziert werden.

In einem letzten Schritt wird dem bisher fertigen Bild ein Bokeh-Effekt hinzugefügt. Das Wort Bokeh stammt aus dem Japanischen und bedeutet soviel wie „unscharf“ oder „verschwommen“. Unter einem Bokeh versteht man nach [MRD12] die visuelle Erscheinung der einzelnen Zerstreuungskreise (CoC) an ursprünglich sehr hellen, konzentrierten Lichtpunkten im Bild, welche sich nun in einem unscharfen Bildbereich befinden (siehe Abbildung 6.10, S.60). Die Implementierung basiert auf dem Vorschlag von [Pet11], welcher zuerst gebündelte Lichtpunkte im unbearbeiteten Bild ausfindig macht und die entsprechenden CoCs nach der Verarbeitung des Dof-Filters in ihrer Helligkeit verstärkt. Um die hellen Lichtpunkte ausfindig

[6] Core2Quad 2.1GHZ, 8GB Ram, Auslastung eines Kerns

[7] 640*480*(5*5) = 7680000, CoC = 4 beschreibt ein Quadrat von 5x5 Pixeln, welche durchlaufen werden müssen, um Pixel innerhalb des Kreises zu finden. Die Quadrate müssen eine ungerade Anzahl von Pixeln aufweisen, um einen Kreis optimal beinhalten zu können

Abbildung 6.10: Cineastische Verwendung des Bokeh Effekts, Bild entnommen aus dem Kurzfilm *Fatum*, Yaglu et al. (2012)

zu machen, wird von jedem Pixel sein Helligkeitswert errechnet[8] und mit dem durchschnittlichen Helligkeitswert seiner umliegenden 5x5 Pixel verglichen. Überschreitet die Differenz dieser beiden Werte einen bestimmten Faktor und besitzt der Pixel einen CoC Durchmesser von mindestens fünf Pixeln (es handelt sich also um einen unscharfen Bereich), so wird der CoC-Kreis an dieser Stelle erneut überzeichnet und in seiner Helligkeit verstärkt. Der Nutzer hat in den Filtereinstellungen die Möglichkeit, den Faktor der Stärke zu beeinflussen und die Ausprägung des Bokehs selbst zu bestimmen.

In der Realität ist die Form eines CoC von der Qualität des verwendeten Objektives abhängig. In dieser Arbeit wurde für die Darstellung eines CoC bewusst ein perfekter Kreis gewählt, da sich dieser programmatisch gesehen zeiteffektiver erzeugen lässt, als die der Wirklichkeit nachempfundenen Abbilder eines CoC (Pentagon, Hexagon,...).

Die grafische Benutzeroberfläche des Dof-Filters wurde auf Wunsch einiger Anwender dahingehend erweitert, dass sich die Fokusebene des Filters automatisch dem Kameraziel (Camera Aim) in Mayas Viewport anpasst. So muss der Anwender die Distanz zur Fokusebene nicht manuell eingeben, sondern kann die Option wählen, die Distanz zur Fokusebene automatisch ermitteln zu lassen. Diese Option wird im Filter Autofokus genannt und lässt sich bei Bedarf aktivieren. Der Anwender ist nun in der Lage, den Scharfpunkt in der 3D-Szeniere mit Hilfe des Kameraziels zu wählen. Die Abstand zwischen der Kamera und ihrem Ziel wird dabei im Hintergrund automatisch von einem Mel-Script berechnet.

[8] Nach [GW01]: Luminance = 0,299*R + 0,587*G + 0,114*B

Kapitel 7

Evaluation und Ergebnisse

In den vorangegangenen Kapiteln wurde eine Methode vorgestellt, Post Production Filter in den Renderprozess von Mental Ray zu integrieren und die Einstellungen im Renderview grafisch zu visualisieren. Anhand dieser Implementierung werden im Folgenden subjektive und objektive Messdaten ermittelt, um die gestellten Fragen und die gesetzten Ziele dieser Arbeit beantworten und auswerten zu können. Dabei soll vor allem die Frage geklärt werden, ob die Integration von Post Production in eine 3D-Software bei den Nutzern auf Akzeptanz stößt und ob die Verwendung des zuvor vorgestellten Plugins Zeitersparnis in den Arbeitsprozessen ermöglicht.

7.1 Erhebung subjektiver Messdaten anhand von Umfragen

Im Folgenden wurden drei verschiedene Interessengruppen mit Hilfe eines Fragebogens befragt. Zu diesen Interessengruppen gehören zum einen Studenten, welche sich zu diesem Zeitpunkt in der Ausbildung befinden und sich zum ersten Mal mit 3D-Animationspaketen auseinander setzen. Als zweite Gruppe wurden Personen aus einem privaten Umfeld ausgewählt, welche sich hobbymäßig mit dem Thema 3D beschäftigen. Zuletzt wurden als dritte Gruppe Personen befragt, welche professionell 3D-Animationspakete nutzen.

Die Unterscheidung der Ergebnisse in Bezug auf die Analyse der Umfrage ist wichtig, da diese Interessengruppen zum einen auf verschiedenen Schwierigkeitsniveaus arbeiten und zum anderen verschiedene Erwartungen an ein Softwareprodukt haben. Zu diesen Erwartungen gehört z.B. die Produktivität einer Software und wie schnell man mit ihr Ergebnisse erzielen kann. Im professionellen Bereich ist dies ein entscheidender Faktor wohingegen im universitären Umfeld viel Zeit mit dem Erlernen von Software verbracht werden kann.

An der Umfrage haben 10 Studenten, 14 Hobbyentwickler und 9 professionell tätige Personen teilgenommen. Die Messdaten werden in Prozentzahlen angegeben, um einen Vergleich der Gruppen zu ermöglichen. Jeder Person wurde das in dieser Arbeit entwickelte

Plugin (in den Umfragen *Tool* genannt) entweder vom Autor in Form einer Live- oder Videopräsentation vorgestellt oder er konnte es selber installieren und testen.

Während der näheren Beschreibung der Messdaten wird vermehrt auf die Antworten der Studenten und professionell tätigen Personen eingegangen. Die Wertebereiche der Personen aus dem Hobbybereich liegen zumeist genau zwischen den Werten der Studenten und professionell tätigen Personen.

In einem ersten Schritt werden Messdaten zur Häufigkeit des Einsatzes von Post Production-Software erhoben und ferner auch erfasst, welche Hauptmerkmale einer Post Production-Software überhaupt verwendet werden. Damit wird ein Überblick geschaffen, in welchen Dimensionen sich die Interessengruppen mit Post Production beschäftigen. Dem Diagramm

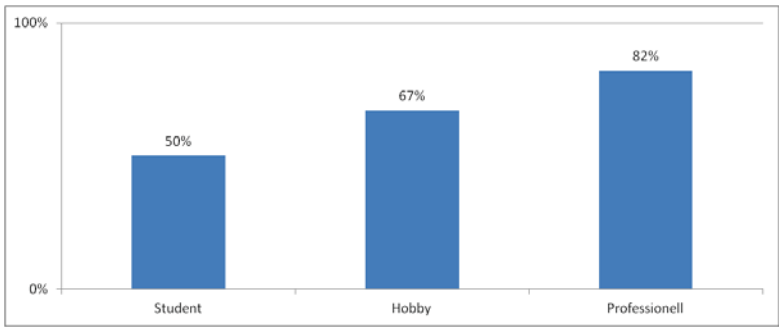


Diagramm 7.1: Häufigkeit der Verwendung von Post Production

7.1 liegt die Frage zugrunde, wie oft die Nutzer Post Production-Software verwenden. Dabei wurde den Befragten eine Auswahl von 0% bis 100% in zehn Prozentschritten vorgegeben. Das Diagramm gibt den gemittelten Wert aller Antworten wieder. Dabei ist festzuhalten, dass die Verwendung von Post Production vom Grad der Professionalität abhängig ist. Mit zunehmenden Wissensstand der Nutzer wird auch häufiger Gebrauch von Post Production-Software gemacht. Bei der Befragung professioneller Anwender ist auch anzumerken, dass 7 von 9 Personen angegeben haben, Post Production-Software in *jeder (100%)* ihrer Produktionen zu verwenden.

Um im späteren Verlauf dieser Arbeit Aussagen über mögliche Weiterentwicklungen des Plugins treffen zu können, werden in den nächsten 6 Diagrammen (7.2 bis 7.7) Aussagen zu dem Nutzungsverhalten von Post Production-Software getroffen. Hierbei ist wichtig festzustellen, wie oft 2D-Filter angewendet werden, wie oft mit mehreren Ebenen gearbeitet wird, wie oft Motion Tracking und Color Keying (Bluescreen) angwendet wird, in welchen Ausmaßen Kompositionen miteinander verschachtelt werden und wie oft Text auf einer Animation platziert wird.

