

Institut für Parallele und
Verteilte Systeme
Abteilung Parallele Systeme

Universität Stuttgart
Universitätsstraße 38
D-70569 Stuttgart

Diplomarbeit Nr. 2996

Modellierung und Simulation bildgebender Systeme für partikelbeladene Strömungen

Uwe Schächterle

Studiengang:	Informatik
Prüfer:	Prof. Dr. Sven Simon
Betreuer:	Lars Rockstroh
begonnen am:	03.Dezember, 2009
beendet am:	10.Mai, 2010
CR-Klassifikation:	D.2.1, I.3.3, I.6.4

Inhaltsverzeichnis

1	Einleitung	1
1.1	Motivation	1
1.1.1	Aufgabenstellung	2
1.2	Hintergrund	3
2	Anforderungen	6
2.1	Anforderungen aus der Aufgabenstellung	6
2.2	Konkretisierung der Problemstellung	7
2.2.1	Partikel	7
2.2.2	Kamerasystem	7
2.2.3	Hohe Genauigkeit	19
2.2.4	Hohe Performance	20
2.2.5	Konzept der Simulation	21
2.2.6	Hohe Flexibilität	22
3	Simulator-Konzept	24
3.1	Schrittweise Simulation der Zeit	24
3.2	Entwicklung der Partikel-Texturformel	27
3.3	Ablauf einer Simulation	29
4	Formeln und Abhängigkeiten	30
4.1	Einheiten im Simulator	30
4.2	Formeln der Partikelerzeugung	30
4.3	Positionsabhängige Effekte	32
4.4	Positionsunabhängige Effekte	33
5	Implementierung	36
5.1	Vorbereitende Schritte	36
5.2	Durchführung der Simulation	40
5.2.1	Schrittweise Simulation der Zeit	41
5.2.2	Aufnahme eines Zeitpunktes	41
5.3	Nachbereitung	45
5.4	Callbacks	45
5.4.1	Setzen der Custom Bytes für alle Partikel	46
5.4.2	Erzeugen der Partikel mittels Callback	47
5.4.3	Setzen der Custom Bytes der Partikel, nach regulärem Erzeugen	48
5.4.4	Bewegen eines einzelnen Partikels	48
5.4.5	Bewegen aller Partikel, inklusive Löschen	49
5.4.6	Erzeugen einer Textur für die Rendering eines Partikels	50
5.4.7	Berechnung des Rauscheffektes der Ausgabe	52

6	Validierung des Simulators	53
7	Zusammenfassung und Ausblick	55
8	Anhang	56
8.1	Der Plotter	56
8.1.1	Mathematische Grundlagen	57
8.1.2	Normierte Darstellung	58
8.2	Der Info Viewer	58
8.3	Aufbau der Konfigurationsdatei	59
8.3.1	Die Allgemeinen Einstellungen (General)	60
8.3.2	Die Kameraeinstellungen (Cameras)	61
8.3.3	Die Partikelkanonen-Einstellungen (Particle Guns)	64
8.4	Konfigurationsbeispiele	65

Abbildungsverzeichnis

1.1	Übersicht Partikelsystem entnommen aus [20]	3
1.2	Übersicht Partikelerzeugung entnommen aus [20]	4
2.1	Gegenüberstellung telezentrische / perspektivische Linse, v.l.n.r.: Aufbau, telezentrisch, perspektivisch	8
2.2	Sichtfelder von Pixeln bei verschiedenen Blenden	9
2.3	Veranschaulichung des Vignettierungseffektes	9
2.4	Beispiele Verzerrungsformen: Kissenförmige, keine, tonnenförmige Verzerrung	10
2.5	Objekte innerhalb und außerhalb des Fokus	11
2.6	Verbesserung des DOF durch kleinere Blende	11
2.7	Darstellung der Frequenzzzerlegung eines Signales	12
2.8	Diskretisierung eines eindimensionalen Signales	13
2.9	Faltung zweier Signale	13
2.10	Missachtung des Abtasttheorems, Extraktion des Rücktransformationsspektrums	13
2.11	Rekonstruktion des Frequenzspektrums	14
2.12	Links Orginal, Rechts Moiré-Effekt aufgrund einer Unterabtastung	14
2.13	Schematische Darstellung zweier Helligkeitssensoren bei der Ermittlung ihrer Ausgabewerte	14
2.14	Originalbild einer exponentiell anwachsenden Ortsfrequenz	15
2.15	Ausgabebild des Bildsensors einer exponentiell anwachsenden Ortsfrequenz	15
2.16	Darstellung der Amplitude der Farbwerte des Originalbildes und des ermittelten Bildes	15
2.17	Siemensstern mit Bemaßung	16
2.18	Beugungsscheibchen an verschiedenen Blendenformen	17
2.19	Szene mit Berücksichtigung der Punktspreizfunktion	17
2.20	Entstehung von Bewegungsunschärfe	18
2.21	Szene mit geblurten Objekten	19
2.22	Unterschiedliche Abtastraten bei variierender Auflösung v.l.n.r.: Original, 8-Bit Stufen, 32-Bit Stufen	20
3.1	Skizze zum Aufbau einer Szene	24
3.2	Beispielhafter Zeitstrahl einer Simulation	25
3.3	Darstellung Lebensraum mit Partikeln	25
3.4	Partikel durch Quader dargestellt	27
3.5	Partikel durch eine Textur dargestellt	27
3.6	Partikel durch drei Texturen dargestellt	28
3.7	Funktionsweise der Partikeltextur-Erzeugungsfunktion	28
3.8	Partikel durch Texturformel dargestellt	29

3.9	Darstellung der Szene mit Zusatzinformationen	29
4.1	Abhängigkeiten der Partikelparameter	30
5.1	Darstellung Lebensraum mit Partikeln	37
5.2	links : Darstellung einer Szene mit Tiefe, rechts Beispiel Alfablending . .	39
5.3	Auswirkungen der begrenzten Auflösung	43
5.4	Darstellung des Verzerrungseffektes v.l.n.r. : Original, tonnenförmige, kistenförmige Verzerrung	44
5.5	Auswirkungen der Vignettierung	44
5.6	Auswirkungen des Rauschens	45
5.7	Speicherbelegung eines Partikels	47
5.8	Speicherbelegung aller Partikel	49
5.9	Veranschaulichung Partikel Callback	51
5.10	Speicheraufbau einer Textur	51
6.1	Aufnahme einer realen Szene	53
6.2	Simulierte Szene mit zwei Partikelkanonen	54
6.3	Aufnahmen alle 100 Zeitschritte einer Simulation	54
8.1	Der Plotter, Darstellung einer Formel eindimensional	56
8.2	Der Plotter, Darstellung einer Formel zweidimensional	56
8.3	Der Info Viewer, Darstellung eines Datensatzes	58

Tabellenverzeichnis

4.1	Formeln der Partikel Erzeugung	31
4.2	Kameraeffekte kategorisiert nach Partikelposition	32
4.3	Formeln zur Realisierung der Schärfentiefe	32
4.4	Parametrisierung der Texture Erzeugungs Funktion	33
4.5	Parametrisierung der Fourier-Transformation	33
4.6	Parametrisierung des Verzerrungseffektes	34
4.7	Parametrisierung des Vignettierungseffektes	34
4.8	Parametrisierung des Rauscheffektes	35
5.1	Übersicht des Mode Parameters der Rausch Callback	52
8.1	Übersicht Mathematische Operatoren und ihre Bindungsstärke	57
8.2	Übersicht der Wahrscheinlichkeitsmodelle	57

Listings

5.1	C-Header für Set_Custom_Bytes_Callback	46
5.2	C-Header für Create_Particle_Callback	47
5.3	C-Header für Set_Custom_Bytes_Callback	48
5.4	C-Header für Move_Particle_Callback	48
5.5	C-Header für Move_All_Particle_Callback	49
5.6	C-Header für Create_Texture_Callback	50
5.7	C-Header für Noise_Callback	52
8.1	Konfiguration der XML-Datei für Abbildung 6.2 und 6.3	65
8.2	Konfiguration der XML-Datei für Abbildung 2.19	69

1 Einleitung

1.1 Motivation

In dieser Diplomarbeit ist das Ziel, einen möglichst realistischen Nachbau bereits vorhandener Versuchsaufbauten für partikelbeladene Strömungen in Software zu entwickeln. Anhand der Ausgaben dieser Software sollen die Auswertelgorithmen von Bild basierenden Messmethoden für Strömungsanalysen validiert und deren Fehler bestimmt werden können. Bei der Realisierung der Software spielen besonders die physikalischen Parameter des verwendeten Kamerasystems, sowie der Partikel und ihrer Trägerströmungen eine Rolle.

Für die korrekte Simulation eines Kamerasystems gilt es, seine für die Bildgebung relevanten Eigenschaften entsprechend umzusetzen. Dazu müssen die dafür in Frage kommenden Parameter erarbeitet und spezifiziert werden. Anschließend muss eine geeignete Implementierung diese korrekt nachahmen. Ein Kamerasystem besteht aus den Komponenten Linsen, Shutter und Bildsensor. Durch die Linsen entstehen die Effekte Vignettierung, Verzeichnung, Geisterflecken und Schärfentiefe. Der Shutter legt die Belichtungszeit einer Aufnahme fest und hat damit Einfluss auf die Bewegungsunschärfe. Durch den Bildsensor entstehen die Effekte Alias-Effekt und Begrenzte Auflösung. Zu all diesen Effekten fügt sich noch der Effekt Rauschen hinzu. Das Rauschen kann sowohl auf dem Bildsensor, als auch von außen durch die Linsen entstehen.

Ein Maß für die Qualität eines Kamerasystems stellt die Modulationsübertragungsfunktion (MTF) [12] dar. Die MTF gibt die Abschwächung des Kontrastes der in der aufgenommenen Szene vorkommenden Ortsfrequenzen (auch als Linienpaare pro Millimeter bezeichnet) an. Diese Abschwächung kommt durch die Effekte Vignettierung, Schärfentiefe, Alias-Effekt und Begrenzte Auflösung zu Stande. Durch die normierte Darstellung der MTF lassen sich verschiedene Kamerasysteme schnell und einfach vergleichen.

Die Umsetzung der partikelbeladenen Strömungen wird mittels eines zu entwerfenden Partikelsystems [20] realisiert. Hierfür müssen die speziellen Ansprüche der Aufgabenstellung für das Partikelsystem erarbeitet und umgesetzt werden. Das Partikelsystem soll hierbei möglichst flexibel gehalten werden. Durch diese Maßnahme soll ein Einsatz in anderen Gebieten ebenfalls ermöglicht werden.

1.1.1 Aufgabenstellung

Eine Vielzahl von Fertigungsprozessen wie das Lackieren und thermokinetische Beschichten basieren auf partikelbeladenen Strömungen. partikelbeladene Strömungen bestehen aus einem Transportmedium wie Gas, Luft oder Flüssigkeit, welches flüssige oder feste Partikel befördert. Charakteristika der Partikel, wie Verteilung, Geschwindigkeit, Größe und Temperatur, haben entscheidenden Einfluss auf Qualität und Effizienz des Produktionsprozesses mit Sensoren erfasst und ausgewertet. Bildbasierte Messmethoden nutzen bildgebende Systeme auf Basis von CMOS- oder CCD-Sensoren, um die Partikel auf Bildaufnahmen als Partikelabbildungen zu erfassen. Bildaufnahmen mit bis zu mehreren tausend Partikelabbildungen werden ausgewertet und Partikel-Charakteristika mit Algorithmen der Bildererkennung extrahiert.

Die Bestimmung des Fehlers einer Messmethode ist essentiell für die Beurteilung der Messergebnisse. Der Fehler bildbasierter Messmethoden wird durch die Auswertalgorithmen und das bildgebende System bestimmt. In dieser Diplomarbeit ist eine Simulationsumgebung für bildgebende Systeme im Bereich partikelbeladener Strömungen mit dem Ziel zu entwickeln, den Fehler bildbasierter Messmethoden zu bestimmen. Die Modellierung des bildgebenden Systems soll Parameter des Bildsensors (z.B. Pixelanzahl, Rauschen) als auch Eigenschaften der abbildenden Optik (z.B. Blendeneinstellung) umfassen. Ebenso sollen alle für die Abbildung der Partikel relevanten Eigenschaften, wie Temperatur, Verteilung, Geschwindigkeit, Größe sowie Dichte und Farbe des Transportmediums simuliert werden.

Die Anwendung der Auswertalgorithmen auf generierte Bilddaten soll die Bestimmung des Fehlers bildbasierter Messmethoden (bildgebendes System und Auswertalgorithmen) ermöglichen. Hierfür müssen Charakteristika der abgebildeten Partikel zusammen mit den Bilddaten ausgegeben werden. Eine Zuordnung von Bildposition zu Partikel-Charakteristika soll möglich sein.

Entwicklung und Realisierung der Auswertalgorithmen sind nicht Bestandteil dieser Diplomarbeit. Die Simulationsumgebung ist universell und erweiterbar auszulegen.

1.2 Hintergrund

Wie [20] zu entnehmen ist, wurden die ersten Partikelsysteme in den 1960er Jahren entwickelt. Partikelsysteme werden hauptsächlich in Computerspielen [8], zum Beispiel als Triebwerksstrahlen von Raumschiffen oder Simulationen von Rauch, Gas,[4] Feuer, Explosionen[26] und Schwärmen von Objekten eingesetzt. William T. Reeves legte mit seinem in [20] beschriebenen Partikelsystem den Grundstein für heutige Simulationen, indem er als erster die Partikel als Objekte zu modellieren versuchte.

Jedes Partikelsystem befindet sich in seinem eigenen lokalen Koordinatensystem, dessen Ursprung ein Punkt im dreidimensionalen Raum bzw. Weltkoordinatensystem ist. Die Orientierung des lokalen Koordinatensystems im Weltkoordinatensystem wird mit-

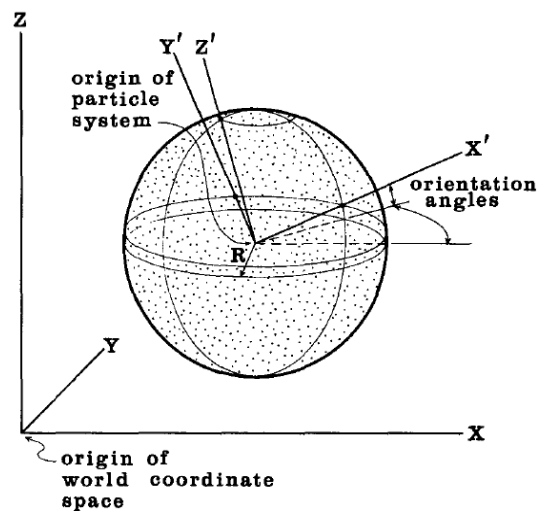


Abbildung 1.1: Übersicht Partikelsystem entnommen aus [20]

tels zweier Rotationswinkel und einer Verschiebung beschrieben. Um den Ursprung des lokalen Systems befindet sich eine sogenannte „generation shape“, zu Deutsch „Erzeugungsfäche“, innerhalb der die Partikel generiert werden können. Sie kann kugelförmig, kreisförmig oder rechteckig sein. Bei einer kugelförmigen Fläche bewegen sich die Partikel vom Ursprung weg, siehe Abbildung 1.1. Bei der kreisförmigen und der rechteckigen Erzeugungsfäche bewegen sie sich senkrecht von der Emissionsfläche weg, wobei ein sogenannter „ejection angle“, zu Deutsch „Austrittswinkel“, dem Partikel eine geringe zufällige Abweichung zur Senkrechten gibt siehe Abbildung 1.2. Die gesamte Simulation besteht aus den vier folgenden Schritten :

1. Erzeugung der Partikel
2. Bewegen der Partikel
3. Rendern der Partikel
4. Löschen der Partikel

1. Erzeugung der Partikel

Zur Erzeugung neuer Partikel kann es verschiedene je nach Anwendung spezifische

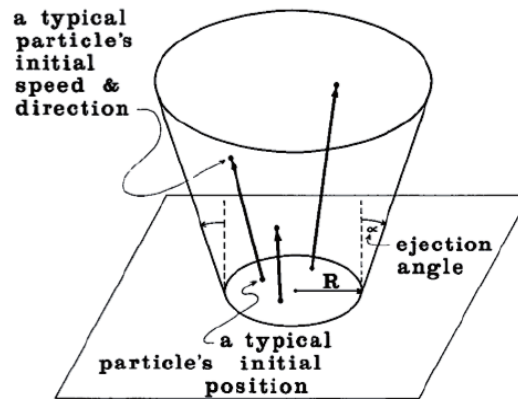


Abbildung 1.2: Übersicht Partikelerzeugung entnommen aus [20]

Gründe geben. So gibt es Anwendungen, welche die Gesamtzahl der Partikel vorschreiben, ein gelöschtes Partikel wird hier sofort durch ein neues ersetzt. Oder der Nutzer gibt eine durchschnittliche Anzahl an Partikeln an, welche pro simuliertem Zeitschritt erzeugt werden sollen. Die Anzahl der Partikel kann auch aufgrund der Größe, welche das Objekt auf der abgebildeten Szene einnimmt, bestimmt werden.

Bei der Erzeugung eines Partikels müssen verschiedene, von stochastischen Prozessen abhängige, Variablen gesetzt werden. Diese sind

- Initiale Position
- Initiale Geschwindigkeit und Richtung
- Farbe / Transparenz
- Größe
- Form
- Lebenszeit

2. Bewegung der Partikel

Zu jedem simulierten Zeitschritt werden alle erzeugten Partikel bewegt. Je nach Art der Simulation werden in diese Bewegungsberechnungen verschiedene Gravitationseffekte und andere Einflüsse mit einbezogen.

3. Rendern der Partikel

Ein gerendertes Partikel wird als Punkt-Licht-Quelle realisiert. Dies bedeutet, dass jedes Partikel für sich gesehen ein leuchtendes Objekt darstellt. Diese Form der Rendering eignet sich jedoch nicht zur Darstellung von Wasser oder Wolken.

4. Löschen der Partikel

Für die Löschung eines Partikels gibt es wie bei der Erzeugung verschiedene Ursachen. Die bei der Erzeugung erteilte Lebenszeit bildet die obere Schranke. Ein Partikel kann jedoch auch gelöscht werden, wenn seine Lichtintensität zu gering geworden ist, oder es sich in nicht mehr abgebildeten Bildbereichen aufhält. Bei der Simulation von Feuer

könnte ein Auftreffen eines Partikels auf ein Wasserobjekt ebenfalls ein Grund für ein Löschen sein.

2 Anforderungen

Dieses Kapitel versucht die genauen Funktionalitäten des späteren Simulators zu erörtern.

2.1 Anforderungen aus der Aufgabenstellung

Aus der Aufgabenstellung ergeben sich direkt folgende Anforderungen:

Partikel

Der Simulator soll beliebig viele Partikel simulieren, welche sich mit einer vorgeschriebenen Bewegungsart, Größe und Farbe im Raum bewegen. Zur Aufnahme dieser Partikel wird ein Kamerasystem benötigt.

Kamerasystem

Ein Kamerasystem besteht aus beliebig vielen Kameras, welche die gesamte Szene von unterschiedlichen Positionen und aus unterschiedlichen Richtungen aufnehmen. Jede Kamera soll die Szene mit eigenen Parametern aufzeichnen können. Die von den Kameras erzeugten Daten sollen in einer geeigneten Form gespeichert und später validiert werden können. Die Simulation soll hierbei eine möglichst hohe Genauigkeit aufweisen.

Hohe Genauigkeit

Nur eine numerisch möglichst genaue Simulation kann auch genaue Kameraausgaben erzeugen. Hierbei sind zum einen möglichst hochauflösende Daten und zum anderen eine möglichst exakte Nachbildung physikalischer Effekte des Linsen-Systems der Kameras erforderlich. Ebenfalls soll die Simulation in einer möglichst effektiven Art und Weise stattfinden.

Hohe Performance

Der gesamte Simulationsprozess soll möglichst effizient auf beliebigen Architekturen stattfinden. So sollen Mehrfachberechnungen möglichst vermieden werden und Schwächen des Speicherverwaltungssystems durch geeignete Implementierungen umgangen werden. Alle weiteren Ansprüche werden unter dem Konzept der Simulation zusammengefasst.

Konzept der Simulation

Eine Simulation soll wiederholbar, speicherbar und zu späterem Punkt fortsetzbar sein. Ebenfalls sollen unvorhersehbare Programmabstürze nicht zum Kompletterlust aller Simulationsdaten führen. Zuletzt ist eine möglichst hohe Flexibilität gefordert.

Hohe Flexibilität

Um eine hohe Flexibilität gewährleisten zu können, soll der Nutzer einen möglichst hohen Einfluss auf Funktionalitäten des Simulators haben. Ebenfalls soll der gesamte Funktionsumfang nachträglich erweiterbar sein.

2.2 Konkretisierung der Problemstellung

In diesem Abschnitt werden alle Anforderungen aus 2.1 nochmals betrachtet. Aus dieser Betrachtung heraus wird dann die konkret zu lösende Problemstellung erarbeitet.

2.2.1 Partikel

Ein Partikel ist ein punktsymmetrisches Objekt, welches sich auf einer geradlinigen Flugbahn bewegt. Der Simulator soll sehr viele Partikel simulieren können. Diese sollen nach groben Kriterien zusammengefasst werden. Eine Zusammenfassung nach diesen Kriterien wird als Partikelkanone bezeichnet. Jede Partikelkanone soll hierbei jeweils nur eine einzige Gruppe von Partikeln erzeugen. Zu den Eigenschaften einer Partikelkanone werden folgende Attribute zusammengefasst:

- Bereich der Erzeugung der Partikel
- Größe der Partikel
- Vorzugsrichtung der Bewegung
- Geschwindigkeit
- Aussehen (in Form von Helligkeit und Form)
- Entstehungshäufigkeit

Diese Eigenschaften sollen alle mit Hilfe von mathematischen Formeln definiert werden können. Zusätzlich soll ein Zufallsgenerator für entsprechende Varianzen sorgen. Hierbei sollen 2 Zufallsmodelle verfügbar sein:

- Gleichverteilung, die Wahrscheinlichkeit des Wertes einer Zufallsvariable ist über den gesamten Wertebereich gleich groß
- Gaußverteilung, die Wahrscheinlichkeit des Wertes einer Zufallsvariable entspricht der Gauß Glockenkurve (Standardnormalverteilung).

2.2.2 Kamerasystem

Zur korrekten Realisierung des Kamerasystems müssen die aus 1.1 aufgelisteten Kamera-Eigenschaften für jede einzelne Kamera im System umgesetzt werden. Je nach verwendetem Linsensystem ergibt sich ein telezentrisches oder perspektivisches Objektiv. Die Art des Objektivs soll ebenfalls frei wählbar sein. Daraus ergeben sich die folgenden Effekte für eine Kamera:

- telezentrisches, perspektivisches Objektiv
- Vignettierung
- Verzeichnung
- Schärfentiefe
- Alias-Effekte
- Begrenzte Auflösung
- Geisterfleck
- Rauschen
- Bewegungsunschärfe

Im Folgenden werden die einzelnen Kameraeigenschaften genauer betrachtet.

telezentrisches, perspektivisches Objektiv

Die Wahl des Objektivs ist entscheidend für das spätere Ausgabebild. Der gravierendste Unterschied zwischen perspektivischem und telezentrischem Objektiv ist der sogenannte Abbildungsmaßstab des abgebildeten Objektes. Ebenfalls wird die Position des Abgebildeten Objektes durch die Objektivart beeinflusst. Ein Beispiel zeigt hier Abbildung 2.1.

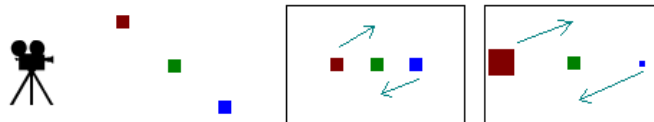


Abbildung 2.1: Gegenüberstellung telezentrische / perspektivische Linse, v.l.n.r.: Aufbau, telezentrisch, perspektivisch

Links im Bild ist der schematische Aufbau der Szene zu sehen. Das rote Objekt ist der Kamera am nächsten und das blaue am weitesten entfernt. In der Mitte sieht man die idealisierte Aufnahme durch ein telezentrisches Objektiv und rechts die idealisierte Aufnahme durch eine perspektivisches Objektiv. Idealisiert deswegen, da der Effekt der Schärfentiefe ebenfalls eine Rolle spielt, sich aber bei beiden Objektivtypen gleich auswirkt und deswegen später gesondert betrachtet wird. Im Vergleich der beiden Aufnahmen, ist deutlich der Größenunterschied der einzelnen Objekte zu erkennen. Auch ist der Abstand der Objekte im perspektivischen Bild größer. Beide Effekte gründen auf dem Abstand des Objektes zur Kamera. In telezentrischen Objektiven wird dieser Effekt fast vollständig unterdrückt. Bei telezentrischen Aufnahmen muss die Größe des Objektivs an die Größe der aufzunehmenden Objekte angepasst werden. Das Objektiv muss mindestens so groß sein wie die aufgenommenen Objekte.

