Entwicklung eines Systems zur dynamischen Beleuchtung von 3D-Szenen

Michael Horsch

Hochschule Darmstadt
Fachbereich Informatik

19.05.2007

Entwicklung eines Systems zur dynamischen Beleuchtung von 3D-Szenen

Bachelorarbeit

Sommersemester 2007

Hochschule Darmstadt
Fachbereich Informatik

von
Michael Horsch

Referent: Prof. Dr. Elke Hergenröther
Koreferent: Prof. Dr. Wolf-Dieter Groch

Extern durchgeführt am:
Fraunhofer Institut für graphische Datenverarbeitung (IGD)
Darmstadt

Unter Betreuung von:
Dipl.-Inform. Martin Knuth

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Bachelorarbeit selbstständig und unter ausschließlicher Verwendung der angegebenen Hilfsmittel angefertigt habe.

Michael Horsch

Altheim, 19.05.2007

Inhaltsverzeichnis

Kapitel 1

Einleitung

Wohin man heutzutage auch sieht, überall kommt Computer Graphik zum Einsatz. Häuser werden am Computer geplant, Designs für Autos und andere Produkte werden zuerst nur virtuell entworfen, Zeitschriften und deren Artikel und Anzeigen werden mit Grafik-Programmen gestaltet, aber auch für Unterhaltungsmedien wie Filmen, Fernsehen und Computerspielen wird die Computer Graphik genutzt.
In vielen Bereichen (z.B. bei der Produktvisualisierung) ist eine möglichst realitätsnahe Darstellung erforderlich. Um dies zu erreichen wird eine genaue Modellierung der Beleuchtung gebraucht, da das menschliche Auge nur Licht wahrnimmt und so durch Konturen und Schattierungen Objekte voneinander unterscheiden kann. Eine genaue Modellierung der Beleuchtung ist jedoch durch die Komplexität von Licht nur schwer zu erreichen und sehr kompliziert.
Bei graphischen Simulationen wird außerdem eine Beschreibung der Objekte benötigt, um diese auf dem Bildschirm darstellen zu können. Dazu werden die Objekte am Computer möglichst realitätsnah modelliert. So können auch Umgebungen modelliert werden, die in Wirklichkeit noch gar nicht existieren. Solche Objekte und Umgebungen werden als 3D-Szenen bezeichnet.
Diese Bachelor-Arbeit zeigt, wie ein System zur dynamischen Beleuchtung von 3D-Szenen entwickelt wurde.

1.1 Motivation

Ein dynamisches Beleuchtungssystem erlaubt die freie interaktive Positionierung von Lichtquellen in 3D-Szenen. Dabei wird die Beleuchtung die von

den Lichtquellen ausgeht dynamisch berechnet, d.h. die Beleuchtung muss nicht vorberechnet werden. So können Beleuchtungsverhältnisse verändert werden und die Ergebnisse können direkt angezeigt werden. Auch optische Eigenschaften wie Transparenz oder Dichte fließen mit in die Berechnungen ein. Die Reaktionen des menschlichen Auges auf Helligkeitsänderungen können ebenfalls simuliert werden.
Die dynamische Beleuchtung von 3D-Szenen hat sehr viele Anwendungsgebiete. Obwohl ein solches System vor allem für Produktvisualisierungen und Unterhaltungsmedien wie Filmen und Computerspielen genutzt wird besteht auch in medizinischen und wissenschaftlichen Bereichen ein Bedarf an realistischen Visualisierungen.
Abbildung 1.1 zeigt zwei sehr unterschiedliche Anwendungen für ein System zur dynamischen Beleuchtung von 3D-Szenen, eine Produktvisualisierung und ein Computerspiel. Während man bei Produktvisualisierungen versucht reale Beleuchtungsverhältnisse zu modellieren, wird die Beleuchtung in Computerspielen wie in Filmen zur Steigerung der Dramatik der 3D-Szene eingesetzt.

Abbildung 1.1: Beispiele für die Einsatzmöglichkeiten eines dynamischen Beleuchtungssystems: Oben: Produktvisualisierungen für die Automobilindustrie [VREC]. Unten: Szenen aus einem Computerspiel [DOOM3].

1.2 Aufgabenstellung

Diese Arbeit wird im Kontext der Visualisierung von Simulationsdaten virtueller Brände durchgeführt. Simulationen von Bränden erlauben eine realistische Abschätzung der Beleuchtungs- und Sichtverhältnisse bei einem Feuer, wodurch sich z.B. die sichere Funktionsweise von Fluchtwegen virtuell testen lässt. Ein großer Nachteil des bestehenden Systems liegt darin, dass zwar das Feuer sehr realistisch dargestellt wird, nicht aber die Beleuchtungsverhältnisse die vom Feuer erzeugt werden. Ein anderer wichtiger Aspekt ist die Interaktivität der gewählten Beleuchtungsmethode. Eine möglicher Lösungsansatz besteht darin, viele Lichtquellen so im Raum zu verteilen, das diese den Beleuchtungseffekt des Feuers approximieren können. Mit moderner Grafik-Hardware lassen sich hier sehr viele Lichtquellen gleichzeitig bei interaktiven Frameraten berechnen. Hierbei ist zu berücksichtigen, dass Feuer ein dynamisches Medium ist, sich also die Lichtquellen frei im Raum bewegen. Es können also keine Annahmen über die Position und Bewegung der Lichtquellen im Raum durchgeführt werden.
In der vorliegenden Arbeit soll hierzu ein Framework entwickelt werden, welches das Beleuchtungsproblem von 3D-Szenen mit vielen Lichtquellen adressiert. Hierbei ist auf eine möglichst einfache Beleuchtungsberechnung zu achten, damit möglichst viele Lichtquellen benutzt werden können. Die Lichtquellen sollen sich frei im Raum bewegen lassen. In einer optionalen Ausbaustufe kann auf vorhandene Simulationsdatensätze zurückgegriffen werden, um die aktuellen Positionen der Lichtquellen zu errechnen.
Zusammenfassend sind folgende Aufgaben im Rahmen der Praxisphase zu bearbeiten, bzw. als Softwaremodul zu realisieren:

- Portierung des bestehenden Szenegraphen von Java nach C++
- Laden und Beleuchten der Raumgeometrie
- Optimieren der Beleuchtungsberechnung
- Darstellung der Simulationsdatensätze

Als Grundlage für diese Arbeit kann auf vorhandene Bibliotheken für Basisfunktionalitäten zurückgegriffen werden. Das zu entwickelnde Softwaremodul soll leicht erweiterbar sein.

Ausgabe der Arbeit: 01.10.2006
Bearbeiter: Michael Horsch

1.3 Anforderungen an das System

Aus der Aufgabenstellung ergeben sich bereits verschiedene Anforderungen an das System. Die Einzelheiten dieser Anforderungen werden in diesem Abschnitt beschrieben.

1.3.1 Performance

Wie bei Filmen werden in der Computer Graphik Animationen durch die Abfolge von Einzelbildern (Frames) erzeugt. Während man bei Filmen eine feste Anzahl an Frames pro Sekunde hat, richtet sich hier die Anzahl der Frames pro Sekunde nach der benötigten Rechenzeit um ein einzelnes Frame anzuzeigen. Das heißt je länger die Rechenzeit für ein einzelnes Frame dauert, desto weniger Frames pro Sekunde können angezeigt werden. Damit das Auge die Abfolge der Bilder nicht bemerkt werden etwa 25 Frames pro Sekunde benötigt, interaktives Arbeiten ist jedoch schon bei weniger Frames pro Sekunde möglich.
Anwendungen die mit dem System realisiert werden sollen mit interaktiven Frameraten laufen. Die minimale Anzahl der Frames pro Sekunde soll 5 Bilder pro Sekunde nicht unterschreiten. Dazu müssen die verwendeten Berechnungen vereinfacht und optimiert werden, wobei die erzeugten Bilder trotzdem realistisch aussehen sollen.

1.3.2 Volumendaten

Das System soll es ermöglichen statische sowie animierte Volumendatensätze anzuzeigen und als Lichtquellen zu benutzen. Die Volumendatensätze werden mit einem externen Programm vorberechnet und als Binärdateien gespeichert. Für jeden Volumendatensatz werden mehrere Dateien gespeichert, eine Konfigurationsdatei mit Informationen über das Volumen und für jeden Animationsschritt jeweils eine Datei mit den Daten des Volumens. Da die Volumendatensätze Feuer darstellen, handelt es sich bei diesen Daten um Temperaturwerte. Die Auflösung des Volumendatensatzes, die Anzahl der Animationsschritte, der verwendete Datentyp und der Name des Volumens

sind in der Konfigurationsdatei gespeichert.
Für die Visualisierung der Volumendaten sollen ebenfalls die Temperaturen genutzt werden. So soll für jeden Punkt im Volumen ein Grauwert angezeigt werden der der Temperatur an diesem Punkt entspricht. Um dem Feuer eine Farbe zu geben, soll der Grauwert mit einer rötlichen Farbe moduliert werden. Temperaturen die einen einstellbaren Wert unterschreiten sollen nicht angezeigt werden.
Die Beleuchtung des Feuers soll durch mehrere Punktlichtquellen angenähert werden. Das Feuer soll realistische Schatten werfen. Die Beleuchtung und die Schatten sollen sich nach den Bewegungen des Feuers richten. Die Bewegungen des Feuers sollen aus den Änderungen der Temperaturwerte zwischen den einzelnen Animationsschritten abgeleitet werden.

1.3.3 3D-Szenen

Das System soll komplexe 3D-Szenen mit einer realistischen Beleuchtung darstellen. Die 3D-Szenen werden in einem externen Programm aus Polygonen modelliert und im .obj-Dateiformat gespeichert. Je komplexer eine 3D-Szene ist, desto mehr Polygone werden zur Modellierung gebraucht. Die 3D-Szene ist in einzelne Objekte unterteilt. Ein Objekt besteht aus mehreren Polygonen, die alle das gleiche Material verwenden. Bei allen Polygonen handelt es sich um Dreiecke. Zu jedem Dreieck gehören 3 Vertices (Eckpunkte) und deren Normalen Vektoren sowie die Textur-Koordinaten. Informationen über die in der 3D-Szene verwendeten Materialien werden in einer eigenen Datei gespeichert. Ein Material besteht aus einer Textur und verschiedenen Konstanten für die Beleuchtungsberechnung. Die darzustellenden 3D-Szenen können aus bis zu 100.000 Polygonen aufgebaut sein. Auch bei der Darstellung muss auf die Performance geachtet werden um interaktive Frameraten zu gewährleisten.

1.3.4 Skalierbarkeit

Das System soll skalierbar sein. Mit schnellerer Hardware soll das System schneller laufen oder es soll bei gleicher Performance eine bessere visuelle Qualität erreicht werden.

1.3.5 Zusammenfassung

Das System soll es ermöglichen 3D-Szenen bei interaktiven Frameraten mit einem animierten Volumendatensatz zu beleuchten und zu visualisieren. Für die einzelnen Teilprobleme können vorhandene Algorithmen und Techniken genutzt werden. Diese sollen zu einem Gesamtsystem zusammengefügt werden, das einfach erweiterbar und skalierbar ist. Auf Erweiterbarkeit und Skalierbarkeit soll auch bei der Wahl der genutzten Verfahren geachtet werden.

Kapitel 2

Stand der Technik

Da die Computer Graphik ein sehr aktives Forschungsgebiet ist, gibt es große Mengen an Publikationen zu den verschiedensten Themen. In diesem Kapitel werden die wichtigsten Techniken zur Beleuchtungsberechnung für 3D-Szenen vorgestellt.