Auch hier fällt auf, dass mit zunehmenden Grad der Professionalität mehrere Bereiche der Post Production öfters verwendet werden. Verwenden fast 80% der kommerziell wirkenden Nutzer 2D-Filter *in jedem ihrer Projekte* so geben dies nur 20% der Studenten an. Weitere 20% der Studenten geben an, Post Production-Software gar nicht zu verwenden. Dies hängt

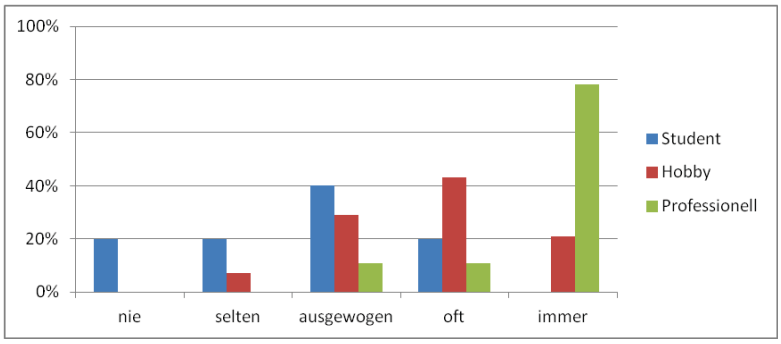


Diagramm 7.2: Häufigkeit der Verwendung von 2D-Filtern in der Post Production

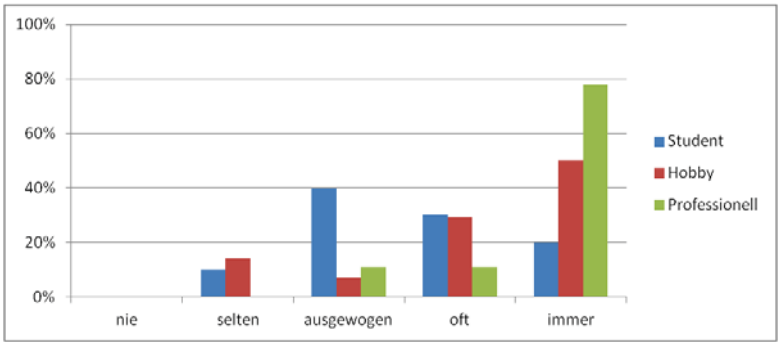


Diagramm 7.3: Häufigkeit der Verwendung von Ebenen in der Post Production

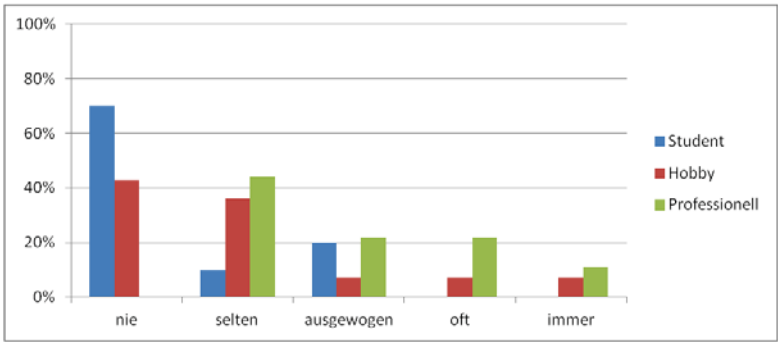


Diagramm 7.4: Häufigkeit der Verwendung von Motion Tracking in der Post Production

sicherlich damit zusammen, dass sie sich mit dieser Art von Software noch nicht auseinander gesetzt haben.

Die nächsten Fragen beschäftigen sich mit der Akzeptanz des vorgestellten Plugins. Dabei soll jede Person Aussagen darüber treffen, ob sie sich vorstellen kann, das Plugin in einer eigenen Produktion zu verwenden und in welchen Umfeld sie sich den Einsatz des Plugins generell vorstellen kann. Die Diagramme 7.8 bis 7.11 zeigen die erhobenen Messdaten.

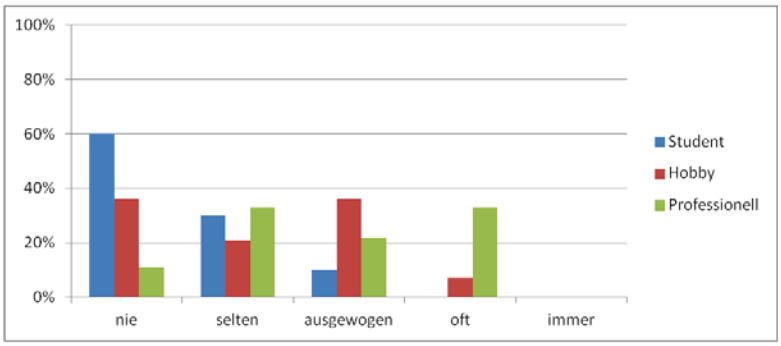


Diagramm 7.5: Häufigkeit der Verwendung von Color Keying in der Post Production

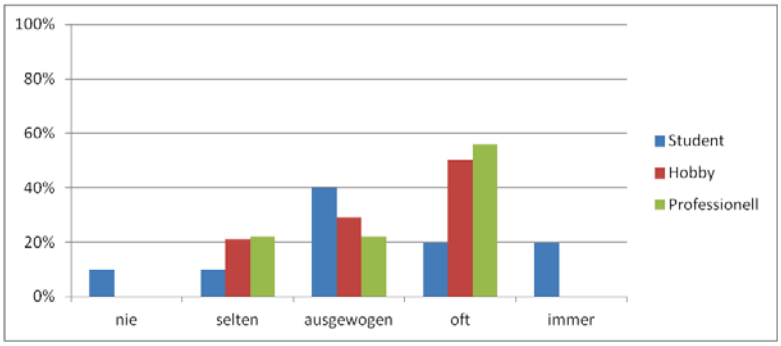


Diagramm 7.6: Häufigkeit der Einbettung von Text in der Post Production

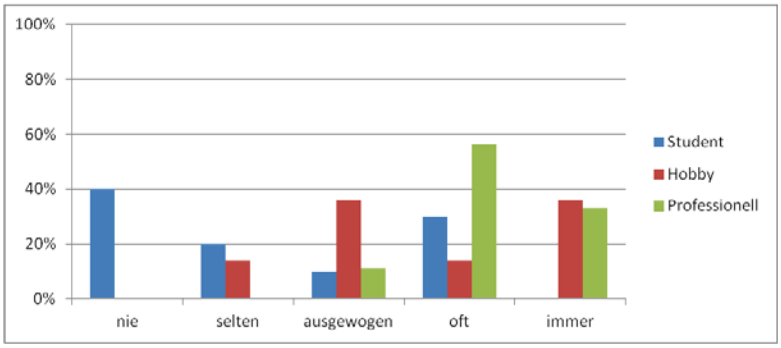


Diagramm 7.7: Häufigkeit der Verwendung von verschachtelten Kompositionen in der Post Production

Im Allgemeinen fällt bei den vier Diagrammen auf, dass die wenigsten Personen den Einsatz des Plugins komplett verneinen. Die kritischen Stimmen, welche in dem Plugin keine Verwendung finden, liegen meistens um die 20%. Gerade im Bereich der Ausbildung würden im Mittel 75% der drei Interessengruppen das Plugin nutzen. 61% unterstützen das Plugin im Hobbybereich und nur noch 19% würden das Plugin im kommerziellen Sektor *definitiv* nutzen. Der Großteil antwortet hier jedoch zu 72% mit der Antwort *Vielleicht* und nur 9%

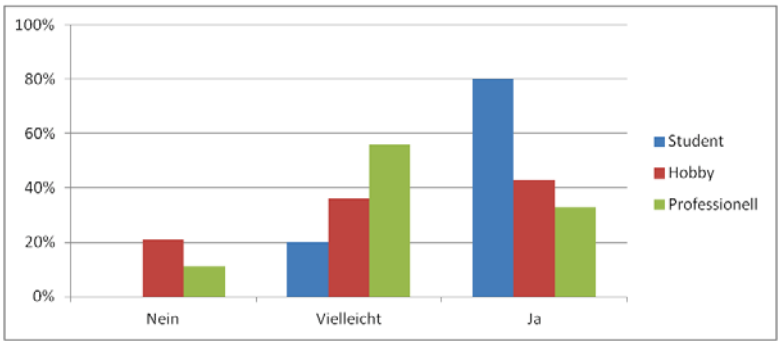


Diagramm 7.8: Können Sie sich vorstellen, das Tool in Ihrem Projekt zu verwenden?

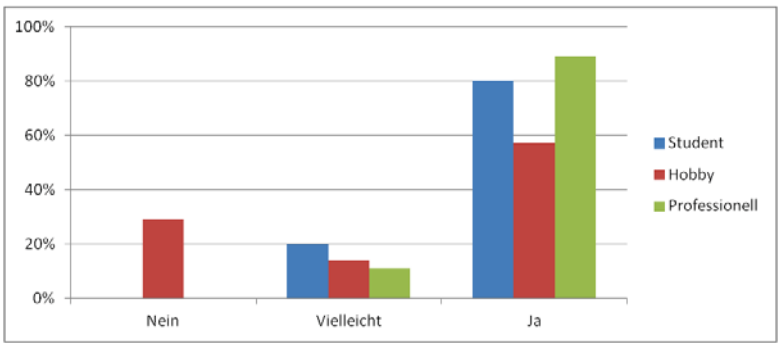


Diagramm 7.9: Verwendung des Tools im Einsatzgebiet der Ausbildung

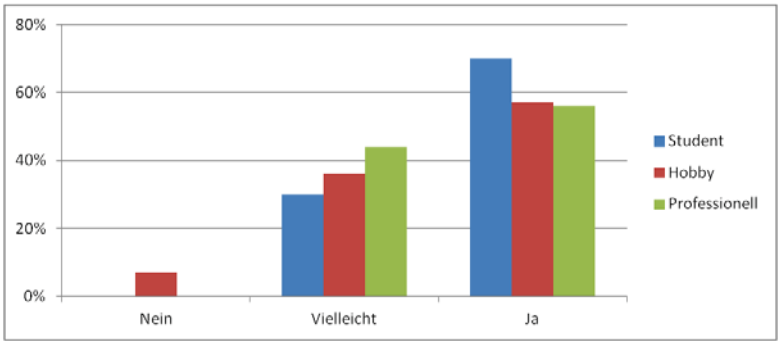


Diagramm 7.10: Verwendung des Tools im privaten Hobbybereich

schließen die Nutzung vollständig aus.