Vignettierung

Der Effekt der Vignettierung, auch als Randlichtabfall bezeichnet, tritt grundsätzlich bei allen Kamerasystemen auf. Jedoch ist die Vignettierung bei Weitwinkelobjektiven und bei sehr großen Blenden stärker ausgeprägt als bei kleinen Blenden oder Teleobjektiven. Von Vignettierung spricht man, wenn auf dem aufgenommenen Bild eine Abdunklung des gesamten Bildes zu den Rändern hin ersichtlich ist [28]. Die Ursache der Vignettierung

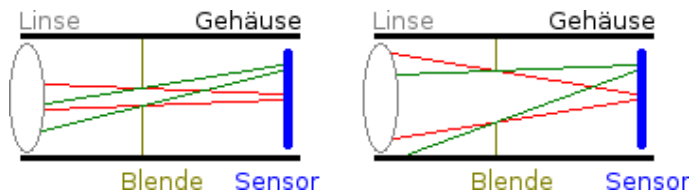


Abbildung 2.2: Sichtfelder von Pixeln bei verschiedenen Blenden

liegt im Zusammenspiel der Linsen im Objektiv und der Länge des Objektivs, sowie der Größe der Blende. Je größer die Blende oder je länger das Objektiv ist, desto mehr kann der aufnehmende Sensor vom Rand und über diesen hinaus sehen. Abbildung 2.2 veranschaulicht dies. Auf der linken Seite die schematische Darstellung bei einer kleinen Blende, das Sichtfeld des Sensors ist für beide Position frei von Vignettierung. Auf der rechten Seite die Darstellung bei einer weiter geöffneten Blende. Das Sichtfeld der mittleren Position (rot) ist immer noch frei von Vignettierung. Das Sichtfeld an der äußeren Position (grün) ist von der Vignettierung betroffen. Durch das Öffnen der Blende wird der Effekt der Vignettierung verstärkt, bei gleichzeitiger Aufhellung des gesamten Bildes. Auf Abbildung 2.3 ist links eine Aufnahme eines gleichmäßigen Graubildes ohne Vignettierung und rechts eine Aufnahme des selben Bildes mit Vignettierung zu sehen.



Abbildung 2.3: Veranschaulichung des Vignettierungseffektes

Die einzigen beiden Korrekturmöglichkeiten gegen den Vignettierungseffekt sind eine sehr kleine Blende oder eine Art künstlicher Abdunklungsmaske zu verwenden. Eine kleine Blende wirkt sich ebenfalls positiv auf die Schärfentiefe aus, schränkt aber den gesamt abgebildeten Bildbereich ein. Eine künstlicher Abdunklungsmaske vor dem Objektiv dunkelt nur im mittleren Bereich des Objektivs das Licht ein klein wenig ab, um den Vignettierungseffekt entsprechend auszugleichen. Zu berücksichtigen ist allerdings, dass eine solche Korrekturmaske für jede verwendete Blendengröße angepasst werden müsste. Die Abdunklungsmaske hat auch noch den weiteren Nachteil, dass sie sich auf die Gesamthelligkeit des Bildes auswirkt und somit die Helligkeitsinformationen verfälscht.

Verzerrung, Verzeichnung

Die optische Verzerrung oder auch Verzeichnung genannt ist ein weiterer zu berücksichtigender Fehler in Kamerasystemen. Von Verzerrung spricht man, wenn das resultierende Bild zu den Rändern hin verzerrt ist. Je nach Art der Verzerrung spricht man von einer kissenförmigen oder einer tonnenförmigen Verzerrung. Ein verzerrungsfreies Linsensystem wird orthoskopisches Linsensystem genannt [18]. Bei Linsensystemen mit mehreren Linsen kann durch Verwendung verschieden gekrümmter Linsen, und spezieller Glassorten die Verzerrung bis zu einem gewissen Grad kompensiert werden. Die Restverzerrung kann mittels folgendem Testverfahren bestimmt werden: Um die Verzerrung eines bestehenden Kamerasystems am besten messen zu können benötigt man ein Quellbild, welches zum Beispiel aus einem Rechteckmuster besteht. Hierbei ist es unbedeutend, ob dieses Muster aus Linien, Punkten oder kleinen Kreuzen besteht. Die einzige Voraussetzung muss die pixelgenaue Bestimmung der Position der abgebildeten Objekte relativ zueinander sein. Zur Bestimmung der Verzerrung werden die Positionen der Muster auf dem

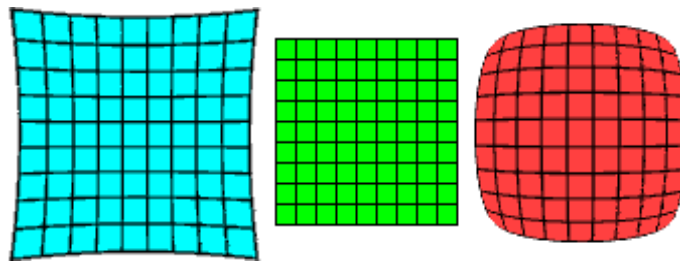


Abbildung 2.4: Beispiele Verzerrungsformen: Kissenförmige, keine, tonnenförmige Verzerrung

Bild mit den Positionen verglichen, auf denen sie eigentlich liegen sollten. Meist werden diese Verzerrungseffekte in ein entsprechendes Vektorfeld übertragen, oder als spezifisches Polynom dargestellt [28]. Abbildung 2.4 zeigt die zwei wesentlich vorkommenden Verzerrungstypen kissenförmige und tonnenförmige Verzerrung. Diese Effekte können sich aber auch überlagern und dann wellenförmig verlaufende Verzerrungen ermöglichen. Hat man die verzerrungsspezifischen Daten des verwendeten Kamerasystems erst einmal bestimmt, so ist es im Nachhinein möglich diese umzukehren [27] und aus den Bilddaten wieder herauszurechnen. Der Verzerrungseffekt tritt bei Weitwinkelobjektiven als tonnenförmige Verzeichnung am deutlichsten auf.

Schärfentiefe

Der Begriff der Schärfentiefe [11], besser bekannt unter seiner englischen Bezeichnung (depth of field DOF), beschreibt den minimalen beziehungsweise maximalen Abstand vom Kamerasystem, bei welchem alle Objekte die sich hierin befinden scharf auf dem Ergebnisbild abgebildet werden. Die Distanz in welcher sich die Strahlengänge eines Objektes genau auf der Bildebene kreuzen wird als Fokus bezeichnet (siehe Abbildung 2.5: roter und brauner Strahlengang). Befindet sich ein Objekt vor oder hinter der Fokusebene, so liegt der Kreuzungspunkt des Strahlenganges entsprechend Abbildung 2.5 nicht genau auf der Bildebene, sondern davor (blauer Strahlengang) beziehungsweise dahinter (grüner Strahlengang). In beiden Fällen wird jeder Punkt des Objektes als

Unschärfescheibchen auf der Bildebene abgebildet, wodurch eine unscharfe Abbildung entsteht. Diese Unschärfe wird im digitalen Abbild erst detektierbar, sobald die Ent-

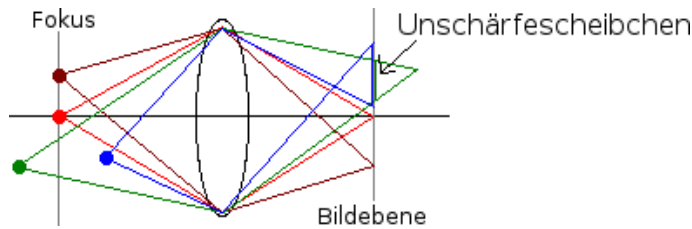


Abbildung 2.5: Objekte innerhalb und außerhalb des Fokus

fernung des Objektes zur Fokus Ebene Unschärfescheibchen erzeugt, die mehr als ein Pixel des Bildsensors überdecken. Kleinere Unschärfescheibchen von der Größe eines Pixels und darunter, hervorgerufen durch Objekte in der Nähe der Fokus-Ebene, sind nicht detektierbar und führen zu perfekt scharfen Abbildern.

Jeder Bildsensor besteht aus vielen kleinen lichtsensitiven Elementen, welche bei CCD- oder CMOS-Sensoren als Pixel bezeichnet werden. Ein Pixel entspricht dabei, je nach Typ des Sensors, einem einzigen Helligkeitssensor, dies ist bei Schwarzweißsensoren der Fall. Oder aus mehreren, dies ist bei Farbsensoren so. Da das menschliche Auge aber unterschiedlich empfindlich auf die einzelnen Grundfarben (Rot, Grün, Blau) reagiert [23], muss dies bei den Farbsensoren ebenfalls berücksichtigt werden. Ein lichtsensitives Element wird mit Hilfe von Farbfiltern auf eine bestimmte Grundfarbe hin orientiert. Die Anordnung der Farbfilter und die Menge der einzelnen Grundfarben soll dabei der Empfindlichkeit des menschlichen Auges angepasst werden. Die Verwendung sogenannter „Bayer-Matrizen“ hat sich auf diesem Bereich durchgesetzt. Bei den Bayer-Matrizen ist die Anordnung im Gegensatz zu „Foveon X3 Direkt-Bildsensoren“ nebeneinander und nicht übereinander. Bei den Bayer-Matrizen werden immer vier lichtensitive Elemente zu einem Pixel verschaltet. Von diesen vier Elementen messen zwei den Grünanteil und jeweils eines den Rot- bzw. Blauanteil. Aus diesen Rohdaten werden dann die Ergebnisbilder zusammengesetzt [21].

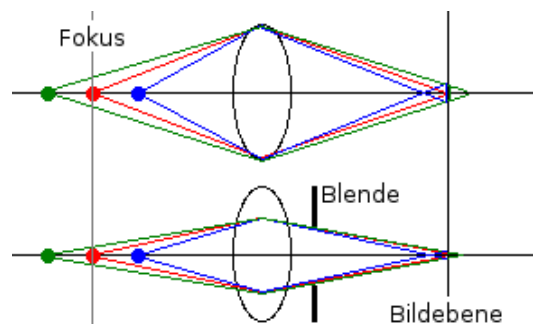


Abbildung 2.6: Verbesserung des DOF durch kleinere Blende

Die Größe der Blende hat einen sehr großen Einfluss auf den Schärfebereich. Die Größe der Blende wirkt sich indirekt auf die Größe der Linse aus und verringert oder erweitert den nutzbaren Bereich der Linse. Auf Abbildung 2.6 ist eine Linse ohne Blende oben und mit Blende unten zu sehen. In der Darstellung mit Blende kann man erkennen, dass sich die Unschärfescheibchen der abgebildeten Objekte durch den Einfluss der Blende verringert haben. Deren Unschärfe auf dem Abbild ist demnach geringer. Der Bereich der Schärfentiefe wird bei immer kleineren Blenden größer. Eine kleinere Blende lässt jedoch auch weniger Licht auf den aufnehmenden Sensor, die Gesamthelligkeit des Bildes wird dadurch also abgeschwächt.

Alias-Effekt

Der Alias-Effekt ist ein Begriff aus der Signaltheorie. Er tritt auf, wenn die Bedingungen des Shannon Abtasttheorems verletzt werden. Das Shannon Abtasttheorem besagt hierbei, dass ein analytisches Signal bei der Digitalisierung mit der doppelten maximalen im Signal vorkommenden Frequenz abgetastet werden muss. Die maximale im Signal vorkommende Frequenz wird als Nyquist-Frequenz bezeichnet. Geschieht dies nicht, überlagern sich die Frequenzspektren der einzelnen Frequenzen und das ursprüngliche Signal kann nicht mehr korrekt rekonstruiert werden. Es treten unerwünschte Interferenz-Erscheinungen auf. Dieser Effekt ist durch die Betrachtung im Frequenzraum am einfachsten zu erkennen. Die Frequenzraumdarstellung ist eine andere Darstellung einer Funktion. Anstatt der Zeit werden die Frequenz und deren Betrag eines Signales dargestellt. Mit Hilfe der Fourier-Transformation [6] kann man eine Funktion vom Ortsraum in den Frequenzraum überführen. Die Fourier-Transformation geht grundsätzlich von

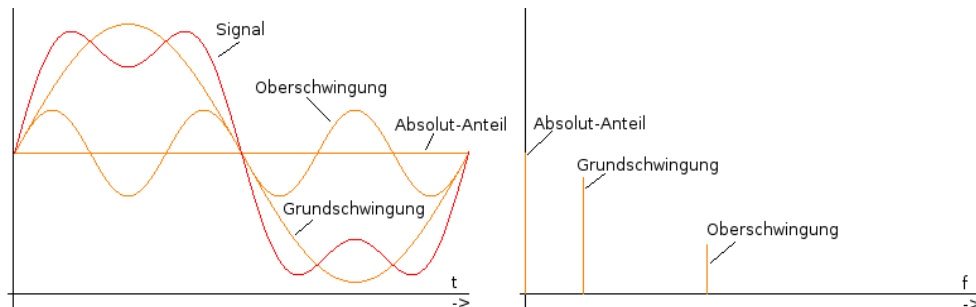


Abbildung 2.7: Darstellung der Frequenzerlegung eines Signales

periodischen Signalen aus. Ein Bild ist prinzipiell erst mal kein periodisches Signal. Betrachtet man ein Bild allerdings als die Darstellung exakt einer Periode (nimmt also an, dass sich das Bild nach allen Richtungen unendlich oft wiederholt), erhält man ein, wenn auch nur theoretisch, periodisches Signal. Dieses lässt sich dann mit der Fourier-Transformation bearbeiten. Abbildung 2.7 zeigt links ein eindimensionales Signal (rot) bestehend aus 3 Frequenzen (orange), und rechts die Darstellung des Frequenzspektrums.

Ein diskretisiertes Signal entspricht einer analytischen Funktion multipliziert mit einer Kamm-Funktion, welche die Abtastrate vorgibt. Abbildung 2.8 zeigt die Diskretisierung eines eindimensionalen Signales. Bilder sind zweidimensional. Durch geschicktes Umformen lassen sich mehrdimensionale Signale auf die eindimensionale Fourier-



Abbildung 2.8: Diskretisierung eines eindimensionalen Signales

Transformation zurückführen. Für den Beweis und die genaue Technik sei hier allerdings auf entsprechend weiterführende Literatur [22] verwiesen. Eine Multiplikation im Urbild entspricht einer mathematischen Faltung der Funktion im Frequenzraum. Der Faltungs-Operator wirkt sich hierbei derart aus, dass die Frequenzspektren der ursprünglichen Funktion an jedem Peak der transformierten Kamm-Funktion wiederholt werden (siehe Abbildung 2.9). Dies hätte eine Verfälschung des Frequenzspektrums



Abbildung 2.9: Faltung zweier Signale

zur Folge. Zur Rekonstruktion des Signales wird die Funktion des Frequenzspektrums mit einer Box-Funktion multipliziert und mit der so erhaltenen Funktion die inverse Fourier-Transformation durchgeführt. In Abbildung 2.10 ist die Multiplikation mit der

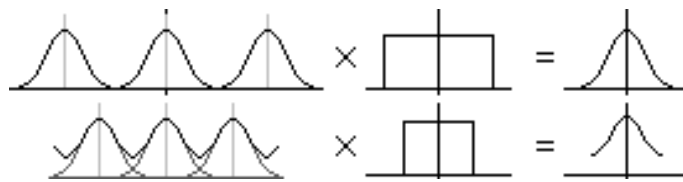


Abbildung 2.10: Missachtung des Abtasttheorems, Extraktion des Rücktransformationsspektrums

Box-Funktion für den Fall der Einhaltung des Abtasttheorems (oben) und für den Fall des nicht einhalten des Abtasttheorems (unten) gezeigt. Abbildung 2.11 zeigt die jeweiligen Rekonstruktionen. Die durch die Überlagerung der Frequenzspektren entstehenden zusätzlichen Oberschwingungen, auch als Rauschen bezeichnet, sind nun klar im rekonstruierten Signal erkennbar.

Dieses kleine Beispiel zeigt, dass die Abtastrate einer Szene möglichst hoch sein sollte, denn nur dann sind die Peaks der Kamm-Funktion auch weit genug auseinander, um eine Überlagerung der Spektralkurve des Eingangesignales zu verhindern. Die Breite des aufgenommenen Bildes dividiert durch die Anzahl der Pixel eines Sensors pro Zeile (auch als Linienpaare pro Millimeter bezeichnet), legt den Abstand der Peaks der Kamm-Funktion fest. Der einzige daraus folgende Schluss muss also eine Erhöhung der Pixelanzahl des CCD- oder CMOS-Sensors sein. Die maximale Anzahl an Pixeln pro Sensor wird durch viele Faktoren beschränkt. Diese sind zum Beispiel die minimale Größe eines Helligkeitssensors, oder durch Fertigungsprozesse bedingte Einschränkungen. Daraus ergibt sich, dass die Nyquist-Frequenz des bildaufnehmenden Systems nicht beliebig in die Höhe geschraubt werden kann. Der Alias-Effekt wird also auf jeden Fall auftreten.

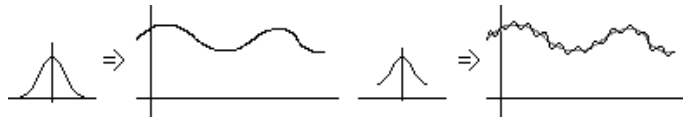


Abbildung 2.11: Rekonstruktion des Frequenzspektrums

Abbildung 2.12 zeigt links eine Fresnel-Zonenplatte im Original und rechts die Abbildung der Zonenplatte durch ein System, welches nicht in der Lage war, die im Original auftretenden maximalen Ortsfrequenzen aufzulösen. Die so entstehenden eigentlich nicht existierenden Muster werden Moiré-Muster [5] genannt.

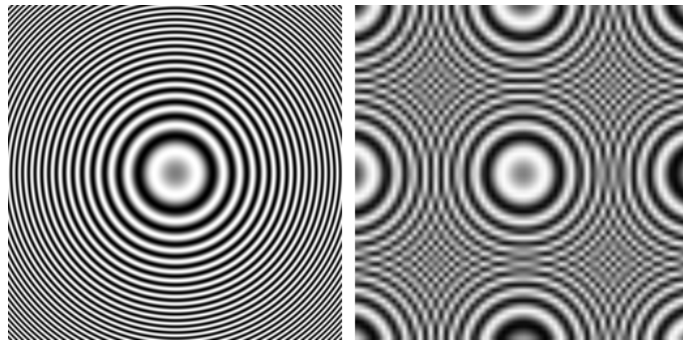


Abbildung 2.12: Links Original, Rechts Moiré-Effekt aufgrund einer Unterabtastung

Begrenzte Auflösung

Der Effekt der begrenzten Auflösung entsteht ebenfalls durch eine Unterabtastung des Quellbildes. Hohe Bildfrequenzen entstehen nicht nur durch Muster auf den abgebildeten Objekten, welche sich in der Szene befinden. Sie können auch durch die Objekte der Szene selbst entstehen. Wenn sich zum Beispiel die Flugbahnen zweier Partikel immer näher kommen um sich zu kreuzen, bildet der Abstand zwischen diesen beiden eine stetig ansteigende Ortsfrequenz. Wie beim Alias-Effekt bereits erläutert, begrenzt der Bildsensor durch seine Bauweise die maximale Ortsfrequenz eines Punktes im Bild. Trifft ein sehr hochfrequentes Signal auf ein Pixel des Sensors, so hat es mehrere Helligkeitsschwankungen im Aufnahmebereich des Pixel. Das Pixel integriert die gesamte Helligkeit des ankommenden Lichtes auf der Pixelfläche über die Belichtungszeit und gibt dessen Mittelwert aus. Abbildung 2.13 veranschaulicht dies. Auf der linken Seite ist ein Pixel



Abbildung 2.13: Schematische Darstellung zweier Helligkeitssensoren bei der Ermittlung ihrer Ausgabewerte

zu sehen, welcher eine Fläche gleicher Helligkeit aufnimmt. Rechts wird das hochfrequente Bild zu einem Helligkeitswert integriert. Das erzeugte Bild wirkt verwaschen.



Abbildung 2.14: Originalbild einer exponentiell anwachsenden Ortsfrequenz

Dieser Effekt tritt aufgrund der Bayer-Matrizen bei Farbsensoren noch deutlicher auf. Durch diesen Effekt wird die Differenz der Helligkeitswerte zweier benachbarter Pixel minimiert. Die Differenz der Helligkeit wird als Kontrast bezeichnet. Ein Beispiel mit welchem sich dieser Effekt sehr gut darstellen lässt, ist die Abbildung einer ansteigenden Ortsfrequenz. Abbildung 2.14 zeigt ein binäres Signal, dessen Ortsfrequenz von links nach rechts exponentiell ansteigt. Der Kontrast zwischen den Hell-Dunkel-Bereichen ist Maximal. Abbildung 2.15 zeigt die Aufnahme durch den Bildsensor. Der Verwaschungs-



Abbildung 2.15: Ausgabebild des Bildsensors einer exponentiell anwachsenden Ortsfrequenz

effekt im rechten Bildbereich deutlich erkennbar. Aber auch an den Übergängen der großflächigen Bereichen hat eine Verwaschung statt gefunden. Diese Verwaschung begründet sich darin, dass die Kante zwischen den Bereichen nur scharf abgebildet würde, wenn sie exakt auf eine Pixelreihe abgebildet wird, und auf der nächsten nicht. Doch selbst dann würde sie durch den oben beschriebenen Beugungseffekt an der Blende trotzdem noch leicht verwaschen wirken, da jede Kante hochfrequente Signalanteile aufweist, welche wiederum an der Blende gestreut würden. Der Abfall des Kontrastes, kann durch die Betrachtung der Helligkeitswerte deutlich ersehen werden. Abbildung 2.16 zeigt die

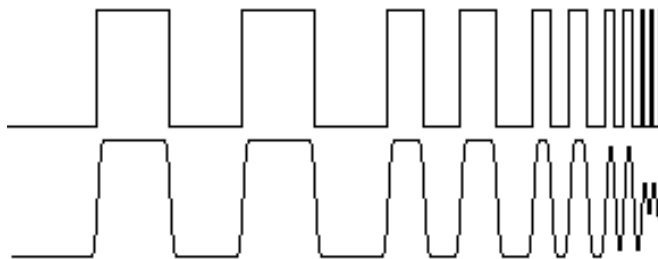


Abbildung 2.16: Darstellung der Amplitude der Farbwerte des Originalbildes und des ermittelten Bildes

Amplituden beider Graphiken im Querschnitt. Bedingt durch die Auflösung des Bildsensors ist der Kontrast an den Kanten gesunken. Bei immer höheren Ortsfrequenzen geht er gegen 0. Bedingt durch den Abbildungsmaßstab des Objektivs und der Pixelanzahl

des Sensors existiert immer eine Ortsfrequenz, bei welcher der Kontrast auf 0 abfällt. Die Darstellung des Abfalls des Kontrastes aller Ortsfrequenzen entspricht der MTF [12]. Allen MTF-Kurven gemein ist, monoton fallend zu sein.

Mit Hilfe der Eigenschaft des Kontrastabfalls kann die maximale Auflösung eines Kamerasystems bestimmt werden. Dazu wurde von der Firma Siemens die sogenannte Siemensreflexprüfmethode entwickelt. Bei dieser Methode wird ein Siemensstern (siehe Abbildung 2.17) aufgenommen. Ein Siemensstern ist eine Graphik, bei der die Ortsfrequenz

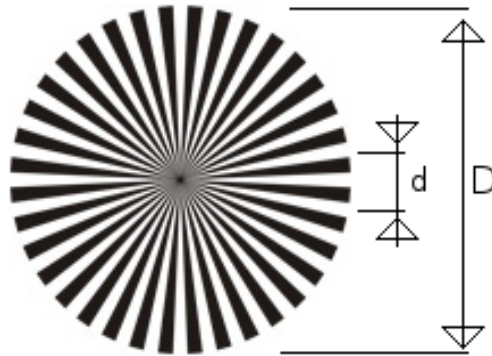


Abbildung 2.17: Siemensstern mit Bemaßung

zum Mittelpunkt des Bildes kontinuierlich ansteigt. Anhand der Ausgabe durch das Kamerasystem, kann dann die Maximale Auflösung für die aktuelle Aufnahme bestimmt werden. Bei perspektivischen Objektiven ist diese Auflösung vom Abstand des Siemenssterns zum Objektiv abhängig. Zur Bestimmung der maximalen Auflösung werden die beiden Größen D und d in Pixeln bestimmt. Der Abstand d entspricht dem Durchmesser des grauen Bereiches in der Bildmitte. Alle Ortsfrequenzen innerhalb dieses Bereiches konnten durch das verwendete Kamerasystem nicht mehr abgebildet werden. Im Fall von Abbildung 2.17 entspricht dieses Verhältnis dem Skalierungsfaktor $r = \frac{d}{D} = 0,16$. Mit Hilfe der Formel :

$$l = \frac{\pi * r * D_0}{n} \quad (2.1)$$

lässt sich die maximale Auflösung l des Kamerasystems berechnen. Hierbei entspricht n der Anzahl der Hell-Dunkel-Bereiche, im Beispiel 32 und D_0 der realen Größe des Siemenssterns in Metern. Wenn zum Beispiel der hier gezeigte Siemensstern im Original 1 cm groß war. So würde sich für das hier verwendete Kamerasystem eine maximale Auflösung von

$$l = \frac{\pi * r * D_0}{n} = \frac{\pi * 0,16 * 0.01[m]}{32} \approx 0,000157[m] = 157[\mu m] \quad (2.2)$$

pro Pixel ergeben. Ein Partikel, welches also $157\mu\text{Meter}$ groß wäre, würde genau ein Pixel im Ergebnisbild überdecken. Kleinere Partikel würden mit dem Helligkeitswert der Umgebung gemittelt. Da diese in der Regel schwarz also gleich 0 ist, bedeutet dies, dass Partikel mit einer Größe von weniger als $157\mu\text{Meter}$ in ihrer Helligkeit abgedunkelt dargestellt würden. Entsprechend würden Partikel welche größer als $157\mu\text{Meter}$ wären, mehrere Pixel überdecken und zum Rand hin durch das ebenfalls auftretende Mischen mit der Umgebung abgedunkelt.