2.1 Beleuchtungsberechnung

Um die Beleuchtung berechnen zu können, ist es wichtig zu wissen wie Licht funktioniert. Das sichtbare Licht ist physikalisch gesehen elektromagnetische Strahlung im Bereich von ca. 380 bis 780 Nanometer die entsteht, wenn ein Atom von einem hohen Energieniveau auf ein niedrigeres sinkt[Wikipedia]. Dies ist z.B. bei einer Verbrennung der Fall, wenn die Atome reagieren und dadurch Energie abgeben. Außerdem hat Licht die Besonderheit, das es Eigenschaften von Wellen und Teilchen hat. Dadurch wird eine Modellierung der Funktionsweise natürlichen Lichts komplizierter. Dennoch kann man die Interaktionen zwischen Licht und Materie durch die Quantenphysik erklären und relativ genau simulieren. Dieser hohe Detailgrad wird jedoch nicht benötigt, da sich die Wahrnehmung des Menschen auf die Reflexion, Refraktion und Absorbierung von Licht beschränkt.
Sogenannte globale Beleuchtungsverfahren versuchen die kompletten Beleuchtungsberechnungen durchzuführen. Doch die Berechnung dieser Effekte benötigt noch sehr viel Zeit. Für Anwendungen mit interaktiven Frameraten werden deshalb vereinfachte Beleuchtungsverfahren benutzt. Diese Verfahren modellieren nicht die Reflexionen des Lichts und werden als lokale Beleuchtungsverfahren bezeichnet.

2.1.1 Globale Beleuchtung

Bei globalen Beleuchtungsverfahren wird versucht, auch das Licht in die Berechnungen miteinzubeziehen, das von anderen Oberflächen reflektiert wird. Globale Beleuchtungsverfahren erzeugen dadurch sehr realistische Bilder, benötigen aber auch viel Rechenleistung.
Bevor man aber überhaupt eine Beleuchtung berechnen kann, braucht man erstmal die entsprechenden Formeln. Die Beleuchtung kann zwar auf unterschiedliche Weisen berechnet werden, die mathematischen Grundlagen sind aber immer die selben. Diese Grundlagen können mit einer Gleichung, der sog. Rendering-Gleichung, beschrieben werden.
Die Rendering-Gleichung[Kajiya1986] wurde im Jahr 1986 von James T. Kajiya vorgestellt. Die Rendering-Gleichung berechnet die Beleuchtung die von einem Punkt in eine bestimmte Richtung abgestrahlt wird.
Die Besonderheit an der Gleichung ist, dass jedes existierende Beleuchtungsmodell aus ihr abgeleitet werden kann. Leider ist die Lösung der Gleichung nicht direkt mit einem Computer zu berechnen, da ein Integral enthalten ist um die Beleuchtungen aufzuaddieren die aus allen Richtungen kommen. Da man die Gleichung zumindest für eine begrenzte Anzahl an Richtungen lösen kann, gibt es verschiedene Lösungsansätze, die sich aber durch die langen Rechenzeiten noch nicht für interaktive Anwendungen eignen. Weil die Beleuchtung mit diesen Lösungsansätzen für statische Lichter in statischen 3D-Szenen vorberechnet werden kann und es nur eine Frage der Zeit ist bis die Hardware die nötigen Berechnungen schnell genug ausführen kann, soll ein kurzer Überblick über die Ideen von drei populären Lösungsansätzen gegeben werden: Radiosity, Path-Tracing und Photon-Mapping.
Radiosity ist eines der ältesten Verfahren zur globalen Beleuchtungsberechnung. Radiosity löst die Rendering-Gleichung für diffuses Licht und einer begrenzten Anzahl an Lichtreflexionen. Zur Berechnung von Radiosity wird die Szene zuerst in sogenannte Patches unterteilt. Ein Patch ist eine Fläche oder ein Punkt für den die Beleuchtungsberechnung ausgeführt wird. Für jeden Patch wird dabei berechnet, wieviel Licht er von allen anderen Patches erhält. Je höher die Unterteilung, desto höher ist die Qualität der Lösung.
Ein Vorteil ist das Radiosity unabhängig vom Betrachter berechnet wird. So ist es möglich die Beleuchtungsdaten unbeweglicher Lichter für Anwendungen in denen die Szene frei betrachtet werden kann vorzuberechnen.
Die folgenden Lösungsansätze basieren alle auf Ray Tracing. Beim Ray Tra-

cing werden Strahlen von einem Punkt aus zurückverfolgt. So kann man mit Ray Tracing auch Lichtstrahlen zurückverfolgen. Mit Ray Tracing können auch Bilder erzeugt werden, indem vom Betrachter aus für jeden Bildpunkt ein Strahl in die Szene geschossen wird. Trifft der Strahl auf eine Oberfläche, wird dem Bildpunkt die Farbe der Oberfläche zugewiesen. Auch Schatten und spiegelnde Reflexionen können mit Ray Tracing erzeugt werden. Um Schatten zu berechnen, wird von jedem Punkt aus ein Strahl zum Licht gesendet. Trifft der Strahl bevor er an der Lichtquelle ankommt auf ein Objekt, liegt der Punkt im Schatten. Für spiegelnde Reflexionen wird der einfallende Strahl reflektiert und zurückverfolgt.
Path-Tracing[Kajiya1986] ist ein weiterer Ansatz zur Lösung der Rendering-Gleichung. Hierbei werden für jeden Punkt möglichst viele Lichtstrahlen aus zufälligen Richtungen zurückverfolgt, wodurch die Komplexität der Berechnungen sehr hoch ist. Dafür können mit Path-Tracing sehr realistische Bilder erzeugt werden. Im Gegensatz zum normalen Ray Tracing werden beim Path-Tracing die Lichtstrahlen auch für diffuse Oberflächen zurückverfolgt.
Beim Photon Mapping[Jensen2001] werden Photonen von der Lichtquelle emittiert. Trifft ein Photon eine Oberfläche, wird es entweder reflektiert oder absorbiert. Die Treffer werden dann in einer Karte, der Photon Map, gespeichert. Um die Beleuchtung an einem Punkt auszuwerten, werden alle Photonen in der Nähe des Punktes berücksichtigt.
Photon Mapping ist im Vergleich zu den anderen Verfahren schneller zu berechnen und erzeugt auch sehr realistische Bilder. Es unterstützt zudem die Berechnung von sogenannten Kaustiken. Kaustiken sind Bündelungen von gebrochenem oder reflektiertem Licht. Kaustiken können zwar auch mit Path-Tracing berechnet werden, Photon Mapping ist durch verschiedene Optimierungen aber deutlich schneller.
Abbildung 2.1 zeigt einen Vergleich der Techniken. Für die Erstellung der Bilder wurde Ray Tracing verwendet. Die spiegelnden Reflexionen auf der hinteren Kugel und die Lichtbrechungen der rechten Kugel wurden ebenfalls mit Ray Tracing erzeugt.

2.1.2 Lokale Beleuchtung

Bei lokalen Beleuchtungsverfahren wird nur das Licht simuliert, das direkt von den Lichtquellen abgestrahlt wird. Dadurch sinkt die Komplexität der

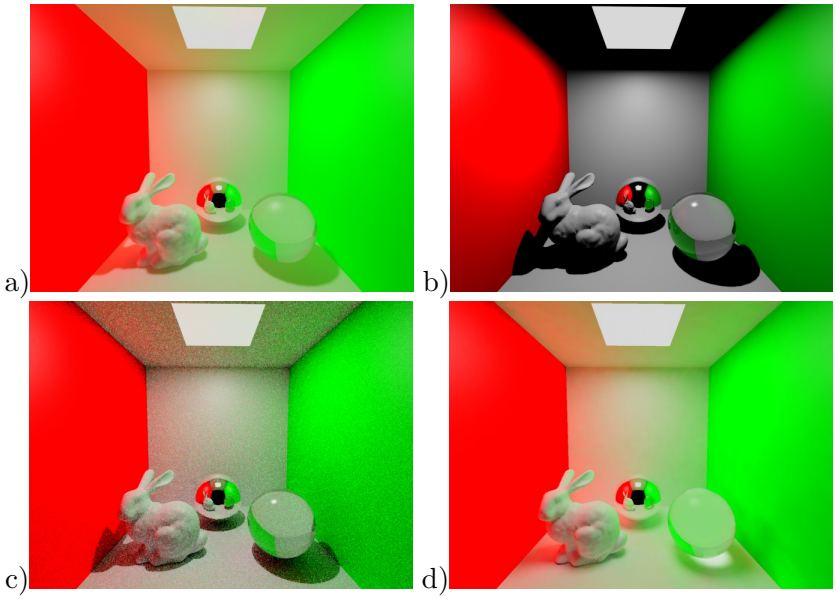

Abbildung 2.1: Vergleich der vorgestellten Verfahren. a) Radiosity; b) Ray Tracing; c) Path-Tracing; d) Photon-Mapping.

Berechnungen und somit die benötigte Rechenleistung.

Phong'sches Beleuchtungssmodell

Das Phong'sche Beleuchtungsmodell[Phong1975] ist wohl das populärste Modell zur Beleuchtungsberechnung. Es ist relativ realistisch, einfach zu implementieren und schnell zu berechnen, jedoch physikalisch nicht korrekt. Das Beleuchtungsmodell nach Phong unterteilt die Beleuchtungsberechnungen in die Berechnung von ambientem Licht, diffusem Licht und Glanzlichtern. Abbildung 2.2 zeigt die einzelnen Teile der Gleichung zusammen mit dem Ergebnis. Mit dem ambienten Licht wird die globale Beleuchtung sehr grob angenähert, indem der 3D-Szene eine an allen Punkten gleiche Beleuchtung zugewiesen wird, die sozusagen von der Umgebung ausgeht. Diffuses Licht ist das Licht, das von der Oberfläche in unterschiedliche Richtungen reflektiert wird. Durch das diffuse Licht erhält die 3D-Szene ihre Schattierung. Glanzlichter entstehen wenn das Licht direkt in das Auge reflektiert wird.

Gleichung 2.1 zeigt die Berechnung der Beleuchtung nach Phong. K ist die

Abbildung 2.2: Links Oben: Ambiente Beleuchtung. Rechts Oben: Diffuse Beleuchtung. Links Unten: Glanzlichter. Rechts Unten: Ergebnis.

Intensität der Beleuchtung. A steht für das ambiente Licht und ist ein Konstanter Wert. Die diffuse Beleuchtung ergibt sich aus dem Skalarprodukt zwischen dem Normalen Vektor N eines Punktes und der Richtung L vom Punkt zur Lichtquelle. Das Glanzlicht wird mit dem Skalarprodukt zwischen der Richtung R in die das Licht reflektiert wird und der Richtung zum Betrachter V berechnet. Mit dem Glanzlicht-Exponenten E kann die Schärfe des Glanzlichts kontrolliert werden. Für matte Oberflächen wird ein niedriger Exponent benutzt, für glänzende und spiegelnde Oberflächen eine hoher Exponent.
Abbildung 2.3 zeigt eine Visualisierung der benötigten Vektoren.

$$K = A + max(0.0, N \circ L) + pow(max(0.0, R \circ V), E) \tag{2.1}$$

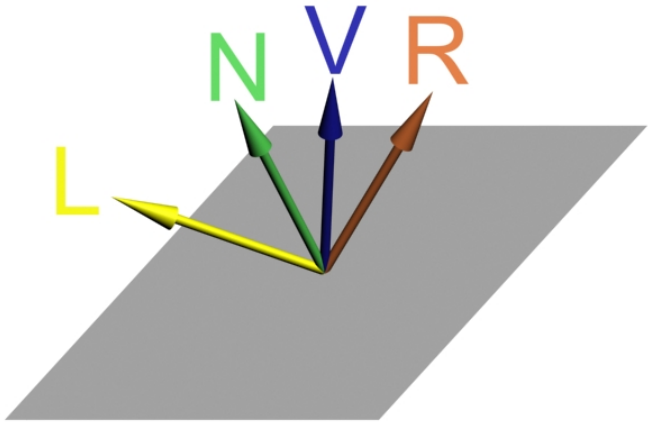

Abbildung 2.3: Die zur Beleuchtungsberechnung nötigen Vektoren.