Eine weitere wichtige Fragestellung lässt Aussagen darüber treffen, in welcher Form das Plugin genutzt werden könnte. Zum einen lässt sich das Plugin dazu nutzen, Previews zu erzeugen und zum anderen die angewendeten 2D-Filter direkt in das Rendering zu integrieren. Diagramm 7.12 lässt erkennen, dass alle drei Interessengruppen einen sehr starken Nutzen in der Preview-Erzeugung sehen. Dieselbe Meinung vertitt auch die studentische Gruppe in

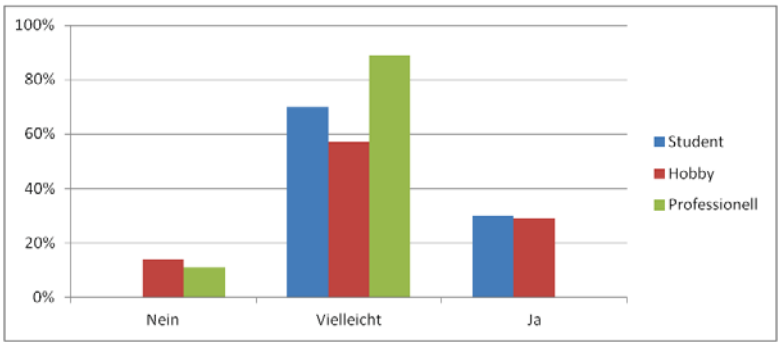


Diagramm 7.11: Verwendung des Tools im professionellen Einsatzgebiet

Bezug auf die Integration der Filter direkt in den Renderprozess. Die kommerziell agierenden Personen teilen diese Meinung nicht. Nur 20% würden die Filter in den Renderprozess integrieren. Im Schnitt über die Werte aller drei Interessengruppen glaubt jeder Zweite, dass die Verwendung des Plugins das Projekt schneller zu einem Ziel führt.

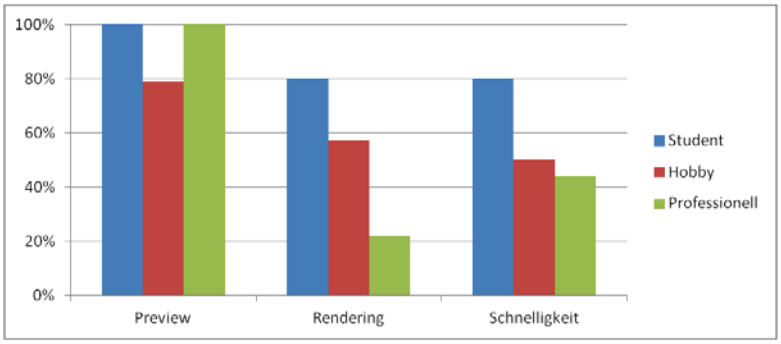


Diagramm 7.12: Einsatz des Tools unter bestimmten Aspekten

In einem dritten Abschnitt wurden den Personen allgemeine Fragen auf einem höheren Abstraktionsniveau zu dem Thema Integration von Post Production in eine 3D-Software gestellt. Dabei beziehen sich die Fragen nicht mehr auf das vorgestellte Plugin, sondern unterstellen eine durchweg fertig gestellte Lösung, mit welcher sich Post Production in 3D-Software betreiben lässt. Diagramm 7.13 zeigt, dass 70% der Studenten Post Production direkt in einer 3D-Software betreiben würden. Diesen Weg würden jedoch nur 22% der kommerziell wirkenden Personen einschlagen. 44% dieser Personen sehen auch gar keine Möglichkeit einer Vereinigung von Post Production in eine 3D-Software. In Diagramm 7.14 ergibt ein Mittelwert über die Antworten aller Interessengruppen jedoch aus, dass 79% der Personen der Meinung sind, mit dieser Methode dennoch Zeit sparen zu können.

In einem letzten Schritt soll sich ein Bild davon gemacht werden, welche der zahlreichen vorhandenen 2D-Filter von Nöten sind, um die Post Production innerhalb einer 3D-Software zufriedenstellend abzudecken. Diagramm 7.15 (S. 68) zeigt eine Übersicht der Filter, welche

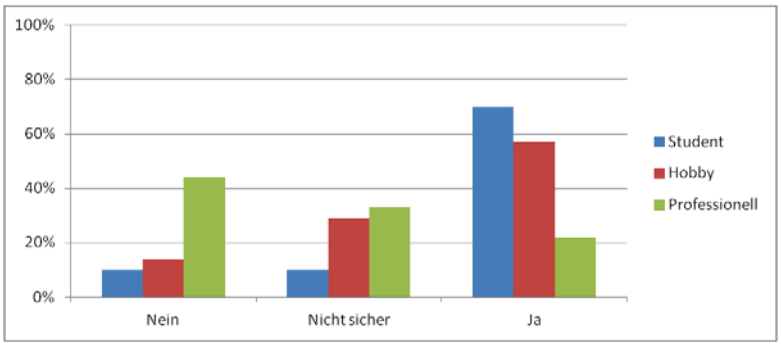


Diagramm 7.13: Integration von Post Production in 3D-Software

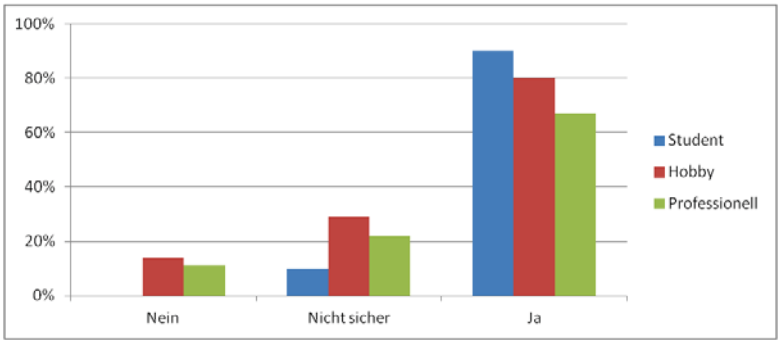


Diagramm 7.14: Zeitersparnis

sich die Personen vorstellen können, in den Renderprozess fest zu integrieren. Diagramm 7.16 (S. 69) sind die von den Personen gewünschten Filter, welche sie zu Preview-Zwecken einsetzen würden. Generell heben sich hierbei folgende Filter hervor: Depth of Field, Blur, Levels, Hue & Saturation, Gamma / Pedestrial / Gain, Exposure, Curves, Color Balance und Brightness & Contrast. Personen aus dem professionellen Bereich sehen hauptsächlich den Depth of Field-Filter dazu in der Lage, in den Renderprozess integriert zu werden und wenden die anderen Filter weiterhin in einer getrennten Post Post Production-Umgebung an. Im Bereich der Preview-Erzeugung steigt das Interesse an den anderen Filtern jedoch stark an.