Punktspreizfunktion

Die Punktspreizfunktion [13] gibt die Abbildung einer Punktlichtquelle (auch Dirac-Funktion oder Stoßfunktion genannt) durch das abbildende System an. Bei diesem Vorgang spielen Effekte wie Beugungerscheinungen an sehr kleinen Blenden [14], Abbildungsfehler und der Einfluss der Sensorfläche bzw. der Apertur eine Rolle. Das Ergebnis der Punktspreizfunktion wird in der Optik als Beugungsscheibchen oder auch Airy-Scheibchen (nach dem englischen Astronomen George Biddel Airy) genannt. Die Beugungsscheibchen entstehen dabei durch den Beugungseffekt am Rand der Blende oder der Linse. Die Größe des Beugungsscheibchens ist umgekehrt proportional zum kleinsten Blendendurchmesser im verwendeten System. Dabei ist es unerheblich ob es sich um die Randbegrenzung der Linse oder die Größe der Blende handelt. Die Form dieses kleinsten Blendendurchmessers bestimmt das Aussehen der Beugungsscheibchen. Abbildung 2.18 zeigt das entstehende Beugungsscheibchen (großes Bild) eines Lichtstrahles an verschiedenen Blendenformen (kleines Bild). Eine sehr kurze Verschlusszeit führt zu

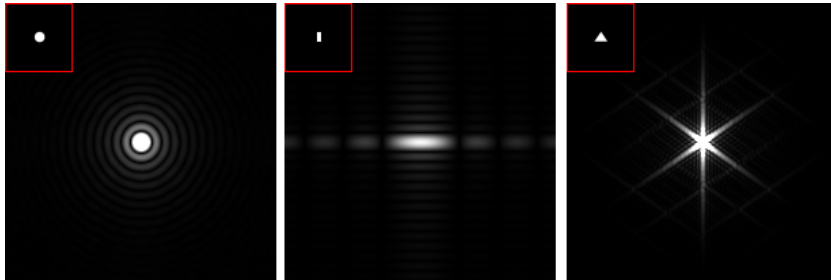


Abbildung 2.18: Beugungsscheibchen an verschiedenen Blendenformen

punktartigen Abbildungen der Partikel, folglich sind alle in der Szene abgebildeten Partikel von diesem Effekt betroffen. Abbildung 2.19 zeigt die Aufnahme einer Szene unter Berücksichtigung der Punktspreizfunktion.

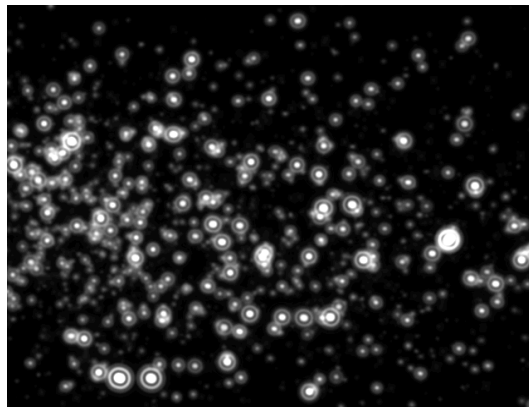


Abbildung 2.19: Szene mit Berücksichtigung der Punktspreizfunktion

Geisterfleck

Geisterflecken wird ein Effekt genannt, der auftritt, wenn Unreinheiten auf der Linse oder kleine Objekte extrem nah an der Linse auftauchen. Der Effekt kommt meistens beim Einsatz von Blitzlichtern zum Einsatz. Hierbei wird das Blitzlicht an den kleinen Teilchen - meist Staub - gestreut und teilweise zurückgeworfen. Die Form der Geisterflecken entspricht hierbei meist der Form der verwendeten Blende.

Rauschen

Als Rauschen werden alle Arten von im Bild auftretenden Fehlern bezeichnet, die nicht weiter durch einen anderen Effekt erklärbar sind. Rauschen kann sich sowohl auf einzelne Pixel auswirken, bei Farbbildern sogar auf nur einen einzigen Farbkanal, als auch auf ganze Bildbereiche. Rauscheffekte, welche einzelne Pixel betreffen, entstehen meist durch Übertragungsfehler zum Beispiel hervorgerufen durch fehlende Abschirmung des Übertragungskanal. Großflächiges Rauschen kann durch Überhitzung von Teilbereichen des Bildsensors, aber auch durch direkte Sonneneinstrahlung oder andere großflächige Störimpulse (etwa Magnetfelder) entstehen. Das großflächige Rauschen ist meistens erklärbar und lässt sich beheben, während das pixelweise Rauschen meist vollkommen zufällig ist.

Bewegungsunschärfe, Motion Blur

Bewegungsunschärfe tritt auf, wenn die Abbildung eines Objektes auf dem Bildsensor während der Belichtungszeit über mehrere Pixel hinweg bewegt wird. Die Anzahl der überschrittenen Pixel ist dabei das Maß der Unschärfe. Abbildung 2.20 zeigt eine solche Situation im Detail. Das Objekt (rot) bewegt sich während der Aufnahme und überstreicht dabei mehrere Pixel (grün) bis zum Ende der Aufnahme. Der Bereich, in

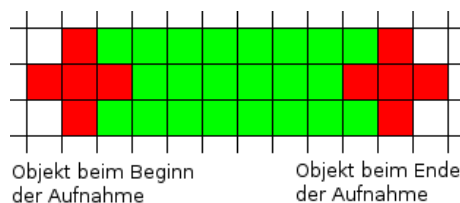


Abbildung 2.20: Entstehung von Bewegungsunschärfe

dem keine Bewegungsunschärfe auftritt, ist hierbei sehr klein. Zur Verhinderung der Bewegungsunschärfe muss die Belichtungszeit so klein gewählt werden, dass das abgebildete Objekt während dieser Zeit eine Bewegung kleiner der Breite eines Pixel macht. Die Belichtungszeit kann hierbei allerdings nicht beliebig kurz gemacht werden. Die unterste Schranke für die Belichtungszeit ergibt sich bei analogen Kameras durch die Öffnungs- und Schließdauer des Shutters und bei digitalen Kameras durch Signallaufzeiten der Steuerungselektronik. Je kürzer die Belichtungszeit, desto lichtschwächer ist die Abbildung auf dem Ausgabebild. Daraus folgert die Notwendigkeit einer möglichst langen Belichtung. Dem gegenüber steht die sehr hohe Geschwindigkeit der abgebildeten Objekte. Je länger die Belichtungszeit und je schneller sich ein Objekt bewegt, desto länger wird der durch die Bewegungsunschärfe entstehende Verwischungseffekt. Abbildung 2.21 zeigt



Abbildung 2.21: Szene mit geblurten Objekten

eine Szene mit verschiedenen unterschiedlich schnellen Objekten. Wenn die verwendete Belichtungszeit bekannt ist, kann man sich die Bewegungsunschärfe allerdings auch zunutze machen. Im rückwärts betrachteten Fall gibt sie an, in welcher Zeit das Objekt wie weit gekommen ist. So lassen sich aus den durch Bewegungsunschärfe entstandenen Verschmierungen die Geschwindigkeiten einzelner Objekte annäherungsweise bestimmen. Mit den in der Aufgabenstellung erwähnten nachfolgenden Auswertelgorithmen soll unter anderem genau diese Geschwindigkeit ermittelt werden.

2.2.3 Hohe Genauigkeit

Die hohe Genauigkeit für die Simulation und deren Ausgabebilder wird durch eine interne 32-Bit-Darstellung erreicht. Durch diese Darstellung werden Rundungsfehler so gut es geht verhindert. Ebenfalls sollen alle Kameraeffekte aus 2.2.2 in dieser Genauigkeit nachgebildet werden. Dies beinhaltet auch die getrennte Verarbeitung der einzelnen Farbkanäle. Die 32-Bit-Darstellung ist genauer als die typischerweise 10- bis 14-Bit-Ausgabedaten der Kameras und kann somit verlustfrei auf dieses Format abgebildet werden. Wie Abbildung 2.22 zeigt ist Dank der viel feineren Auflösung eine exaktere Abtastung eines z. B. Graukeils möglich. Ebenfalls können dadurch Skalierungen in andere Zahlenbereiche (sogenanntes Windowing) mit deutlich weniger Artefakten vorgenommen werden. Für eine Darstellung in 32-Bit pro Farbkanal sind mehrere Bildformate möglich. Die Bildformate JPG, OpenEXR und TGA unterstützen alle eine 32-Bit Darstellung pro Farbkanal. Eine genaue Analyse hat jedoch ergeben, dass die Nutzung eines dieser Bildformate zwar möglich, jedoch nicht notwendig ist. Dies gründet in der nachfolgenden Analysestufe. Hier sind lediglich die Daten von Relevanz, nicht aber ihre Repräsentation als Bilder. Aus diesem Grund wird ein eigenes Dateiformat verwendet. Dieses Dateiformat soll lediglich die reinen Daten verlustlos abspeichern. Um dem Nutzer jedoch die Möglichkeit der schnellen Ansicht der Ausgabedaten zu ermöglichen, soll eine Ausgabe in einem standardisierten Dateiformat gegeben sein. Das PNG-Dateiformat lässt diese Aufgabe unter minimalem Speicherverbrauch und ohne Verluste durch die Komprimierung zu. Eine weitere Ausgabe speichert die Szene als transferierte Rohdaten, so ist es

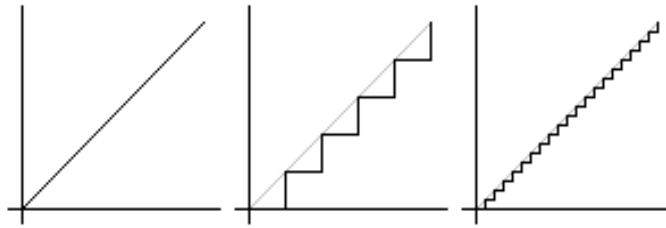


Abbildung 2.22: Unterschiedliche Abtaststrategien bei variierender Auflösung v.l.n.r.: Original, 8-Bit Stufen, 32-Bit Stufen

möglich, jedem Pixel des Ausgabedatensatzes ein oder mehrere Partikel der simulierten Szene zuzuordnen.

2.2.4 Hohe Performance

Eine hohe Performance kann mittels drei sich ergänzenden Ansätzen erreicht werden:

- Effiziente Speicherausnutzung
- Clipping im Objekt wie im Bildraum
- Verhindern von Mehrfachberechnungen

Effiziente Speicherausnutzung

Eine sehr schnelle und gut geeignete Ausnutzung des Speichers ist zu realisieren. Ein einfaches dynamisches Array scheidet hierbei sofort aus, da eine Erweiterung des Arrays um nur ein Element eine sofortige komplette Neuallokierung des gesamten Speichers des Arrays nach sich zieht. Eine Speicherung der „frei“ werdenden Plätze im Array und deren Wiederverwendung behebt dieses Problem nicht, bzw. nur ungenügend. Die Speicherung als Kette scheidet ebenfalls aus, hier müssten zwar jeweils nur die neu benutzten Speicherplätze allokiert werden. Da diese allerdings nicht zusammenhängend im Speicher liegen, erzeugt ein Zugriff durch das Programm ständig sogenannte Seitenfehler. Diese bremsen die Anwendung sehr stark aus, da das Betriebssystem die entsprechenden Speicherbereiche erst aus dem Cache laden muss. Liegt dieser unter Umständen sogar auf der Festplatte, so kann ein Zugriff sogar mehrere hundert CPU-Zyklen dauern.

Die Lösung wird eine Mischung aus dynamischem Array und Liste verwenden. Der Simulator wird jeweils sehr große Blöcke an zusammenhängendem Speicher für sehr viele Partikel im Voraus allokiert, und diesen Platz dann mittels einer Freispeicherliste verwalten. Reicht der Platz nicht aus, so wird ein weiterer großer Speicherblock allokiert. Durch diese Verfahrensweise wird eine deutlich höhere räumliche Lokalität der Daten geschaffen und die Anzahl der Seitenfehler minimiert, bei gleichzeitiger dynamischer Gesamtanzahl der Partikel. Die Verwendung einer Freispeicherliste erspart zusätzlich noch ein aufwendiges Suchen von freien Speicherbereichen und minimiert so die Zeit des Neuanlegens eines Partikels.

Clipping im Objekt wie im Bildraum

Das Clipping soll den gesamten Rendervorgang beschleunigen. Dies wird erreicht zum einen durch eine minimale Lebenszeit der Partikel und zum anderen durch eine gute Strategie zum Nichtrendern eines Partikels:

„Jedes nicht gerenderte Partikel ist ein gutes Partikel“.

Indem der gesamten Szene ein „Lebensraum“ der Partikel zugeteilt wird, können diese beim Verlassen dieses Lebensraumes sofort gelöscht werden. Auf diese Weise ist sichergestellt, dass Partikel, welche zum Beispiel an allen Kameras vorbei geflogen sind und somit also nie wieder gerendert werden können, nicht unnötig lange mitgeschleppt werden. Dies spart nicht nur Renderzeit, sondern auch Speicher. Ein Partikel, das zwar außerhalb des Lebensraumes erzeugt wird, dessen Flugbahn aber durch den Lebensraum hindurchgeht, soll nicht gelöscht werden, beziehungsweise erst beim Verlassen des Lebensraumes.

Weiterhin sollen vor jedem Rendern eines Partikels mittels einer Technik namens „View-FrustumCulling“ [1] alle diejenigen Partikel ausgeschlossen werden, welche für die aktuell verwendete Kamera ohnehin nicht sichtbar sind. Gerade bei sehr vielen Partikeln in sehr großen Szenen verspricht diese Technik eine enorme Leistungssteigerung, da durch die Kamera lediglich ein sehr kleiner Bereich der Szene sichtbar gemacht wird.

Verhindern von Mehrfachberechnungen

Das Verhindern von Mehrfachberechnungen soll die Gesamtanzahl der Operationen minimieren. Alle statischen Effekte der Kamera sind hierfür hervorragend geeignet. So muss zum Beispiel der Vignettierungseffekt für jedes ausgegebene Bild berücksichtigt werden. Der Vignettierungseffekt ist aber auf all diesen Bildern gleich. Eine entsprechende Vorberechnung seines Einflusses vor der eigentlichen Simulation erspart bei jedem erzeugten Bild diese Rechenzeit. Die vorberechneten Abschwächungseffekte können nun ohne zusätzlichen Berechnungsaufwand direkt auf die Ausgabe wirken.

2.2.5 Konzept der Simulation

Um das Konzept der Speicherbarkeit einer Simulation umsetzen zu können, muss der Zustand des Zufallsgenerators ebenfalls berücksichtigt werden. Die mehrfache Simulation einer gespeicherten Szene muss immer wieder zu denselben Ergebnissen führen. Um dies zu bewerkstelligen, muss der Initialwert des Zufallsgenerators mit berücksichtigt werden. Bei jedem Abspeichern einer Szene darf hierbei aber nicht der ursprüngliche Initialwert übernommen, werden. Denn dies bedeutete, dass in der wieder geladenen Szene dieselben Zufallszahlen wie in der Version vor dem Speichern gezogen würden, ein signifikanter Einfluss auf deren Wahrscheinlichkeiten. Als Lösung wird bei jedem Abspeichern einer Szene ein neuer zufälliger Initialwert mit abgespeichert. Dies ermöglicht eine Wiederholbarkeit der Abspeicherung bei gleichzeitiger Berücksichtigung der Unabhängigkeit der zu ziehenden Zufallszahlen. Dieses Verfahren hat zur Folge, dass sich die geladene und wieder fortgesetzte Szene von der im ursprünglichen Zeitstrang simulierten Szene ab dem Zeitpunkt der Speicherung unterscheidet. Dieser Effekt kann auch nicht verhindert werden, da es unmöglich ist den aktuellen Wert des Zufallszahlengenerators zum Speichern der Szene auszulesen. Wäre dies möglich, bedeutete dies das Aus für viele Verschlüsselungstechniken. Die einzige Lösung dieses Problems bestünde in der Implementierung eines eigenen Zufallszahlengenerators [9], dieser ließe sich auslesen und entsprechend speichern. Zusätzlich zum Initialwert des Zufallsgenerators müssen auch noch der aktuelle Zeitpunkt der Simulation sowie alle momentan in der Simulation befindlichen Partikel gespeichert werden.

Für das Konzept der Robustheit soll der Simulator gegen Abstürze des Betriebssystems geschützt werden. Dies soll durch das regelmäßige Speichern der gesamten Szene erreicht

werden. Das Speichern soll derart geschehen, dass immer zwei Sicherungen existieren. So ist gewährleistet, dass wenn die Anwendung ungewollt während des Speicherns einer Szene beendet wird, auf jeden Fall die vorletzte Speicherung vorhanden ist. Diese Speicherung muss auf jeden Fall korrekt sein, da der Simulator sonst nicht bis zum aktuellen Zeitpunkt simulieren hätte können. Mehr als zwei Speicherungen sollen nicht vorgehalten werden, denn die Speicherungen könnten unter Umständen sehr viel Speicherplatz benötigen und sind für weitere Simulationen irrelevant.

2.2.6 Hohe Flexibilität

Der Simulator soll möglichst flexibel sein, dies bedeutet, dass alle Parameter, welche bisher erwähnt wurden, später vom Nutzer in geeigneter Form editierbar sein sollen. Zusätzlich wird eine möglichst übersichtliche Darstellungsform dieser Parameter benötigt. Ebenfalls soll der gesamte Funktionsumfang des Simulators im Nachhinein noch durch den Nutzer erweitert werden können. Um dies zu bewerkstelligen, wird das Konzept der Callbacks eingeführt. Es soll später möglich sein, mittels einer zusätzlichen Library, welche die Callbacks beinhaltet, die Funktionalitäten des Simulators an geeigneten Stellen zu ersetzen/ erweitern.

Um eine kurze Übersicht über die Fülle dieser zu geben, sind hier die sich aus den Abschnitten 2.2.1 bis 2.2.5 ergebenden Einstellungsmöglichkeiten aufgelistet:

Generelle Eigenschaften

- Automatische Speicherung
- Lebensraum der Partikel
- Einstellungen für den Zufallszahlengenerator
- Speicherverwaltungsspezifische Angaben
- Einstellungen für die Library

Einstellungen der Kameras

- Auswahl des Ausgabeformates
 - Bilder
 - Daten
 - Zusatzinformationen
- Objektiv Eigenschaften
 - Art (Telezentrisch / Perspektivisch)
 - Belichtungszeit / Zeit bis zur nächsten Aufnahme
 - Vignettierung
 - Verzeichnung
 - Schärfentiefe
 - Alias-Effekte

- Begrenzte Auflösung
- Rauschen + Rausch Callback

Einstellungen der Partikelkanonen

- Bereich der Erzeugung der Partikel
- Partikeleigenschaften
 - Größe
 - Vorzugsrichtung der Bewegung
 - Geschwindigkeit
 - Aussehen (Form, Helligkeit)
- Entstehungshäufigkeit
- Einstellungen zur Texturerzeugung
- Callbacks (Erzeugung, Textur, Bewegung)

Um trotzdem die Übersicht über all diese Parameter zu behalten und gleichzeitig eine einfache Konfigurierbarkeit zu gewährleisten, soll das XML-Format zur Speicherung der Konfigurationen verwendet werden. Es ermöglicht einen hierarchischen Aufbau aller Konfigurationen, damit eine gute Übersichtsmöglichkeit. Die Konfigurationsdatei ist aufgrund ihrer Ausführung im Textformat mit einfachsten Mitteln editierbar.

3 Simulator-Konzept

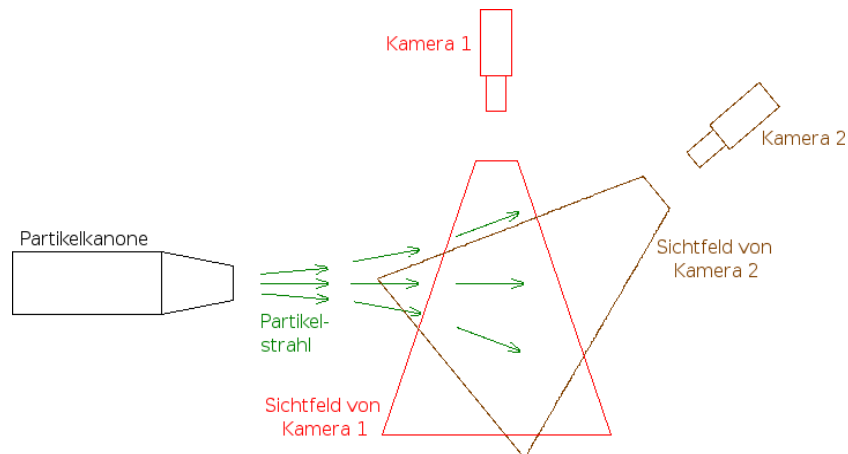


Abbildung 3.1: Skizze zum Aufbau einer Szene

Eine Szene besteht aus drei Elementen. Eine beliebige Anzahl von Partikelkanonen erzeugt in gleichmäßigen Intervallen Partikel. Eine Partikelkanone steht für die grobe Klassifizierung der zu erzeugenden Partikel nach Entstehungsort, Größe, ungefähre Flugrichtung, Geschwindigkeit und Helligkeit. Diese Partikel bewegen sich auf linearen Flugbahnen durch den Raum, und werden wiederum von einem Kamerasystem aufgenommen. Das Kamerasystem hat verschiedene Effekte, für diese gibt es Formeln. Diese Effekte setzen sich zusammen aus der Vignettierung, der Verzerrung, der Schärfentiefe, dem Alias-Effekt, der begrenzten Auflösung, den Geisterflecken, dem Rauschen und der Bewegungsunschärfe. Mit Hilfe von nutzerdefinierten Callbacks lassen sich einige dieser Effekte individuell anpassen und ersetzen. Zu den ersetzbaren Effekten zählen der Alias-Effekt, das Rauschen und die Bewegungsunschärfe.

3.1 Schrittweise Simulation der Zeit

Der Simulator rechnet in diskreten Zeitschritten. Dennoch kann die simulierte Zeit der Belichtung einer Kamera kontinuierlich dargestellt werden. Die berechneten Zeitpunkte springen in unterschiedlich großen Zeitschritten. Die Länge dieser Zeitschritte wird im Wesentlichen von zwei Faktoren beeinflusst.

- Partikel-Erzeugungsintervalle
- Kameraaufnahmen

Weitere Einflüsse sind das Autospeichern und Erzeugen von Partikeln während der Belichtung einer Kamera. Durch das Springen der Zeit können Zeitintervalle, in denen nichts von Interesse passiert, übersprungen werden. Durch dieses Verfahren müssen insgesamt deutlich weniger Iterationen berechnet werden. Am einfachsten kann dieser Prozess anhand eines Zeitstrahls verdeutlicht werden. Abbildung 3.2 zeigt einen solchen Zeitstrahl für eine fiktive Simulation mit einer Partikelkanone und zwei Kameras. Die Partikelka-

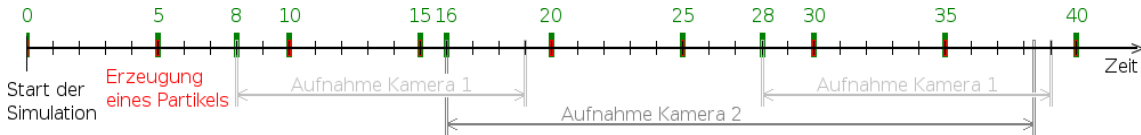


Abbildung 3.2: Beispielhafter Zeitstrahl einer Simulation

none ist so konfiguriert, dass sie alle 5 Zeitschritte ein Partikel erzeugt. Diese Einstellung legt damit die maximale Sprunglänge der Zeit fest. Kamera 1 beginnt die Aufnahme an Zeitschritt 8 und hat eine Belichtungszeit von 11 Zeitschritten. Kamera 1 erzeugt alle 20 Zeitschritte ein neues Bild. Kamera 2 beginnt an Zeitschritt 16 mit der Aufnahme und hat eine Belichtung von 22,4 Zeitschritten. Prinzipiell müssen alle Events zu einem vollen Zeitschritt beginnen. Die Belichtungszeit der Kameras muss als einzige Ausnahme nicht auf einem ganzen Zeitschritt enden. Der Simulator simuliert die in Grün gezeigten Zeitschritte. Je nach Konstellation kann ein Zeitsprung zwischen ein und fünf Zeiteinheiten lang sein.

Bei Erreichen eines jeden simulierten Zeitpunktes werden zuerst alle Partikel bis zum aktuellen Zeitpunkt hin bewegt. Während dieser Bewegung findet das Objekt Raum Clipping statt. Dieses wird mit Hilfe des Konzeptes des "Lebensraums" umgesetzt.

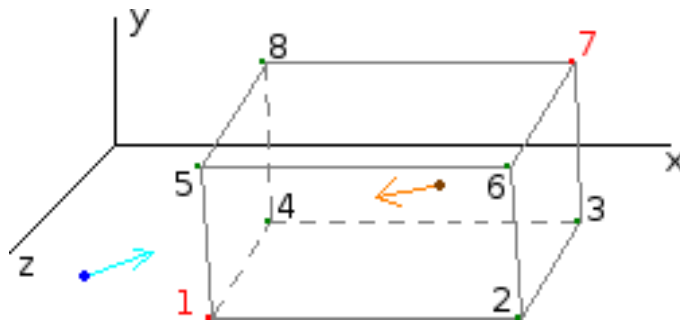


Abbildung 3.3: Darstellung Lebensraum mit Partikeln

Die Partikelkanonen erzeugen Partikel in einem Erzeugungsbereich und durchfliegen den Raum innerhalb des Lebensraums. Beim Verlassen dieses Lebensraumes werden die Partikel wieder gelöscht. Abbildung 3.3 zeigt einen solchen Lebensraum. Der Nutzer gibt hierbei lediglich die beiden Vektoren 1 und 7 (rot) an. Aufgrund der Achsenparallelität des Lebensraumes lassen sich alle restlichen Eckpunkte (grün) durch Kombination der jeweiligen Vektorkomponenten berechnen.

Partikel Erzeugungsintervalle

Die Partikel Erzeugungsintervalle werden jeweils von den einzelnen Partikelkanonen festgelegt. Dabei sind zwei Varianten möglich:

1. Ein Partikel wird nur alle n Zeitschritte erzeugt
2. Bei jedem Zeitschritt werden k Partikel erzeugt

Ist der Zeitpunkt der Erzeugung eines Partikels erreicht, so wird dieses mit all seinen Parametern erzeugt, oder die entsprechende Callback zur Erzeugung des Partikels aufgerufen.

