

Hauptseminar
Prof. Dr. Nischwitz
Volumetrische Effekte mit Voxelsystemen

Heiko Senger
Matr.-Nr.: 03447602
Studiengruppe: IGM3
heiko.senger@gmx.net

30. Juni 2009

Erklärung

gemäß § 35 Abs. 7 RaPO

Heiko Senger
geboren am 11.04.1981 in Weilheim i. Obb.

Matrikelnummer 03447602
Studiengruppe IGM3
Sommersemester 2009

Hiermit erkläre ich, dass ich die Studienarbeit selbständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel benützt, sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

München, am 30. Juni 2009

Heiko Senger

Inhaltsverzeichnis

1	Einführung	4
2	Volumentrische Effekte	4
3	Volume-Rendering	6
3.1	Volumendaten	6
3.2	Transferfunktion	7
3.3	Renderverfahren	7
4	Rauch und Feuer mit Volumenrendering	11
4.1	Simulation	11
4.1.1	Modellierung	11
4.1.2	Lösung	13
4.1.3	Berücksichtigung von Objekten	15
4.1.4	Effektspezifische Einflussgröße	16
4.2	Rendern	17
4.2.1	Vorverarbeitung	17
4.2.2	Raycasting	18
4.3	Rauch und Feuer	19
4.3.1	Weitere Problemstellungen	20
4.3.2	Artefakte	21
5	Vergleich zu Partikelsystemen	23
6	Zusammenfassung	27

1 Einführung

In dieser Arbeit wird ein Verfahren zum Rendern von volumetrischen Effekte durch Volume-Rendern auf der Grafikkarte vorgestellt. In Abschnitt 2 werden hierfür die Grundlagen vermittelt. Dazu wird zunächst kurz erklärt, was man unter volumetrischen Effekten versteht, was Volume-Rendern ist und welche Ansätze es hierfür gibt. Danach wird in Abschnitt 3 auf ein konkretes Verfahren eingegangen, das Feuer und Rauch durch Fluidynamik in einem Volumen simuliert. Zur Visualisierung des Effektes wird Raycasting verwendet. Abschließen werden volumenbasierte Ansatz und der heutige Standard, die Partikelsystem, miteinander verglichen.

Was in dieser Arbeit nicht besprochen wird, sind Schatten, Self-Shadowing oder Beleuchtung. Hierfür sei auf [AS09] und [FSJ01] verwiesen.

2 Volumetrische Effekte

Unter den Begriff der volumetrischen Effekte fallen in der Computergrafik eine breite Palette von Effekten. Hierzu zählen Effekte wie Feuer, Staub, Nebel, Wolken, Rauch, Explosionen oder auch Flüssigkeiten. Im Unterschied zu den „normalen“ Objekten in der Computergrafik, haben diese Effekte keine Oberfläche oder feste Form. Sie bestehen nicht nur aus einer leeren Hülle von Polygonen, sondern existieren, wie der Name schon sagt, über ein Volumen. Zwar ist es möglich einige dieser Effekte über animierte Texturen darzustellen wie zum Beispiel Feuer oder Wolken, jedoch bringt die Natur dieser Effekte es mit sich, dass die Kamera auch im Inneren dieser Effekte sein kann, wodurch der „Schwindel“ sofort auffliegt.

Der klassische Ansatz zu Realisierung von volumetrischen Effekten sind Partikelsysteme. Hierbei werden viele einzelne Objekte – Partikel – erzeugt die als Ganzes einen Effekt erzeugen. Ein Partikel wird dabei durch ein Objekt (z.B. Kugel), eine Textur (z.B. Feuer-textur) oder einfach nur ein Punkt mit einem Farbwert dargestellt (Abbildung 1).

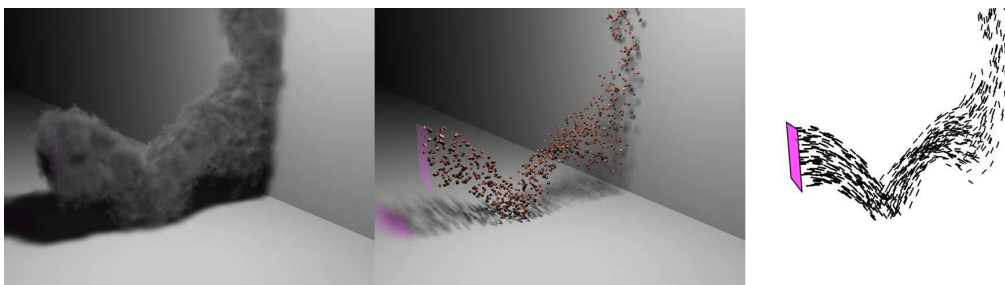


Abbildung 1: Partikelsystem: Einmal als Rauch und einmal als Würfelpartikel gerendert [Wik09b]

Ein anderer Ansatz zum Erzeugen von volumetrischen Effekten, der auch der Kern dieser Arbeit ist, basiert auf dem Volume-Rendering. Volume-Rendering ist eine Darstellungsverfahren, das sehr stark im medizinischen Umfeld im Einsatz ist [Wat00]. Hierbei werden im Gegensatz zum oberflächenbasierten Rendern nicht nur die Oberflächen eines Objektes berücksichtigt, sondern auch dessen Inneres. Dies ist besonders bei medizinischen Bildern wichtig, da oft der Querschnitt vom Objektinneren von Interesse ist (z.B. für Knochen und Organe).