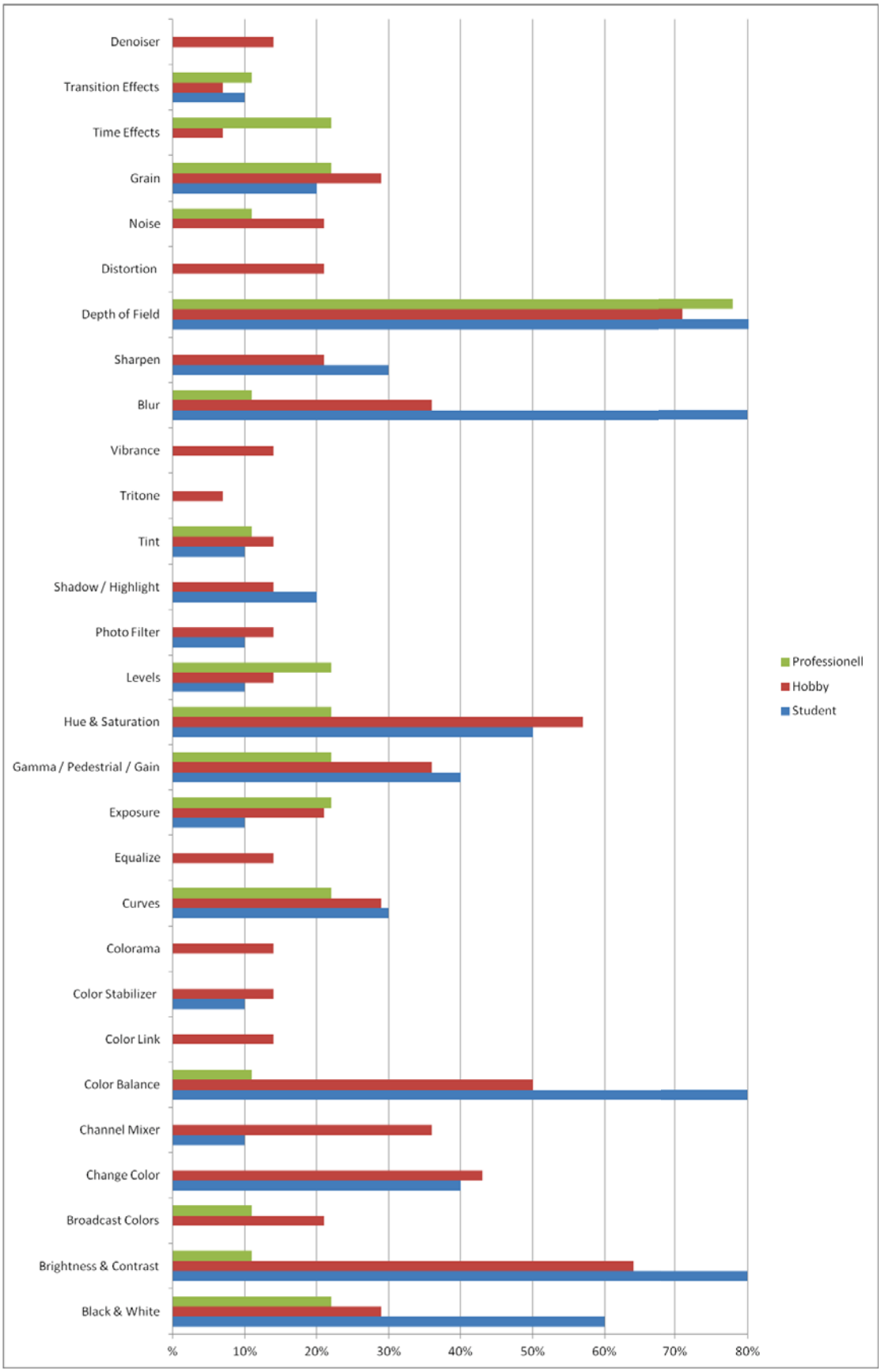


Diagramm 7.15: Wunschfilter zur Integration in den Renderprozess

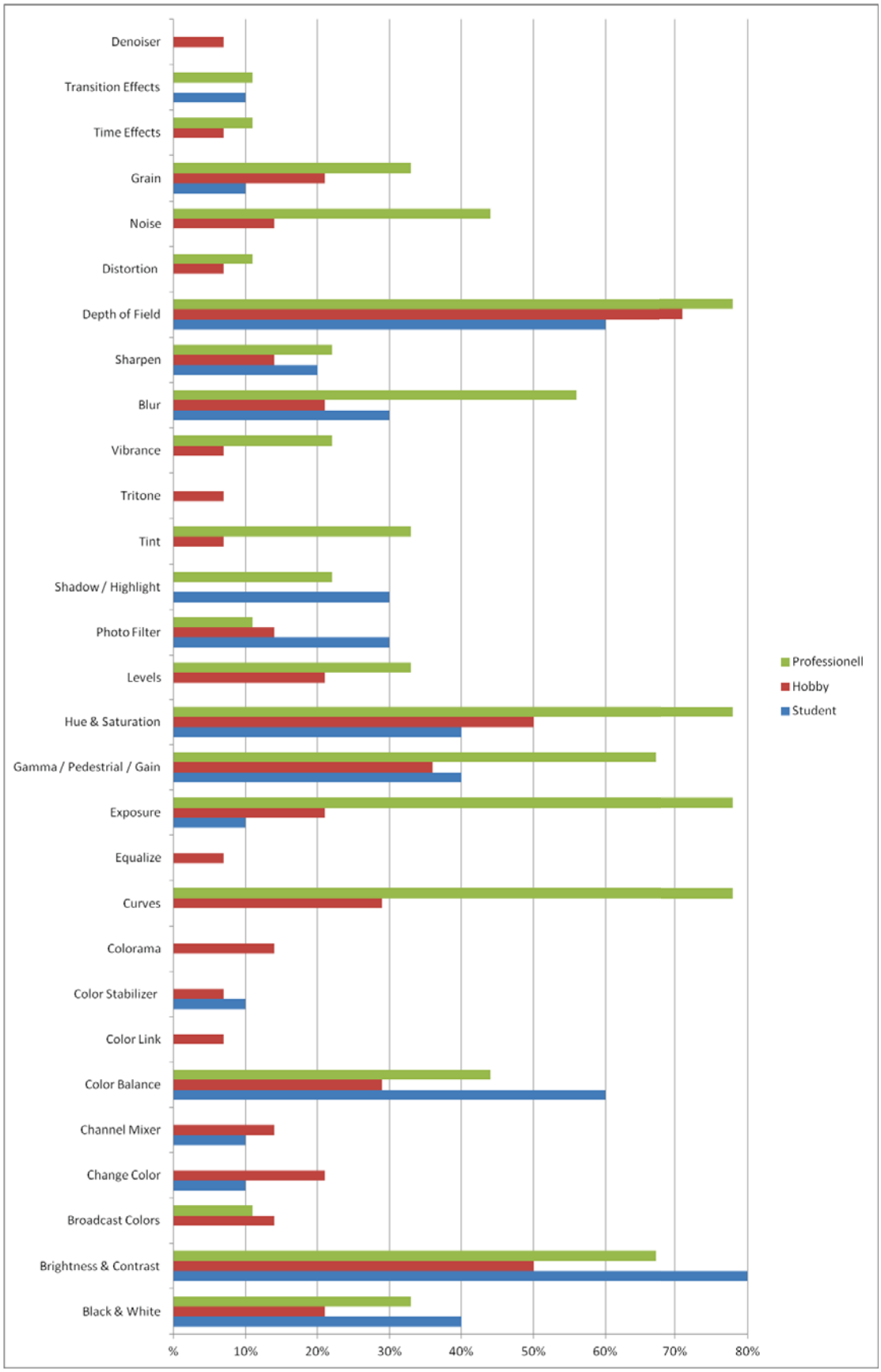


Diagramm 7.16: Wunschfilter zum Erstellen von Previews im Renderview

7.2 Auswertung objektiver Messwerte

In diesem Unterkapitel werden objektive Messwerte ermittelt und mit anderen Systemen verglichen. Dabei wurden alle Werte auf demselben Computer erstellt. Dieser besteht aus einem Intel Core 2 Quad Q6600 Vierkernprozessor mit 2.1 GHz, 8 GB Arbeitsspeicher und läuft auf einem Windows 7 64Bit Betriebsystem.

In einem ersten Schritt werden die Renderzeiten der zweidimensionalen Depth of Field-Implementierung mit dem Mental Ray-eigenen dreidimensionalen Depth of Field-Effekt verglichen. Dabei wurde eine 3D-Szene mit mehreren einfachen Geometrien versehen und dreimal mit verschiedenen Einstellungen in Mental Ray gerendert. Die gerenderte Animation ist fünf Sekunden lang und besteht aus 120 einzelnen Bildern bei einer Bildrate von 24 Bildern pro Sekunde. Gerendert wurde einmal ohne einen Depth of Field-Effekt, um eine Referenzzeit zu erhalten. Das zweite Rendering beinhaltet die Mental-Ray eigene Dof-Implementierung und das dritte Rendering wurde mit dem in dieser Arbeit implementierten Dof-Lösung verarbeitet. Die 3D-Szene wurde dabei so konstruiert, dass die eigene Dof-Implementierung lange Berechnungszeiten durchlaufen muss, um einen möglichen *worst case* aufzuzeigen.

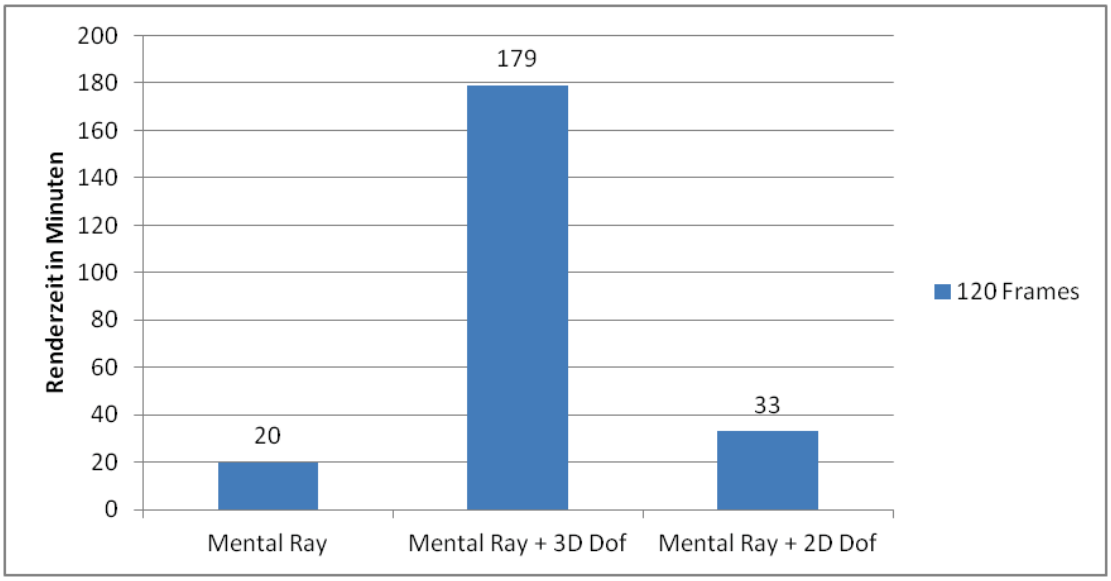


Diagramm 7.17: 1. Vergleich Renderzeiten 2D-3D-Dof

Wie im Diagramm 7.17 zu erkennen ist, benötigt das Rendering ohne Dof-Implementierung für 120 gerenderte Frames 20 Minuten. Mit der Dof-Implementierung von Mental Ray vergrößert sich sich die Renderzeit auf insgesamt 179 Minuten. Bei Verwendung des vorgestellten zweidimensionalen Dof-Filters vergrößert sich die Renderzeit auf insgesamt 33 Minuten.

In einem zweiten Test wurde eine komplexere 3D-Szene mehrfach gerendert, für welche Mental Ray ohne Dof pro Bild circa 5,5 Minuten braucht. Die Objekte sind hierbei für

die eigene zweidimensionale Dof-Implementierung in günstiger Art und Weise angeordnet. Für eine Sekunde Filmmaterial (24 Bilder) benötigt Mental Ray ohne Dof 132 Minuten.

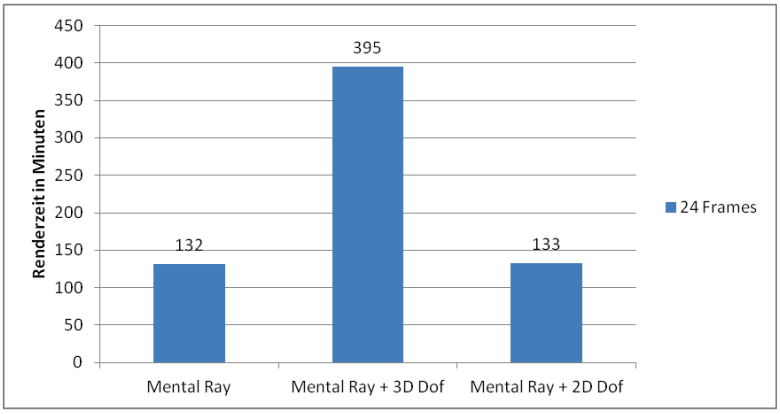


Diagramm 7.18: 2. Vergleich Renderzeiten 2D-3D-Dof

Dieses Mal nimmt die eigene Dof-Implementierung insgesamt nur eine Minute mehr an Renderzeit in Anspruch. Im Vergleich hierzu verdreifacht die Mental Ray Dof-Lösung die gesamte Renderzeit auf 395 Minuten.