Cook-Torrance Beleuchtungsmodell

Beim Cook-Torrance Beleuchtungsmodell [Cook1982] wird angenommen, dass alle Oberflächen aus winzigen Mikrofacetten mit unterschiedlicher Rauheit bestehen. Das Glanzlicht ist dadurch beim Cook-Torrance Modell realistischer als beim Phong Modell. Mit unterschiedlichen Rauheiten werden unterschiedliche Materialien erzeugt, z.B. eine niedrige Rauheit für glat-

tes Metall und eine hohe Rauheit für zerfurchtes Holz. Außerdem wird der sogenannte Fresnel-Effekt simuliert, wodurch es möglich wird metallische Oberflächen von anderen zu unterscheiden.

2.2 Schattenberechnung

Schatten gehören genauso zu einer realistischen Visualisierung wie die Beleuchtung. Ohne Sie würden die erzeugten Bilder unnatürlich aussehen und Sie helfen Abstände und Entfernungen in 3D-Szenen besser einzuschätzen.
Normalerweise würden die Schatten durch die Beleuchtungsberechnungen automatisch erzeugt werden. Da bei den lokalen Beleuchtungsverfahren aber auf starke Vereinfachungen der Berechnungen zurückgegriffen wird, werden auch keine Schatten erzeugt. Bei lokalen Beleuchtungsverfahren stehen nur Informationen eines Punktes der 3D-Szene zur Verfügung. Andere Objekte der 3D-Szene, durch die ein Schatten entstehen könnte, werden bei der Beleuchtungsberechnung nicht berücksichtigt. Deshalb müssen die Schatten getrennt von der Beleuchtung berechnet werden. Für die Berechnung der Schatten gibt es drei verschiedene Ansätze, von denen es jeweils verschiedene Variationen gibt. Die drei Ansätze sind Ray Traced Shadows, Volume Shadows und Shadow Mapping.
Abbildung 2.4 zeigt eine beleuchtete 3D-Szene mit Schatten. Die Schatten wurden mit Shadow Mapping berechnet.

2.2.1 Ray Traced Shadows

Bei Ray Traced Shadows wird für jeden Punkt ein Strahl zum Licht zurückverfolgt. Trifft der Strahl auf ein Objekt bevor er an der Lichtquelle ankommt, liegt der Punkt im Schatten. Um zu prüfen ob ein Strahl ein Objekt schneidet, muss berechnet werden ob der Strahl eines der Polygone des Objekts schneidet. Diese Berechnung ist gerade bei komplexen 3D-Szenen sehr aufwändig und wird zur Zeit noch nicht von der Grafik-Hardware unterstützt.

2.2.2 Volume Shadows

Die Idee bei Volume Shadows ist, den Schatten als Volumen zu modellieren. Um zu prüfen ob ein Punkt im Schatten liegt, wird berechnet ob sich der

Abbildung 2.4: Schatten sind für eine realistische Visualisierung sehr wichtig.

Punkt innerhalb eines Schatten-Volumens befindet.
Die Konstruktion der Schattenvolumen ist sehr rechenaufwändig, darum wird versucht die Komplexität der Schattenvolumen möglichst gering zu halten. Um die Volumen zu konstruieren, müssen die Kanten gefunden werden, die gleichzeitig zu einem Polygon das zum Licht und einem das vom Licht weg zeigt gehören. Diese Kanten werden extrudiert, d.h. sie werden vom Licht aus nach hinten geschoben. Mit den extrudierten Kanten hat man dann genug Informationen um das Schattenvolumen aus Polygonen zu modellieren. Mit Hilfe der Grafik-Hardware können dann die Teile der 3D-Szene markiert werden, die sich innerhalb des Schattenvolumens befinden. Um die Schatten einer Lichtquelle darzustellen, wird die beleuchtete 3D-Szene nur an den Stellen gezeichnet die nicht markiert sind. Für jede Lichtquelle wird ein eigenes Schattenvolumen benötigt. Die Schattenvolumen müssen immer dann neu berechnet werden, wenn sich die Geometrie der 3D-Szene oder die Position einer Lichtquelle geändert hat.

2.2.3 Shadow Mapping

Shadow Mapping [Williams1978] ist ein Textur-basierter Ansatz zur Schattenberechnung. Die Idee bei Shadow Mapping ist, dass das was das Licht nicht sieht im Schatten liegt. Dazu wird die Szene von der Position der Lichtquelle aus gezeichnet und die Werte des Tiefenpuffers werden in einer Textur gespeichert, der sogenannten Shadow Map. Diese Textur wird dann wieder vom Licht aus auf die Szene projeziert. Die in der Textur gespeicherten Tiefenwerte des Lichts werden mit den Tiefenwerten des Lichts verglichen die der Betrachter sieht. Ist der Tiefenwert in der Textur kleiner, befindet sich der Punkt im Schatten.

2.3 High-Dynamic-Range Rendering

Um Details sowohl in dunklen und schattigen als auch hell beleuchteten Teilen der Szene realitätsgetreu wiederzugeben wird ein hoher Dynamik-Bereich benötigt. Um einen möglichst hohen Dynamik-Bereich zu erreichen wird High Dynamic Range (HDR) Rendering benutzt.
Low Dynamic Range (LDR) Rendering umfasst nur Werte mit einer Präzision von 8 Bit im Bereich von 0,0 bis 1,0 bzw. 0 bis 255. So könnte man z.B. die Intensität der Sonne auf 1,0 setzen und die Intensität einer Glühlampe

auf 0,1. In der Realität ist die Sonne aber sehr viel heller als eine Glühlampe. Durch die begrenzte Präzision kann dieser hohe Kontrast nicht dargestellt werden.
High Dynamic Range Rendering dagegen arbeitet mit einer viel höheren Präzision und begrenzt nicht den Wertebereich. Dadurch können die Intensitäten der Lichtquellen realistisch simuliert werden.

2.3.1 Tone Mapping

Moderne PC-Monitore können mit HDR Rendering erzeugte Bilder nicht korrekt anzeigen. Das liegt zum einen daran, dass HDR Rendering eine noch sehr neue Technik ist die vorher fast nur für Filme genutzt wurde und nun plötzlich einen Boom erlebt, zum anderen würden die hohen Lichtintensitäten die Augen schädigen. Man stelle sich nur vor, der Bildschirm würde so hell wie die Sonne leuchten. Stattdessen geht man einen anderen Weg, indem man das mit HDR erzeugte Bild wieder in ein LDR Bild umwandelt was auf dem PC-Monitor angezeigt werden kann. Den Vorgang der Umwandlung bezeichnet man als Tone Mapping, die genutzte Rechengleichung wird Tone Mapping-Operator genannt.
Gleichzeitig versucht man beim Tone Mapping das menschliche Auge bzw. eine Kamera zu simulieren. Sieht man z.B. direkt in die Sonne, ziehen sich die Pupillen zusammen um weniger Licht durchzulassen. Bei einer Kamera würde man die Belichtungszeit verringern um eine Überbelichtung zu verhindern. Außerdem werden Helligkeiten nicht linear wahrgenommen, d.h. ein Licht mit der doppelten Intensität eines anderen Lichtes wird nicht als doppelt so hell empfunden.
Das Tone Mapping kann auf viele verschiedene Wege durchgeführt werden. Wegen hohen Rechenzeiten eignen sich jedoch nur wenige Tone Mapping-Operatoren für Anwendungen mit interaktiven Frameraten. In [Calver2005] wurde eine für die Grafik-Hardware leicht vereinfachte Implementierung des Tone Mapping-Operators nach Reinhard [Reinhard2002] vorgestellt. Bei diesem Operator wird die mittlere Luminanz der 3D-Szene berechnet und zusammen mit dem weißen Punkt genutzt um die HDR-Werte zu skalieren. Der weiße Punkt ist die maximale Luminanz in der 3D-Szene. Ein noch einfacherer Tone Mapping-Operator wurde von Hugo Elias [Elias] vorgestellt. Um einen unbegrenzten Wertebereich in den zum LDR gehörenden Wertebereich umzuwandeln wird eine invertierte Exponential-Funktion genutzt,

wodurch gleichzeitig die nicht-lineare Wahrnehmung von Helligkeiten simuliert wird. In Abhängigkeit von einer frei einstellbaren Belichtungszeit wird der HDR-Wert entsprechend skaliert.

2.4 Volume Rendering

Nebel, Rauch und Gase können durch die optischen Eigenschaften oft nur sehr schwer mit Polygonen direkt modelliert werden. Stattdessen wird häufig ein Volumendatensatz benutzt, der für jeden Punkt in einem Volumen die zugehörigen Werte (z.B. die Dichte) speichert. Dieser Datensatz wird in Analogie zur 2D-Textur auch als 3D-Textur bezeichnet. Zur Visualisierung eines Volumens gibt es unterschiedliche Techniken.

2.4.1 Ray Casting

Die einfachste, aber auch rechenaufwändigste Technik ist das Ray Casting. Dabei wird ein Strahl durch das Volumen zum Betrachter zurückverfolgt. Die Werte der 3D-Textur die auf dem Strahl liegen werden dann zur Visualisierung benutzt.

2.4.2 Slice-Based Rendering

Beim Slice-Based Rendering wird das Volumen sozusagen scheibenweise gezeichnet. Mehrere Ebenen werden auf einer Achse in gleichbleibenden Abständen im Volumen platziert. Die Ebenen sind immer zum Betrachter orientiert und werden von hinten nach vorne gezeichnet. Durch die Verwendung von vielen Ebenen können die Lücken zwischen den Ebenen ausgeglichen werden.

2.5 Deferred Shading

Deferred Shading ist ein Verfahren mit dem die Beleuchtungsberechnungen optimiert werden können. Anstatt die Beleuchtung für jedes Polygon einzeln zu berechnen wie es üblich ist, werden die für die Beleuchtungsberechnung nötigen Informationen der sichtbaren Teile der 3D-Szene in Texturen gespeichert und anschließend für jedes Licht ausgewertet. Damit die Informationen in einer Textur gespeichert werden können, wird die 3D-Szene ganz normal vom Betrachter aus gezeichnet, nur mit dem Unterschied das anstatt

der Farben der Oberflächen die Beleuchtungsinformationen der Oberflächen angezeigt werden. Um dann die 3D-Szene anzuzeigen und zu beleuchten, wird ein Rechteck mit den Texturen über den Bildschirm gelegt. Mit Hilfe programmierbarer Grafik-Hardware kann anhand der Beleuchtungsinformationen die sich in den Texturen befinden die Beleuchtung berechnet werden. Der Benutzer sieht zum Schluß also nur ein zweidimensionales Bild der 3D-Szene, dass mit Hilfe der gespeicherten Beleuchtungsinformationen beleuchtet wird.
Deferred Shading hat den Vorteil, dass die Beleuchtungsberechnungen nicht mehr abhängig von der Komplexität der 3D-Szene ist. Die Berechnung der Beleuchtung findet unabhängig von der Verarbeitung der Geometrie statt. Für die Berechnung der Beleuchtung muss so nicht mehr die gesamte 3D-Szene gezeichnet werden, sondern nur noch ein Rechteck. Gerade in Anwendung in denen es nötig ist die 3D-Szene mehrmals zu zeichnen kann deshalb mit Deferred Shading sehr viel Rechenzeit eingespart werden.
Abbildung 2.5 zeigt Texturen mit den Beleuchtungsinformationen einer 3D-Szene. In diesem Beispiel beschränken sich die Beleuchtungsinformationen auf die Farben, Normalen, Positionen und Schatten. Je nachdem wie die Beleuchtung berechnet wird, können auch andere Informationen gespeichert werden.
Abbildung 2.6 zeigt die mit mehreren hundert Lichtquellen beleuchtete 3D-Szene. Durch die Optimierung mit Deferred Shading können die nötigen Berechnungen bei interaktiven Frameraten durchgeführt werden.