Im Folgenden werden einige Effekte beschrieben, die in der Computergrafik häufig vorkommen sind. Dabei ist zu beachten, dass die Beschreibung nicht den Anspruch einer korrekten physikalischen Definition hat. Vielmehr handelt es sich hierbei um eine allgemeine Beschreibung um ein grobes Verständnis dafür zu bekommen.

Feuer Bei Feuer handelt es sich um eine chemische Verbrennung, die sich optisch durch eine Flammenbildung zeigt. Bei diesem Vorgang wird Licht und Wärme abgegeben. Notwendig für die Entstehung eines Feuers sind Sauerstoff, Brennmaterial und die davon abhängige Zündtemperatur. Diese sind auch für das grundlegende Erscheinungsbild verantwortlich, das noch durch Luftströmungen (z. B. Wind) beeinflusst werden kann.

Staub / Rauch / Wolken / Nebel Bei diesen Effekten handelt es sich um ähnliche Phänomene die unterschiedliche Ursachen haben. So ist zum Beispiel Rauch das Ergebnis eines Verbrennungsprozesses und Wolken das Ergebnis einer Verdunstung von Wasser. Allen gemein ist, dass ihr Aussehen (Farbe, Form und Bewegung) von der Dichte und Größe der Schwebeteilchen, Gasen, Temperatur und externen Faktoren wie Lichtquellen oder Krafteinwirkung (Gravitation, Wind) abhängig sind. Je nach Effekt werden bestimmte Eigenschaften stärker gewichtet um das charakteristische Aussehen zu erhalten.

Explosionen Eine Explosion ist ein plötzlicher Anstieg von Temperatur und/oder Druck, welcher eine plötzliche Volumenausdehnung von Gasen und der Freisetzung von Energie auf engen Raum zur Folge hat. Die häufigsten Explosionen in der Computergrafik sind wohl die chemischen Explosionen, bei denen es zu Flammen-, Rauch- und Staubbildung kommt.

Flüssigkeiten Hierbei handelt es sich um einen Stoff in einem flüssigen Aggregatzustand. Flüssigkeiten leisten Formveränderungen so gut wie keinen Widerstand und sind nahezu inkompressibel. Das Erscheinungsbild sind auch von Art, Größe und Dichte der Teilchen in der Flüssigkeit und von externen Faktoren wie Lichtquellen und Krafteinwirkung abhängig. Im Unterschied zu anderen Effekten verfügen Flüssigkeiten über eine Oberfläche auf der sich andere Effekte wie Reflexion oder der Fresnel-Effekt zeigen können.

3 Volume-Rendering

Wie im vorherigen Abschnitt bereits erwähnt, konzentriert sich diese Arbeit auf die Darstellung von volumetrischen Effekten durch Volume-Rendering. Unter Volume-Rendering versteht man das Visualisieren von Volumendaten. Dieser Abschnitt geht nun auf Techniken ein, mit den dies realisiert werden kann.

3.1 Volumendaten

Im Gegensatz zum oberflächenbasierten Rendern liegen die Daten nicht als Dreiecke / Polygone vor, sondern als Volumen. Diese werden durch Voxel (Volumenelement) dargestellt. Für ein Voxel gibt es zwei verschiedene Formen bzw. Betrachtungsmöglichkeiten: Einerseits kann ein Voxel als ein Abtastpunkt betrachtet werden, der sich in der Mitte eines Volumens befindet. Der Wert des Voxels ist hierbei im ganzen Volumen gleich. Eine andere Betrachtungsweise ist die der Voxelzelle. Sie wird durch acht Abtastpunkte dargestellt, die einen Würfel bilden. Der Wert an einer Position innerhalb eines Voxels ergibt sich dann durch Interpolation der Werte an den Eckpunkten. Ein Voxel kann ein oder mehrere Werte haben. Diese sind in der Regel keine Farbwerte, sondern physikalische Größen wie Dichte oder Temperatur. Diese werden über eine Transferfunktion auf einen Farb- und Transparenzwert abgebildet. Die Voxel werden in einem regulären kartesischen Gitter angeordnet, die dann den Volumendatensatz bilden. Abbildung 2 verdeutlicht nochmal die beiden Formen.