Im nächsten Schritt wird ermittelt, wie viel Arbeitszeit innerhalb der einzelnen Systemen benötigt wird, um Post Production-Filter auf ein Rendering anzuwenden. Dabei kommt der Aspekt zum tragen, wie schnell man ein Preview seiner Arbeit erlangen kann. Verglichen wird die eigene Lösung u.a. mit dem konventionellen Weg. Bei diesem exportiert man das gerenderte Bild zuerst in eine Bilddatei und überführt diese in eine klassische Post Production-Umgebung. In diesem Fall wurde Adobe After Effects gewählt.

Weiterhin werden Vergleiche zu der in V-Ray und Blender integrierten Post Production-Lösung aufgezeigt. Bei allen Vergleichen wurde die eigentliche Renderzeit abgezogen. Jedes der vier Systeme muss zumindest einmal auf ein bereits fertig gerendertes Bild zugreifen. Die gemessenen resultierenden Zeiten entsprechen dem Arbeitsaufwand des Nutzers. Dabei wurden alle Messungen dreimal durchgeführt und ein Mittelwert gebildet, welcher hier vorgestellt wird. Vor den Messungen gab es Probedurchläufe, um die optimale Vorgehensweise zu verinnerlichen. In den Messergebnissen steckt also die optimale Arbeitszeit, um die vorgestellten Schritte durchzuführen.

Zuerst wurde gemessen, wie lang man benötigt, die Post Production-Umgebung einzurichten und den ersten Filter (Hue & Saturation) dem Bild hinzuzufügen. Im Fall von After Effects kommt man hier auf ungefähr 62 Sekunden. Dabei muss zuerst das Bild aus dem Renderview von Maya in eine Datei gespeichert werden. Weitere Sekunden verbraucht dann der Start von After Effects sowie das Importieren der Bilddatei in eine neue Komposition. Nach diesen Schritten kann dann der Effekt schnell auf das Bild angewendet werden.

In V-Ray muss man zuerst in den Rendereinstellungen den V-Ray-eigenen Renderview (Framebuffer) aktivieren. Nach dem Rendern hat man dann die Wahl, entweder durch Akti-

	After Effects	V-Ray	Blender	Maya Plugin
Ersteinrichtung	62s	20s-30s	110s	5s
Effekt hinzufügen	3s	6s	20s	2s
Inhaltliche Änderung	16s	0s	0s	0s
Dof einrichten	380s	nicht möglich	25s	10s
Fokusebene	manuelle Eingabe		manuelle Eingabe	Camera Aim
Effektvielfalt	hoch	mittel	hoch	gering
Batch 3D-Rendering	nein	nein	ja	ja

Abbildung 7.1: Vergleich der Arbeitszeit bei verschiedenen Systemen

vierung einiger Optionen die Helligkeit des Bildes zu manipulieren. Dies dauert insgesamt 20 Sekunden. Der Nutzer kann das Bild zudem auch gleich in den externen Pdplayer übergeben und dort auf eine größere Vielzahl von Filtern anwenden zu können (30s).

Bei Blender gestaltet sich die einmalige Einrichtung komplexer. Über mehrere Optionen in diversen Untermenüs wird die Post Production-Umgebung zunächst aktiviert. Dann ist das Fensterlayout umzustellen, um die verschiedenen Editoren, die wichtig für die Filterbearbeitung sind, auf einen Blick zu haben. In einem letzten Schritt muss ein Node-System aufgebaut werden. Dafür wird eine Filter-Node dem Editor hinzugefügt und ihre Ein- und Ausgänge entsprechend verbunden. Mit dem hier vorgestellten Maya-Plugin wird die geringste Zeit für die Ersteinrichtung benötigt. Man startet das Plugin zuerst über ein Symbol in der Toolpalette, wählt die zu bearbeitende Kamera und fügt einen Filter aus einer Liste hinzu.

Ist die Post Production bereits in allen Systemen eingerichtet, so gestaltet sich das Hinzufügen von neuen Filtern bei allen vier Kandidaten sehr schnell. In Blender werden zusätzliche Schritte benötigt, da hier wieder neue Nodes hinzugefügt und verbunden werden müssen.

Ändert sich der Inhalt des Renderings, z.B. durch eine Verschiebung der Kamera, und man möchte die zuvor eingestellten Effekte auch auf das neue Bild anwenden, so geschieht dies bei allen Systemen, bis auf After Effects, automatisch. Die Renderzeit für das neue Bild wird hier nicht berücksichtigt. Um das neue Rendering in After Effects sichtbar zu machen, muss das Bild jedoch erst neu exportiert werden. Speichert man es unter dem selben Dateinamen wie zuvor, kann man in After Effects das Material neu laden und die zuvor eingestellten Filter wirken dann sofort auf das neue Rendering. Dies dauert circa 16 Sekunden.

Die erstmalige Einrichtung eines Dof-Filters verbraucht besonders in After Effects viel Zeit. After Effects braucht hier zusätzlich noch eine Tiefenmap, welche separat aus Maya heraus gerendert und gespeichert werden muss. Nachdem beide Bilder in After Effects importiert wurden, müssen diese auf zwei verschiedenen Ebenen platziert und miteinander in den Filtereinstellungen verknüpft werden. Mitsamt der Einstellung der korrekten Fokusebene verbraucht diese Prozedur mehr als 6 Minuten. In V-Ray und dem Pdplayer ist keine Möglichkeit vorgesehen, Depth of Field auf 2D-Basis anzuwenden. In Blender wird erneut

eine Node erstellt, welche doppelt (Tiefenmap) verbunden werden muss. In dem eigenen Maya-Plugin geschieht das Management der Tiefenmap im Hintergrund. Es reicht aus, den Dof-Filter hinzuzufügen und man kann sofort beginnen, die Einstellungen vorzunehmen. In After Effects und Blender wird die Fokusebene manuell eingegeben. Hier erreicht man das gewünschte Ergebnis entweder durch Entfernungsmessung oder durch Ausprobieren. Im Maya-Plugin besteht die Möglichkeit neben der manuellen Eingabe der Fokusebene, das *Camera Aim* der Maya Kamera als Fokus zu bestimmen. Damit wird der Schärfebereich direkt in der 3D-Szenerie gesetzt.

Ein Nachteil des Maya Plugins ist im direkten Vergleich die Anzahl der zu verwendenden 2D-Filter. Mit der Auswahl von drei Filtern liegt das Plugin weit hinter den anderen Systemen, befindet sich aber auch noch im prototypischen Stadium. Ein Vorteil wiederum ist, dass die angewendeten Filter bei Bedarf direkt in das 3D-Rendering integriert werden können. Bei Blender ist dies auch möglich, bei V-Ray nicht.

Ein weiterer Vorteil der Integration von 2D-Filtern in das Rendering erschließt sich im Vergleich des Speicherplatzbedarfs zwischen der konventionellen Vorgehensweise mit dem Maya-Plugin. Bei einer konventionellen Vorgehensweise wird unterstellt, dass die Animation in eine verlustfreie Einzelbildsequenz (Tiff) gespeichert wird. Diese wird in eine Post Production-Software (After Effects) importiert und anschließen bearbeitet. Nach diesem Schritt wird die Sequenz erneut in eine verlustfreie Einzelbildsequenz gespeichert und letztendlich einem Videoschnittprogramm übergeben. Durch Verwendung des vorgestellten Post

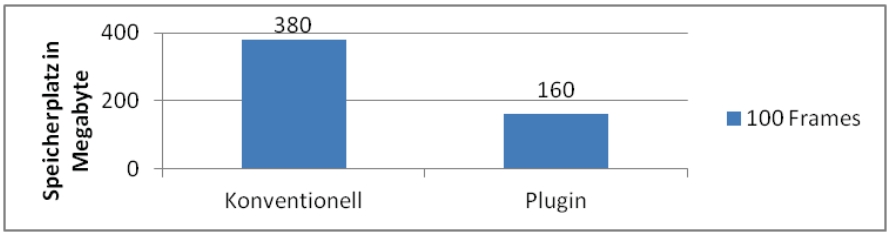


Diagramm 7.19: Speicherplatzersparnis bei Verwendung des Plugins

Production-Plugins für Maya, kann der Bedarf an Speicherplatz halbiert werden, wenn man davon ausgeht, dass der Nutzer After Effects nicht mehr zu verwenden braucht. Das Bildmaterial kann sofort nach dem Rendering dem Videoschnittprogramm übergeben werden. In Diagramm 7.19 wurde auch die zusätzlich zu speichernde Tiefenmap berücksichtigt, welche bei Verwendung eines Dof-Effektes benötigt wird.