2.6 Programmierbare Grafik-Hardware

Durch neue Technologien wird die Grafik-Hardware immer flexibler. Während die Grafikkarten früher nur eine begrenzte Funktionalität hatten, sind sie inzwischen frei programmierbar. Das ermöglicht Programmierern relativ einfach neue Techniken und Effekte zu entwickeln.
Die frei programmierbaren Einheiten werden Shader genannt. Für den Programmierer unterscheiden sie sich hinsichtlich der Funktionalität. So gibt es verschiedene Shader für die verschiedenen Schritte in der Rendering-Pipeline. Auf der Hardware-Ebene sind die Shader mittlerweile vereinheitlicht. Das bedeutet, dass jeder Shader jeden Shader-Befehl ausführen kann. Dadurch kann die Rechenlast auf alle Shader-Einheiten gleichmäßig vertei-

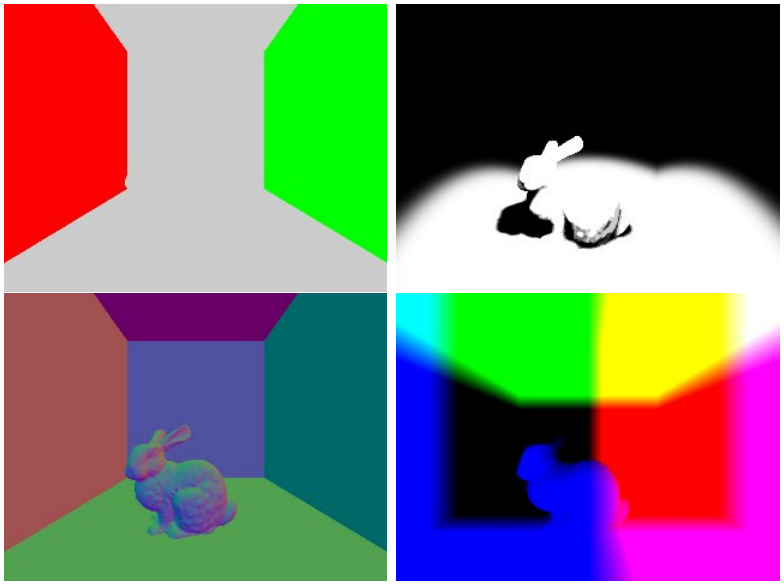

Abbildung 2.5: Beleuchtungsinformationen der Oberflächen der 3D-Szene. Links Oben: Farben. Links Unten: Normalen. Rechts Unten: Positions-Koordinaten. Rechts Oben: Schatten.

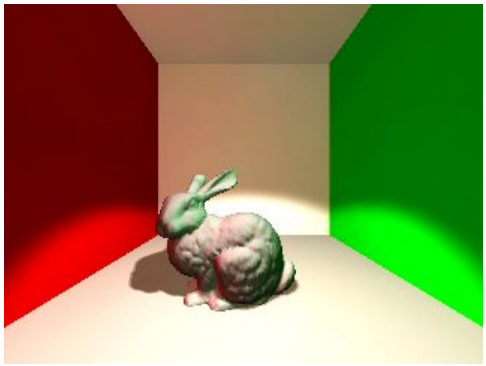

Abbildung 2.6: Ergebnis der Beleuchtungsberechnungen mit Deferred Shading.

len. Ein Programm für einen Shader wird ebenfalls als Shader bezeichnet. Für die Programmierung der Shader werden spezielle Programmiersprachen von den Programmierschnittstellen zur Verfügung gestellt.

Vertex Shader

Ein Vertex Shader berechnet die Transformationen und Attribute eines Vertex. Zu den Attributen gehört z.B. die Position oder die Farbe. Auch benutzerdefinierte Attribute können angegeben werden. Mit einem Vertex Shader kann eine Geometrie auch verformt werden.

Geometry Shader

Mit den Geometry Shaders hat man Zugriff auf Informationen über benachbarte Kanten und Polygone. Außerdem ist es möglich neue Geometrien zu erzeugen.

Fragment Shader

Fragment Shader, auch Pixel Shader genannt, berechnen die Farbe und Tiefe der Pixel. Durch die freie Programmierbarkeit ist theoretisch jeder beliebige Effekt möglich.

Rendering-Pipeline

Die Rendering-Pipeline beschreibt die Schritte die nötig sind um eine 3D-Szenen darzustellen. Mit der Einführung programmierbarer Grafik-Hardware änderte sich auch die klassische Rendering-Pipeline, wie sie von Programmierschnittstellen genutzt wurde[Wright2005]. Die Shader ersetzen viele Teile und vereinfachen dadurch den Aufbau der Pipeline. Die Funktionalität steigt aber. Die Rendering-Pipeline kann in zwei Stufen unterteilt werden. In der ersten Stufe werden die Vertices verarbeitet, in der zweiten Stufe die Fragmente.

Alte Rendering-Pipeline Die alte Rendering-Pipeline beginnt mit den Transformationen der Vertices und der zugehörigen Normalen. Danach wird die Beleuchtung für jeden Vertex berechnet. Im nächsten Schritt werden

Textur-Koordinate generiert und transformiert. Die Textur-Koordinaten werden in der zweiten Stufe wieder benötigt. Beim sogenannten Clipping werden Polgone, bei denen Vertices außerhalb des Blickfeldes liegen, neu angeordnet, so dass alle Vertices im Blickfeld liegen. Dazu werden neue Vertices erzeugt, die Attribute werden interpoliert. Als letzter Schritt der ersten Stufe kommt die Rasterisierung. Hier wird die 3D-Szene auf das Raster des Bildschirms abgebildet.
Die zweite Stufe fängt mit der Texturierung an. Dabei werden die Texturen entsprechend den Textur-Koordinaten auf die Polygone gelegt. Nach der Texturierung wird der Nebel berechnet. Danach kommt das Anti-Aliasing um die Aliasing-Artefakte der Rasterisierung auszufiltern. Im letzten Schritt werden die Operationen auf die Fragmente, wie z.B. Alpha Testing, Blending und Depth Testing, ausgeführt.
Fast alle Schritte in der Pipeline sind konfigurierbar und können je nach Bedarf aktiviert und deaktiviert werden.

Neue Rendering-Pipeline In der neuen Rendering-Pipeline ersetzen die Shader große Teile. So wird als erster Schritt der Vertex Shader ausgeführt. Der zur Zeit noch neue Geometry Shader befindet sich in der Rendering-Pipeline direkt hinter dem Vertex Shader. Danach kommt wie in der alten Pipeline das Clipping und die Rasterisierung. Damit ist auch schon die erste Stufe abgeschlossen. Der Fragment Shader macht in der zweiten Stufe den Anfang. Die restliche Pipeline hat sich nicht geändert.
Erst durch diese einfachen Änderungen in der Rendering-Pipeline ist die freie Programmierung der Grafik-Hardware möglich geworden.

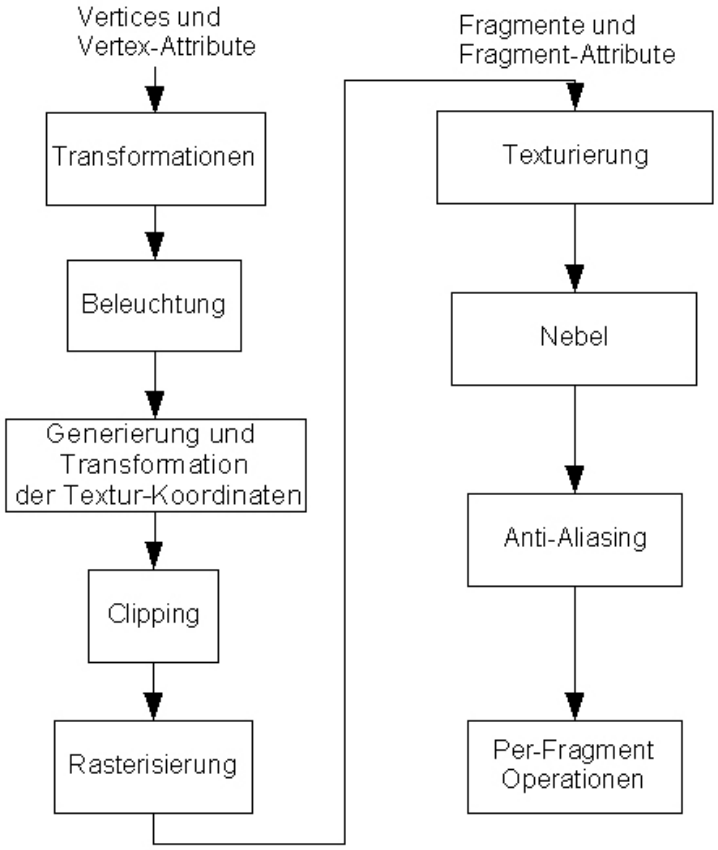

Abbildung 2.7: Die alte Rendering-Pipeline.

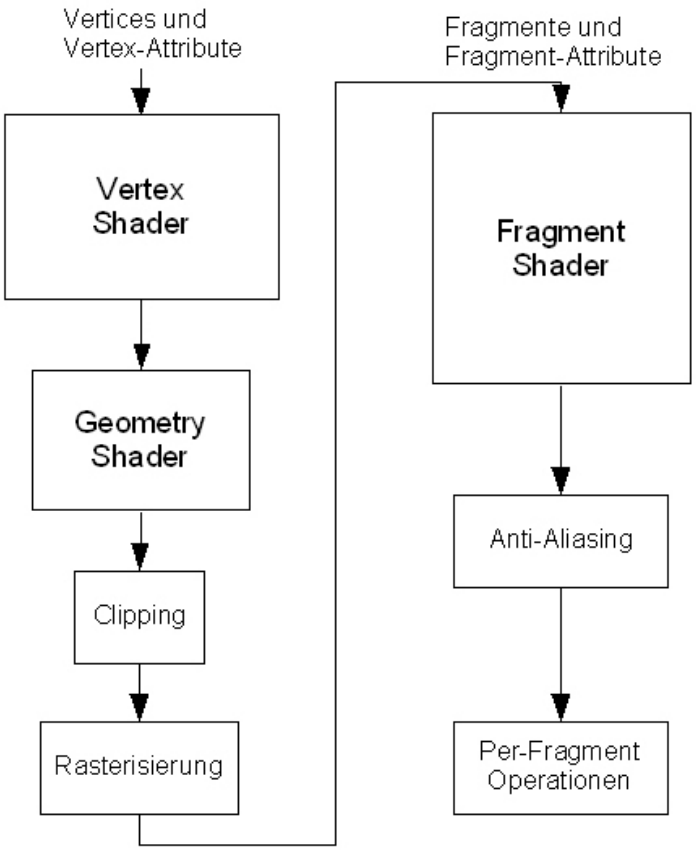

Abbildung 2.8: Die neue Rendering-Pipeline von programmierbarer Grafik-Hardware.

Kapitel 3

Konzept

Bei der Entwicklung eines dynamischen Beleuchtungssystems müssen die neuesten Technologien ausgenutzt werden um 3D-Szenen mit realistischer Beleuchtung bei interaktiven Frameraten darzustellen. Dabei soll die benötigte Rechenleistung möglichst niedrig bleiben. Bei allen Techniken muss deshalb ein Kompromiss zwischen visueller Qualität und Leistung gefunden werden. Dazu gehört auch die Nutzung von verschiedenen Optimierungen für aufwändige Rechenaufgaben. Nur so kann das System entsprechend den Anforderungen implementiert werden.