Kameraaufnahmen

Der zweite Zeitpunkt, an welchem die Simulation berechnet wird, ist zugleich der einzige an dem Daten erzeugt werden. Zum Anstoßen des Abspeicherns der Daten gibt es drei Ursachen:

- Autosave alle n Minuten
- Autosave zu einem bestimmten Zeitpunkt
- Rendern einer Kamera

In 2.2.5 wurde eine Robustheit gegen nicht vorhersehbare Störeinflüsse gefordert. Um dieser Forderung gerecht zu werden, wurde der Mechanismus des automatischen Speicherns eingeführt. Das automatische Speichern funktioniert derart, dass der Simulator beim Starten die aktuelle Uhrzeit nimmt. Läuft eine Simulation länger als eine vom Nutzer vorgegebene Zeitspanne (angegeben in Minuten), dann speichert der Simulator automatisch die komplette Szene wie in 2.2.5 beschrieben ab. Eine angegebene Zeitspanne kleiner einer Minute sorgt dafür, dass die Szene nach jedem simulierten Zeitschritt gespeichert wird. Von dieser Einstellung ist allerdings abzuraten, da sie einen immensen Speicher-Overhead erzeugt.

Zusätzlich zum regelmäßigen Speichern der Simulation, kann der Nutzer einen Simulationszeitpunkt (angegeben in Zeitschritten) frei wählen, an welchem ebenfalls die komplette Simulation gespeichert werden soll. Dies ist notwendig, da bei kurzer Laufzeit des Simulators eventuell gar keine automatische Speicherung stattfindet. Durch die explizite Angabe eines Speicherpunktes kann zumindest ein einmaliges Speichern der Szene garantiert werden.

Die letzte Ursache für eine Kameraaufnahme ist das regelmäßige Aufnehmen einer Kamera. Jede Kamera wird zu einem vom Nutzer festgelegtem Zeitpunkt aktiviert. Nach der Aktivierung erzeugt jede Kamera in vom Nutzer definierbaren Intervallen Aufnahmen. Diese Aufnahmen wiederum haben eine ebenfalls vom Nutzer festlegbare Belichtungszeit. Während einer Aufnahme werden alle von der Kamera sichtbaren Partikel aufgezeichnet. Jedes Partikel hat dabei ein durch eine Formel bestimmtes Aussehen. Das Aussehen eines Partikels kann mittels eines Plotters bereits vor der eigentlichen Simulation geprüft und angepasst werden. Der Plotter ist ein weiteres Programm, mit welchem sich die vom Nutzer eingegebenen Formeln darstellen lassen. Die Formeln sind von einer oder zwei Eingangsgrößen abhängig. Jedes Partikel bewegt sich auf einer eigenen Flugbahn, hat eine eigene Geschwindigkeit und Helligkeit, und wird beim Durchqueren des Sichtfeldes einer Kamera von dieser aufgenommen.

3.2 Entwicklung der Partikel-Texturformel

Durch das Zusammenspiel von Belichtungszeit einer Kamera und der Fluggeschwindigkeit des Partikels entsteht der Bewegungsunschärfe-Effekt. Für dessen Umsetzung wurden verschiedene Konzepte getestet:

1. Darstellung durch einen Quader
2. Darstellung durch eine gestreckte Textur
3. Darstellung durch drei Texturen
4. Darstellung mittels partikel-individuell berechneter Textur

1. Darstellung durch einen Quader

Der erste Ansatz war das Rendern eines Partikels durch einen einfachen Quader. Dieser Quader würde durch die später folgende Nachbearbeitung begrenzte Auflösung (siehe 2.2.2) geglättet und bekäme so seine endgültige Form. Abbildung 3.4 zeigt links Partikel

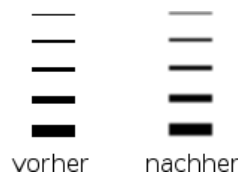


Abbildung 3.4: Partikel durch Quader dargestellt

verschiedener Größe vor dem Glätten, und rechts nach dem Glätten. Man kann erkennen, dass bei kleinen Partikelgrößen der gewünschte Effekt eintritt. Ist das Partikel jedoch größer als der Filterkern, mit welchem das Bild bearbeitet wurde, nehmen die Partikel eine immer eckigere Form an. Um diesem Effekt entgegenzuwirken wurde der Quader durch eine punktförmige Textur ersetzt.

2. Darstellung durch eine gestreckte Textur

Durch die Darstellung des Partikels mittels einer Textur wurde die freie Gestaltung der Form des Partikels ermöglicht. Abbildung 3.5 zeigt links eine solche Textur. Die Helligkeit der einzelnen Pixel gibt die Transparenz der jeweiligen Position an. Zur Darstellung der



Abbildung 3.5: Partikel durch eine Textur dargestellt

Geschwindigkeit wird die Breite der Textur benutzt. Je höher die Geschwindigkeit des Partikels ist, um so breiter wird sein Abbild durch den Bewegungsunschärfe-Effekt 2.2.2

dargestellt. Auf Abbildung 3.5 sind rechts zwei Partikel unterschiedlicher Geschwindigkeit dargestellt. Dabei wird das obere, schnellere Partikel durch das Strecken der Textur extrem spitz. Dasselbe Partikel bei deutlich geringerer Geschwindigkeit unten im Bild wird korrekt dargestellt. Das Spitzwerden der Enden bei steigender Partikelgeschwindigkeit ist jedoch nicht gewünscht.

3. Darstellung durch drei Texturen

Um das Spitzwerden der Partikel bei steigender Geschwindigkeit zu verhindern, wurden nun drei Texturen verwendet. Zwei Texturen für den Rand und eine weitere zur Darstellung der Geschwindigkeit des Partikels. Abbildung 3.6 zeigt links die drei einzelnen Texturen. Rechts ist das Ergebnis zu sehen. Man kann erkennen, dass durch die Be-

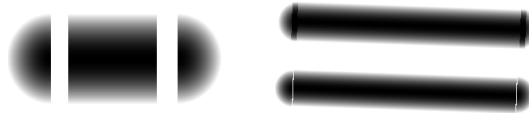


Abbildung 3.6: Partikel durch drei Texturen dargestellt

rechnung der Start- und Endposition der Texturen eine Anfälligkeit auf Rundungsfehler sichtbar wird. Je nachdem ob auf oder abgerundet wird, überlappen die drei Texturen (oben) oder es entsteht ein kleiner Spalt (unten).

4. Darstellung mittels partikel-individuell berechneter Textur

Um das Problem der Rundungsfehler bei der Zusammenstückelung der drei Einzeltexturen lösen zu können, darf nur eine einzige Textur verwendet werden. Das Strecken der Textur in Breite und Höhe muss bei der Erzeugung entsprechend berücksichtigt werden. Der Nutzer hat die Möglichkeit, mittels einer einzigen Formel eine solche Textur zu erzeugen. Die Formel gibt die Transparenz der Textur an ihrer jeweiligen Position an. Zur Berechnung stehen normierte Koordinaten innerhalb der Textur sowie die echte Größe auf dem Abbild und die Unschärfewerte bei Beginn und Ende der Aufnahme zur Verfügung. Der Simulator streckt dann selbstständig den sich für $x = 0$ ergebenden



Abbildung 3.7: Funktionsweise der Partikeltextur-Erzeugungsfunktion

Streifen der Textur auf die gesamte Bildbreite der Textur. Abbildung 3.7 zeigt eine vom Simulator erzeugte Partikeltextur. Auf der Abbildung unten ist der Zahlenbereich der X-Komponente abgebildet. Im gestreckten Bereich verändert sich der Wert für die X-Komponente nicht.

Durch diese Art der Texturerzeugung kann auf alle der oben aufgezählten Problemstellungen Rücksicht genommen werden, bei gleichzeitiger Beibehaltung der Einflussmöglichkeit auf das Aussehen des Partikels. Abbildung 3.8 zeigt eine Szene mit unterschiedlichen Partikelgrößen und Geschwindigkeiten. Durch den Einsatz der flexiblen Berechnung mit



Abbildung 3.8: Partikel durch Texturformel dargestellt

Formeln kann das runde Aussehen der Start- und Endpunkte der Partikel bei großen ebenso wie bei schnellen Partikeln erhalten werden.

3.3 Ablauf einer Simulation

Die Simulation beginnt mit dem Einlesen aller Parameter aus einer zentralen XML-Datei. Nach dem Einlesen dieser Datei wird gegebenenfalls eine abgespeicherte Szene geladen. Anschließend wird die Szene erstellt und bis zum in der XML-Datei definierten Simulationsende berechnet.

Die Berechnung umfasst dabei das in 3.1 beschriebene Fortschreiten der Zeit, sowie das Erzeugen der Partikel und die Berechnung ihrer Flugbahnen. Ebenfalls umfasst die Berechnung die Erzeugung der Kameraaufnahmen mitsamt aller Effekte für jede Kamera. Dabei wird jede Kamera individuell bearbeitet. Zusätzlich findet ein regelmäßiges Speichern der Simulation statt, sowie das Speichern zu einem bestimmten in der XML-Datei festgelegtem Zeitpunkt.

Je nach Parametrisierung erzeugen die Kameras während der Simulation unterschiedliche Daten. Die „*.dat“ Datensätze entsprechen einem direkten Speicherabbild des Renderbuffers und haben daher die größte Genauigkeit. Sie werden ergänzt durch die „*.info“ Dateien, welche die dazugehörigen Partikelinformationen speichern. Zur einfachen Betrachtung der Ausgaben ohne besondere Anwendungen ist zusätzlich noch die Ausgabe eines auf 8-Bit pro Farbkanal reduzierten Datensatzes in Form von „*.png“ Dateien möglich. Auf Abbildung 3.9 ist die Ausgabe einer Szene mit der Darstellung ihrer Zusatzinformationen zu sehen.

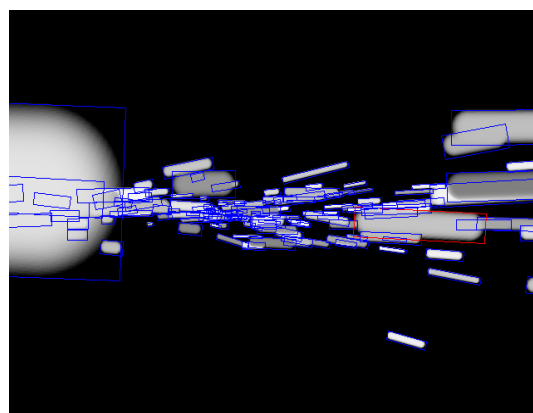


Abbildung 3.9: Darstellung der Szene mit Zusatzinformationen

4 Formeln und Abhängigkeiten

In diesem Kapitel werden alle beteiligten Effekte aus 2.2.2 und Einstellungen aus 2.2.1 nach ihrer Reihenfolge und Umsetzung untersucht.

4.1 Einheiten im Simulator

Um eine möglichst große Flexibilität gewährleisten zu können, benutzt der Simulator keine feste Einheit für die Simulation der Zeit oder der Länge. Die Zeit wird in sogenannten Zeitschritten berechnet. Alle Ereignisse beginnen zu jeweils einem ganzen Zeitschritt. Die Zeitschritte werden Simulatorintern als *Int64* Variable abgespeichert. Jede Simulation beginnt beim Zeitschritt 0. Wird für die Belichtung zum Beispiel ein Femtosekundenlaser verwendet [17], so muss von einem Zeitschritt gleich einer Femto (10^{-15}) Sekunde ausgegangen werden, es ergibt sich eine maximale Simulationszeit von ca. 9223 Sekunden oder ca. 2,5 Stunden.

Dasselbe Prinzip wird für die Längeneinheit angewandt. Alle Positions- und Geschwindigkeitsangaben im Simulator werden jeweils nur in Längeneinheiten oder in $\frac{\text{Laengeneinheiten}}{\text{Zeitschritt}}$ angegeben, damit bleibt dem Nutzer die Freiheit die echte Länge für eine Längeneinheit selbst fest zu legen.

Im Folgenden werden alle Formeln des Simulators mit ihren Parametern erläutert, beginnend mit den zur Partikelerzeugung notwendigen Formeln bis zu den für das Rendern und die Kameraausgabe benötigten Formeln.

4.2 Formeln der Partikelerzeugung

Für die Erzeugung eines Partikels und seiner Parameter sind mehrere Rechenschritte notwendig. Abbildung 4.1 veranschaulicht die Abhängigkeiten (rot) und Berechnungsreihenfolge (blau) sowie die zur Berechnung verwendeten Formeln (grün) und ihre Para-

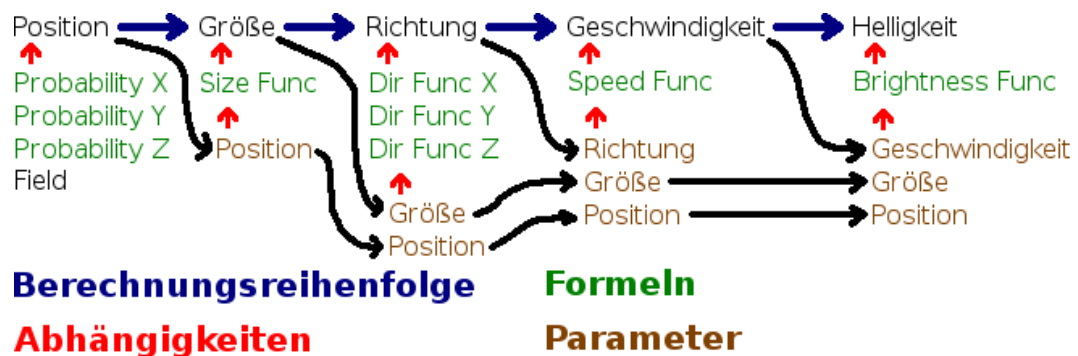


Abbildung 4.1: Abhängigkeiten der Partikelparameter

meter (braun). Zuerst wird die Position eines Partikels berechnet, dies geschieht unter Zuhilfenahme der drei Probability-Funktionen (siehe Tabelle 4.1). Das Ergebnis einer jeden Probability-Funktion ist ein Skalar im Intervall $[-1, 1]$. Dieses Intervall wird auf das sich ergebende Intervall der jeweiligen Komponente aus dem Bereich der Partikelerzeugung (Parameter Field aus 2.2.1) der zugehörigen Kanone abgebildet. Der Bereich der Partikelerzeugung wird für jede Kanone als achsenparallelen Quader angegeben. Nach der Berechnung der Position wird unter Zuhilfenahme der normierten Position die Größe des Partikels mittels der Größenfunktion (Size Func) berechnet. Anschließend wird durch die drei Richtungsfunktionen (Dir Func) die Flugrichtung des Partikels in Abhängigkeit der Größe und der Position bestimmt. Diese Richtung wird normiert und mit der danach berechneten Geschwindigkeit (Speed Func) multipliziert. Eine negative Geschwindigkeit kehrt ebenfalls die Flugrichtung um. Zuletzt wird die Helligkeit (Brightness Func) des Partikels bestimmt. Auch für die Berechnung der Helligkeit stehen alle bisher ermittelten Größen zur Verfügung. Alle übergebenen Parameter sind optional nutzbar.

In Tabelle 4.1 werden alle zur Partikelerzeugung verwendeten Funktionen und deren Parameter aufgelistet. Es können auch farbige Partikel erzeugt werden. Dies ist allerdings

Tabelle 4.1: Formeln der Partikel Erzeugung

Name	Parameter	Bedeutung
Probability X		
Probability Y	Keine	Festlegung der Position im Erzeugungsbereich der Partikelkanone
Probability Z		
Size Func	x, y, z	$x, y, z \in [-1..1]$ = Position des Partikels im Erzeugungsbereich
Dir Func X	x, y, z	$x, y, z \in [-1..1]$ = Position des Partikels im Er-
Dir Func Y		zeugungsbereich
Dir Func Z	S_{ize}	S_{ize} = Größe des Partikels in Längeneinheiten
Speed Func	x, y, z	$x, y, z \in [-1..1]$ = Position des Partikels im Er-
	dir_x, dir_y, dir_z	dir_x, dir_y, dir_z = Normierte Richtung des Partikels
	S_{ize}	S_{ize} = Größe des Partikels in Längeneinheiten
Brightness Func	x, y, z	$x, y, z \in [-1..1]$ = Position des Partikels im Er-
	dir_x, dir_y, dir_z	dir_x, dir_y, dir_z = Geschwindigkeit des Partikels
	S_{ize}	S_{ize} = Größe des Partikels in Längeneinheiten

nur mit Hilfe einer Callback möglich (siehe 5.4.2, 5.4.3). Alle beteiligten Kameraeffekte sind entweder abhängig von der aktuellen Partikelposition oder unabhängig von dieser. Diese Abhängigkeit trennt die Kameraeffekte im Groben, die unten stehenden Tabelle zeigt diese Abhängigkeiten.

Tabelle 4.2: Kameraeffekte kategorisiert nach Partikelposition

positionsabhängige Effekte	positionsunabhängige Effekte
Schärfentiefe	Begrenzte Auflösung
Geisterfleck	Verzerrung
Bewegungsunschärfe	Vignettierung
Begrenzte Auflösung	Rauschen

4.3 Positionsabhängige Effekte

Beim Erzeugen einer Aufnahme, werden zuerst alle von der Partikelposition abhängigen Effekte berücksichtigt. Die Auswirkungen dieser Effekte fließen in die Texturerzeugung ein. Für die Berechnung der Schärfentiefe und des Geisterflecks stehen die Formeln aus Tabelle 4.3 zur Verfügung. Die Ergebnisse der Unsharp Func und der Bright Func wer-

Tabelle 4.3: Formeln zur Realisierung der Schärfentiefe

Name	Parameter	Bedeutung
Fokus	keine	Der Abstand zur Kamera an dem die komplette Bildebene scharf abgebildet wird
Unsharp Func	x, y	x, y $[-1..1]$ = Position im Bild (0/0) = Bildmitte
	z	z Abstand zur Fokusebene 0 = Partikel exakt in der Fokusebene
	a	a Fokus der Kamera
Bright Func	x, y	x, y $[-1..1]$ = Position im Bild (0/0) = Bildmitte
	z	z Abstand zur Fokusebene 0 = Partikel exakt in der Fokusebene
	a	a Fokus der Kamera

den schließlich an die Texture Func (Tabelle 4.4) , welche für die Erzeugung der Partikel Texturen verwendet wird, weitergereicht. Der Bewegungsunschärfefeekt wird in der Texture Func durch die Parameter w und h berücksichtigt. Diese beiden Parameter werden

Tabelle 4.4: Parametrisierung der Texture Erzeugungs Funktion

Name	Parameter	Bedeutung
	x, y	x, y Position im Bild (0/0) = Bildmitte
	w	Echte Breite der Textur auf dem Ausgabebild
	h	Echte Höhe der Textur auf dem Ausgabebild
Texture Func	u_{start}	Ergebnis der Unsharp Func für die Startposition des Partikels
	u_{end}	Ergebnis der Unsharp Func für die Endposition des Partikels

vom Simulator aufgrund der Belichtungszeit und der Geschwindigkeit des Partikels vor dem Aufruf der Texture Func berechnet. Der Effekt der begrenzten Auflösung wirkt sich sowohl direkt für ein Partikel aus, als auch indirekt durch die Kamera. Bei der direkten Auswirkung hat er den als Punktspreizfunktion bekannten Effekt von 2.2.2 zur Folge und muss daher bei der Texture Func oder der Create Texture Callback 5.4.6 berücksichtigt werden.

4.4 Positionsunabhängige Effekte

Die Reihenfolge der umgesetzten Effekte, welche sich auf einen Ausgabedatensatz auswirken, sind der physikalischen Durchlaufreihenfolge des Lichtes durch das Linsensystem nachempfunden. Ein Lichtstrahl der auf das Linsensystem trifft, wird zuerst durch die Sammellinse des Objektivs gebündelt. Aus dieser Bündelung heraus entsteht der Effekt der begrenzten Auflösung 2.2.2. Hiernach durchläuft der Lichtstrahl die verschiedenen Linsen um letztendlich auf den Sensor zu treffen. Beim Durchlaufen dieser Linsen tritt der Verzerrungseffekt 2.2.2 auf, gefolgt vom Vignettierungseffekt 2.2.2, welcher durch die Betrachtung der Ränder des Objektivs entsteht. Zuletzt wirkt sich der Effekt des Rauschens 2.2.2 aus. Zur Umsetzung der einzelnen Kameraeffekte stehen wieder Formeln zur Verfügung. Der Effekt begrenzte Auflösung wird mittels der Fourier-Transformation [6] [22] umgesetzt. Diese bearbeitet den Ausgabedatensatz komponentenweise mit:

Tabelle 4.5: Parametrisierung der Fourier-Transformation

Name	Parameter	Bedeutung
FFT	x, y	$x, y \in [-1..1]$ = Position im Bild, wobei (0/0) = Bildmitte

Der Effekt der Verzerrung wird wie in [27] beschrieben umgesetzt. Die Formel für die Verzerrung wird durch [27] bereits vorgegeben, lediglich die Parametrisierung der Formeln kann beeinflusst werden. Tabelle 4.6 listet alle möglichen Parameter für den Verzerrungseffekt auf. Der umgesetzte Verzerrungseffekt ist in der Lage, einfache Krümmungen in Abhängigkeit von der Linsenposition zu realisieren. Die Umsetzung einer Wellenförmigen

Verzerrung ist nicht vorgesehen.

Tabelle 4.6: Parametrisierung des Verzerrungseffektes

Parameter	Bedeutung
Lens X, Lens Y	Gibt die Mittelpunktposition der Linse an (0/0) = Bildmitte
K1	Gibt die Primärverzeichnung an = Vorfaktor des kubischen Terms der Taylerentwicklung, 0 = Deaktivierung
K2	Gibt die Sekundärverzeichnung an = Vorfaktor der fünften Potenz des Terms der Taylerentwicklung, 0 = Deaktivierung
$\lambda X, \lambda Y$	Gibt die unsymmetrische Verzerrung entlang der X-Achse, Y-Achse an, 0 = Deaktivierung
Zoom	Während der Berechnung der Verzerrung kann in das Bild zum Zentrum der Linse hineingezoomt werden, 1 = Deaktivierung
Over Sampling	Mit Hilfe des Over Sampling Parameters kann eine Überabtastung des Original-Datensatzes erzeugt werden. Durch diesen Effekt kann sichergestellt werden, dass der Ausgabedatensatz keine Löcher/Lücken aufweist.
Interpolation Mode	Hier kann die Art und Weise der Überabtastung festgelegt werden, Nearest Neighbour erhält die Kanten und Artefakte, Bilinair wirkt sich glättend aus.

Der Vignettierungseffekt lässt sich mittels einer einfachen Formel zur Abschwächung des Bildes darstellen:

Tabelle 4.7: Parametrisierung des Vignettierungseffektes

Name	Parameter	Bedeutung
Vignetting Func	x, y	$x, y \in [-1..1]$ = Position im Bild, wobei (0/0) = Bildmitte

Als letztes wird der Rauscheffekt berücksichtigt. Für seine Umsetzung stehen zwei Möglichkeiten zur Verfügung. Zum einen die Callback-Variante 5.4.7 und zum andern eine Formel, welche komponentenweise auf den Ausgabedatensatz angewandt wird. Tabelle 4.8 veranschaulicht die für die Umsetzung des Rauscheffektes benötigte Noise Func. Durch die Übergabe des vorher an der entsprechenden Pixelposition vorherrschenden Helligkeitswertes jeder Farbkomponente wird dem Nutzer die Möglichkeit gegeben, den Rauscheffekt abhängig von diesem Helligkeitswert zu gestalten. Soll ein reines Schwarzweißrauschen realisiert werden, so muss das Rauschen im Roten Farbkanal berechnet, und für die weiteren Farbkanäle der Rotwert übernommen werden. Ist kein Rauschen erwünscht, so muss der alte Helligkeitswert der jeweiligen Farbkomponente unverändert zurückgegeben werden.

Tabelle 4.8: Parametrisierung des Rauscheffektes

Name	Parameter	Bedeutung
Noise Func	x, y	$x, y \in [-1..1]$ = Position im Bild, wobei (0/0) = Bildmitte
	r_{ot}	r_{ot} = Alter Helligkeitswert des Rotkanals der Ausgabe
	g_{ruen}	g_{ruen} = Alter Helligkeitswert des Grünkanals der Ausgabe
	b_{lau}	b_{lau} = Alter Helligkeitswert des Blaukanals der Ausgabe

5 Implementierung

In Kapitel 2 wurden die Anforderungen des Simulators erörtert, und in Kapitel 3 ein Durchlauf einer Simulation. Dieses Kapitel wird sich mit der Implementierung des Simulators befassen.

Um den gesamten Simulationsprozess gemäß der Anforderungen aus 2.1 durchführen zu können, müssen die verschiedenen Effekte und Eigenschaften nach der Berechnungsreihenfolge geordnet werden. Die Umordnung geschieht derart, dass eine korrekte Simulation bei maximaler Performance durchgeführt werden kann.

Die Simulation wird in drei Schritte unterteilt:

- Vorbereitende Schritte
- Durchführung der Simulation
- Nachbereitung

5.1 Vorbereitende Schritte

Die Simulation beginnt mit dem Laden der Konfigurationsdatei. Nach dem erfolgreichen Laden werden alle Parameter auf ihre Gültigkeit hin überprüft. Die Gültigkeitsprüfung gewährleistet einen später reibungslosen Ablauf der gesamten Simulation. Abgesehen vom Überlauf des Speichers und eventueller falscher Rückgaben durch Callbacks lassen sich so alle Fehler der Parameter im Voraus aufdecken und beseitigen.