7.3 Zusammentragung der Ergebnisse

In dieser Arbeit konnte aufgezeigt werden, dass es möglich ist, Filter auf 2D-Basis an den Renderprozess von Mental Ray in Autodesk Maya anzuhängen. Zudem wurde ein Weg vorgestellt, wie sich die 2D-Filter in Mayas Renderview direkt bearbeiten lassen, obwohl dieser

für eine Bildmanipulation nicht ausgelegt ist. Durch die Informationen, welche der Renderer Mental Ray beim Rendervorgang dem Entwickler zur Verfügung stellt, war es auch möglich, einen komplexen Depth of Field-Filter mitsamt einer Darstellung des Bokehs zu implementieren (siehe Abbildung 7.2). Die Vorteile der Renderzeiten mit der integrierten

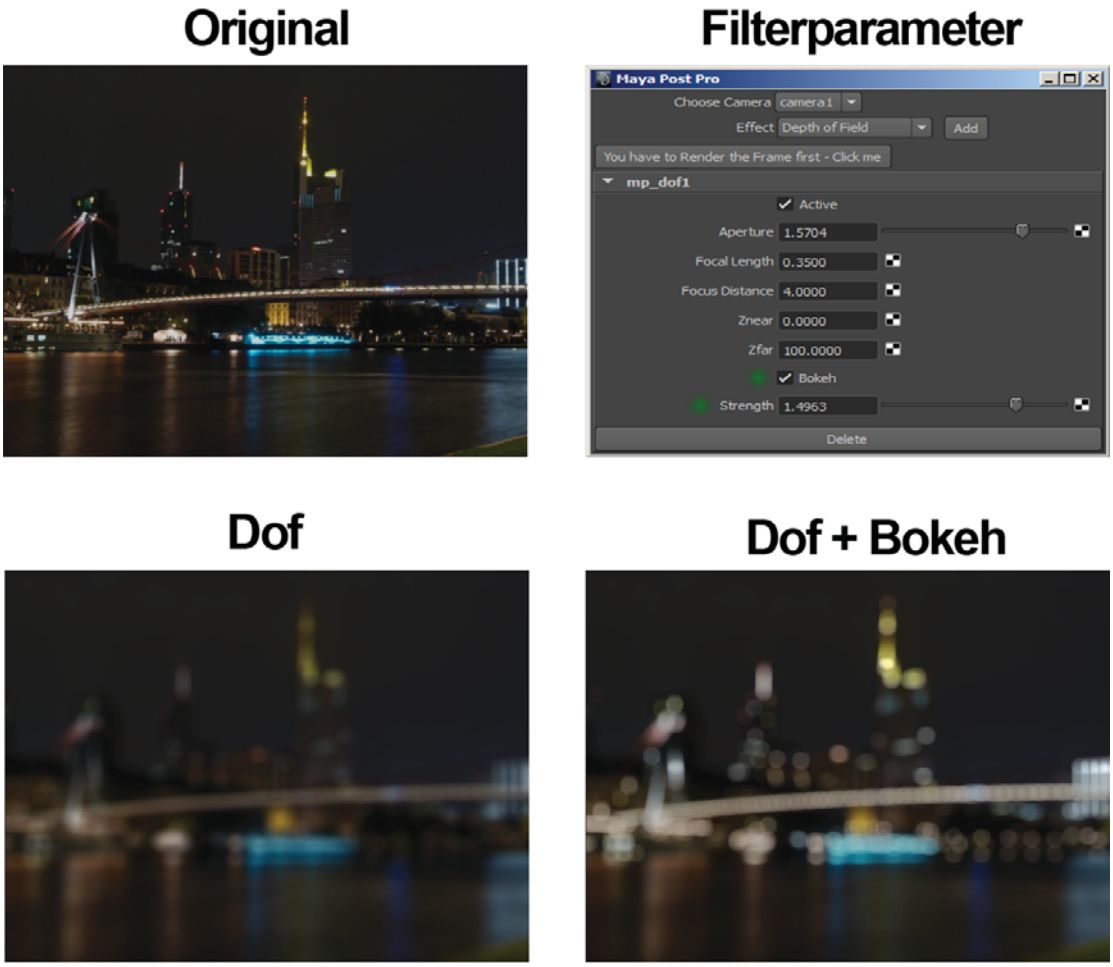


Abbildung 7.2: Auswirkungen des Bokeh-Effekts

zweidimensionalen Dof-Lösung sind sehr deutlich. Die Messdaten haben ergeben, dass sich die Renderzeiten um ganzzahlige Faktoren beschleunigen lassen. Auch zur Erzeugung von Previews lässt sich die vorgestellte Methode verwenden. In den Arbeitsabläufen ergibt sich eine Einsparung der Bruttozeit. Dies kommt besonders zum Tragen, wenn mit vielen Kameraeinstellungen previsualisiert und mit mehren Filtern in Kombination experimentieren werden soll, um bestimmte künstlerische Prozesse zu ermöglichen.

Bei der Auswertung der Fragebögen zeigt sich, dass es zwischen den verschiedenen Interessengruppen große Unterschiede bei der Verwendung von Post Production gibt. Studenten nutzen die Post Production meist weniger intensiv als professionell agierende Nutzer. Dafür

zeigt sich das Interesse an dieser Entwicklung bei den Studierenden sehr groß. Professionelle sehen diese Entwicklung eher skeptisch, was die feste Integration von 2D-Filtern in den Renderprozess anbelangt. Der Einzug des Depth of Field-Effektes in den Renderprozess wird hier gesehen. Zu Preview-Zwecken sind sich jedoch alle Interessengruppen einig. Mit diesem System kann viel Zeit eingespart werden und bietet eine gute Unterstützung bei der schnellen Erzeugung von Previews. In der Ausbildung sieht ein großer Prozentanteil einen berechtigten Einsatz, wo Post Production-Software nicht primär unterrichtet wird, sondern der Schwerpunkt auf die 3D-Programme gelegt wird.

Im Generellen herrscht dennoch eine Diskussion nach dem Nutzen der hier vorgestellten Lösung. Der Einsatz dieser Software ist abhängig von den Interessen und Vorstellungen der Anwender. Es gibt Nutzer, welche schon länger nach einer solchen Lösung gesucht haben. *„You know how slow 3d depth of field is? So the solution is a zdepth pass and a filter in After Effects, Nuke or whatever compositing package anyone uses. Why isn't there a 2d solution from within mental ray to do just that?“* Auf die Vorstellung des Maya Plugins hat dieser Nutzer aus einem Internetforum sehr positiv reagiert. *„Awesome. That's what I was talking about. Why wait for a post program? I don't use Blender, but the thing I like about it is that compositing can happen without leaving the 3D program.“* Aus seiner Aussage lässt sich auch heraus lesen, dass er generell eine Integration von Post Production in eine 3D-Software für möglich hält.

In einem anderen Beispiel, hat ein weiterer Nutzer Stellung zu dieser Entwicklung bezogen und die Möglichkeit zur Erstellung von schnellen Previews bekräftigt: *„I'm using After Effects to generate DOF and motion blur, so having a way to predict my outcome or maybe even implementing them inside Maya in some scenarios could be awesome.“*

Ein anderer Nutzer bezieht kritisch Stellung zur Integration von Post Production in eine 3D-Software: *„Nein, weil wenn ich mir die Benutzeroberfläche von beiden Programmen ansehe, fühle ich mich eh schon oft von den Optionen/Möglichkeiten überfallen und muss ewig suchen, um das zu finden was ich gerade an Werkzeug/Einstellung brauche. An dieser Stelle muss ich sagen, wenn beide Programme verbunden werden, daß ich den Wald vor lauter Bäumen nicht mehr finde ...“*

Man kann feststellen, dass jeder Nutzer für sich selber entscheiden muss, ob ihm das vorgestellte Plugin in seinen Arbeitsprozessen hilft oder nicht. Das Maya Plugin beherrscht momentan nur drei Filter. Weitaus wichtige Gebiete der Post Production fehlen noch in der Implementierung. Die Umfrageergebnisse haben jedoch gezeigt, dass das Plugins in durchaus berechtigten Anwendungsfällen einen Einsatz finden kann.

Abbildung 7.3: Visuelles Ergebnis des Maya-Plugins unter Verwendung des Depth of Field-Filters. Oben ist das Original zu sehen, unten das Ergebnis. Ferner wurde die Farbstimmung mit den beiden implementierten Farbkorrektur-Filtern optimiert. Im Ergebnis ist das Bokeh in Form von hellen Lichtkreisen deutlich zu erkennen. Die Anwendung der Filter auf das originale 3D-Rendering hat bei diesem Beispiel circa eine Sekunde an Rechenzeit benötigt.

Kapitel 8

Zusammenfassung

Post Production wird seit über 20 Jahren in allen professionellen 3D-Produktionen betrieben. Die Separation von 3D-Rendering und Nachbearbeitung bringt große Vorteile in den Arbeitsprozessen mit sich, da 2D-Bildalgorithmen sehr schnell auf fertig gerenderten 3D-Bildern angewendet werden können. Momentan wird die Post Production meist in eine zweite Software ausgelagert, welche auf die Anwendung spezialisiert wurde. In den vergangenen Jahren gibt es immer öfters Versuche, 2D-Processing in den 3D-Rendervorgang zu integrieren. Diese Versuche sind vor allem Standpunkt der Wissenschaft im Bereich des Spielsektors, wo aufwendige cineastische Effekte in eine zeitkritische Spielumgebung integriert werden sollen. Zu diesen Effekten gehört vor allem der Depth of Field-Effekt, welcher versucht, reale optische Systeme von Kameras zu imitieren. Damit wird es ermöglicht wichtige Bildbereiche von unwichtigen zu trennen, indem diese unscharf dargestellt werden.

Im Bereich der 3D-Animationspakete gibt es auch schon erste Ansätze zur Integration von Post Production-Aspekten. Da wäre V-Ray zu nennen, der bereits Belichtungskorrekturen auf einem fertigen Rendering anwenden kann und das Open Source-Programm Blender, welches schon große Fortschritte im Bereich Post Production gemacht hat.

Mit dieser Methode eröffnet sich auch ein neuer Bereich. Previews können sich schneller erzeugen lassen, wenn Programme mehrere Techniken vereinen. Im Filmbereich wurden Spiele-Engines bereits dazu verwendet, um schneller previsualisieren zu können. Dabei können frühzeitig genug Probleme erkannt und rechtzeitig eingegriffen werden.

Das renommierte 3D-Animationspaket Autodesk Maya besitzt mit seinem von Haus aus mitgelieferten Renderer Mental Ray noch keine Möglichkeit, Aspekte der Post Production zu integrieren. Ziel war es daher zu prüfen, ob es Möglich ist, mit Hilfe der programmierfreundlichen Entwicklungsumgebung 2D-Filter in den Rendervorgang zu integrieren und im Renderview zu previsualisieren. Weiterhin sollte durch einen Umfragebogen geklärt werden, ob die Implementierung dieser Idee bei den Nutzern auf Akzeptanz stößt und ob sie glauben damit Zeit zu sparen.

Es hat sich herausgestellt, dass durch die Outputshader-Technologie von Mental Ray

die Integration von 2D-Filtern in den Renderprozess möglich ist. Während diesem Prozess kann der Programmierer auf alle notwendigen Daten zugreifen, um die Filter entwickeln zu können. Um die Möglichkeit der Previsualisierung und der Einstellung von 2D-Filtern im Renderview zu ermöglichen, wurde ein eigener abstrakter 2D-Renderer programmiert, welcher die Renderergebnisse von Mental Ray aufgreift, die Filter auf diese anwendet und schließlich im Renderview platziert.