3.1 Beleuchtungsberechnung

Für eine realistische Beleuchtung wäre ein globales Beleuchtungsmodell am besten geeignet. Doch selbst moderne Grafik-Hardware ist nicht in der Lage die Berechnungen schnell genug auszuführen. Die globale Beleuchtung kann aber mit verschiedenen Techniken für statische Szenen vorberechnet werden. Die lokale Beleuchtung wird dynamisch berechnet. Daher ist ein Beleuchtungsmodell nötig, das sowohl schnell zu berechnen ist als auch realistisch aussieht. Das Beleuchtungsmodell von Phong erfüllt diese Bedingungen.
Die Beleuchtungsberechnungen werden in einem Shader durchgeführt. So kann das Beleuchtungsmodell je nach Bedarf geändert werden. Auch eine globales Beleuchtungsmodell könnte später mit einem neuen Shader hinzugefügt werden.
Die Lichtquellen werden in Primär- und Sekundärlichtquellen unterteilt, die sich dadurch unterscheiden, dass nur die Primärlichtquellen Schatten wer-

fen. In vielen Anwendungen haben die Sekundärlichtquellen eine sehr geringe Intensität und werfen nicht wahrnehmbare Schatten für die man sich die Berechnung sparen kann. Diese Optimierung erlaubt es, viel mehr Lichtquellen in der 3D-Szene zu platzieren.
Die Positionen der Sekundärlichtquellen können aus einem (animierten) Volumendatensatz gelesen werden. Dafür werden nur die Lichtquellen mit der höchsten Intensität berücksichtigt. Zur Positionierung der Primärlichtquelle wird die mittlere Position der Sekundärlichtquellen berechnet. So befindet sich die Primärlichtquelle immer im Mittelpunkt des Volumens und es können realistische Schatten berechnet werden.
Die Beleuchtungsberechnung kann sehr einfach in einem Fragment Shader ausgeführt werden. Da mit Deferred Shading gearbeit wird, müssen die für die Beleuchtung nötigen Information zuerst in einer Textur gespeichert werden. Hierfür ist noch ein weiterer Shader notwendig, der nur die Beleuchtungsinformationen ausgibt. Diese Informationen werden dann direkt in die Textur gezeichnet und zur Beleuchtungsberechnung benutzt.

3.2 Schattenberechnung

Für die Schattenberechnung stehen drei Techniken zur Auswahl, von denen es verschiedene Variationen gibt: Ray Traced Shadows, Volume Shadows und Shadow Mapping. Alle Techniken erzeugen realistische Schatten, haben aber auch ihre Vor- und Nachteile.
Mit Ray Traced Shadows können realistische Schatten erzeugt werden. Durch den hohen Rechenaufwand eignen sie sich aber zur Zeit noch nicht für interaktive Anwendungen mit dynamischer Beleuchtung.
Die Berechnung von Volume Shadows kann mit Grafik-Hardware beschleunigt werden. Zusammen mit verschiedenen Optimierungen kann diese Technik auch für interaktive Anwendungen benutzt werden. Jedoch wird viel Rechenleistung für die Konstruktion der Schattenvolumen benötigt. Für eine bessere Performance werden deshalb Shader benutzt und es wird versucht die Anzahl der Polygone in der Szene möglichst gering zu halten.
Shadow Mapping ist relativ einfach zu implementieren und kann auch mit Grafik-Hardware beschleunigt werden. Es hat aber einige Nachteile die jedoch alle umgangen werden können.
Tabelle 3.1 zeigt einen Vergleich der Techniken zu verschiedenen Aspek-

Anforderung	Shadow Mapping	Volume Shadows	Ray Traced Shadows
Einfache Implementierung	+	0	-
Geeignet für komplexe Szenen	+	-	-
Weiche Schatten	+	0	0
Performance	0	0	-

Tabelle 3.1: Vergleich der Techniken zur Schattenberechnung

ten die berücksichtigt werden müssen. Die Implementierung sollte möglichst einfach sein. Shadow Mapping ist mit relativ wenigen Befehlen realisierbar. Im Vergleich dazu sind Volume Shadows aufwändiger, da noch Code für die Konstruktion der Schattenvolumen benötigt wird. Ray Traced Shadows sind von der Theorie her zwar einfach zu implementieren, aber leider kann auch aktuelle Grafik-Hardware die Berechnungen nicht beschleunigen.
Die Technik sollte sich auch für komplexe 3D-Szenen eignen. Beim Shadow Mapping muss die Szene mehrfach gezeichnet werden, für die Berechnung der Schatten ist die Komplexität der 3D-Szene aber unwichtig. Volume Shadows eignen sich weniger für komplexe 3D-Szenen, da für jedes zusätzliche Polygon auch die Komplexität der Schattenvolumen steigt. Ray Traced Shadows benötigen schon bei kleinen 3D-Szenen viel Rechenzeit. Wegen der fehlenden Hardware-Unterstützung ist diese Technik ungeeignet.
Weiche Schatten bzw. Halbschatten sollen plausibel dargestellt werden. Shadow Mapping eignet sich dafür sehr gut, harte Schattenkanten können mit herkömmlichen Bildbearbeitungsverfahren ausgefiltert werden. Bei Volume Shadows und Ray Traced Shadows muss die Form der Lichtquelle durch viele Punktlichtquellen angenähert werden um realistische Halbschatten zu erreichen. Dies kostet allerdings viel Performance.
Die Performance der Technik ist ebenfalls sehr wichtig. Je nach Komplexität der 3D-Szene und Anzahl der Lichtquellen ist sie bei Shadow Mapping ein wenig besser als bei Volume Shadows. Bei weichen Schatten liegt Shadow Mapping allerdings deutlich vor Volume Shadows. Ray Traced Shadows hat, wieder durch die fehlende Hardware-Unterstützung, die schlechteste Perfor-

mance.
Da sich die Techniken vom Ergebnis her nicht wesentlich unterscheiden, wurde Shadow Mapping als Algorithmus zur Schattenberechnung gewählt. Shadow Mapping ist einfach zu implementieren, es eignet sich für komplexe 3D-Szenen, weiche Schatten sind ebenfalls einfach zu realisieren und es beansprucht im Vergeich zu den anderen Techniken weniger Rechenleistung.
Auch die Schattenberechnung nutzt die Möglichkeiten der Shader. Bei der Schattenberechnung wird zwischen Richtungs-Lichtquellen und Omnidirektionalen Lichtquellen unterschieden. Bei Richtungs-Lichtquellen kommt die Beleuchtung nur aus einer Richtung, wie wenn die Lichtquelle sehr weit entfernt wäre. Omnidirektionale Lichtquellen dagegen haben eine beliebige Position in der 3D-Szene und strahlen Licht in alle Richtung ab. Bei Richtungs-Lichtquellen genügt es eine Shadow Map aus der Richtung des Lichts zu zeichnen. Für Omnidirektionale Lichtquellen müssen 6 Shadow Maps gezeichnet werden. Bei einem Blickwinkel von 90 Grad wäre das jeweils eine Shadow Map für die Teile der 3D-Szene die sich links, rechts, vor, hinter, über und unter der Lichtquelle befinden.
Um alle Details der Szene in der 3D-Szene zu erfassen ist eine hohe Auflösung der Shadow Map notwendig. Die Auflösung kann je nach Szene variieren. Wählt man die Auflösung zu niedrig werden Aliasing-Artefakte sichtbar und es werden nur grobe Schatten dargestellt.
Shadow Mapping leidet außerdem unter Artefakten die durch mangelnde Präzision entstehen können. Entsprechend ihrem Aussehen werden diese Artefakte auch als Shadow Acne bezeichnet. Diese Artefakte lassen sich aber einfach umgehen. Abbildung 3.1 zeigt zwei Bilder mit Shadow Acne-Artefakten.
Weiche Schatten können mit Shadow Mapping auch relativ einfach realisiert werden. Hierfür wird auf das Ergebnis vom Shadow Mapping einfach ein Unschärfe-Filter angewandt. Um realistische Halbschatten zu erreichen, richtet sich die Größe des Filters nach der Größe der Lichtquelle und der Entfernung zum dem schattenwerfenden Punkt.

3.3 High-Dynamic-Range Rendering

Bei dem Einsatz vieler Lichtquellen wird eine hohe Präzision benötigt. Mit den Beschränkungen von LDR Rendering muss die Intensität der Lichtquel-

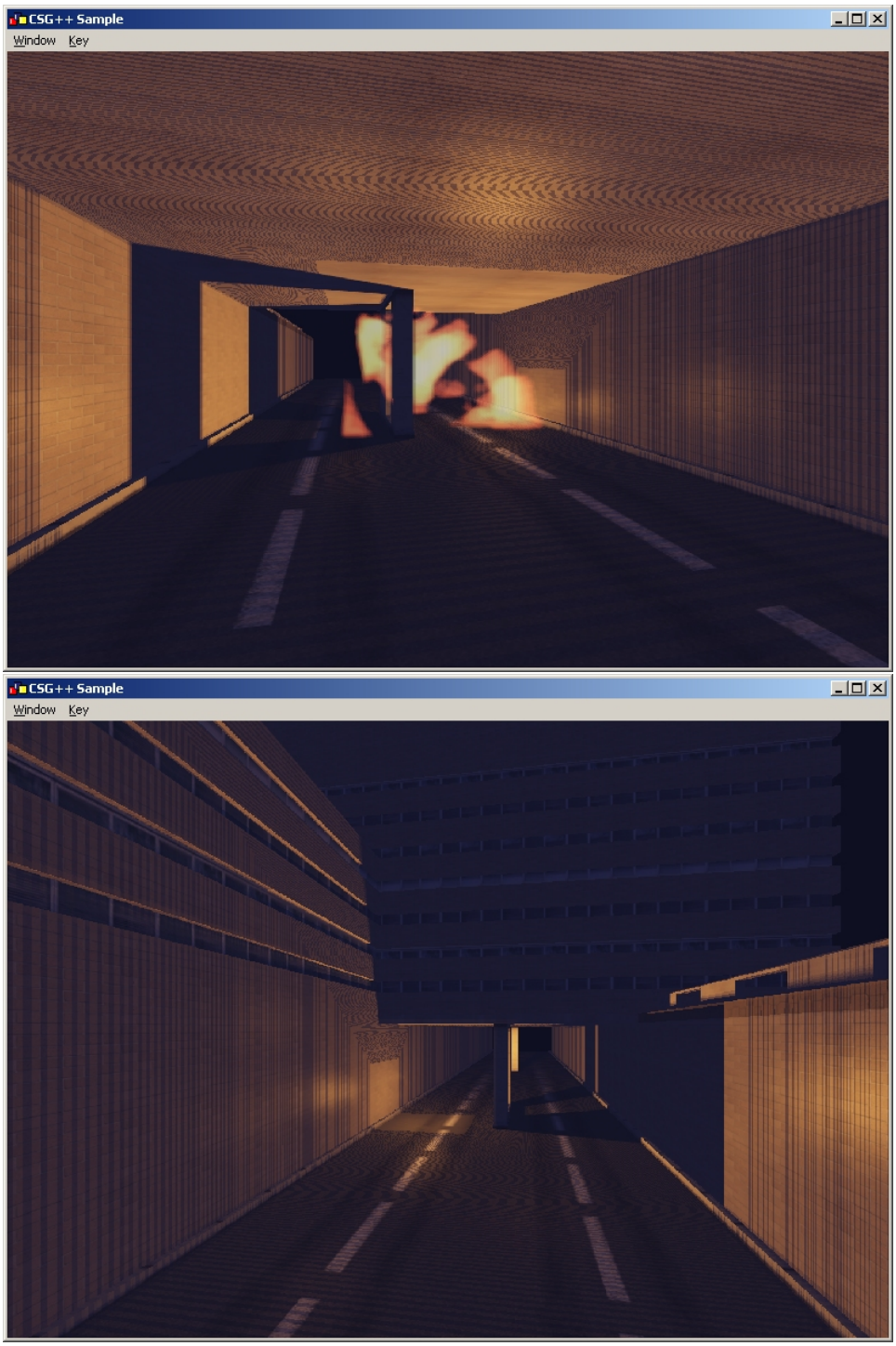

Abbildung 3.1: Shadow Acne-Artefakte durch eine zu geringe Präzision.