Aus den in 2.2.4 ermittelten Parametern ergeben sich folgende vorberechenbare Werte:

- Initialisierung Zufallsgenerator
- Berechnungen zum Lebensraum
- Prüfung der Ausgabeordner
- Umwandeln der Formeln in Zwischencode
- Vorberechnung der Vignettierung
- Allokation des Tiefensortierpuffers für alle Kameras
- Allokieren des ersten Partikelpufferblocks
- Berechnung des Simulationsendes

Initialisierung Zufallsgenerator

Während der Simulation stehen zwei verschiedene Wahrscheinlichkeitsmodelle 2.2.1 zur Verfügung. Das Wahrscheinlichkeitsmodell der Gleichverteilung ist Teil der verwendeten Programmiersprache für den Simulator. Für das zweite Wahrscheinlichkeitsmodell

wurde die Box-Muller-Transformation [3] [10] verwendet. Mit Hilfe der Box-Muller-Transformation lassen sich gleich verteilte Variablen in normal verteilte Zufallszahlen umrechnen. Beide Wahrscheinlichkeitsmodelle sind somit abhängig von nur einem Zufallszahlengenerator. Dieser wird zum Zwecke der Wiederholbarkeit bei jeder Simulation durch die Konfigurationsdatei gesetzt.

Berechnungen zum Lebensraum

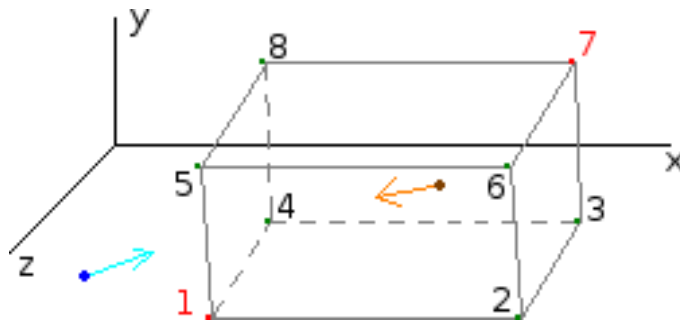


Abbildung 5.1: Darstellung Lebensraum mit Partikeln

Die Beschleunigung der Simulation wird unter anderem durch das möglichst frühe Löschen eines Partikels erreicht. Hierzu muss festgestellt werden, ob sich ein Partikel innerhalb oder außerhalb des Lebensraumes befindet. Befindet er sich außerhalb, so muss geprüft werden, ob sich das Partikel in Richtung des Lebensraumes bewegt, diesen also irgendwann noch betreten wird, oder ob sich das Partikel vom Lebensraum weg bewegt, und gelöscht werden kann. Da das Objektraum Clipping besonders häufig während der Simulation durchgeführt werden muss, wurde nach einer besonders effizienten Lösung gesucht. Für das Löschen eines Partikels ist ausschlaggebend ob dieser den Lebensraum verlässt. Das auf Abbildung 5.1 braun dargestellte Partikel befindet sich innerhalb des Lebensraumes. Es muss gelöscht werden, wenn es eine der drei folgenden Ebenen durchbricht, E1 aufgestellt durch die Punkte (1,4,5), E2 aufgestellt durch die Punkte (1,2,5) und E3 aufgestellt durch die Punkte (1,2,4). Der Lebensraum besteht insgesamt aus sechs dieser Ebenen. Jede Ebene unterteilt das Koordinatensystem in zwei Halbräume. Einer der beiden Halbräume beinhaltet den Lebensraum, der andere nicht. Da sich das Partikel innerhalb des Lebensraumes befindet, muss der Test mit den verbleibenden drei Ebenen nicht durchgeführt werden. Somit wurde das Objektraum Clipping reduziert auf drei Tests gegen eine Ebene. Sobald nur eine dieser drei Ebenen durchbrochen wird, darf das Partikel gelöscht werden. Ein weiterer beschleunigender Faktor ist die Eigenschaft der Achsenparallelität des Lebensraumes. Dank dieser Eigenschaft reduziert sich der Test, ob eine Ebene durchbrochen wurde, auf zwei Vergleiche. Der erste Vergleich prüft das Vorzeichen der Bewegungsrichtung des Partikels der jeweiligen Komponente und der zweite Test prüft, ob die jeweilige Komponente innerhalb oder außerhalb des Halbraumes liegt, in welchem sich der Lebensraum befindet. Durch dieses Verfahren ist es mit nur sechs Fließkommaoperationen möglich, ein Objektraum Clipping für ein Partikel innerhalb des Lebensraumes durchzuführen. Abbildung 5.1 zeigt noch ein weiteres Partikel (blau), dieses befindet sich außerhalb des Lebensraumes, bewegt sich aber

in Richtung des Lebensraumes, darf also nicht gelöscht werden. Der oben beschriebene Ansatz bewerkstelligt auch diese Aufgabe. Erreicht wird das durch die Betrachtung der Richtung, in welche sich das Partikel bewegt. Durch die Betrachtung der Bewegungsrichtung der jeweiligen Richtungskomponente wird die entsprechende Halbebene ausgewählt, gegen welche getestet wird. Dies ist möglich, weil im vorbereitenden Schritt eine genaue Identifizierung der Halbebenen vorgenommen wurde. Dadurch wird auch im Falle eines Partikels außerhalb des Lebensraumes nur gegen drei Ebenen geprüft. Im speziellen Beispiel von Abbildung 5.1 sind dies die den Ebenen E1-3 gegenüberliegenden Ebenen.

Prüfung der Ausgabeordner

Während der vorbereitenden Phase wird das Lese-/Schreibrecht auf alle Ausgabeordner überprüft. Dies geschieht, indem zuerst ein unbenutzter Dateiname gesucht wird, und dann eine Datei in den entsprechenden Ausgabepfad geschrieben wird. Ist dieser Vorgang für alle Ausgabepfade erfolgreich, wird davon ausgegangen, dass sich die Schreibrechte während der Simulation nicht ändern.

Umwandeln der Formeln in Zwischencode

Wie in 2.2.1 bereits festgelegt sollen alle Parameter in der Simulation mittels Formeln einstellbar sein. Zu diesem Zweck wurde ein entsprechender Formel Solver entwickelt. Dieser besteht aus zwei Komponenten. In einem ersten Schritt werden alle Formeln in einen Zwischencode umgeformt [25]. Liegt eine Formel im Zwischencode vor, so kann sie mehrfach bei unterschiedlicher Belegung einzelner Variablen ausgerechnet werden. Enthält die Formel Zufallsvariablen, so darf für jedes Wahrscheinlichkeitsmodell jeweils nur einmal pro Evaluierung der Formel eine Zufallszahl gezogen werden. Ein mehrfaches Vorkommen einer Zufallsvariable eines Wahrscheinlichkeitsmodells ergibt während einer Evaluierung stets den selben Wert. Die Darstellung durch Zwischencode erspart sehr viel Rechenzeit. Die Alternative der textuellen Ersetzung der Variablen und dann Auswertung des resultierenden Strings ist der Darstellung im Zwischencode aufgrund der nicht effizient umsetzbaren String- Operationen daher weit unterlegen. Die Darstellung im Zwischencode beinhaltet ebenfalls die verschiedenen Bindungsstärken der unterschiedlichen Operatoren, bei einer Implementierung mittels String-Operatoren, müsste diese Bindung stets durch einen sogenannten „Recursive descent parser“ [25] [19] gelöst werden. Dieser hat den Nachteil, dass er unter Umständen sehr viele rekursive Aufrufe erzeugt und somit unübersichtlich viel Speicher verbraucht. Bei Formeln, die besonders häufig aufgerufen werden, wie etwa alle bei der Partikelerzeugung benötigten Formeln oder der Rauschformeln, kann durch die Zwischencodedarstellung eine signifikante Beschleunigung erreicht werden.

Vorberechnung der Vignettierung

Der Effekt der Vignettierung ist ein von der Partikelposition unabhängiger Effekt. Dies bedeutet, er wirkt sich zwar auf alle Ausgabebilder einer Kamera aus, ist bei diesen aber gleich. Aus diesem Grund genügt es, wenn sein Einfluss nur ein einziges Mal berechnet wird. Die Berechnung des Vignettierungs Effektes geschieht mit Hilfe der zuvor erstellten Zwischencodedarstellung der Vignettierungsformel. Für den Einfluss der Vignettierung wird ein zweidimensionales Array in der Größe des Ausgabebildes im Speicher angelegt. Für jede Pixelposition wird nun mittels der Vignettierungsformel der entsprechende Einfluss berechnet und abgespeichert. Beim Erzeugen eines Ausgabebildes durch den Si-

mulator kann dann jeweils auf diese Informationen ohne eine Neuberechnung zugegriffen werden.

Allokation des Tiefensortierpuffers für alle Kameras

Für die korrekte Darstellung der Partikel müssen diese tiefensortiert (Sortierung nach Abstand zur Kamera) gerendert werden. Nur so kann sichergestellt werden, dass ein Partikel im Vordergrund diejenigen Partikel, welche sich im Hintergrund befinden, überdecken kann. Normalerweise wird eine derartige Funktion durch den Tiefenpuffer der OpenGL-Schnittstelle [15] ermöglicht. Die Tiefenpuffer-Funktion von OpenGL kann im Fall des implementierten Simulators nicht verwendet werden. Dies ist durch die Verwendung der Technik namens „Alphablending“ [2] begründet. Das Alphablending ist eine spezielle Anwendungsform des Alpha Compositings. Mit Hilfe des Alphablending ist es möglich, eine semitransparente Textur darzustellen. Die Verwendung des Tiefenpuffers ist nur bei einer komplett opaken Szene korrekt. In OpenGL besteht eine Graphik nicht nur aus den drei Farbkanälen Rot, Grün und Blau, sondern auch noch aus einem vierten Kanal, dem sogenannten Alpha Kanal. Das Alphablending nutzt diesen zusätzlichen Kanal um die Opazität eines jeden Pixels in der Graphik zu berechnen. Durch diese Technik ist es möglich, eine Textur zu erzeugen, welche zum Beispiel im Mittelpunkt opak und an den Rändern transparent ist. Das Alphablending arbeitet hierbei nach der Formel:

$$C_{Buffer+1}^{ij} = C_{Image}^{kl} * A_{Image}^{kl} + C_{Buffer}^{ij} * (1 - A_{Image}^{kl}) \quad (5.1)$$

$C_{Buffer+1}^{ij}$ Steht für die Farbe des Renderpuffers an Pixelkoordinate (i/j) nach dem Zeichnen des Bildes. Entsprechend C_{Buffer}^{ij} vor dem Zeichnen, C_{Image}^{kl} für den Farbwert des Bildes an der interpolierten Stelle (k/l) und A_{Image}^{kl} für den entsprechenden Alpha Wert. Auf Abbildung 5.2 ist die Anwendung der Formel auf der rechten Seite nochmals bildlich dargestellt. Durch die Abhängigkeit der Alphablending-Formel vom bisherigen

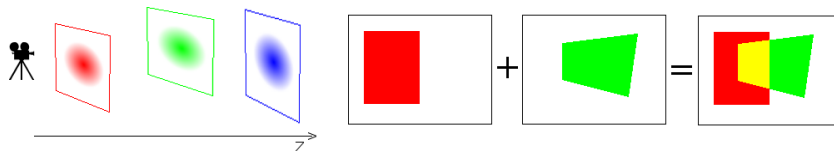


Abbildung 5.2: links : Darstellung einer Szene mit Tiefe, rechts Beispiel Alphablending

Renderpuffer ergibt sich die Notwendigkeit der tiefensortierten Rendering. Auf Abbildung 5.2 links ist eine beispielhafte Szene zu sehen. Die korrekte Render-Reihenfolge für diese Szene lautet: zuerst das blaue, dann das grüne und zuletzt das rote Partikel. Das in 2.2.4 beschriebene Clipping im Bildraum ermöglicht zwar eine Bestimmung des Abstandes des Partikels zur Kamera. Aufgrund der sequentiellen Verarbeitung der Partikel ist es zum Zeitpunkt des Renderns eines einzelnen Partikels jedoch nicht möglich zu entscheiden, ob dieser verdeckt wird oder nicht. Um die hohen Rechenkosten einer Sortierung der Partikel nach deren Abstand zur Kamera zu sparen wurde das Konzept des Tiefensortierpuffers eingeführt. Jede Kamera in OpenGL besitzt eine sogenannte Z_{near} und Z_{far} Clipping Ebene. Dies sind zwei Ebenen, welche eine Beschränkung der Sichte ebene nach vorn beziehungsweise nach hinten festlegen. Alle Objekte, deren Abstand zur Kamera größer als Z_{far} ist, werden von der OpenGL Render-Pipeline ignoriert, folglich

also nicht gerendert. Ebenso alle Objekte, welche sich näher an der Kamera befinden als Z_{near} . Aufgrund der perspektivischen Division ist in OpenGL eine Z_{near} Ebene mit Abstand größer 0 erforderlich. Die Beschränkungen von Z_{near} und Z_{far} nutzt das Konzept des Tiefensortierpuffers aus. Die Distanz zwischen Z_{near} und Z_{far} wird in diskrete Schritte von jeweils einer Längeneinheit unterteilt. Zu Beginn eines Rendervorganges wird das Array entsprechend zurückgesetzt. Soll dann ein Partikel gerendert werden, so wird es in einem ersten Schritt vollständig berechnet und in den Tiefensortierpuffer an entsprechender Stelle eingetragen. Durch das in 2.2.4 beschriebene View Frustum Culling, werden nur diejenigen Partikel betrachtet, welche wirklich gerendert werden. Und durch die korrekte Eintragung in den Tiefensortierpuffer werden diese automatisch sortiert. In einem zweiten Schritt werden anschließend alle Elemente des Tiefensortierpuffers in entsprechender Reihenfolge von hinten nach vorne mittels Alphablending gerendert. Das Konzept des Tiefensortierpuffers arbeitet hinreichend exakt bei minimalem Rechenaufwand. Das Verfahren ist nicht perfekt, da Partikel, deren Abstand zur Kamera sich um weniger als eine Längeneinheit unterscheidet, entsprechend ihrer Reihenfolge im Speicher und nicht bezüglich ihrer Tiefe gerendert werden. Der Vorteil der Rechenkomplexität wird durch einen Mehrverbrauch an Speicher erreicht. Dieser ist ca. so groß wie die maximale Distanz von $Z_{far} - Z_{near}$ aller verwendeten Kameras. Die Minimierung der Zeit zum Sortieren von $O(n * \log(n))$ auf $O(n)$ erfordert den zusätzlichen Tiefensortierpuffer.

Allokieren des ersten Partikelpufferblocks

Wie in 2.2.4 diskutiert wird bei den vorbereitenden Schritten, unter der Annahme, dass in der Zukunft Partikel erzeugt werden, ein erster Partikelblock allokiert. Der Simulator allokiert hierbei einen Speicherblock, welcher durch zwei Größen beeinflusst wird. Zum einen die Größe in Byte, welche ein einzelnes Partikel im Speicher benötigt und zum anderen die Anzahl der Partikel, welche jeweils zusammenhängend allokiert werden sollen. Der so erhaltene Speicherbereich wird als leer markiert und in die entsprechenden Einzelblöcke für jeweils ein Partikel zerlegt. Anschließend wird jeder dieser Blöcke in einen Freispeicher-Stack eingetragen. Diese Vorgehensweise erspart ein späteres Suchen nach freien Speicherblöcken. Reicht der Platz eines Speicherblockes während der Simulation nicht aus, so wird ein weiterer Block allokiert, mit welchem dann gleich verfahren wird. Die Wahl der Blockgröße sollte sich hierbei an den Eigenschaften der zu simulierenden Szene orientieren. Ein zu großer Wert wirkt sich nachteilig auf die Geschwindigkeit aus, da der Simulator beim Simulieren eines Zeitschrittes alle Elemente dieses Blockes abfragen muss. Wird die Blockgröße hingegen zu klein gewählt, wird das in 2.2.4 angestrebte Konzept der Speicherlokalität außer Kraft gesetzt.

Berechnung des Simulationsendes

Das Ende der Simulation kann ebenfalls bereits im Voraus berechnet werden. Die Simulation endet, wenn alle Kameras oder die Szene auslesenden Events abgeschlossen wurden. Wenn eine Kamera eine Belichtung über diesen Zeitpunkt hinaus durchführt, wird diese berücksichtigt.

5.2 Durchführung der Simulation

Während der Simulation wechselt der Simulator zwischen den beiden Zuständen:

- Schrittweise Simulation der Zeit
- Aufnahme eines Zeitpunktes

5.2.1 Schrittweise Simulation der Zeit

Der Simulator startet immer mit dem Zeitpunkt 0, es sei denn, bei der Initialisierungsphase wurde eine gespeicherte Szene geladen. Ausgehend von diesem Zeitschritt wird der nächste zu simulierende Zeitschritt berechnet. Dieser Zeitschritt kann aufgrund einer der folgenden Ereignisse eintreten:

- Erzeugung eines Partikels
- Speichern der Szene
- Aufnahme eines Zeitpunktes

Erzeugung eines Partikels

Ein Partikel wird entweder durch eine Partikelkanone erzeugt, oder durch eine Callback. Bei der Erzeugung des Partikels durch die Partikelkanone, wird jeweils der nächste Zeitpunkt zur Erzeugung eines weiteren Partikels berechnet. In der Callback-Variante, wird dies durch die Rückgabeparameter der Callback geregelt.

Speichern der Szene

Das Speichern der Szene wird entweder durch das automatische Speichern alle n Minuten ausgelöst. Hierzu misst der Simulator die echte vergangene Zeit und triggert so das automatische Speichern. Bei dieser Art der Speicherung wird der jeweils vorherrschende simulierte Zeitschritt als Dateiname der Szene mit abgespeichert. Als weitere Form der Speicherung kann der Nutzer einen Zeitpunkt in der XML-Datei definieren. Ist dieser erreicht wird die Szene nach dem gleichen Verfahren wie bei der automatischen Abspeicherung gesichert.

5.2.2 Aufnahme eines Zeitpunktes

Das wichtigste Element neben der hohen Genauigkeit 2.2.3 des Simulators ist die Aufnahme eines Zeitpunktes. Die Aufnahme erzeugt aus den Simulationsdaten einen dem Abbild der Szene entsprechenden Datensatz. Für diese Aufgabe wurde die freie Graphik-Bibliothek Mesa 3D [16] verwendet. Mesa 3D ist auf der Spezifikation von OpenGL [15] aufgebaut und stellt somit alle notwendigen Operationen zur Verfügung. Im Gegensatz zur echten OpenGL API ist es unter Mesa 3D möglich, einen Rendering Context mit einer Auflösung von 32 Bit pro Farbkanal wie in 2.2.3 gefordert zu erstellen.

Zur Erzeugung eines Datensatzes sind mehrere Schritte notwendig:

- Vorbereiten der Kameraeinstellungen
- Berechnen aller Render-Parameter für ein Partikel
- Rendern der tiefensortierten Partikel
- Umsetzung der Kameraeffekte

- Erzeugen der Ausgabedatensätze

Vorbereiten der Kameraeinstellungen

Zu Beginn einer Aufnahme wird als erstes der gesamte Rendering Context gelöscht. Anschließend wird die Größe des Rendering Context an die Ausgabegröße der aktuell verwendeten Kamera angepasst. Dieser Schritt ist notwendig, da Mesa 3D genau wie OpenGL nur eine Größe für einen Rendering Context zulässt. Mittels der Viewport Funktion kann dieser Nachteil jedoch ausgeglichen werden. Hiernach findet die Kameratransformation statt. Bei einer perspektivischen Kamera werden die Parameter Öffnungswinkel der Kamera sowie Seitenverhältnis gesetzt, bei einer telezentrischen Kamera die Breite und Höhe des Sichtfeldes. Beide Kamertypen verwenden die in 5.1 erwähnten Einstellungen für Z_{near} und Z_{far} und den ebenfalls in 5.1 beschriebenen Tiefensortierpuffer. Nachdem diese Parameter gesetzt wurden, unterscheiden sich die beiden verwendeten Kamerasysteme im Programmverlauf nicht mehr. Alle weiteren Operationen sind für beide Kamerasysteme gleich.

Berechnen aller Render-Parameter für ein Partikel

Ein Partikel ist wie in 2.2.1 definiert einen volumenartigen Körper. Dieser wird mit der Billboard Technik [7] [26] gerendert. Dank dieser Technik ist es möglich mit zweidimensionalen Texturen einen dreidimensionalen Körper darzustellen. Für das Billboarding muss das Partikel durch die Augtransformation in das Koordinatensystem der Kamera überführt werden. Die Positionsdaten des Partikels, liegen dann relativ zur Kameraposition vor. Aus diesen Positionsdaten lassen sich mit nur einer weiteren Matrizenmultiplikation die normalisierten Gerätekoordinaten berechnen. Diese Koordinaten stellen einen Würfel der Kantenlänge zwei dar, dessen Schwerpunkt im Ursprung liegt. Mit Hilfe der normalisierten Gerätekoordinaten, kann ein Bildraum Clipping 2.2.4 durchgeführt werden. Wenn das Partikel durch diese Optimierungsstufe nicht vom Rendern ausgeschlossen wurde, werden mit Hilfe der Positionsdaten relativ zur Kamera Werte für die Unschärfe und die Helligkeit in Abhängigkeit vom Abstand zur Fokusebene bestimmt. Bei der Texturerzeugung wird die Texture Func mit all ihren Parametern für die sich ergebende Skalierungsstufe aufgerufen und die Partikeltextur erzeugt. Nach der erfolgreichen Erstellung der Textur für die Partikel werden diese samt all der zum Rendern nötigen Daten gemäß 5.1 in den Tiefensortierpuffer eingetragen.

Rendern der tiefensortierten Partikel

Nachdem alle zu rendernden Partikel in den Tiefensortierpuffer eingetragen wurden, kann dieser von hinten nach vorne durchlaufen werden. Jedes Element dieses Puffers wird durch einen Stack repräsentiert, welcher wiederum die für die jeweilige Tiefe zu rendernden Partikel beinhaltet. Sofort nach dem Rendern des entsprechenden Partikels wird die für das Rendern benötigte Textur aus dem Speicher entfernt. Die Darstellung der einzelnen Tiefen durch einen Stack wird hierbei nicht mittels eines echten Stacks simuliert. Ein echter Stack hätte den Nachteil, ständig auf- und wieder abgebaut zu werden, und würde so immer wieder im Speicher hin- und herkopiert werden müssen. Ein Stack mittels Zeigerstrukturen würde sich zusätzlich negativ auf die Datenlokalität auswirken und Seitenfehler erzeugen. Aus diesem Grund wird der Stack der jeweiligen Tiefen mittels eines der Speicherverwaltung ähnlichen Prinzips realisiert. Bei Beginn der Simulation sind alle Stacks in allen Tiefen leer. Während eines Renderschrittes wer-

den dann vom Simulator in die einzelnen Tiefen des Sortierbuffers Partikel eingetragen. Die jeweiligen Tiefen werden dabei durch dynamisch anwachsende Arrays und einen Zähler repräsentiert. Die Arrays werden erst bei Simulationsende wieder gelöscht. Der Zähler wird nach jedem Rendschritt zurückgesetzt. Auf diese Weise werden die gepufferten Partikeldaten lokal in Arrays gehalten. Da die Größe der einzelnen Puffer monoton wächst und die Partikelzahl im Mittel gleich bleibt, werden im Laufe der Simulation immer weniger Neuallokierungen für die einzelnen Tiefenstacks nötig.

Umsetzung der Kameraeffekte

Das Erzeugen des Ausgabedatensatzes wird durch das Einrechnen der Kameraeffekte abgeschlossen. Berücksichtigt werden hierbei folgende Effekte in der gegebenen Reihenfolge:

1. Begrenzte Auflösung
2. Verzerrung
3. Vignettierung
4. Rauschen

1. Begrenzte Auflösung

Die in 2.2.2 eingeführte begrenzte Auflösung wird mittels einer Fourier-Transformation auf dem gesamten Ausgabedatensatz realisiert. Hierzu wird jeder Farbkanal einzeln Fourier-transformiert. Dies ist notwendig, da der Nutzer mittels Callbacks auch farbige Partikel erzeugen kann. Eine Realisierung der begrenzten Auflösung ohne Berücksichtigung der Farbkanäle hätte Graustufenbilder zur Folge und würde somit den Funktionsumfang einschränken. Im Fourier-Raum kann der Nutzer dann mittels dreier Formeln (für jeden Farbkanal eine Formel) festlegen, wie stark die jeweiligen Frequenzanteile gewichtet werden sollen. Ein Gewicht von Null unterdrückt die jeweilige Frequenz komplett. Durch die Verwendung von drei Formeln, für jeden Farbkanal eine, können unterschiedliche maximalen Ortsfrequenzen der einzelnen Farbkanäle berücksichtigt werden, dies ermöglicht eine entsprechende Anpassung an das Bayer-Matrizen-Prinzip.



Abbildung 5.3: Auswirkungen der begrenzten Auflösung

Auf Abbildung 5.3 sind die Auswirkungen des Effektes begrenzte Auflösung in einem Graustufenbild bei gleicher begrenzter Auflösung in allen drei Farbkanälen zu erkennen. Links die Abbildung zweier Linien als Repräsentant für zwei Partikelflugbahnen. Rechts das Ergebnis nach dem Effekt: Die beiden Linien sind ab einem gewissen Abstand nicht mehr von einander unterscheidbar und verschmelzen.

2. Verzerrung

Anschließend wird die in 2.2.2 eingeführte Verzerrung in den Ausgabedatensatz eingerechnet. Je nach Parametrisierung kann hier eine einfache tonnen- oder kissenförmige

Verzerrung realisiert werden. Der Verzerrungseffekt wird ebenfalls wie die begrenzte Auflösung auf allen drei Farbkanälen mittels eigener Parametrisierung getrennt berechnet. Durch diese Vorgehensweise ist eine maximale Konfigurationsvielfalt gegeben. Eine Realisierung der tonnenförmigen Verzerrung im Rotkanal und der kissenförmigen Verzerrung im Blaukanal ist dank der Unabhängigkeit ebenfalls möglich.