Eine eigens entwickelte Benutzeroberfläche in Maya versteckt dabei die Prozesse vor dem Anwender. Wenn er seinem Rendering Filter hinzufügt, erhält er sofortige Rückmeldung im Renderview über seine getätigten Filtereinstellungen. Implizit wird bei seiner Arbeit im Hintergrund der 2D-Renderer aufgerufen, wenn sich das Bild visuell verändert. Dieser zeigt das veränderte Bild sofort im Renderview an. Dabei wird auf erneutes 3D-Rendern verzichtet, welches den großen Geschwindigkeitsvorteil darstellt.

In einem weiteren Schritt wurden prototypisch drei in der Post Production oft genutzte Filter implementiert. Zwei Farbkorrektur-Filter und der komplexere Depth of Field-Filter. Durch objektive Messwerte konnte gezeigt werden, dass die Verwendung dieses Filters anstatt der dreidimensionalen Version von Mental Ray erhebliche Zeitvorteile mit sich bringt. Dies wurde auch durch die Umfragen bestätigt. Eine Mehrzahl der Befragten bestätigt, dass sich mit dem entwickelten Tool Zeit einsparen lässt.

Es zeigt sich auch, dass es Unterschiede in der Akzeptanz und dem Nutzen der hier vorgestellten Idee gibt. Wo noch studierende Personen einen großen Nutzen in der Entwicklung sehen, auch in Bezug auf die feste Integration von 2D-Filtern in den Renderprozess, stehen professionell und professionell agierende Personen der Entwicklung skeptisch gegenüber. Eine Vereinigung von 3D Production und Post Production wird hier nicht gesehen. Vielmehr ließe sich die Entwicklung dazu nutzen, schneller ein Preview von ihrer 3D-Animation erzeugen zu können. Ausnahme ist jedoch der Depth of Field-Effekt. Hier wird die Möglichkeit einer Integration dieses Effektes gesehen.

Kapitel 9

Ausblick

Diese Arbeit konnte aufzeigen, dass die Bearbeitung von 2D-Filtern aus der Post Production in Maya umsetzbar ist. Dabei befindet sich die Implementierung noch in einem prototypischen Zustand. Es wurden exemplarisch drei Basisfunktionen integriert. Die Umfrage hat ergeben, dass folgende, noch nicht implementierte 2D-Filter besonders häufig eingesetzt werden: Blur, Levels, Gamma / Pedestrial / Gain, Exposure, Curves und Brightness & Contrast. Mehrere Anwender haben zusätzlich noch darauf hingewiesen, den Vignetten-Effekt anzubieten. Diese Filter ließen sich in einem weiteren Verlauf der Entwicklung implementieren.

In Hinblick auf den Depth of Field-Effekt offenbart sich eine weitergehende Aufgabenstellung. Die von Mental Ray zur Verfügung gestellte Tiefenmap berücksichtigt bei den Objektoberflächen keine Darstellung von Transparenz und Reflexionen. So kommt es bei der Verarbeitung durch den Algorithmus zu Fehlinterpretationen. Diagramm 9.1 vergleicht ein problematisches 3D-Rendering mit seiner Tiefenmap, die Mental Ray zur Verfügung stellt. Hier ist zu erkennen, dass alle Objekte gleich behandelt werden unabhängig davon, welche Eigenschaften die Oberflächenmaterialien besitzen. Diese Problematik ist heutzutage weiterhin Gegenstandpunkt der wissenschaftlichen Arbeiten und bedarf einer Lösung.

Das Maya-Plugin bettet die 2D-Filter direkt in das 3D-Rendering ein. Fertig gerenderte Animationen müssen bei Korrekturen mit hohen weiteren Zeitauflagen erneut gerendert werden. Dieses Problem ist so nicht zu lösen, jedoch ist es möglich, beim Rendern eine zweite Version der Animation abzuspeichern. Diese Version beinhaltet eine vom Plugin nicht verarbeitete Version des Renderings. Sie besteht aus dem originalen Mental Ray-Rendering. Auf diese Weise bleibt die Entscheidung dem Anwender überlassen, ob er die erste Version des Renderings letztendlich nutzt oder zu Preview-Zwecken verwenden. Ihm bleibt jedoch weiterhin die Wahl, mit der zweiten unangetasteten Version in eine Post Production-Umgebung zu gehen. In diesem Fall hätte er durch die erste Version bereits eine vorzeigbare Filmsequenz, arbeitet dann aber auf konventionelle Art und Weise weiter.

Aus Usability-Gründen macht es Sinn, die einer Kamera zugewiesenen Effekte mitsamt den Einstellungen und der Reihenfolge in *Presets* abspeichern zu können. Besteht eine längere Filmsequenz aus mehreren Kameraeinstellungen, so müssen die Effekte momentan für jede

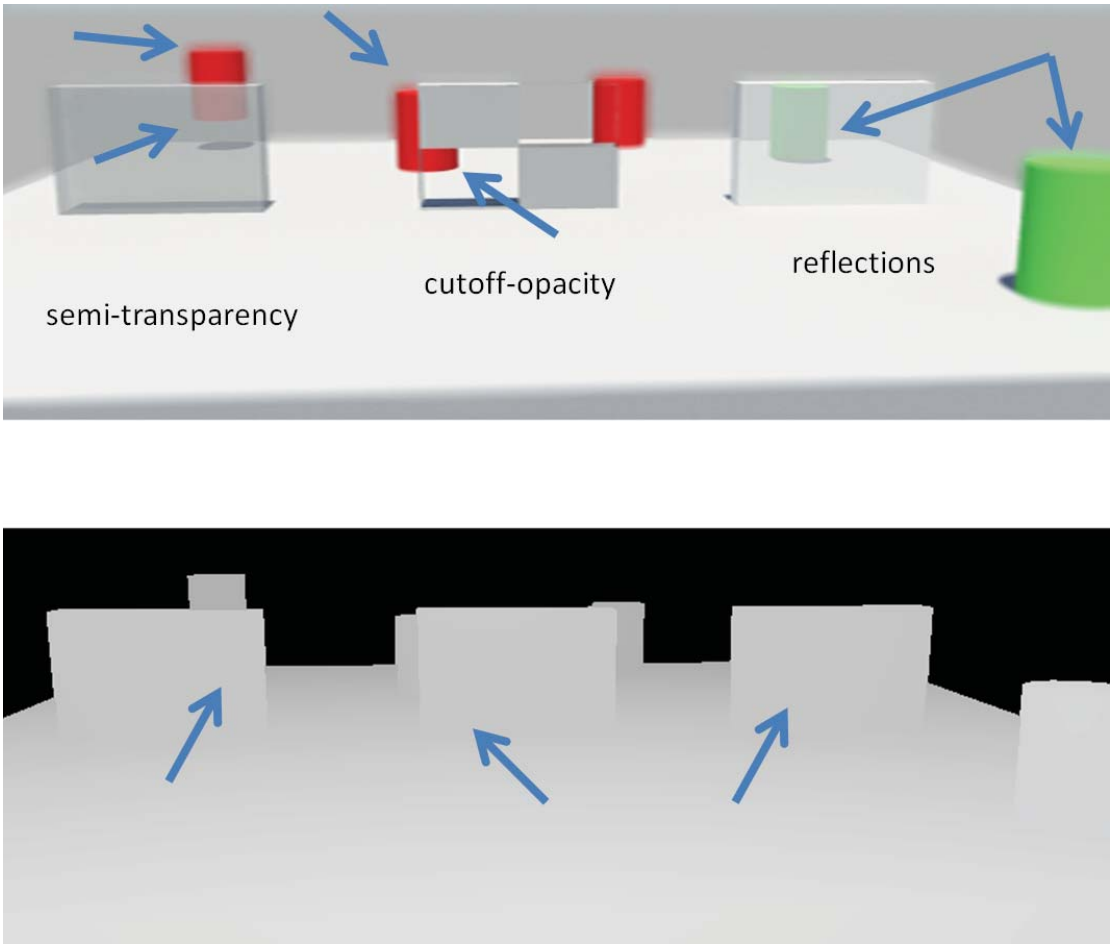


Abbildung 9.1: Problematik einer Tiefenmap bei Verwendung von Transparenz und Reflexion

Kamera neu hinzugefügt und konfiguriert werden. Mit einer Preset-Architektur könnte dieser Prozess in seiner zeitlichen Dauer optimiert werden. Die Architektur von Maya ermöglicht es, mehrere Shader in einem Netzwerk gemeinsam abzuspeichern und erneut zu öffnen. Auf diese Funktionalität müsste dann das Mel-Script der Benutzeroberfläche aufgreifen.

Der letzte Punkt beschäftigt sich mit der Zurverfügungstellung des Plugins in verschiedene Maya-Versionen. Die Implementierung dieser Arbeit basiert auf Maya in der Version 2012 (64-Bit). In der Regel lässt sich ein Plugin ohne Komplikationen auf eine andere Zielversion kompilieren. Im Fall von Maya 2011 wurde dies erfolgreich gemacht. In Maya 2013, der aktuellen Version, wurde die Architektur der Outputshader leicht modifiziert und die Ordnerstrukturen abgeändert. Es sind einige Anpassungen im Quellcode nötig, dürfte letztendlich jedoch im Bereich des Möglichen liegen.

Anhang A

Quellcodes - Outputshader

A.1 Verzweigungsshader

In den folgenden Codezeilen 2-9 befindet sich die Deklaration des Mental Ray Shaders. Diese wird gesondert in einer .mi-Datei abgespeichert und in den Ordner Maya2012\mentalray\include kopiert. Mit Hilfe der Zeile 7 wird der Shader zu einem Outputshader deklariert. Als Inputlot erhält er, wie in Zeile 4 zu sehen, ein Array bzw. eine Liste an weiteren Outputshadern, welche im Hypershade eine Verbindung mit ihm eingehen können.