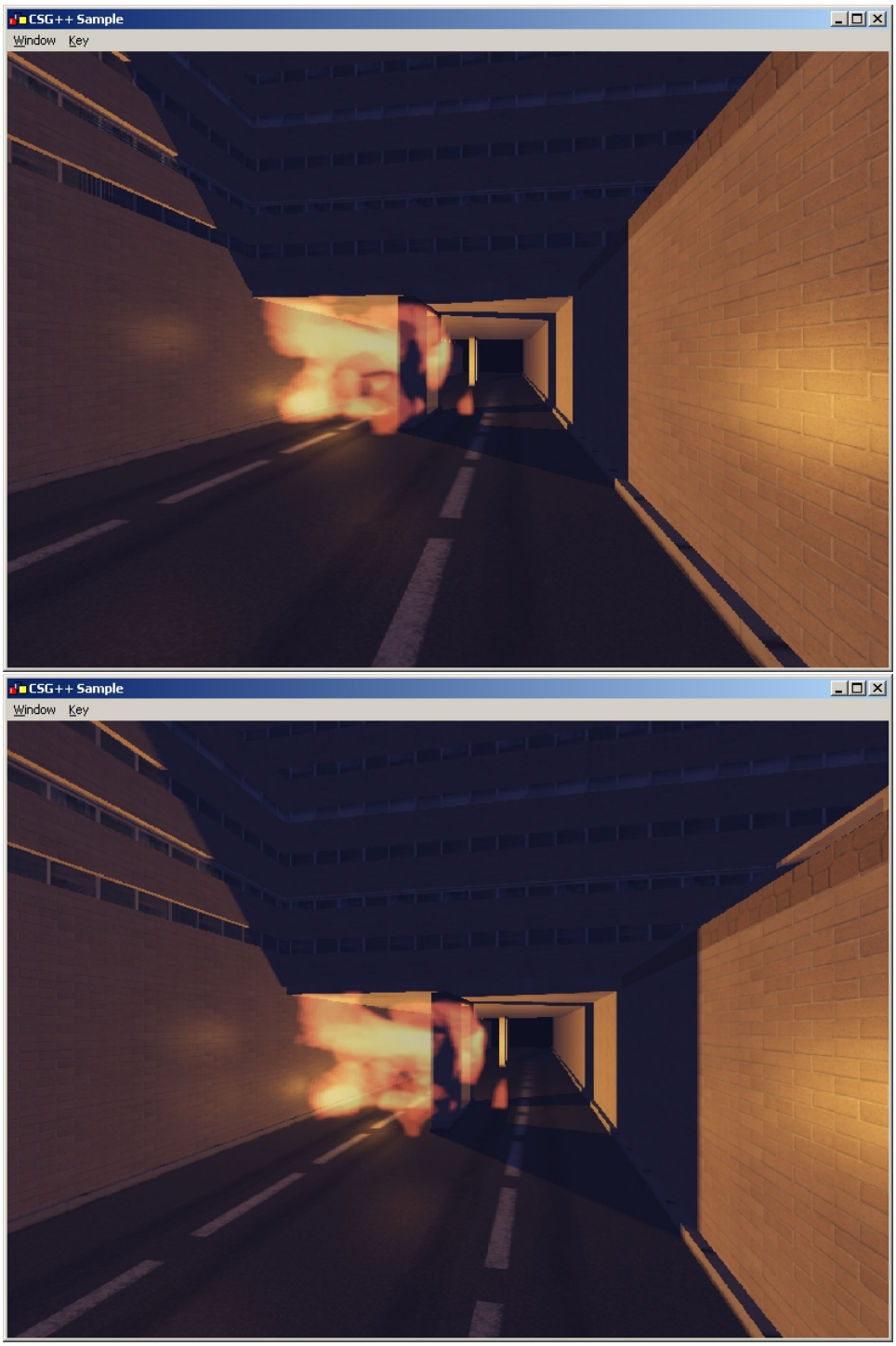

Abbildung 3.2: Oben: Harte Schatten. Unten: Weiche Schatten

len so skaliert werden, dass die Beleuchtung von allen Lichtquellen mit 8 Bit Präzision dargestellt werden kann. Je mehr Lichtquellen vorhanden sind, desto niedriger muss die Intensität skaliert werden. Werden zu viele Lichtquellen verwendet, entstehen dadurch sogenannte Banding-Artefakte, durch die eine Abstufung der Schattierung entsteht (siehe Abbildung 3.3). Ist die Skalierung zu niedrig, wird sie durch die mangelnde Präzision auf 0 gerundet, das Ergebnis ist ein schwarzes Bild.
HDR Rendering erlaubt es den Lichtquellen realistische Intensitäten ohne künstliche Skalierungen zu verleihen. Außerdem können die Banding-Artefakte mit Tone Mapping verhindert werden. Die Beleuchtung wirkt so realistischer und natürlicher.
Um die für HDR-Rendering benötigte hohe Präzision zu erreichen werden Fließkomma-Texturen eingesetzt. Diese speichern im Gegensatz zu normalen Texturen die Farbinformationen nicht als Bytes, sondern wie der Name schon sagt als Fließkomma-Zahlen.
Für HDR-Rendering wird zuerst die beleuchtete 3D-Szene in eine Fließkomma-Textur gezeichnet. Dabei werden realistische Intensitäten für die Lichtquellen verwendet. Im nächsten Schritt, dem Tone Mapping, müssen die in der Textur gespeicherten Intensitäten wieder in den für Monitore anzeigbaren Wertebereich umgewandelt werden. Für diese Umwandlung gibt es verschiedene Operatoren, implementiert wurde der Tone Mapping-Operator nach Hugo Elias. Dieser basiert auf der nicht-linearen Wahrnehmung von Helligkeiten, wie es bei den menschlichen Augen und auch Kameras der Fall ist. Außerdem ist die Berechnung mit Hilfe der Grafik-Hardware sehr einfach. Das Tone Mapping wurde mit einem Fragment Shader realisiert, so kann je nach Bedarf auch ein anderer Tone Mapping-Operator benutzt werden.
Die Belichtungszeit kann als Parameter angegeben werden. Alternativ könnte aber auch die mittlere Luminanz der Fließkomma-Textur, die die 3D-Szene zeigt, für eine automatische Einstellung der Belichtungszeit genutzt werden.
Abbildung 3.4 zeigt eine 3D-Szene mit unterschiedlichen Belichtungszeiten. Die Belichtungszeiten steigt in den Bildern a)-d) um jeweils 25 Prozent, bei Bild d) ist sie bei 100 Prozent angelangt. Bild e) zeigt die 3D-Szene mit der doppelten Belichtungszeit, Bild f) mit der zwölffachen Belichtungszeit.

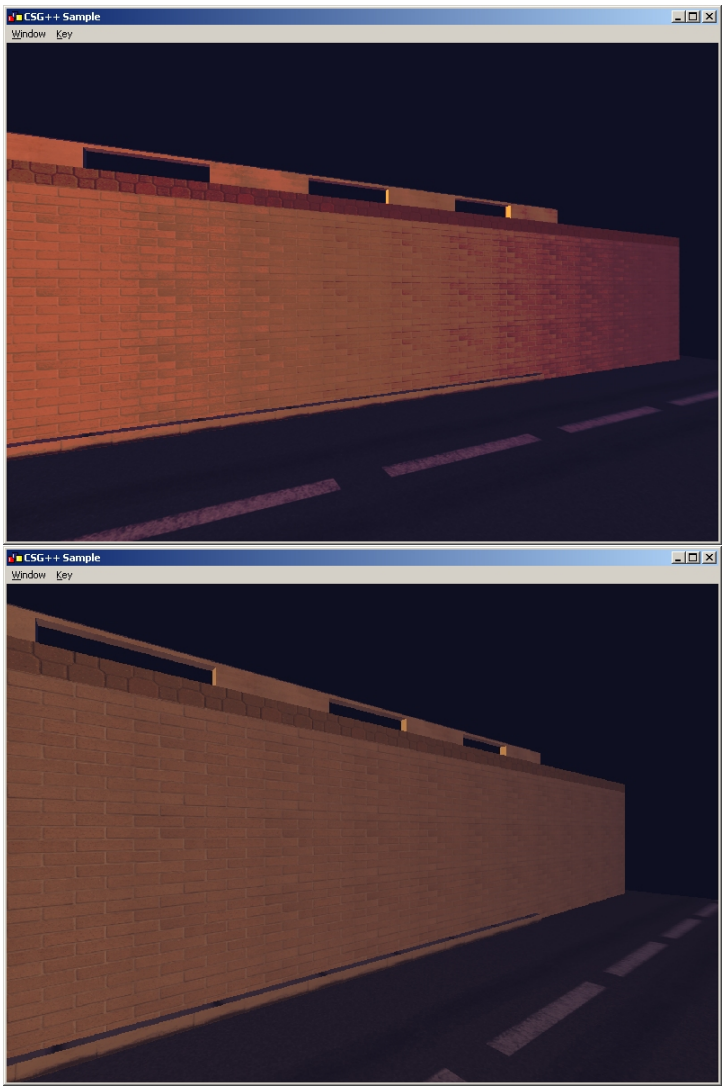

Abbildung 3.3: Oben: Banding-Artefakte ohne HDR-Rendering. Unten: Keine Banding-Artefakte mit HDR-Rendering und Tone Mapping.

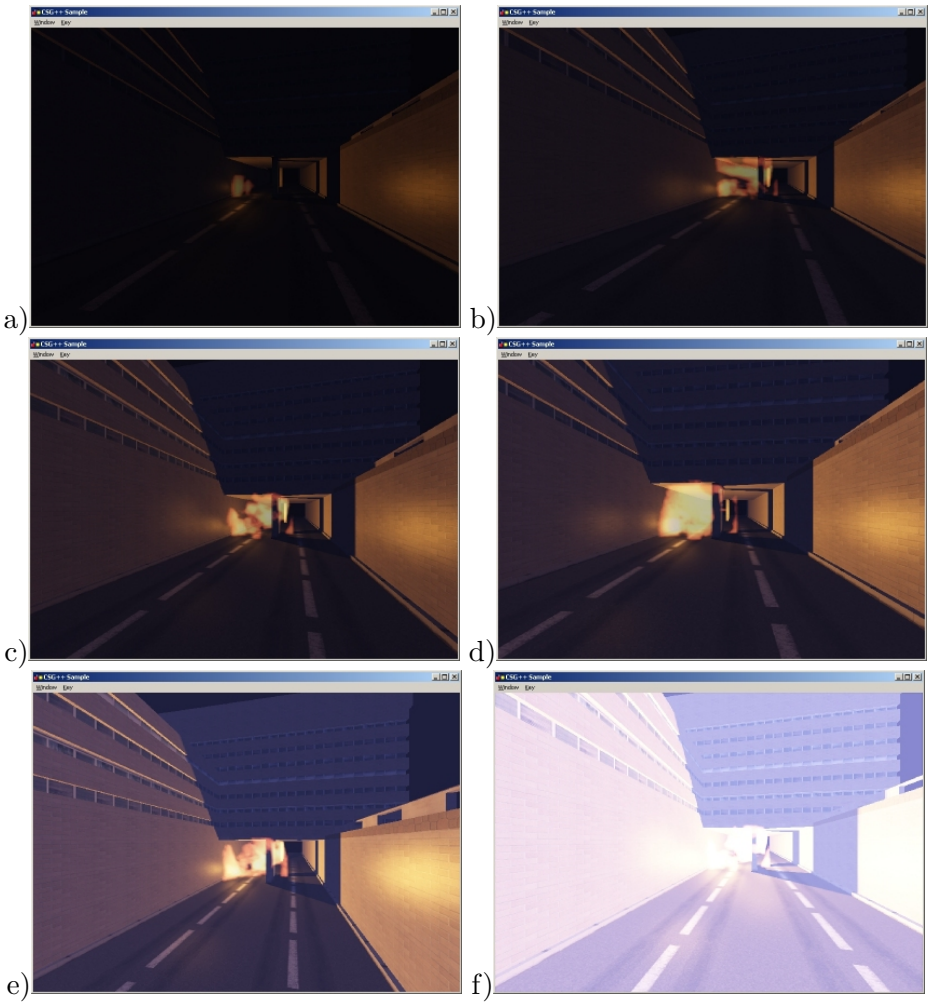

Abbildung 3.4: Die gleiche 3D-Szene mit unterschiedlichen Belichtungszeiten.