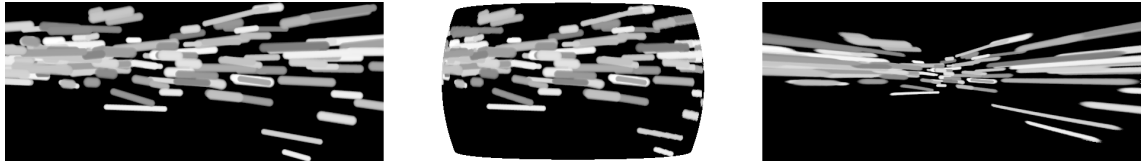


Abbildung 5.4: Darstellung des Verzerrungseffektes v.l.n.r. : Original, tonnenförmige, kissenförmige Verzerrung

Die Nachbildung einer wellenförmigen Verzerrung ist nicht vorgesehen. Abbildung 5.4 zeigt den jeweiligen Einfluss der Verzerrungsform auf die Graustufen-Ausgabedaten.

Vignettierung

Weiter wird die Vignettierung berücksichtigt. Die in 5.1 vorberechneten Daten werden in die Szene eingerechnet. Hierzu wird jeder Helligkeitswert der einzelnen Farbkanäle mit dem durch die Vignettierungsformel berechneten Gewichtungsfaktor multipliziert.



Abbildung 5.5: Auswirkungen der Vignettierung

Zur besseren Veranschaulichung des Vignettierungseffektes ist Abbildung 5.5 farblich invertiert dargestellt. Abbildung 5.5 zeigt links das Original, und rechts die Auswirkungen der Vignettierung. Die Vignettierung wirkt sich auf alle Farbkanäle gleich aus und ist mit nur einer einzigen Formel für das gesamte Bild realisiert.

Rauschen

Als letztes wird der Rauscheffekt in den Ausgabedatensatz eingerechnet. Durch die verwendete Formelvariante ist ein Rauschen um den Signalwert, ebenso wie ein Hotpixel-Rauschen und ein Bereichsrauschen möglich. Abbildung 5.6 zeigt links eine Szene ohne Rauschen, in der Mitte die selbe Szene mit weißem Rauschen in Abhängigkeit der Helligkeit des Pixels im Bild, und rechts die Szene mit einem weißen Rauschen, welches sich über das gesamte Bild erstreckt.

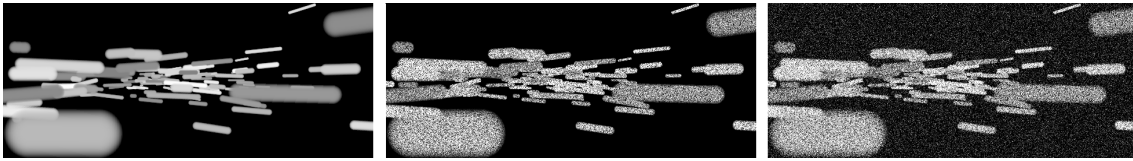


Abbildung 5.6: Auswirkungen des Rauschens

Erzeugen der Ausgabedatensätze

Als letztes werden die vom Nutzer eingestellten Ausgabedaten erzeugt. Je nach Konfiguration ist eine beliebige Untermenge der drei folgenden Datensätze verfügbar:

1. 32-Bit-Datensatz
2. 32-Bit-Infodatei
3. 8-Bit reduzierter Datensatz

1. 32-Bit-Datensatz

Der in 2.2.3 festgelegte 32-Bit-Datensatz ist die genaueste Darstellung der Szene, die der Simulator erzeugen kann. Dieser Datensatz wird als Rohdaten direkt aus dem Rendering Context ausgelesen und nach der Bearbeitung durch die verschiedenen Kameraeffekte gespeichert.

2. 32-Bit-Infodatei

Der 32-Bit-Infodatensatz wird benötigt, wenn man später die exakten Partikelinformationen zu einzelnen Pixeln des Ausgabedatensatzes auslesen möchte.

3. 8-Bit reduzierter Datensatz

Der reduzierte Datensatz dient der schnellen und einfachen Validierung der Szenerie. Das Ausgabeformat ist eine 8-Bit pro Farbkanal verlustfrei abgespeicherte PNG-Datei.

5.3 Nachbereitung

In der Nachbearbeitungsphase werden abschließend nochmals Informationen zum Verlauf der Simulation ausgegeben. Dies beinhaltet die gesamte Laufzeit der Simulation den letzten simulierten Zeitschritt, so wie die Anzahl der insgesamt erzeugten Partikel. Zuletzt werden sämtliche allokierten Speicherbereiche wieder freigegeben und die Anwendung beendet sich selbstständig.

5.4 Callbacks

Das in 2.2.6 geforderte Prinzip der hohen Flexibilität wird nicht nur durch die Anwendung verschiedenster mathematischer Formeln ermöglicht, sondern auch durch die Erweiterbarkeit des Simulators. Durch eine Library werden Callbacks zur Verfügung gestellt. Der Einsatz von Callbacks ist an insgesamt sieben Stellen im Simulator möglich, diese sind:

1. Setzen der Custom Bytes für alle Partikel

2. Erzeugen der Partikel mittels Callback
3. Setzen der Custom Bytes der Partikel, nach regulärem Erzeugen
4. Bewegen eines einzelnen Partikels
5. Bewegen aller Partikel, inklusive Löschen
6. Erzeugen einer Textur für die Rendering eines Partikels
7. Berechnung des Rauscheffektes der Ausgabe

Eine Callback ersetzt hierbei die jeweilige Funktion im Simulator. Lediglich der Einsatz der Custom Bytes für Partikel stellt eine komplett neue Funktion dar. Der Zugriff auf diese Custom Bytes ist ebenfalls nur durch Callbacks möglich.

5.4.1 Setzen der Custom Bytes für alle Partikel

Ein im Simulator verwendetes Partikel besteht aus insgesamt elf Werten:

- ID - Zugehörigkeit zu einer bestimmten Partikelkanone und Unterscheidung der Partikel untereinander
- Position (drei Werte) - Absolut Position im Raum, an der sich das Partikel aktuell befindet
- Speed (drei Werte) - Geschwindigkeitsvektor des Partikels
- Helligkeit (drei Werte) - Helligkeit getrennt für jeden Farbkanal
- Größe - Durchmesser des Partikels

Sollen weitere Werte für ein Partikel festgehalten oder berechnet werden, so können diese mit Hilfe der Custom Bytes realisiert werden. So ist es zum Beispiel möglich, jedem Partikel eine eigene „Lebenszeit“ zuzuordnen. Dies ermöglicht eine Simulation eines Partikels, welches sich nach einer bestimmten Zeitspanne in „Luft“ auflöst. Mittels der „Custom_Bytes_Callback“ Callback kann so der zusätzlich zur Speicherung der Lebenszeit benötigte Speicherplatz in Bytes reserviert werden. Mit Hilfe der „Set_Custom_Bytes_Callback“ 5.4.3 können dann bei der Erzeugung des Partikels diese Bytes gesetzt werden. Die „Move_All_Particle_Callback“ 5.4.5 ermöglicht mittels dieser Informationen ein Löschen der Partikel nach Ablauf der Lebenszeit. Die Callback „Custom_Bytes_Callback“ wird als parameterlose Funktion, welche einen Integer Wert zurückgibt realisiert.

Listing 5.1: C-Header für Set_Custom_Bytes_Callback

```
1 // Liefert die Anzahl der Custom Bytes zurueck
2 int Custom_Bytes_Callback ();
```

5.4.2 Erzeugen der Partikel mittels Callback

Neben dem Setzen der Custom Bytes 5.4.3 ist es auch möglich das gesamte Partikel mittels einer Callback zu erzeugen. Für die Erzeugung des Partikels sind zwei Werte wichtig.

Zum einen das Partikel selbst, und zum anderen der Zeitpunkt, an welchem das Partikel erzeugt werden soll. Für die Berechnung dieser Werte stellt der Simulator der Callback den aktuell im Simulator existierenden Zeitpunkt und einen entsprechend großen Speicherplatz für das Partikel zur Verfügung.

Listing 5.2: C-Header für Create_Particle_Callback

```

1 // Schreibt an Address die Daten des Partikels ,
2 // welches zum Zeitschritt Time erzeugt werden soll
3 void Create_Particle_Callback (void *Address , long long *Time );

```

Address	ID
Address + 4	Position X
Address + 8	Position Y
Address + 12	Position Z
Address + 16	Speed X
Address + 20	Speed Y
Address + 24	Speed Z
Address + 28	Brightness R
Address + 32	Brightness G
Address + 36	Brightness B
Address + 40	Size
Address + 44	Custom Bytes

Abbildung 5.7: Speicherbelegung eines Partikels

Zu beachten ist hierbei allerdings, dass der Simulator das Setzen der Partikel-Bytes und der Custom Bytes verlangt. Der Zeiger *Address* zeigt auf das 1. Element des zu erzeugenden Partikels. Dieses Feld ist eine 32-Bit Integer Zahl, welches vom Simulator auf die Partikelkanonen ID gesetzt wird. Alle weiteren Werte des Partikels sind vom Typ float, welcher einer 32-Bit Fließkommazahl entspricht. Die Adressen der jeweiligen Parameter werden in 5.7 gezeigt. Der Parameter *Time* hält den aktuellen Zeitpunkt der Simulation. Der Simulator ruft die Callback so lange auf bis diese den Wert von *Time* auf einen Wert in der Zukunft setzt. Durch diese Verfahrensweise ist es möglich zu einem Zeitpunkt mehrere Partikel zu erzeugen. Der Zukunftswert von *Time* legt fest, zu welchem Zeitschritt der Simulator das nächste Mal die Create_Particle_Callback aufrufen wird. Das Setzen der Create_Particle_Callback deaktiviert folgende Parameter der Partikelkanone:

- ProbabilityX, ProbabilityY, ProbabilityZ
- Size_Func
- Dir_FuncX, Dir_FuncY, Dir_FuncZ
- Speed_Func
- Brightness_Func

- Count_Per_Tick
- Set_Custom_Bytes_Callback

Da der Simulator in der Initialisierungsphase diese Werte trotzdem auf ihre Gültigkeit hin überprüft, sollten die Default Werte der jeweiligen Funktionen angegeben werden.

5.4.3 Setzen der Custom Bytes der Partikel, nach regulärem Erzeugen

Wenn der Nutzer lediglich die selbst definierten Custom Bytes setzen will, so eignet sich die „Set_Custom_Bytes_Callback“. Ist diese Callback definiert, so wird sie nach der erfolgreichen Erstellung eines Partikels aufgerufen. Ebenfalls wie bei der Create_Particle_Callback steht der aktuelle Zeitpunkt der Simulation zur Verfügung.

Listing 5.3: C-Header für Set_Custom_Bytes_Callback

```
1 // Schreibt an Address die Custom Daten des Partikels ,
2 // welches zum Zeitschritt Time erzeugt werden soll
3 void Set_Custom_Bytes_Callback(void *Address, long long Time);
```

ACHTUNG !

Der Parameter *Address* zeigt wie in 5.4.2 beschrieben auf die ID des Partikels, dies kann von Vorteil sein, will man die vom Simulator für das Partikel berechneten Parameter im Nachhinein doch noch ändern. Das hat aber auch zur Folge, dass man gemäß Abbildung 5.7 erst bei Adresse $address + 44$ 5.4.2 mit dem Schreiben der Custom Bytes beginnen darf.

Das Setzen der Set_Custom_Bytes_Callback deaktiviert keine Parameter, welche über das XML-Dokument gesetzt werden können.

5.4.4 Bewegen eines einzelnen Partikels

Für die Bewegung der Partikel gibt es zwei unterschiedliche Callbacks. Die Callback zur Bewegung eines einzelnen Partikels („Move_Particle_Callback“) wird immer bei einer Kameraaufnahme aufgerufen. Wird die „Move_All_Particle_Callback“, welche alle Partikel während einer Simulation außerhalb von Kameraaufnahmen bewegt, nicht gesetzt. So werden die Partikel auch außerhalb einer Kameraaufnahme mittels Move_Particle_Callback bewegt. Zusätzlich zu den Parametern Aktueller Zeitschritt und Zeit, die simuliert werden soll, wird der Callback noch der Zugriff auf alle Partikel zur Verfügung gestellt. Dies ermöglicht zum Beispiel die Berücksichtigung einer Partikelkollision auch während einer Kameraaufnahme.

Listing 5.4: C-Header für Move_Particle_Callback

```
1 // Bewegt ein Partikel um Delta Zeitschritte
2 void Move_Particle_Callback(void *Address, void *All, int Count,
3                             unsigned long int BlockCount,
4                             long long Time, double Delta);
```

Um auf alle Partikel zugreifen zu können, muss an dieser Stelle das Speichersystem für alle Partikel erläutert werden. Abbildung 5.8 zeigt die schematische Abbildung des

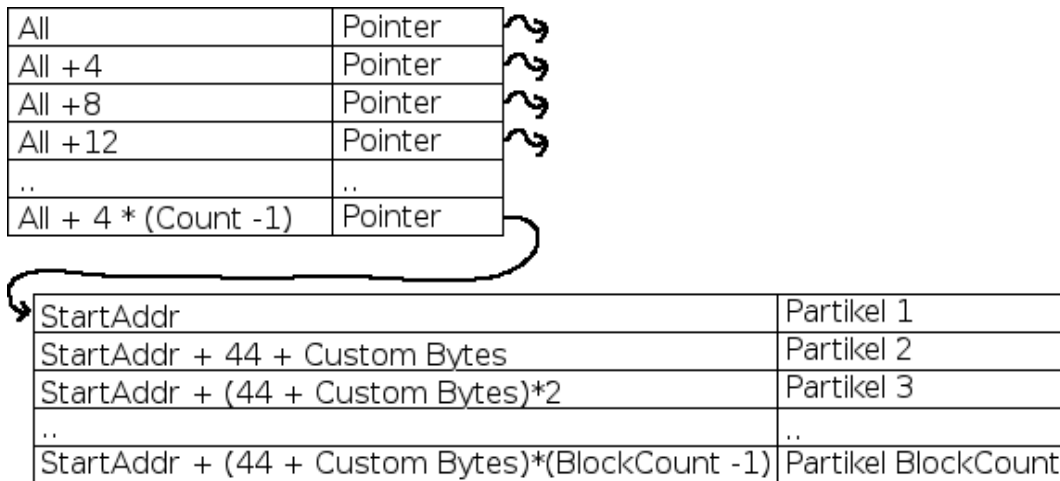


Abbildung 5.8: Speicherbelegung aller Partikel

Speichers aller Partikel. Der Parameter *All* zeigt auf den Beginn der Partikelliste. Diese Liste wiederum beinhaltet alle Startadressen der Partikelblöcke. Der Parameter *Count* legt die Anzahl dieser Blöcke fest. Der Parameter *BlockCount* entspricht dem Parameter „Particle_Block_Size“ und beschreibt die Anzahl der Partikel pro Speicherblock. Die Adresse eines Partikels ergibt sich aus der minimalen Partikelgröße von 44 Byte 5.4.2 plus Custom Bytes. Da der Custom Bytes Wert ebenfalls über eine Callback definiert wird, wird er nicht als Parameter zur Verfügung gestellt. Wurde die korrekte Adresse eines Partikels berechnet, so muss zuerst geprüft werden, ob sich an der entsprechenden Speicherstelle ein gültiges Partikel befindet. Dies ist anhand der Partikel ID erkennbar. Ist die Partikel ID gleich 0, so ist der nachfolgende Speicherplatz nicht gültig. Eine ID ungleich 0 weist auf ein gültiges, momentan in der Simulation aktives Partikel hin.

Der Parameter *Address* zeigt auf das aktuell zu bewegende Partikel. Dieses befindet sich ebenfalls in der Liste aller Partikel; da der Simulator intern aber mit einer Kopie des Partikels arbeitet, ist ein einfacher Adressvergleich zur Bestimmung des Partikels innerhalb der Partikelliste nicht möglich. *Time* zeigt den aktuellen Zeitschritt an, und *Delta* die zu simulierenden Zeitschritte.

Das Setzen der Move_Particle_Callback deaktiviert keine Parameter, welche über das XML-Dokument gesetzt werden können.

5.4.5 Bewegen aller Partikel, inklusive Löschen

Der Simulator berechnet die Flugbahn eines Partikels als lineare Funktion nach der Zeit. Sollen komplexere Bewegungsabläufe simuliert werden, so ist eine eigene Berechnung der Bewegung der Partikel notwendig. Dies schließt auch den in 5.4.1 beschriebenen Fall der begrenzten Lebenszeit ein. Aber auch eine Simulation von Schwerkraft/Magnetfeldern, oder anderer Auswirkungen auf die Flugbahn aller Partikel kann so umgesetzt werden. Der C-Header für diese Callback lautet wie folgt:

Listing 5.5: C-Header für Move_All_Particle_Callback

```

1 // Typ der Partikel loesch Callback
2 typedef void TKillCallback(void *Address);

```

```

3
4 // Bewegt alle Partikel um Delta Zeitschritte
5 void Move_All_Particle_Callback(void *All, int Count,
6                               unsigned long int BlockCount,
7                               TKillCallback KillCallback,
8                               long long Time, double Delta);

```

Zusätzlich zu allen Partikeldaten ist noch eine Callback für das Löschen eines Partikels notwendig. Durch das Verwenden von Freispeicherlisten kann nur der Simulator selbst ein Partikel freigeben. Wird die ID des Partikels auf 0 gesetzt, so berücksichtigt der Simulator das Partikel zwar nicht mehr, da es aber nicht in die Freispeicherliste aufgenommen wurde, wird der Speicherplatz auch nicht mehr verwendet, für die Zeit der Simulation entsteht an dieser Speicherposition ein „memory leak“ oder auch Speicherloch. Durch den Parameter *KillCallback* kann die Adresse des freizugebenden Partikels übergeben werden. Der Simulator ist dann in der Lage, das Partikel in die Freispeicherliste aufzunehmen. Die *KillCallback* setzt ebenfalls die ID auf 0. Die anderen Parameter entsprechen denen der *Move_Particle_Callback* aus 5.4.4.

Ist die *Move_All_Particle_Callback* definiert, so wird die *Move_Particle_Callback* zur Bewegung aller Partikel deaktiviert. Während einer Belichtung einer Kamera werden diverse Partikel einzeln bewegt. Diese Bewegung findet weiterhin durch die *Move_Particle_Callback* statt.

5.4.6 Erzeugen einer Textur für die Rendering eines Partikels

Für die Umsetzung der Simulation von nicht geradlinigen Flugbahnen ist nicht nur ein Eingriff in die Bewegung der Partikel erforderlich, ebenfalls ist es notwendig, die zu verwendende Textur, mit welcher das Partikel in der Szene dargestellt wird, entsprechend anzupassen. Die „*Create_Texture_Callback*“ Callback ermöglicht dies.

Listing 5.6: C-Header für *Create_Texture_Callback*

```

1 // Typdefinition fuer die Differenzvektoren
2 typedef struct{
3     float x;
4     float y;
5 }Tvector2;
6
7 // Callback zur Bestimmung der Partikeltextur und deren Position
8 void Create_Texture_Callback(float UnsharpS, float UnsharpE,
9                             float Brightness, void *Address,
10                            void *All, int Count,
11                            unsigned long int BlockCount,
12                            long long Time, double Delta,
13                            void *Tex,
14                            int *W, int *H,
15                            Tvector2 *TL, Tvector2 *BL,
16                            Tvector2 *TR, Tvector2 *BR);

```

Die Parameter (*All*, *Count*, *BlockCount*, *Time*, *Delta*) dienen dem Zugriff auf alle Partikel sowie der notwendigen Zeitinformationen und haben dieselbe Bedeutung wie schon

in 5.4.4 beschrieben. Der Parameter *Address* verweist auf das Partikel, dessen Textur erzeugt werden soll.

Die Parameter *UnsharpS* sowie *UnsharpE* beinhalten die Ergebnisse der Unsharp Funktion 4.3 jeweils zu Beginn und Ende der Aufnahme. Der Parameter *Brightness* die ermittelte Helligkeit der Brightness Funktion 4.3 zu Beginn der Aufnahme. Die Verbleibenden Parameter sollen anhand Abbildung 5.9 verdeutlicht werden. Durch die Parameter *W*

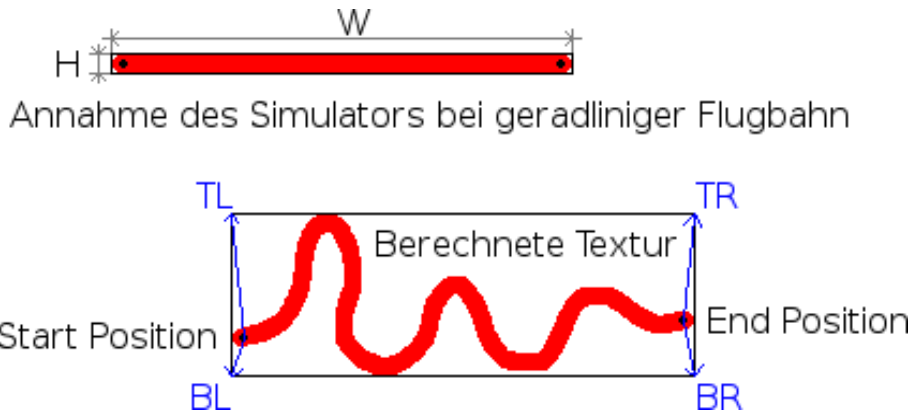


Abbildung 5.9: Veranschaulichung Partikel Callback

und *H* wird die vom Simulator berechnete Größe der Textur beim Rendern einer geradlinigen Flugbahn auf dem Ausgabebild übergeben. Anhand dieser Werte kann die Callback die optimale Größe der zu berechnenden Textur ermitteln und als Ausgabe-parameter in *W* und *H* schreiben. Die Größe der berechneten Textur muss, bedingt durch die OpenGL Schnittstelle, einer Potenz von 2 sowohl in Breite als auch Höhe entsprechen. Ist dies nicht der Fall wird die Simulation abgebrochen. Der Zeiger *Tex* muss beim Verlassen der Callback auf einen von der Library zur Verfügung gestellten Spei-

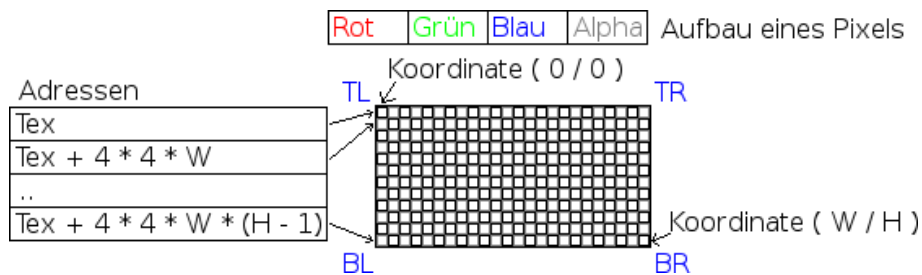


Abbildung 5.10: Speicheraufbau einer Textur

cherplatz für die berechnete Textur verweisen. Dieser muss bis zum nächsten Aufruf der *Create_Texture_Callback* gültig sein. Die Textur muss dabei dem auf Abbildung 5.10 gezeigten Schema folgen. Ein Pixel der Textur besteht aus vier 32-Bit Fließkommawerten, den Farbkanälen Rot, Grün, Blau und der Transparenz. Ein Wert von 0 für die Transparenz bedeutet keine Transparenz, entsprechend ein Wert von 1 volle Transparenz. Die Reihenfolge der Pixel muss Zeilenweise von links oben nach rechts unten erfolgen. Mit Hilfe der vier Vektoren *TL*, *BL*, *TR* und *BR* kann die exakte Position der Textur in der Billboard-Ebene relativ zur Start- und Endposition angegeben werden. *TL* und *BL*

müssen relativ zur berechneten Startposition des Partikels angegeben werden, TR und BR relativ zur berechneten Endposition. Durch die Angabe aller vier Eckpunkte sind auch nicht rechteckige Formen der abgebildeten Textur möglich. Jedoch ist zu beachten, dass durch die Triangulation [24] Bildfehler auftreten können.

Das Setzen der `Create_Texture_Callback` deaktiviert folgende Parameter der XML-Datei:

- `Texture_Func`
- `Anti_Alias`

5.4.7 Berechnung des Rauscheffektes der Ausgabe

Die „Noise_Callback“ Callback stellt die Möglichkeit der abschließenden Manipulation des Datensatzes vor dem Abspeichern zur Verfügung. Sie ist definiert durch:

Listing 5.7: C-Header für Noise_Callback

```
1 // Callback zur Berechnung von Rauschen
2 void Noise_Callback(void *Tex, int Mode, int Width, int Height);
```

Aufgrund der zwei unterschiedlichen unterstützten Ausgabeformate 2.2.3 ist bei der Noise Callback ebenfalls eine Unterscheidung notwendig. Diese Funktion bewerkstelligt der Parameter *Mode*. Die Tabelle 5.1 veranschaulicht seine Bedeutung. *Tex* zeigt auf das ers-

Tabelle 5.1: Übersicht des Mode Parameters der Rausch Callback

Mode	Bedeutung
0	Ein Pixel besteht aus 3 * 1 Byte RGB Daten (PNG Ausgabe)
1	Ein Pixel besteht aus 4 * 4 Byte RGBA Daten (32 - Bit Fließkomma)

te Pixel, der nachfolgende Speicher ist für beide Modi wie in Abbildung 5.10 aufgebaut. Das Setzen der Noise Callback deaktiviert den Parameter „Noise_Func“ aus der XML-Datei.

Da die Noise Callback den letzten Zugriff auf den Datensatz darstellt, kann sie auch als Nachbearbeitungsstufe umfunktioniert werden. Nach der Noise Callback werden die Daten vom Simulator auf die Festplatte gespeichert.