Ab Zeile 12 beginnt in C-Sprache die eigentliche Implementierung des Verzweigunsshaders. Funktionsname und Funktionsparamater unterliegen einem strikten Muster, welches durch die Mental Ray und Maya-API vorgeschrieben wird. In Zeile 25 werden die im Hypershade anghängten weiteren Outputshader durch Iteration über das Array einzeln aufgerufen und ausgeführt. Die kompilierte Dll-Datei wird in den Ordner Maya2012\mentalray\lib kopiert.

```
1  //Mental Ray Shader Declaration
   declare shader
3    "mayapost" (
       array shader "outputshader"
5    )
     #: nodeid 1391649
7    apply output
     version 1
9  end declare

11 //C-DLL
   struct mayapost {
13     int i_shader;
       int n_shader;
15     miTag shader[1];
       };
17
   extern "C" DLLEXPORT miBoolean mayapost(void *result, miState *state,
       struct mayapost *paras)
```

```
{
  miTag *shader =  mi_eval_tag(paras->shader);
  int i_shader = *mi_eval_integer(&paras->i_shader);
  int n_shader = *mi_eval_integer(&paras->n_shader);
  int i;
  for (i=0; i < n_shader; i++){
    mi_call_shader(NULL ,miSHADER_OUTPUT, state, shader[i_shader + i]);
  }

  return miTRUE;
}
```

A.2 Typendefinition - miImg_image

Quelle: C:\Program Files\Autodesk\Maya2012\devkit\mentalray\include\shader.h
(Bestandteil jeder Maya-Installation)

```
struct miImg_image {
  miScalar  filter;   /* filtered lookups if > 0 */
  int    dirsize;  /* valid # of filter levels */
  int    dir[miIMG_DIRSIZE]; /* offs from this to other imgs */
  int    width, height;  /* width and height in pixels */
  int    bits;   /* requested bits per comp, 8/16/32 */
  int    comp;   /* requested components/pixel, 1..4 */
  miUint1   flags;    /* additional flags */
  miUint1   writable; /* writable texture incl. image format */
        miCBoolean  cacheable;  /* cache in parts */
  miCBoolean  remap;    /* image is remapped */
  int    type;   /* requested miIMG_TYPE_* */
  miTag   real_name;  /* (local) file name to open */
  miTag   colorprofile; /* color profile for image data */
  int    c[4];   /*  indexed by miIMG_*; 4*height */
          /*  component indices follow, */
          /*  then component scanlines */
};
```

A.3 Konvertierungsfunktionen Custom Image - Mental Ray Framebuffer

```
//Konvertierung Mental Ray Framebuffer -> Custom Image Format
extern "C" DLLEXPORT void miImgToCustomImage(customImage *ci, miImg_image
    *fb_color, int width, int height){
  unsigned int index = 0;

  for (int y=0; y < height; y++) {
    for (int x=0; x < width; x++) {
```

```
8         mi_img_get_color(fb_color, (miColor *)&ci[index], x, y);

10        index++;
      }
12    }

14  }

16  //Konvertierung Custom Image Format -> Mental Ray Framebuffer
    extern "C" DLLEXPORT void customImageToMiImg(customImage *ci, miImg_image
        *fb_color, int width, int height){
18    unsigned int index = 0;

20    for (int y=0; y < height; y++) {
        for (int x=0; x < width; x++) {
22
          mi_img_put_color(fb_color,(const miColor *)&ci[index], x, y);
24
          index++;
26
        }
28    }

30  }
```

Anhang B

Referenzen - Autodesk Maya - Mental Ray

B.1 Mental Ray

Dies ist ein Auszug aus Filmproduktionen, in welchen der 3D-Renderer Mental Ray verwendet wurde.

Quelle : http://www.mentalimages.com/gallery/motion-pictures.html

TRON: Legacy, LOOM, TRON: Legacy Light Runner, Dante 01, Même Les Pigeons Vont Au Paradis, Spiderman 3, Tekkonkinkreet, Arthur and the Minimoys - Arthur and the Invisibles, Poseidon, 2046, Blade Trinity, Son Of The Mask, Superbowl, The Day After Tomorrow, Alexander, The Matrix Revolutions, The Matrix Reloaded, Star Wars: Episode II Attack of the Clones, The Hulk, Terminator 3: Rise of the Machines, Frankenred, Brothers Grimm, Aero Mice, The City Of The Lost Children, Eve Solal, Fight Club, Panic Room, The Cell, The Grinch, Tightrope, Walking with Dinosaurs

B.2 Autodesk Maya

Quelle : http://www.animaya.co.il

20 Years of Innovation
Made With Maya Films

Maya software is used by production studios around the world to achieve onscreen special effects reality. Our customers have taken Maya and used it to raise the creative bar in filmmaking in such movies as:

102 Dalmations
A.I. Artificial Intelligence
Atlantis
Behind Enemy Lines
Black Hawk Down
Bulletproof Monk
The Cell
Charlie's angels: Full Throttle
The ChubbChubbs!
Darkness Falls
Die Another Day
Dinosaur
Dreamcatcher
Enemy at the Gates
Final Fantasy: The Spirits Within
Finding Nemo
For the Birds
Harry Potter & the Chamber of Secrets
Harry Potter & the Sorcerer's Stone
Hollow Man
How The Grinch Stole Christmas
Ice Age
Inspector Gadget
The Iron Giant
Jonah: A VeggieTales Movie
Jurassic Park III
The Living Forest
Lilo & Stitch
The Lord of the Rings: The Fellowship of the Ring
The Lord of the Rings: The Two Towers
The Matrix
Men in Black II
Mighty Joe Young
Mike's New Car
Minority Report
Mission to Mars
Monsters, Inc.
The Mummy
The Mummy Returns
Panic Room
Pearl Harbor
The Perfect Storm
Pitch Black
Reign of Fire
Scooby-Doo
Shrek
Signs
Solaris
Spider-Man
Spirit: Stallion of the Cimarron
Star Trek: Nemesis
Star Wars: Episode I - The Phantom Menace
Star Wars: Episode II - Attack of the Clones
Stuart Little
Stuart Little 2
Swordfish
Terminator 3: Rise of the Machines
The Hulk
The League of Extraordinary Gentlemen
The Time Machine
Treasure Planet
Triple X
Vertical Limit
X-Men
X-Men 2
XXX

Anhang C

Inhalt der CD

CD:/Masterarbeit_Peter_Salziger.pdf Masterarbeit im PDF-Format

CD:/Quellcode/ Quellcode des Maya-Plugins

CD:/Umfrage.pdf Eine Kopie der Umfrage

CD:/Quellen/ Quellen: Dof- und Bokeh-Implementierung

Literaturverzeichnis

[Dem04] Demers, Joe: Depth of Field: A Survey of Techniques. In: Fernando, Randima (Hrsg.): *GPU Gems*, Pearson Education, 4 2004, S. 375–390

[Gio11] Giordana, Francesco: High quality previewing of shading and lighting for Killzone3. In: *ACM SIGGRAPH 2011 Talks*, ACM, 7 2011 (SIGGRAPH '11)

[Gou03] Gould, David: *Complete Maya Programming: An Extensive Guide to MEL and C++ API*. Morgan Kaufmann, 2003

[GW01] Gonzalez, Rafael C. ; Woods, Richard E.: *Digital Image Processing*. 2nd. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 2001

[Ham07] Hammon, Earl: Practical Post-Process Depth of Field. In: Nguyen, Hubert (Hrsg.): *GPU Gems 3*, Addison-Wesley Professional, 2007

[HP10] Huang, Liying ; Pei, Yijian: Film and television animation design based on Maya and AE. In: *Image and Signal Processing (CISP), 2010 3rd International Congress on* Bd. 1, 2010, S. 135 –137

[Hur10] Hurkman, Alexis V.: *Color Correction Handbook: Professional Techniques for Video and Cinema*. 1. Peachpit Press, 2010

[Kaj86] Kajiya, James T.: The rendering equation. In: *SIGGRAPH Comput. Graph.* 20 (1986), August, Nr. 4, S. 143–150

[Kri04] Kritzenberger, Huberta: *Multimediale und interaktive Lernräume*. Oldenbourg Wissenschaftsverlag, 2004

[Men08] Mental images: *mental ray Production Shader Library*, 11 2008. http://docs.autodesk.com/MENTALRAY/2012/CHS/mental%20ray%203.9%20Help/files/tutorials/production.pdf

[MRD12] McIntosh, L. ; Riecke, B. E. ; DiPaola, S.: Efficiently Simulating the Bokeh of Polygonal Apertures in a Post-Process Depth of Field Shader. In: *Comp. Graph. Forum* 31 (2012), September, Nr. 6, S. 1810–1822

[Mus06] Muscha, Elona: Translating 2D german expressionist woodcut artwork into 3D, Texas A&M University, 7 2006

[Nit08] Nitsche, Michael: Experiments in the use of game technology for previsualization. In: *Proceedings of the 2008 Conference on Future Play: Research, Play, Share*. New York, NY, USA : ACM, 2008 (Future Play '08), S. 160 –165

[Pet11] Pettineo, Matt: How To Fake Bokeh. (2011), 2. http://mynameismjp.wordpress.com/2011/02/28/bokeh/. – aufgerufen am 06.01.2013, eine Kopie befindet sich auf der beiliegenden CD

[Sch08] Schaarschmidt, André: Techniken und Umgang mit Tiefenschärfe in 3D-Programmen, THE GERMAN FILM SCHOOL, 7 2008

[YM04] Yu, Jingyi ; McMillan, Leonard: General Linear Cameras. In: *Computer Vision - ECCV 2004* Bd. 3022. Springer Berlin Heidelberg, 2004, S. 14–27

[Yu04] Yu, Tin-Tin: DEPTH OF FIELD IMPLEMENTATION WITH OPENGL. In: *Michigan Technological University* (2004). – eine Kopie befindet sich auf der beiliegenden CD