3.4 Volume Rendering

Zur Darstellung der Volumendatensätze muss ein Verfahren gewählt werden, das schnell zu berechnen ist und das Volumen möglichst genau wiedergibt. Mit moderner Grafik-Hardware wäre es zwar möglich das Volumen mit dem Ray Casting Verfahren zu berechnen, jedoch würde die Performance sinken. Auch eine bessere Qualität wäre nicht zu erwarten. Aus diesen Gründen wird Slice-Based Rendering zur Darstellung der Volumen benutzt. Durch den relativ geringen Rechenaufwand ist diese Technik auch auf älterer Grafik-Hardware realisierbar.
Das für die Darstellung der Volumen gewählte Slice-Based Rendering lässt sich relativ schnell implementieren. Zuerst werden die Ebenen (Slices), deren Anzahl frei einstellbar ist, parallel zur z-Achse positioniert. Nun müssen die Ebenen noch so ausgerichtet werden, dass sie immer zum Betrachter zeigen. Hierfür ist es nötig eine 3x3-Matrix zu berechnen, die die Ebenen entsprechend transformiert. Die Elemente der Matrix entsprechen der x-,y- und z-Achse des Koordinatensystems in dem die Ebenen zum Betrachter orientiert sind. Sie lassen sich mit wenigen Befehlen berechnen. Die z-Achse des Koordinatensystems ist gleich dem normalisierten Vektor vom Mittelpunkt des Volumens zum Betrachter. Bildet man das Kreuzprodukt zwischen der errechneten z-Achse und einer beliebigen y-Achse (meistens y=(0.0, 1.0, 0.0)) erhält man die x-Achse des gesuchten Koordinatensystems. Die noch fehlende y-Achse ergibt sich aus dem Kreuzprodukt zwischen z- und x-Achse. Nun sind alle Information für die Darstellung der Volumendatensätze vorhanden.
Da mit moderner Grafik-Hardware keine unendlichen Ebenen dargestellt werden können, werden stattdessen einfache Quadrate verwendet. Die Berechnung der benötigten 3x3-Matrix wird in einem Vertex Shader ausgeführt, ebenso wie die Transformation der Ebenen. Auch die Textur-Koordinaten müssen transformiert werden. Würden sie nicht transformiert werden, würde man immer die gleiche Seite des Volumens sehen.
Oft will man Daten die einen bestimmten Wert nicht überschreiten auch nicht anzeigen. Solche Daten werden im Alpha-Kanal der 3D-Textur gespeichert. Beim Zeichnen des Volumens wird dann einfach das Alpha Testing aktiviert um die Berechnung von Fragmenten mit unerwünschten Werten zu vermeiden.

3.5 Deferred Shading

Um eine realistische Beleuchtung zu erreichen ist der Einsatz von vielen Lichtquellen nötig. Im Idealfall gäbe es ein Licht für jeden Punkt in der 3D-Szene, dessen Intensität sich nach dem einfallenden Licht richten würde. Der Rechenaufwand dafür ist jedoch für interaktiven Anwendungen zu hoch. Deshalb wird versucht, möglichst viele Lichter zu berechnen. Auch zur Berechnung der Beleuchtung eines Feuers werden viele Lichtquellen benötigt, da jeder Punkt im Feuer Licht abstrahlt. Um den Beleuchtungseffekt eines Feuers zu berechnen, muss also mit vielen Lichtquellen die Form des Feuers angenähert werden.
Oft schränken die Fähigkeiten der Grafik-Hardware die Anzahl der Lichter, die man pro Rendering-Durchgang berechnen kann, ein. Diese Einschränkungen kann man jedoch umgehen indem man mehrere Rendering-Durchgänge kombiniert in denen nur ein paar Lichter berechnet werden. Dieses Vorgehen ist aber sehr langsam, weil für jeden Rendering-Durchgang die gesamte 3D-Szene neu gezeichnet werden muss. Durch den Einsatz von Deferred Shading lässt sich die Beleuchtungsberechnungen beschleunigen. Mit Deferred Shading wird die 3D-Szene nur einmal gezeichnet um die Beleuchtungsinformationen in einer Textur zu speichern. Für jeden Rendering-Durchgang muss nur noch ein Rechteck in der Größe des Bildschirms mit dieser Textur gezeichnet werden. Ein Shader benutzt dann die in der Textur gespeicherten Informationen um die Beleuchtung zu berechnen. Ohne Deferred Shading müsste also die 3D-Szene für jeden Rendering-Durchgang neu gezeichnet werden, mit Deferred Shading wird für jeden Rendering-Durchgang nur noch ein Rechteck gezeichnet, das die 3D-Szene als Bild zeigt. Deferred Shading verschiebt sozusagen die Beleuchtungsberechnungen.
Mit Deferred Shading kann sehr viel Rechenleistung gespart werden. Auch die Qualität der Visualisierung wird durch diese Optimierung nicht beeinträchtigt. So ist es möglich eine große Anzahl an Lichtern in komplexen 3D-Szenen zu berechnen.
Um die Beleuchtungsberechung zu optimieren wird aus diesen Gründen Deferred Shading benutzt. Zuerst müssen die für die Beleuchtungsberechnung nötigen Informationen in eine Textur gespeichert werden. Welche Informationen benötigt werden, hängt von dem verwendeten Beleuchtungsmodell ab. Für die Implementierung des Beleuchtungsmodell nach Phong werden die Farben und Normalen der Oberfläche benötigt, sowie die Position jedes

Fragments. Auch die Schatten und die Volumen müssen gespeichert werden. Mit den Fähigkeiten aktueller Grafik-Hardware lässt sich das relativ schnell realisieren. So bietet die Grafik-Hardware die Möglichkeit, Szenen direkt in eine oder sogar gleichzeitig in mehrere Texturen zu zeichnen. Zeichnet man in nur eine Textur, muss die 3D-Szene für jede der benötigten Informationen einmal gezeichnet werden. Dazu wird für jede Information jeweils ein Fragment Shader genutzt um die jeweilige Informationen auszugeben. Zeichnet man in mehrere Texturen gleichzeitig, dann muss ein einziger Fragment Shader die Ausgabe aller Texturen berechnen. Diese Option hat den Vorteil, dass die Szene nur einmal gezeichnet werden muss und nur ein Fragment Shader benötigt wird. Jedoch wird diese Fähigkeit nicht von älterer Grafik-Hardware unterstützt, weshalb sie vom entwickelten System zwar unterstützt, aber standartmäßig nicht benutzt wird.
Nachdem diese Informationen gespeichert wurden, kann mit der Beleuchtungsberechnung und der eigentlichen Darstellung der Szene begonnen werden. Dazu muss nur noch für jeden Rendering-Durchgang ein Quadrat über den gesamten Bildschirm gezeichnet werden, denen ein Shader und die Texturen mit den Beleuchtungsinformationen zugewiesen wird. Der Shader berechnet dann mit Hilfe der Beleuchtungsinformationen aus den Texturen die Beleuchtung. Ein Rendering-Durchgang berechnet die Beleuchtung von 8 Lichtern. Die Beleuchtung der Primären Lichtquelle wird alleine berechnet, so könnte man für sie ein komplexeres Beleuchtungsmodell verwenden während man für die sekundären Lichtquellen ein weniger rechenaufwändiges Beleuchtungsmodell verwendet.
Zuerst wird die Beleuchtung der primären Lichtquelle berechnet. In den darauf folgenden Rendering-Durchgängen wird die Beleuchtung der sekundären Lichtquellen berechnet. Die Ergebnisse der Beleuchtungsberechnung des gerade aktuellen Rendering-Durchgangs müssen mit den vorherigen Ergebnissen kombiniert werden. Da sich Licht additiv verhält müssen also die Ergebnisse addiert werden. Diese Operation kann mit dem sogenannten Blending(Mischen) realisiert werden. Dabei kann bestimmt werden wie die Farben eines zu zeichnenden Objektes mit den Farben die sich bereits im Framebuffer befinden gemischt werden. So wird einfach ein additives Blending nach dem ersten Rendering-Durchgang aktiviert und die Grafik-Hardware erledigt den Rest. Als Ergebnis zeigt sich die fertig beleuchtete 3D-Szene.

3.5.1 Deferred Shading und Volume Rendering

Deferred Shading hat den großen Nachteil das transparente Objekte nicht dargestellt werden können, da an einer Stelle immer nur für ein Fragment die Beleuchtungsinformationen gespeichert werden können. Für die Darstellung der Volumen wird aber Transparenz benötigt. Um trotzdem die Vorteile von Deferred Shading nutzen zu können, wird ein kleiner Umweg gegangen. Wie die anderen Beleuchtungsinformationen wird das Volumen vom Betrachter aus mit normalem Shading vor einem schwarzen Hintergrund in eine Textur gezeichnet. Damit das Volumen korrekt von anderen Teilen der 3D-Szene die sich vor dem Volumen befinden verdeckt wird, wird vor dem Zeichnen des Volumens zuerst die 3D-Szene nur in den Tiefenpuffer gezeichnet. Beim Deferred Shading wird dann zum Schluss einfach die Textur mit dem Volumen über die beleuchtete 3D-Szene gelegt.

Kapitel 4

Implementierung

Bei der Implementierung des Systems müssen die verwendeten Techniken sinnvoll miteinander kombiniert werden. So wirken sich die Techniken auch auf den Ablauf des Systems aus, dieser kann nicht beliebig verändert werden. Als Programmierschnittstelle zur Grafik-Hardware wird die OpenGL API [OpenGL] benutzt. Die OpenGL API besitzt die nötige Funktionalität für die verwendeten Techniken. Alle Techniken nutzen Shader, die mit der OpenGL Shading Language relativ einfach programmiert werden können.

4.1 Ablauf des Gesamtsystems

Der Ablauf des Gesamtsystems besteht aus vier Schritten: Initialisierung, Darstellung der 3D-Szene, Aktualisierung der 3D-Szene und De-Initialisierung. Während der Initialisierung werden alle für die Darstellung der 3D-Szene nötigen Daten geladen. Dazu gehört die 3D-Szene inklusive der Texturen sowie die Volumendatensätze. Die genutzten Shader werden aus einfachen Text-Dateien geladen. Außerdem werden die für das Shadow Mapping und Deferred Shading verwendeten Texturen initialisiert. Die Positionen der Lichtquellen für jeden Animationsschritt werden berechnet und gespeichert.
Für die Darstellung der 3D-Szene werden zuerst die Shadow Maps der Primärlichtquelle gezeichnet. Um Rechenzeit zu sparen wird die 3D-Szene zum Zeichnen der Shadow Maps nur in den Tiefenpuffer gezeichnet. Damit auch kleine Objekte in den Shadow Maps erfasst werden, wird eine Auflösung von 2048x2048 Pixel verwendet. Danach werden die Beleuchtungsinformationen für Deferred Shading in Texturen gespeichert. Hier reicht für die Texturen eine etwas geringere Auflösung von 1024x1024 Pixeln. Zuerst kommen

die Farben der Oberfläche, im Alpha-Kanal der Textur werden die Schatten gespeichert. Dann kommen die Positionen der sichtbaren Punkte und deren Normalen Vektoren, gefolgt von dem Volumen. Mit diesen Informationen kann die 3D-Szene mit Deferred Shading gezeichnet und beleuchtet werden. Zuerst wird die Primärlichtquelle berechnet, danach die Sekundärlichtquellen. Das Volumen wird am Ende über die beleuchtete 3D-Szene gelegt. Die beleuchtete 3D-Szene wird für das HDR-Rendering in eine Textur gezeichnet. Mit Tone Mapping werden die Werte in dieser Texture für die Anzeige auf dem Monitor skaliert. Das fertige 3D-Szene wird dann dem Benutzer angezeigt.
Nach der Darstellung muss die 3D-Szene aktualisiert werden. So wird z.B die neue Kameraposition berechnet und die Positionen der Lichtquellen werden neu gesetzt. Die Animation des Volumens wird ebenfalls aktualisiert, nach jeweils 1/25 Sekunden wird der nächste Schritt in der Animation angezeigt. Während der Aktualisierung werden auch Benutzereingaben, z.B. für die Bewegung der Kamera, verarbeitet. Sobald die Anwendung vom Benutzer beendet wird kommt die De-Initialisierung.
Zum Schluss werden wieder alle Daten gelöscht die während der Initialisierung geladen wurden. Der genutzte Speicher wird wieder freigegeben.
In Abbildung 4.1 wird dieser Ablauf illustriert.