6 Validierung des Simulators

In diesem Kapitel sollen die Ausgaben mit echten Ausgabedaten von real aufgebauten Szenarien verglichen werden. Abbildung 6.1 zeigt die Aufnahme einer echten Szene, während Abbildung 6.2 eine Annäherung durch den Simulator an Abbildung 6.1 dar-

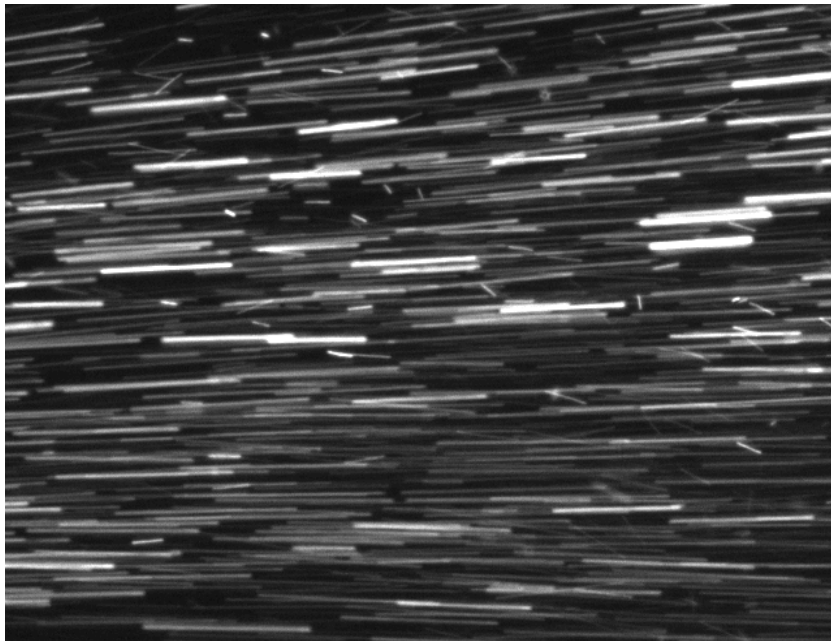


Abbildung 6.1: Aufnahme einer realen Szene

stellt. Für die Berechnung der Simulation wurden die Kameraeffekte :

- telezentrische Kamera
- begrenzte Auflösung
- Schärfentiefe

Die Partikel wurden mit Hilfe zweier Partikelkanonen erzeugt. Die erste Partikelkanone erzeugt den Hauptstrahl, mit Erzeugungsfeld links des Kamerasichtfeldes. Die zweite Partikelkanone erzeugt die Stör-/Rauschpartikel, deren Erzeugungsfeld sich über das gesamte Sichtfeld der Kamera erstreckt.

Der Vergleich der beiden Abbildungen 6.1 und 6.2 zeigt, dass die von der Aufgabenstellung geforderte möglichst realistische Darstellung der Szene erreicht werden konnte. Durch die Verwendung von zwei Partikelkanonen können die Störpartikel beliebig erhöht werden. Da der Simulator keinerlei Einschränkungen in der Anzahl der Partikel vorgibt, sind noch weitere Störgrößen realisierbar. Die für Abbildung 6.2 erzeugte Bildsequenz

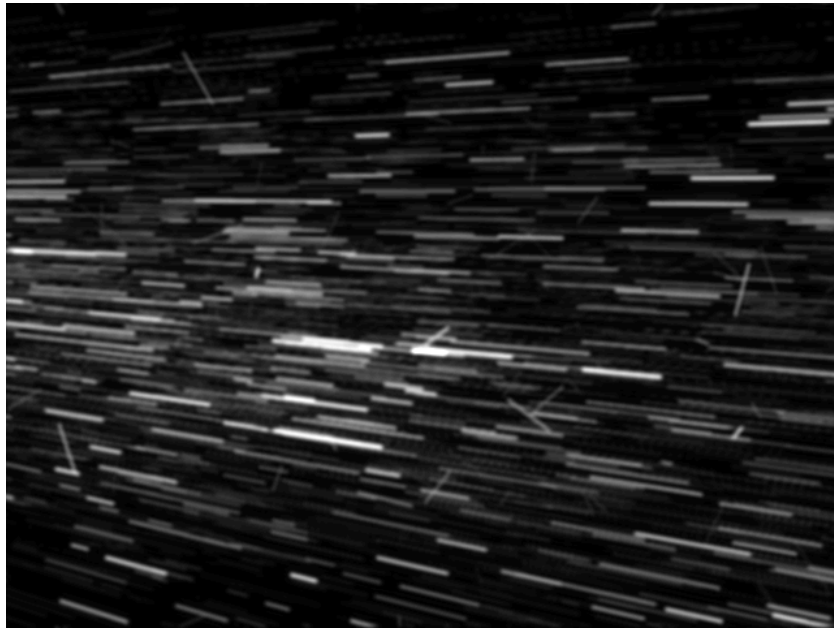


Abbildung 6.2: Simulierte Szene mit zwei Partikelkanonen

bestand aus acht Aufnahmen, welche einen Zeitraum von 710 Zeitschritten abdecken. Abbildung 6.3 zeigt die Ausgabe der telezentrischen Kamera, welche alle 100 Zeitschrit-

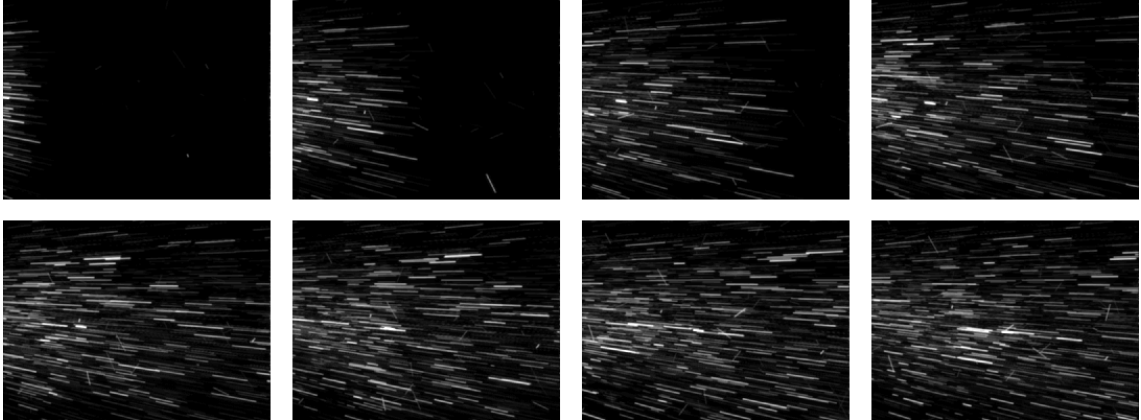


Abbildung 6.3: Aufnahmen alle 100 Zeitschritte einer Simulation

te eine Aufnahme von 50 Zeitschritten gemacht hat. Die Gesamtzeit für die Berechnung der Simulation betrug auf einem handelsüblichen PC (Intel Core Duo 2.00 GHZ) unter Linux Kubuntu 9.10 etwa 30 Sekunden.

7 Zusammenfassung und Ausblick

Im Rahmen dieser Arbeit wurde ein Konzept zur Erzeugung von simulierten Bilddaten entworfen. Diese Bilddaten zeigen den Flug von Partikeln einer virtuellen, modellierten Umgebung. Um den Nutzen der Software sicherzustellen, fanden Analyse und Entwurf anhand einer in der Praxis relevanten Szenerie statt. Neben der Analyse von Kamerasystemen und deren Aufbau, sowie von Partikeln in Gasen wurde auch die Simulation von Flüssigkeiten ermöglicht. Besonderes Augenmerk wurde dabei auf die Umsetzung der Kameraeffekte und die größtmögliche Flexibilität der Implementierung gelegt.

Die programmiertechnische Realisierung fand in der Entwicklungsumgebung Lazarus statt. Die Visualisierung wurde unter Zuhilfenahme der freien Graphik-Bibliothek Mesa 3D [16], welche auf dem OpenGL-Standard [15] aufbaut, realisiert. Ergänzt wird das Programm durch das Callback-Prinzip, welches eine einfache Erweiterungsmöglichkeit der Software im Nachhinein ermöglicht. Das Callback-Prinzip wurde trotz der Implementierung des Simulators in der Sprache Free Pascal für eine spätere Umsetzung durch C-Code angepasst. Die Konfiguration mittels XML-Datei ermöglicht eine maximale Konfigurierbarkeit bei bestmöglicher Übersicht über die Vielfalt der zur Verfügung stehenden Parameter.

Bei der Erprobung der verschiedenen Ansätze zum Speichermanagement und der Visualisierung von Partikeln konnten viele neue Erfahrungen zum Umgang mit großen Datenmengen, zum Beispiel der dynamischen Speicherverwaltung, und den betroffenen physikalischen Effekten wie beispielsweise der Kameraeffekte gewonnen werden.

Darüber hinaus gibt es Punkte, die noch verbessert werden könnten. So könnte der Kameraeffekt der Verzerrung durch eine wellenförmige Art und Weise erweitert werden. Der Rauscheffekt könnte durch Effekte wie Bereichsrauschen und Lensflare erweitert werden. Diese Erweiterungen ließen sich mit Hilfe des Callback-Prinzips umsetzen.

Das Callback-Prinzip ermöglicht auch eine Erweiterung der Simulation der Partikel. So ist es mit der Zuhilfenahme des Callback-Prinzips auf einfache Art und Weise möglich, auf die einzelnen Partikeleigenschaften Einfluss zu nehmen. Eine Realisierung von farbigen Partikeln oder in der Lebenszeit begrenzten Partikeln ist ebenso einfach realisierbar, wie das Einbeziehen von zum Beispiel Flugbahnänderungen durch Wind, Schwerkraft oder Magnetfeldern. Ebenfalls ist eine Interaktion der Partikel untereinander realisierbar. Mit Hilfe der MoveAllPartikel Callback sind sowohl Partikel-Kollisionen als auch Partikel-Verschmelzungen ebenfalls möglich.

Da die komplette Implementierung auf eine möglichst hohe Flexibilität abzielte, konnten nicht alle prinzipiell in Betracht kommenden Laufzeit-Optimierungen umgesetzt werden. Eine Verbesserung der Umsetzung einer einzelnen Aufnahme einer Kamera ohne die redundante Berechnung der Einzelbewegung der abgebildeten Pixel könnte zu deutlich geringeren Laufzeiten führen. Ebenso wäre eine parallele Berechnung der Partikelbewegungen und Partikeltexturen denkbar.

8 Anhang

In diesem Kapitel sollen die in 2.2.1 eingeführten Formeln und deren Verwendung genauer beschrieben werden. Abschließend werden zwei Beispielkonfigurationen dargestellt.

8.1 Der Plotter

Der Plotter dient nicht nur der Veranschaulichung der verwendeten Formel. Zusätzlich stellt er auch deren Validierungsinstrument dar. Eine vom Plotter akzeptierte Formel wird auch in der Initialisierungsphase des Simulators 5.1 akzeptiert, oder entsprechend abgelehnt. Abbildung 8.1 zeigt die Verwendung des Plotters bei der Validierung einer

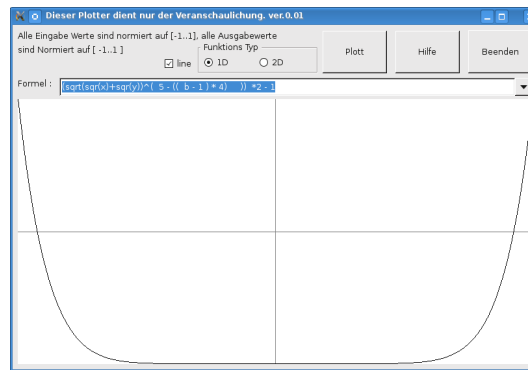


Abbildung 8.1: Der Plotter, Darstellung einer Formel eindimensional

Partikeltextur-Funktion im eindimensionalen Modus und Abbildung 8.2 dieselbe Formel im Bild-Modus.

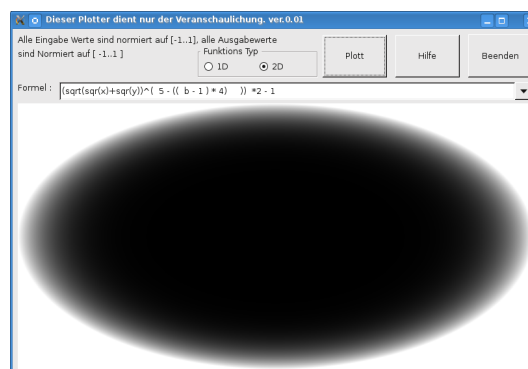


Abbildung 8.2: Der Plotter, Darstellung einer Formel zweidimensional

Um die höhere Flexibilität des Simulators gewährleisten zu können, wurde das Kon-

zept der Formeln in 2.2.1 eingeführt. Das Formel Konzept stützt hierbei auf zwei sich ergänzenden Aspekten.

1. Mathematische Grundlagen
2. Normierte Darstellung

8.1.1 Mathematische Grundlagen

Zur Beschreibung von Funktionen werden mathematische Operatoren benötigt. Ein Operator wird definiert durch sein Symbol, seine Bedeutung und seine Bindung. Im Folgenden sind alle vom Simulator unterstützten Operatoren entsprechend ihrer Bindungsstärke aufgelistet. Der „^“-Operator bindet am stärksten, das binäre „-“ am schwächsten. Zu beachten ist, dass es **kein** unäres „-“ gibt. Soll eine negative Zahl gebildet werden

Tabelle 8.1: Übersicht Mathematische Operatoren und ihre Bindungsstärke

Operator	Bezeichnung	Bindungstyp
^	Potenz	binär
sin	Sinus im Bogenmaß	unär, linksbündig
cos	Cosinus im Bogenmaß	unär, linksbündig
sqr	Quadrieren	unär, linksbündig
sqrt	Wurzel ziehen, negative Wurzeln werden auf 0 abgebildet	unär, linksbündig
abs	Betrag vom Argument	unär, linksbündig
min	liefert den kleineren Wert zurück	binär
max	liefert den größeren Wert zurück	binär
*	Multiplikation	binär
/	Division, beim Teilen durch 0 wird 0 zurückgegeben	binär
+	Addition	binär
-	Subtraktion	binär

so kann diese mittels der Schreibweise $0 - X$ erreicht werden.

Die beiden in 2.2.1 eingeführten Wahrscheinlichkeitsmodelle sind mittels der Variablen *rnd* für die Gleichverteilung und *rndg* für die Gauß-Verteilung verfügbar. Die Varia-

Tabelle 8.2: Übersicht der Wahrscheinlichkeitsmodelle

Variable	Bedeutung
<i>rnd</i>	Liefert eine Gleich verteilte Zufallszahl im Intervall $[0..1[$
<i>rndg</i>	Liefert eine Gauß verteilte Zufallszahl um die 0

blen rnd und $rndg$ dürfen pro Formel nur ein einziges Mal verwendet werden. Bei mehrfacher Verwendung haben die Werte rnd und $rndg$ stets dieselben Werte. Es gilt $rnd * rnd = rnd^2$, ebenso wie $rndg * rndg = rndg^2$.

8.1.2 Normierte Darstellung

Alle Parameter der Funktionen, welche aus vorher bekannten Definitionsbereichen stammen, oder in bekannte Definitionsbereiche abgebildet werden, werden durch eine normierte Darstellung repräsentiert. Im Detail sind dies bei den Eingabeparametern: alle dimensionsabhängigen Angaben wie Breite, Höhe und Richtung. Auf der Ausgabeseite: alle Farbwerte und Dimensionsangaben.

Die normierte Darstellung beschränkt sich stets auf das Intervall $[-1..1]$. Bei Dimensionsangaben steht die -1 für die linke Grenze, entsprechend die 1 für die rechte Grenze der Dimension. Bei Farbwerten wird der Wert -1 der Farbe Schwarz und der Wert 1 der Farbe Weiß gleich gestellt.

8.2 Der Info Viewer

Mit Hilfe des Info Viewers lassen sich die 32-Bit Info-Dateien in Verbindung mit entweder den 8-Bit reduzierten PNG-Dateien, oder den 32-Bit Repräsentationen des Rendering Contextes anzeigen. Durch die 32-Bit Info-Dateien ist eine Zuordnung eines Pixels im Bild zu allen Partikeln und deren Eigenschaften im Nachhinein möglich. Mit Hilfe des Info Viewers lassen sich auf diese Weise die nachfolgenden Analysestufen validieren. Abbildung 8.3 zeigt die Anwendung des Info Viewers, zu einer gegebenen Pixel-

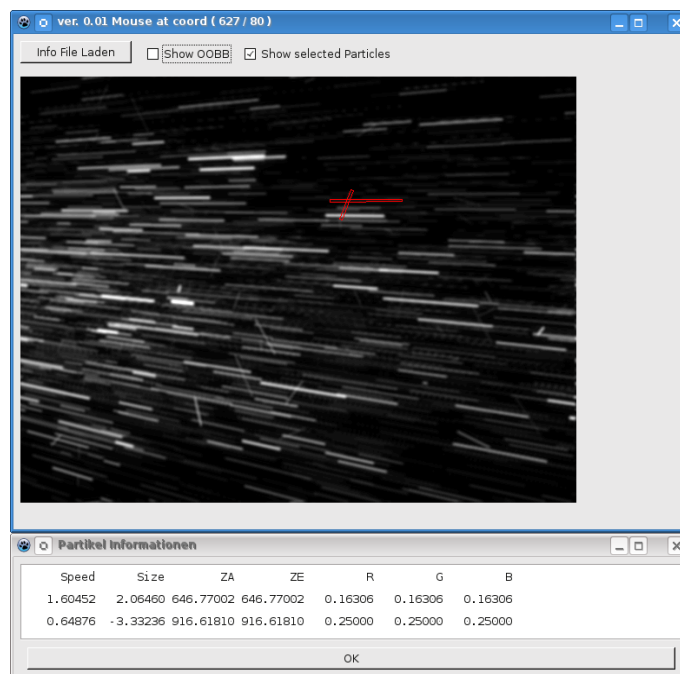


Abbildung 8.3: Der Info Viewer, Darstellung eines Datensatzes

Koordinate werden alle Partikel, sortiert nach Abstand zur Kamera, ausgegeben. Die Partikelinformationen bestehen aus:

- Speed = Geschwindigkeit des Partikels in $\frac{\text{Längeneinheiten}}{\text{Zeitschritt}}$
- Size = Durchmesser des Partikels in Längeneinheiten
- Za = Abstand zur Kamera in Längeneinheiten bei Beginn der Aufnahme
- Ze = Abstand zur Kamera in Längeneinheiten beim Ende der Aufnahme
- R, G, B = Helligkeitswerte des Partikels in der jeweiligen Farbkomponente

Der Info Viewer zeigt die bereits nachbearbeiteten Daten der Info-Datensätze an. Ein Info-Datensatz enthält folgende Informationen:

- Startposition
Position in Pixelkoordinaten
 - Z-Abstand
der Start Position in Längeneinheiten
- Endposition
Position in Pixelkoordinaten
 - Z-Abstand
der Start Position in Längeneinheiten
- Geschwindigkeit
Darstellung als Vektor (drei Komponenten) in $\frac{\text{Längeneinheiten}}{\text{Zeitschritt}}$
- Größe
 - echte Größe in Längeneinheiten
 - Höhe in Pixeln zu Beginn der Aufnahme
 - Höhe in Pixeln am Ende der Aufnahme
- Helligkeit
Helligkeit der drei Farbkomponenten des Partikels

Alle Werte sind als 32-Bit Fließkommazahlen abgespeichert. Die Info-Datei beinhaltet die reinen Info-Daten, alle Datenblöcke sind gleich groß ($15 * 4 = 60$ Byte), welche ohne weitere Angaben gespeichert werden. Die Anzahl der abgebildeten Partikel ergibt sich somit aus der Dateigröße in Byte dividiert durch 60.

8.3 Aufbau der Konfigurationsdatei

Die Konfigurationsdatei ist die Schnittstelle, mit der der Nutzer einen Einfluss auf die Simulation hat. In der Konfigurationsdatei werden alle Programmparameter gesetzt. Um Sprachbarrieren zu umgehen, ist das komplette Dokument in englischer Sprache gehalten. Durch das verwendete XML-Format ist eine hierarchische Gestaltung der Konfigurationsdatei möglich. Es ist untergliedert in drei Bereiche:

- Die Allgemeinen Einstellungen (General)
- Die Kameraeinstellungen (Cameras)
- Die Partikelkanonen-Einstellungen (Particle Guns)

Abweichend vom XML-Standard schreibt der Simulator eine exakte Reihenfolge der spezifizierten Parameter fest. Ist die Konfigurationsdatei ungültig und/oder fehlen einzelne Felder, gibt der Simulator beim Laden eine geeignete Fehlermeldung in der Konsole aus.

8.3.1 Die Allgemeinen Einstellungen (General)

Die Allgemeinen Einstellungen sind Einstellungen, welche sich global auf die Simulation auswirken und nichts mit einzelnen Elementen der Simulation zu tun haben.

- Version
Jede Konfigurationsdatei besitzt eine Versionsnummer, diese ist zur Verifizierung der Konfigurationsdatei gegenüber dem Simulator gedacht, und soll verhindern, dass ein veralteter Simulator eine zu neue Konfigurationsdatei lädt.
- Auto Save
Der Auto Save Bereich verwaltet die Einstellungen für die automatische Speicherung der gesamten Szene 2.2.5.
 - Dir
Legt das Ausgabeverzeichnis für die automatische Speicherung fest.
 - Delta
Gibt die Anzahl in Minuten an, nach der eine automatische Speicherung stattfinden soll. Der Dateiname der Speicherung ist der dann jeweils herrschende Zeitschritt.
 - Save At Tick
Gibt die Möglichkeit, die Szene an einem bestimmten Zeitschritt zu speichern.
Der Wert -1 deaktiviert diese Option.
- Live Range
Gibt den Lebensbereich der Partikel an. Verlässt ein Partikel diesen Bereich, wird es gelöscht 2.2.3 und 5.1. Der Lebensbereich wird mittels zweier Vektoren (Top_Left und Bottom_Right) angegeben.
- Seed
Gibt den Initialwert des Zufallsgenerators an 2.2.5 und 5.1
- Particle Block Size
Legt die Anzahl der Partikel pro allokiertem Speicherblock fest 2.2.4 und 5.1
- Callback File
Gibt den Dateinamen der Library für die Callbacks 5.4 an. Ist dieser Dateiname ungültig, werden alle Callback-Einträge ignoriert und entsprechende Warnungen erzeugt. Alle Callback-Parameter enden mit dem Suffix Callback .

- Custom Bytes Callback
Gibt den Namen der Callback für die Festlegung der Custom Bytes 5.4.1, 5.4.3 an
ACHTUNG!!
Für alle Callback-Namen ist die Groß-/Kleinschreibung zu beachten.
- Move all Particle Callback
Die Callback zur Bewegung aller Partikel außerhalb einer Kameraaufnahme 5.4.5.

8.3.2 Die Kameraeinstellungen (Cameras)

In den Kameraeinstellungen sind alle in der Simulation existierenden Kameras aufgelistet. Dabei müssen nicht alle gelisteten Kameras auch benutzt werden. Sind die drei Werte “PNG Active”, “Data Active” und “Particle Info” auf “False” gesetzt, oder liegt die “End Time” vor der “Start Time” wird die entsprechende Kamera als ausgeschaltet gewertet und in der Simulation nicht berücksichtigt. Alle von einer Kamera erzeugten Daten werden in dem für sie spezifizierten Ausgabeverzeichnis abgelegt. Jede Datei besteht aus dem aktuellen Zeitschritt, sowie der entsprechenden Dateierweiterung des jeweilig abgespeicherten Datensatzes.

- PNG Active
Gibt an, ob die Kamera auf 8-Bit pro Farbkanal reduzierte Daten in das Ausgabeverzeichnis abspeichern soll, oder nicht 2.2.3, 3, 5.2.2. Die Dateierweiterung lautet „*.png“.
- Data Active
Gibt an, ob ein 32-Bit pro Farbkanal Datensatz abgespeichert werden soll 2.2.3, 3, 5.2.2. Die Dateierweiterung lautet „*.dat“.
- Particle Info
Gibt an, ob die pixelspezifischen Zusatzinformationen abgespeichert werden sollen 2.2.3, 3, 5.2.2. Die Dateierweiterung lautet „*.info“.
- Mode
Legt fest, ob es sich um eine telezentrische oder perspektivische Kamera handelt 2.2.2.
- Name
Jede Kamera kann mit einem Namen versehen werden, diese Einstellung ist Optional und hat keinerlei Auswirkungen auf die Simulation.
- Out Width
Gibt die Ausgabebreite der Bilder der Kamera an ($OutWidth * OutHeight = Megapixel$).
- Out Height
Gibt die Ausgabehöhe der Bilder der Kamera an ($OutWidth * OutHeight = Megapixel$).
- Out Filename
Legt den Ausgabeordner der Kamera an. Verweist Out Filename nicht nur auf einen Ordner, wird der Rest als Prefix für die Dateinamen verwendet.