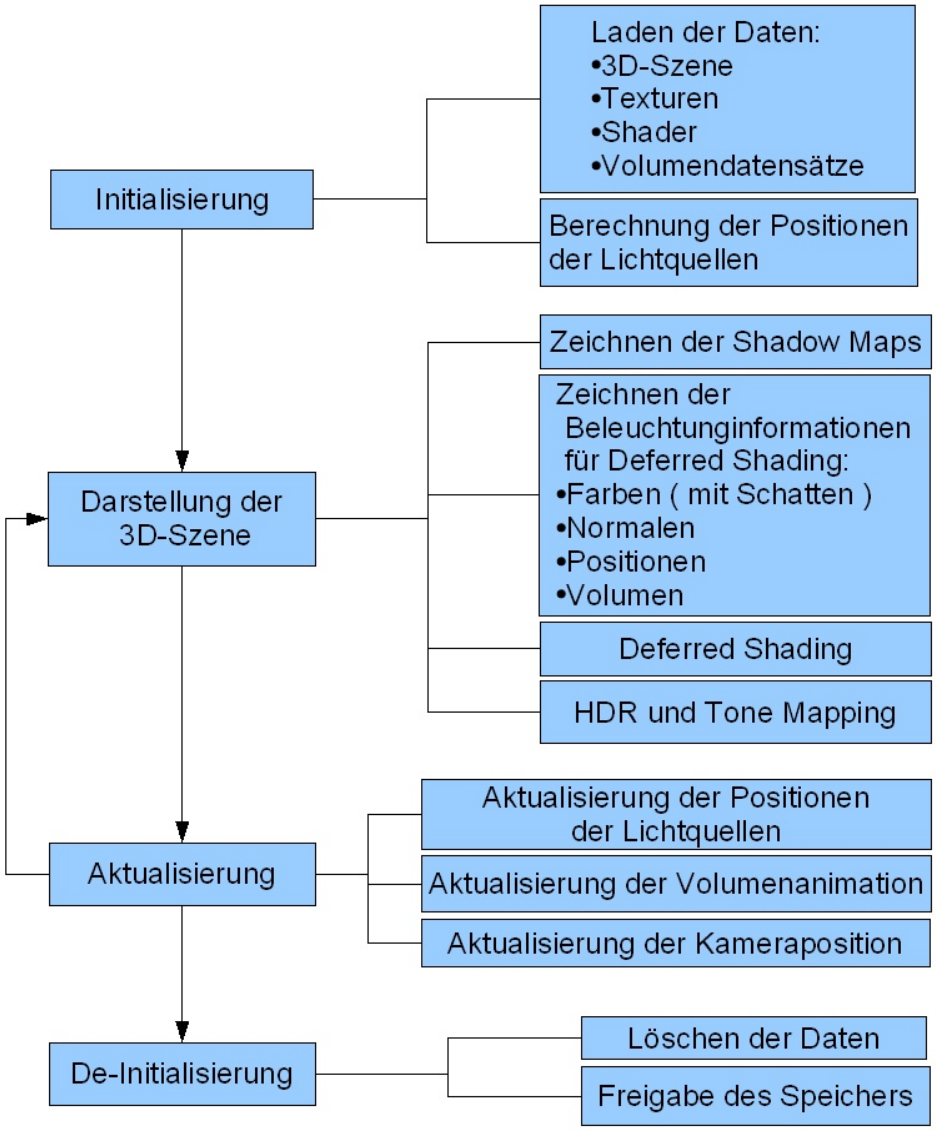

Abbildung 4.1: Ablauf des Systems.

Kapitel 5

Ergebnisse

Während der Entwicklung entstand eine Anwendung die alle Fähigkeiten des Systems ausnutzt. Die Anwendung ist die Simulation der Beleuchtung von Feuern, für die das System entwickelt wurde.

5.1 FireRenderer

FireRenderer ist eine Anwendung in der 3D-Szenen durch ein virtuelles Feuer beleuchtet und dargestellt werden können.
Die Anwendung kann frei konfiguriert werden. So kann die 3D-Szene oder das Volumen für das Feuer ausgetauscht werden, es sind keine Vorberechnungen nötig. Das Feuer kann frei in der 3D-Szene platziert werden. Die Eigenschaften des Feuer-Volumens sowie die Anzahl der Lichtquellen und ein Faktor für die Belichtungszeit können ebenfalls verändert werden.
Die 3D-Szene kann frei betrachtet werden. Die Animation des Feuers läuft in einer Endlosschleife, d.h. wenn die Animation zu Ende ist beginnt sie von neuem. Die Beleuchtung und die Schatten richten sich nach dem Feuer. Dadurch erlaubt die Anwendung unterschiedliche Brand-Szenarios zu testen. Die Beleuchtungsverhältnisse bei einem Brand können mit Hilfe der Anwendung abgeschätzt werden.
Abbildung 5.1 zeigt zwei Bilder aus der Anwendung.

5.2 Fazit

Das System eignet sich für fast alle Anwendungen in denen eine realistische und dynamische Beleuchtung einer 3D-Szene erforderlich ist. Die gewählten

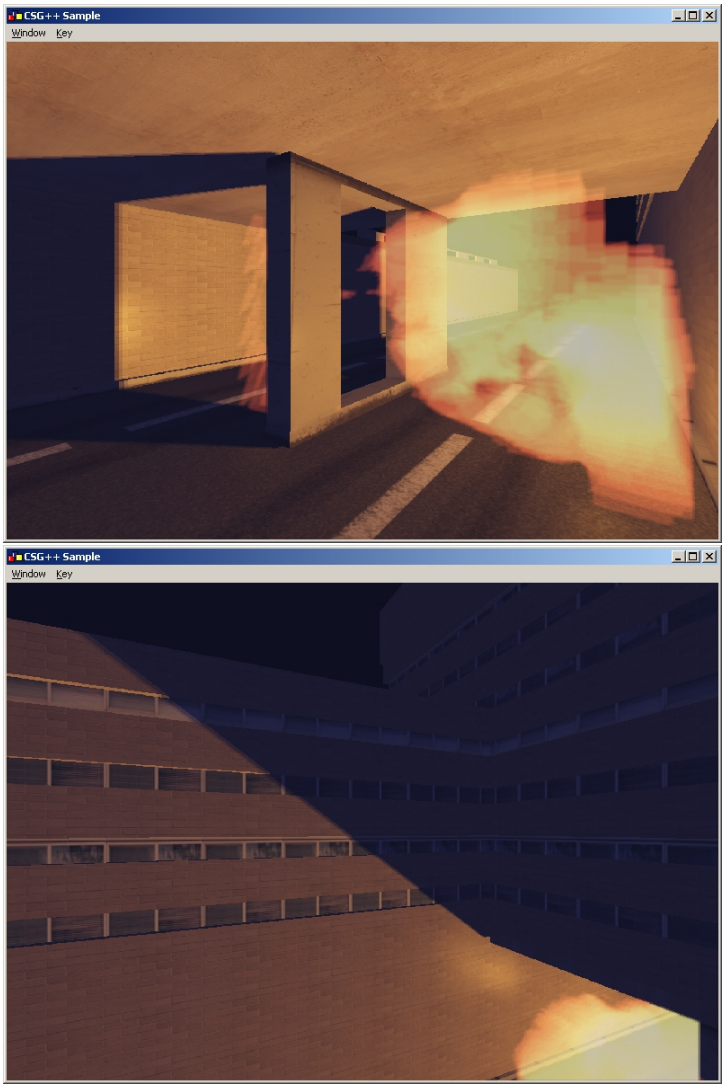

Abbildung 5.1: FireRenderer: Simulation der Beleuchtungsverhältnisse bei Bränden .

Techniken konnten relativ problemlos zu einem Gesamtsystem zusammengefügt werden. Durch die Nutzung von Deferred Shading mussten nur kleine Änderungen an den verschiedenen Algorithmen vorgenommen werden. Auch die geforderte Skalierbarkeit und Erweiterbarkeit konnte erfüllt werden.

Kapitel 6

Ausblick

Aufgrund der nur begrenzt zur Verfügung stehenden Entwicklungszeit konnten das System und die Anwendungen die das System nutzen nicht komplett fertig gestellt werden.
So können mit dem System zur Zeit nur Gasverbrennungen ohne Rauchentwicklung simuliert werden. Diese Einschränkung hat mehrere Gründe. Die vorhandenen Volumendatensätze haben nur eine geringe Auflösung und um Speicherplatz zu sparen wird nur das Volumen des Feuers gespeichert. Die Volumendaten der restlichen 3D-Szene sind somit nicht vorhanden und es können auch keine Annahmen über die Ausbreitung des Rauchs gemacht werden.
Bisher fehlt dem FireRenderer eine Benutzeroberfläche. Alle Einstellung müssen in einer Konfigurationsdatei festgelegt werden.

Literaturverzeichnis

[Wikipedia] http://de.wikipedia.org/wiki/Licht , zuletzt besucht am: 17.05.2007

[Kajiya1986] James T. Kajiya: The Rendering Equation, 13th Proceedings of SIGGRAPH, ACM Press, Dallas(Texas), 1986

[Jensen2001] Henrik Wann Jensen: Realistic Image Synthesis Using Photon Mapping, AK Peters, 2001

[Phong1975] Bui-Tuong Phong: Illumination for Computer Generated Pictures, Juni 1975

[Cook1982] Robert L. Cook and Kenneth E. Torrance: A Reflectance Model for Computer Graphics, ACM Transactions on Graphics Vol. 1 No. 1, Januar 1982

[Williams1978] Lance Williams: Casting Curved Shadows on Curved Surfaces, Computer Graphics Vol.12 No.3 (SIGGRAPH'78 Proccedings): pp. 270-274, 1978

[OpenGL] http://www.opengl.org , zuletzt besucht am: 17.05.2007

[Elias] Hugo Elias: The Exposure Function, http://freespace.virgin.net/hugo.elias/graphics/x_posure.htm , zuletzt besucht am: 17.05.2007

[Keller1997] A. Keller: Instant Radiosity, Proceedings of SIGGRAPH 97, Computer Graphics Proceedings, Annual Conference Series, Los Angeles (California), Atlanta (Georgia), 1997

[Wright2005] Richard Wright and Benjamin Lipchak: The OpenGL Superbible, 3rd Edition, Sams Publishing, 2005

[Calver2005] Dean Calver: Deferred Lighting on PS 3.0 with High Dynamic Range, ShaderX 3 - Advanced Rendering with DirectX and OpenGL, edited by W. Engel, Charles River Media, 2005

[Reinhard2002] E. Reinhard et al.: Photographic Tone Reproduction for Digital Images, ACM, 2002

[VREC] VREC: VRED. Benutzung mit freundlicher Genehmigung von VREC GmbH und Volkswagen AG.

[DOOM3] id Software: Doom III, Activision, 2004