- FFT
Für jeden Farbkanal, repräsentiert durch die Vektorkomponenten (x, y, z) , kann eine Formel zur Bearbeitung der verschiedenen Bildfrequenzen angegeben werden 2.2.2.
- Vignetting Func
Gibt die Verdunkelung durch den Vignettierungseffekt an 2.2.2.
- Aberration
Der Verzerrungseffekt 2.2.2 benötigt mehr als nur eine einfache Formel für die korrekte Darstellung. Diese sind hier aufgeführt.
 - Lens X, Lens Y
Gibt die Position der Linse relativ im Bild an.
Die Koordinate (0,0) steht für den Bild Mittelpunkt.
 - K1
Gibt den Vorfaktor des dritten Exponenten der Taylerentwicklung der Linsenverzerrung an, auch Primärverzerrung genannt (zum Deaktivieren den Wert 0 angeben).
Die Angabe erfolgt als Vektor, jede Vektorkomponente steht für eine Farbe.
 - K2
Gibt den Vorfaktor des fünften Exponenten der Taylerentwicklung der Linsenverzerrung an, auch Sekundärverzerrung genannt. Wirkt sich hauptsächlich auf die Ränder der Ausgabe aus (zum Deaktivieren den Wert 0 angeben).
Die Angabe erfolgt als Vektor, jede Vektorkomponente steht für eine Farbe.
 - Lambda X, Lambda Y
Gibt die unsymmetrische Verzerrung entlang der X-Achse, Y-Achse an (zum Deaktivieren den Wert 0 angeben).
Die Angabe erfolgt als Vektor, jede Vektorkomponente steht für eine Farbe.
 - Zoom
Gibt an, ob während der Verzerrung zusätzlich noch in das Bild hinein gezoomt werden soll (zum Deaktivieren den Wert 1 angeben)
 - Over Sampling
Gibt an, ob der Eingabedatensatz feiner abgetastet werden soll.
Ist diese Einstellung größer Null, so wird durch die in Interpolation Mode angegebene Technik ein Zwischenwert berechnet. Die Berechnung von Zwischenwerten sollte vor allem beim Zoom oder Werten von K1 und K2 größer Null angewandt werden.
 - Interpolation Mode
Es stehen zwei unterschiedliche Modi zur Verfügung.
Nearest Neighbour: Erhaltung von scharfen Kanten, Erhaltung der

Farbwerte

Biliniar: Berechnet einen geglätteten Farbverlauf und dämpft damit hohe Frequenzen im Bild.

- Noise Func
Für jeden Farbkanal, repräsentiert durch die Vektorkomponenten (x, y, z) , kann eine Formel zur Bearbeitung der des Rauschens angegeben werden.
- Noise Callback
Umsetzung des Rauscheffektes mittels Callback 5.4.7, kann auch zu anderen Nachbearbeitungszwecken benutzt werden.
- Z near
Legt den Abstand zur Z-Near Clipping Plane von OpenGL fest 5.1.
- Z far
Legt den Abstand zur Z-Far Clipping Plane von OpenGL fest 5.1.
- Focus
Legt den Punkt fest, an welchem alles scharf abgebildet wird 2.2.2.
- Unsharp Func
Formel zur Berechnung der Unschärfe durch den Abstand zum Fokus 2.2.2.
- Bright Func
Formel zur Berechnung der Helligkeitsschwankung durch den Abstand zum Fokus 2.2.2.
- Delta Shut
Legt die Zeitschritte fest, die vergeht, bis eine neue Aufnahme gestartet wird 3.1.
- Exposure
Legt die Anzahl der Zeitschritte fest, wie lange eine Aufnahme geht 2.2.2, 3.1.
- Start Time
Legt den ersten Zeitschritt fest, ab welchem die Kamera mit der Aufnahme beginnt 3.1.
- End Time
Legt den letztmöglichen Zeitschritt fest, zu welchem eine Aufnahme beginnt 3.1.
- Position
Legt die Position der Kamera im Raum fest 2.1. Die Position wird als Vektor angegeben.
- Dir
Legt die Richtung fest, in welche die Kamera blickt 2.1. Die Richtung wird als Vektor angegeben.
- Up
Legt fest, wo oben für die Kamera ist 2.1. Der Oben Vektor wird als Vektor angegeben.

- telezentric
Die hier stehenden Parameter beziehen sich speziell auf telezentrische Linsensysteme 2.2.2.
 - In Width
Legt die Breite der aufgenommenen Szene in Längeneinheiten fest.
 - In Height
Legt die Höhe der aufgenommenen Szene in Längeneinheiten fest.
- perspective
Die hier stehenden Parameter beziehen sich speziell auf perspektivische Linsensysteme 2.2.2.
 - aspect ratio
Gibt das Seitenverhältnis zwischen Breite und Höhe an.
 - Fovy
Gibt den Öffnungswinkel des Objektivs an.

8.3.3 Die Partikelkanonen-Einstellungen (Particle Guns)

Die Partikelkanonen fassen gemäß 2.2.1 alle Eigenschaften zur Erzeugung und Bewegung von Partikeln getrennt nach ihrem Typ zusammen. Für jeden Partikeltyp stehen folgende Attribute zur Verfügung.

- Name
Gibt den Namen des Partikels an, diese Einstellung ist optional und hat keinerlei Auswirkungen auf die Simulation.
- Field
Gibt den zu den Hauptachsen parallelen Quader im Raum an, in welchem die Partikel erzeugt werden. Das Erzeugungsfeld wird mittels zweier Vektoren (Top_Left und Bottom_Right) angegeben.
- ProbabilityX, ProbabilityY, ProbabilityZ
Mit den Probability Parametern kann die genaue Position des Partikels bei der Erzeugung im Field festgelegt werden. Der Wert 0 entspricht genau dem Mittelpunkt der jeweiligen Komponente 2.2.1 im Field.
- Size Func
Mit Hilfe der Size Func wird der Durchmesser der Partikel bei der Erzeugung berechnet. Diese Berechnung ist optional von der Position abhängig 2.2.1, 3.1.
- Dir FuncX, Dir FuncY, Dir FuncZ
Mit Hilfe der Dir Func wird die Richtung, in welche sich der Partikel bewegen wird, festgelegt. Diese Berechnung ist optional von der Position und der Größe abhängig 2.2.1, 3.1.
- Speed Func
Gibt die Geschwindigkeit in Abhängigkeit der Position, Größe und Richtung an 2.2.1, 3.1.

- ID
Die ID dient der simulationsinternen Identifikation der Partikel, auch bei den Callbacks. Die ID muss stets ein Wert größer Null sein.
- Texture Func
Mit Hilfe der Texture Func wird die dynamisch erzeugte Textur für das zu rendernde Partikel erzeugt 5.2.2.
- Anti Alias
Der Anti Alias Faktor legt eine künstliche Vergrößerung der Texture Func fest, durch diese Einstellung werden Alias-Effekte beim Rendern der Partikel minimiert.
- Brightness Func
Berechnet die Helligkeit des Partikels in Abhängigkeit der Position, Größe, Richtung und Geschwindigkeit 2.2.1, 3.1.
- Count Per Tick
Legt fest, wie viele Partikel pro Zeitschritt erzeugt werden sollen 2.2.1, 3.1.
- Create Particle Callback
Erzeugen der Partikel mittels Callback, deaktiviert „Count Per Tick“ und alle Partikelerzeugungs-Parameter 5.4.2 .
- Set Custom Bytes Callback
Setzen der Custom Bytes nach der Erzeugung der Callbacks, nur aktiv, wenn „Create Particle Callback“ nicht gesetzt wurde 5.4.3.
- Move Particle Callback
Bewegt ein einziges Partikel beim Rendering. Wenn „Move all Particle Callback“ nicht gesetzt ist auch außerhalb 5.4.4 des Renderings.
- Create Texture Callback
Erzeugt die Partikeltextur, welche in der Billboard-Ebene dargestellt wird 5.4.6.

8.4 Konfigurationsbeispiele

Das nachfolgende Listing zeigt die Konfiguration zur Erzeugung der Abbildung 6.2 sowie 6.3. Abbildung 6.2 entspricht der Aufnahme an Zeitschritt 710.

Listing 8.1: Konfiguration der XML-Datei für Abbildung 6.2 und 6.3

```
1 <Scene>
2   <General>
3     <Version> 0.01 </Version>
4     <Auto_Save>
5       <Dir> "/home/username/outputdir/" </Dir>
6       <Delta> 60 </Delta>
7       <Save_At_Tick> -1 </Save_At_Tick>
8     </Auto_Save>
9     <Live_Range>
10    <top_left>
```

```

11     <x> 0 </x>
12     <y> 0 </y>
13     <z> -90 </z>
14 </top_left>
15 <bottom_right>
16     <x> 700 </x>
17     <y> 480 </y>
18     <z> 900 </z>
19 </bottom_right>
20 </Live_Range>
21 <Seed> 0 </Seed>
22 <Particle_Block_Size> 1000 </Particle_Block_Size>
23 <Callback_File> "" </Callback_File>
24 <Custom_Bytes_Callback> "" </Custom_Bytes_Callback>
25 <Move_All_Particle_Callback> "" </Move_All_Particle_Callback>
26 </General>
27 <Cameras>
28     <count> 1 </count>
29     <camera>
30         <PNG_Active> true </PNG_Active>
31         <Data_Active> false </Data_Active>
32         <Particle_Info> false </Particle_Info>
33         <Mode> telezentric </Mode>
34         <Name> "Hauptkamera" </Name>
35         <Out_Width> 640 </Out_Width>
36         <Out_Height> 480 </Out_Height>
37         <Out_Filename> "/home/username/outputdir/" </Out_Filename>
38         <FFT>
39             <x> "(sqrt(sqr(x)+sqr(y))*(0-2)^(+0))^(max(0,1))" </x>
40             <y> "(sqrt(sqr(x)+sqr(y))*(0-2)^(+0))^(max(0,1))" </y>
41             <z> "(sqrt(sqr(x)+sqr(y))*(0-2)^(+0))^(max(0,1))" </z>
42         </FFT>
43         <Vignetting_Func> "1" </Vignetting_Func>
44         <Aberration>
45             <Lens_X> 0 </Lens_X>
46             <Lens_Y> 0 </Lens_Y>
47             <K1>
48                 <x> 0 </x>
49                 <y> 0 </y>
50                 <z> 0 </z>
51             </K1>
52             <K2>
53                 <x> 0 </x>
54                 <y> 0 </y>
55                 <z> 0 </z>
56             </K2>
57         <Lambda_X>

```

```

58     <x> 0 </x>
59     <y> 0 </y>
60     <z> 0 </z>
61 </Lambda_X>
62 <Lambda_Y>
63     <x> 0 </x>
64     <y> 0 </y>
65     <z> 0 </z>
66 </Lambda_Y>
67 <Zoom> 1 </Zoom>
68 <Over_Sampling> 1 </Over_Sampling>
69 <Interpolation_Mode> bilinear </Interpolation_Mode>
70 </Aberration>
71 <Noise_Func>
72     <x> " _z_" </x>
73     <y> " _a_" </y>
74     <z> " _b_" </z>
75 </Noise_Func>
76 <Noise_Callback> "" </Noise_Callback>
77 <Z_near> 10 </Z_near>
78 <Z_far> 1000 </Z_far>
79 <Focus> 500 </Focus>
80 <Unsharp_Func> "(1+_+(z_/_a)_*_2)_max_(0_--_0.9)" </Unsharp_Func>
81 <Bright_Func> "(1+_+(z_/_a)_*_2)_max_(0_--_0.9)" </Bright_Func>
82 <Delta_Shut> 100 </Delta_Shut>
83 <Exposure> 50 </Exposure>
84 <Start_Time> 10 </Start_Time>
85 <End_Time> 710 </End_Time>
86 <Position>
87     <x> 320 </x>
88     <y> 240 </y>
89     <z> -100 </z>
90 </Position>
91 <Dir>
92     <x> 0.0 </x>
93     <y> 0.0 </y>
94     <z> 1.0 </z>
95 </Dir>
96 <Up>
97     <x> 0.0 </x>
98     <y> 1.0 </y>
99     <z> 0.0 </z>
100 </Up>
101 <telezentric>
102     <In_Width> 640 </In_Width>
103     <In_Height> 480 </In_Height>
104 </telezentric>

```

```

105     <perspective >
106         <aspect_ratio > 1.33 </aspect_ratio >
107         <Fovy > 90 </Fovy >
108     </perspective >
109 </camera >
110 </Cameras >
111 <particle_guns >
112     <count > 2 </count >
113     <particle_gun >
114         <Name > "Streustrahlen" </Name >
115         <Field >
116             <top_left >
117                 <x > 0 </x >
118                 <y > 0 </y >
119                 <z > -90 </z >
120             </top_left >
121             <buttom_right >
122                 <x > 700 </x >
123                 <y > 480 </y >
124                 <z > 900 </z >
125             </buttom_right >
126         </Field >
127         <ProbabilityX > "rndg" </ProbabilityX >
128         <ProbabilityY > "rndg_*_0.4" </ProbabilityY >
129         <ProbabilityZ > "0.5*_+(rndg_*_0.2)" </ProbabilityZ >
130         <Size_Func > "(((rndg_*_0.1)+1)*_3)_max_5" </Size_Func >
131         <Dir_FuncX > "rndg" </Dir_FuncX >
132         <Dir_FuncY > "rndg" </Dir_FuncY >
133         <Dir_FuncZ > "0" </Dir_FuncZ >
134         <Speed_Func > "(rnd+0.0125)" </Speed_Func >
135         <ID > 2 </ID >
136         <Texture_Func > "sqrt(_sqr(x)_+_sqr(y))^1*2-1" </Texture_Func >
137         <Anti_Alias > 3 </Anti_Alias >
138         <Brightness_Func >
139             "_((rndg_*_0.1*_z*_1.5)_max_(0_--_0.5))"
140         </Brightness_Func >
141         <Count_Per_Tick > 0.1875 </Count_Per_Tick >
142         <Create_Particle_Callback > "" </Create_Particle_Callback >
143         <Set_Custom_Bytes_Callback > "" </Set_Custom_Bytes_Callback >
144         <Move_Particle_Callback > "" </Move_Particle_Callback >
145         <Create_Texture_Callback > "" </Create_Texture_Callback >
146     </particle_gun >
147 </particle_gun >
148     <Name > "Kanone1" </Name >
149     <Field >
150         <Top_Left >
151             <x > 0 </x >

```

```

152     <y> 0 </y>
153     <z> -90 </z>
154   </Top_Left>
155   <Buttom_Right>
156     <x> 0 </x>
157     <y> 480 </y>
158     <z> 900 </z>
159   </Buttom_Right>
160 </Field>
161 <ProbabilityX> "rndg" </ProbabilityX>
162 <ProbabilityY> "rndg_*_0.4" </ProbabilityY>
163 <ProbabilityZ> "0.5+_+(rndg_*_0.2)" </ProbabilityZ>
164 <Size_Func> "(((rndg_*_0.1)+1)*_3)_max_5" </Size_Func>
165 <Dir_FuncX> "1" </Dir_FuncX>
166 <Dir_FuncY> "(y*0.25)_-_0.1" </Dir_FuncY>
167 <Dir_FuncZ> "0" </Dir_FuncZ>
168 <Speed_Func> "(rnd+0.1)*_2" </Speed_Func>
169 <ID> 1 </ID>
170 <Texture_Func> "sqrt(_sqrt(x)_+_sqrt(y))^1*2-1" </Texture_Func>
171 <Anti_Alias> 3 </Anti_Alias>
172 <Brightness_Func>
173 " (rndg_*_0.1_+_z_*_1.5)_max_(0_+_0.75)"
174 </Brightness_Func>
175 <Count_Per_Tick> 4 </Count_Per_Tick>
176 <Create_Particle_Callback> "" </Create_Particle_Callback>
177 <Set_Custom_Bytes_Callback> "" </Set_Custom_Bytes_Callback>
178 <Move_Particle_Callback> "" </Move_Particle_Callback>
179 <Create_Texture_Callback> "" </Create_Texture_Callback>
180 </particle_gun>
181 </particle_guns>
182 </Scene>

```

Das nachfolgende Listing zeigt die Konfiguration zur Erzeugung der Abbildung 2.19 an Zeitschritt 710.

Listing 8.2: Konfiguration der XML-Datei für Abbildung 2.19

```

1 <Scene>
2   <General>
3     <Version> 0.01 </Version>
4     <Auto_Save>
5       <Dir> "/home/username/outputdir/" </Dir>
6       <Delta> 60 </Delta>
7       <Save_At_Tick> -1 </Save_At_Tick>
8     </Auto_Save>
9     <Live.Range>
10      <top_left>
11        <x> 0 </x>
12        <y> 0 </y>

```

```

13     <z> -90 </z>
14 </top_left>
15 <bottom_right>
16     <x> 700 </x>
17     <y> 480 </y>
18     <z> 900 </z>
19 </bottom_right>
20 </Live_Range>
21 <Seed> 0 </Seed>
22 <Particle_Block_Size> 1000 </Particle_Block_Size>
23 <Callback_File> "" </Callback_File>
24 <Custom_Bytes_Callback> "" </Custom_Bytes_Callback>
25 <Move_All_Particle_Callback> "" </Move_All_Particle_Callback>
26 </General>
27 <Cameras>
28     <count> 1 </count>
29     <camera>
30         <PNG_Active> true </PNG_Active>
31         <Data_Active> false </Data_Active>
32         <Particle_Info> false </Particle_Info>
33         <Mode> telezentric </Mode>
34         <Name> "Hauptkamera" </Name>
35         <Out_Width> 640 </Out_Width>
36         <Out_Height> 480 </Out_Height>
37         <Out_Filename> "/home/username/outputdir/" </Out_Filename>
38         <FFT>
39             <x> "(sqrt(sqr(x)+sqr(y))*(0-2)_)max_(0_1_)" </x>
40             <y> "(sqrt(sqr(x)+sqr(y))*(0-2)_)max_(0_1_)" </y>
41             <z> "(sqrt(sqr(x)+sqr(y))*(0-2)_)max_(0_1_)" </z>
42         </FFT>
43         <Vignetting_Func> "1" </Vignetting_Func>
44         <Aberration>
45             <Lens_X> 0 </Lens_X>
46             <Lens_Y> 0 </Lens_Y>
47             <K1>
48                 <x> 0 </x>
49                 <y> 0 </y>
50                 <z> 0 </z>
51             </K1>
52             <K2>
53                 <x> 0 </x>
54                 <y> 0 </y>
55                 <z> 0 </z>
56             </K2>
57             <Lambda_X>
58                 <x> 0 </x>
59                 <y> 0 </y>

```

```

60     <z> 0 </z>
61 </Lambda_X>
62 <Lambda_Y>
63     <x> 0 </x>
64     <y> 0 </y>
65     <z> 0 </z>
66 </Lambda_Y>
67 <Zoom> 1 </Zoom>
68 <Over_Sampling> 1 </Over_Sampling>
69 <Interpolation_Mode> bilinear </Interpolation_Mode>
70 </Aberration>
71 <Noise_Func>
72     <x> " _z_" </x>
73     <y> " _a_" </y>
74     <z> " _b_" </z>
75 </Noise_Func>
76 <Noise_Callback> "" </Noise_Callback>
77 <Z_near> 10 </Z_near>
78 <Z_far> 1000 </Z_far>
79 <Focus> 500 </Focus>
80 <Unsharp_Func> "(1+_ (z/a)*2) _max_(0-0.9)" </Unsharp_Func>
81 <Bright_Func> "(1+_ (z/a)*2) _max_(0-0.9)" </Bright_Func>
82 <Delta_Shut> 100 </Delta_Shut>
83 <Exposure> 1 </Exposure>
84 <Start_Time> 610 </Start_Time>
85 <End_Time> 710 </End_Time>
86 <Position>
87     <x> 320 </x>
88     <y> 240 </y>
89     <z> -100 </z>
90 </Position>
91 <Dir>
92     <x> 0.0 </x>
93     <y> 0.0 </y>
94     <z> 1.0 </z>
95 </Dir>
96 <Up>
97     <x> 0.0 </x>
98     <y> 1.0 </y>
99     <z> 0.0 </z>
100 </Up>
101 <telezentric>
102     <In_Width> 640 </In_Width>
103     <In_Height> 480 </In_Height>
104 </telezentric>
105 <perspective>
106     <aspect_ratio> 1.33 </aspect_ratio>

```

```

107     <Fovy> 90 </Fovy>
108     </perspective>
109     </camera>
110 </Cameras>
111 <particle_guns>
112     <count> 1 </count>
113     <particle_gun>
114         <Name> "Kanone1" </Name>
115         <Field>
116             <Top_Left>
117                 <x> 0 </x>
118                 <y> 0 </y>
119                 <z> -90 </z>
120             </Top_Left>
121             <Buttom_Right>
122                 <x> 0 </x>
123                 <y> 480 </y>
124                 <z> 900 </z>
125             </Buttom_Right>
126         </Field>
127         <ProbabilityX> "rndg" </ProbabilityX>
128         <ProbabilityY> "rndg*_0.4" </ProbabilityY>
129         <ProbabilityZ> "0.5_+_(rndg*_0.2)" </ProbabilityZ>
130         <Size_Func> "(((rndg*_0.1)+6)*_5)_max_25" </Size_Func>
131         <Dir_FuncX> "1" </Dir_FuncX>
132         <Dir_FuncY> "(y*0.25)_-_0.1" </Dir_FuncY>
133         <Dir_FuncZ> "0" </Dir_FuncZ>
134         <Speed_Func> "(rnd+0.1)*_2" </Speed_Func>
135         <ID> 1 </ID>
136         <Texture_Func>
137             "(cos((x^2+_y^2)*15.8)*(_1_-(x^2+_y^2))_-(x^2+_y^2))*(0-1)"
138         </Texture_Func>
139         <Anti_Alias> 3 </Anti_Alias>
140         <Brightness_Func>
141             "(rndg*_0.1_+_z*_1.5)_max_(0_+_0.75)"
142         </Brightness_Func>
143         <Count_Per_Tick> 4 </Count_Per_Tick>
144         <Create_Particle_Callback> "" </Create_Particle_Callback>
145         <Set_Custom_Bytes_Callback> "" </Set_Custom_Bytes_Callback>
146         <Move_Particle_Callback> "" </Move_Particle_Callback>
147         <Create_Texture_Callback> "" </Create_Texture_Callback>
148     </particle_gun>
149 </particle_guns>
150 </Scene>

```

Literaturverzeichnis

- [1] Ulf Assarsson and Tomas Möller. Optimized view frustum culling algorithms for bounding boxes. *journals of graphics tools*, 2000.
- [2] James F. Blinn. Jim blinn's corner: Compositing, part i: Theory. *IEEE Computer Graphics & Applications*, pages 83–87, 1994.
- [3] G. E. P. Box and E. Muller. A note on the generation of random normal deviates. In *The Annals of Mathematical Statistics, Vol. 29, No. 2*, pages 610–611. Institute of Mathematical Statistics, 1958.
- [4] David S. Ebert, Wayne E. Carlson, and Richard E. Parent. Solid spaces and inverse particle systems for controlling the animation of gases and fluids. *The Visual Computer*, 10:179–190, 1994.
- [5] Mario Figueiredo, Josiane Zerubia, and Anil K. Jain. *Energy minimization methods in computer vision and pattern recognition*, pages 185–190. Springer-Verlag Berlin, 2001.
- [6] Matteo Frigo and Steven G. Jonson. Fftw - fastest fourier transform in the west.
- [7] Ismael Garcia, Mateu Sbert, and Laszlo Szirmay-Kalos. Tree rendering with billboard clouds. *Third Hungarian Conference on Computer Graphics and Geometry, Budapest*, 2005.
- [8] Erin J. Hastings, Ratan K. Guha, and Kenneth O. Stanley. Interactive evolution of particle systems for computer graphics and animation. *IEEE Transactions on Evolutionary Computation*, 2009.
- [9] Wolfgang Hörmann and Gerhard Derflinger. A portable uniform random number generator well suited for the rejection method. *ePub*, 1992.
- [10] Taygeta Scientific Inc. Generating gaussian random numbers, 2001.
- [11] Harold M. Merklinger. *The INs and OUTs of FOCUS*. Harold M. Merklinger, 2002.
- [12] H.H. Nasse. Wie liest man mtf-kurven ?, 2008.
- [13] Michael J. Nasse and Jörg C. Woehl. Realistic modeling of the illumination point spread function in confocal scanning optical microscopy. *Journal of the Optical Society of America A*, 27(2):295–302, February 2010.
- [14] Andrew Norton. *Dynamic fields and waves*. CRC Press, 2000.
- [15] OpenGL, Shreiner, D., M. Woo, Neider, J., and Davis, T. *OpenGL(R) Programming Guide : The Official Guide to Learning OpenGL(R), Version 1.1 (4th Edition)*. Addison-Wesley Professional, 1997.

- [16] Brian Paul. The mesa 3d graphics library, 1993.
- [17] E. Ploetz, B. Marx, and P. Gilch. Disturbing interference patterns in femtosecond stimulated raman microscopy. *Raman Spectroscopy*, 2009.
- [18] Sidney F. Ray. *Applied photographic optics: lenses and optical systems for photography*, chapter 24. Focal Press, 3 edition, 2002.
- [19] Roman R. Redziejowski. Parsing expression grammar as a primitive recursive-descent parser with backtracking. *Proceedings of CS&P'2006*, pages 304–315, 2008.
- [20] William T. Reeves. Particle systems - a technique for modeling a class of fuzzy objects. *ACM Transactions on Graphics*, pages 91–108, 1983.
- [21] B. K. Gunturk, J. Glotzbach, Y. Altunbasak, R. W. Schafer, and R. M. Mersereau. Demosaicking: color filter array interpolation. pages 44–54, 2005.
- [22] Elias Stein and Guido Weiss. *Introduction to Fourier analysis on Euclidean spaces*. Princeton, N.J.: Princeton University Press, 1971.
- [23] Dr. R. Steinbecher. *Bildverarbeitung in der Praxis*. R. Oldenbourg Verlag, 2005.
- [24] Tiow-Seng Tan. Optimal triangulation problems. *Contribution to the International Congress on Industrial & Applied Mathematics*, pages 3–7, 1995.
- [25] Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman. *Compiler Prinzipien, Techniken und Werkzeuge*. Pearson Studium, 2008.
- [26] Tamas Umenhoffer, Laszlo Szirmay-Kalos, and Gabor Szijarto. Spherical billboards and their application to rendering explosions. *Department of Control Engineering and Information Technology Budapest University of Technology, Hungary*, 137:57–63, 2006.
- [27] G. Vass and T. Perlaki. Applying and removing lens distortion in post production. *The Second Hungarian Conference on Computer Graphics and Geometry, Budapest*, 2003.
- [28] Dietmar Wueller and Prof. Dr. Bernd Jähne. Charakterisierung digitaler kamera-systeme nach iso und emva 1288. *Framos*, 2009.

Erklärung

Hiermit versichere ich, diese Arbeit selbständig verfasst und nur die angegebenen Quellen benutzt zu haben.

(Uwe Schächterle)