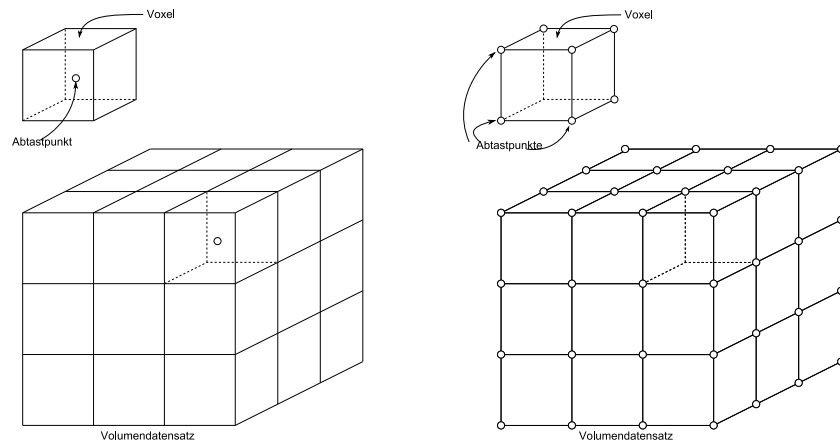


Abbildung 2: Darstellung eines Voxels als Punkt (links) und als Zelle (rechts)

Grundsätzlich kann ein Voxel auch ein Tetraeder und der Datensatz kann auch irregulär sein.

3.2 Transferfunktion

Voxel beinhalten in der Regel keine Farbwerte, sondern physikalische Größen wie beispielsweise Dichte, Temperatur, usw.. Wird nun ein Farbwert benötigt, so muss dieser aus diesen Größen abgeleitet werden. Im einfachsten Fall geschieht dies mit einer Lookup-Table. Hierbei wird der Voxelwert auf einen Farbwert abgebildet. Typischerweise wird dies mit einer eindimensionalen Textur realisiert. Abbildung 3 zeigt dies anhand einer Farbtemperaturskala. Dabei erhält Temperaturwert von 4000 Kelvin die Farbe weiß. Hierbei ist zu beachten, dass dies nicht der menschlichen Farbwahrnehmung entspricht, da dieser die Farbe weiß ab etwa 1500 Kelvin (Weißglut) wahrnimmt.

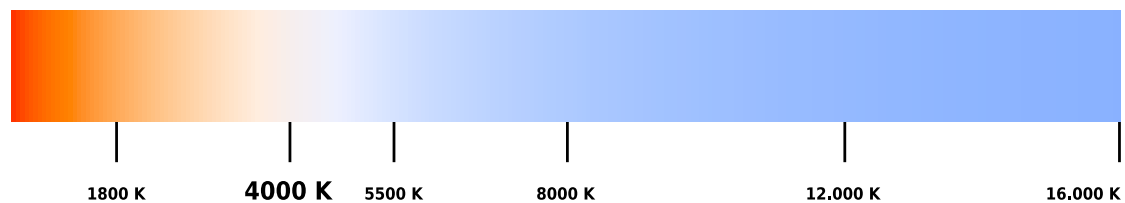


Abbildung 3: Farbtemperatur nach dem planckschen Strahlungsgesetz [Wik09c]

3.3 Renderverfahren

Zu Visualisierung von Volumendaten, gibt es eine Reihe von Verfahren, die sich in zwei Klassen einteilen lassen:

- indirektes Volume-Rendering
- direktes Volume-Rendering

Beim indirekten Volume-Rendering wird versucht aus den Volumendaten die Oberfläche zu extrahieren. Das Ergebnis dieses Prozesses ist eine Menge von Polygonen / Dreiecken, die mit dem klassischen oberflächenbasierten Rendern dargestellt werden können.

Einer der bekanntesten Algorithmen hierfür ist der *Marching-Cube-Algorithmus*. Dies ist ein Verfahren zur Oberflächenextraktion. Hierbei wird jeder Eckpunkt eines Voxels mit einer 1 oder 0 versehen, je nachdem ob sich dieser Punkt im Inneren (hinter einer Oberfläche) oder außerhalb (vor einer Oberfläche) des Objektes befindet. Durch diese Voxelkodierung lässt sich eine acht stellige Binärzahl bilden. Es gibt somit theoretisch 256 Möglichkeiten, wie eine Oberfläche eine Zelle schneiden kann. Aufgrund von einigen geometrischen Eigenschaften, lassen sich diese auf 15 Stück reduzieren [Wat00] (Abbildung 4)

Die letztendliche Position der Dreiecksflächen hängt vom Voxelwert an den Ecken ab. Daraus wird ein Polygon-Mesh erstellt, das an die Grafikkarte übergeben und regulär gerendert wird. Eine Implementierung wird in [WW92] beschrieben. Eine fertige Implementierung des Marching-Cube-Algorithmus gibt es von Nvidia im CUDA SDK [Nvi].

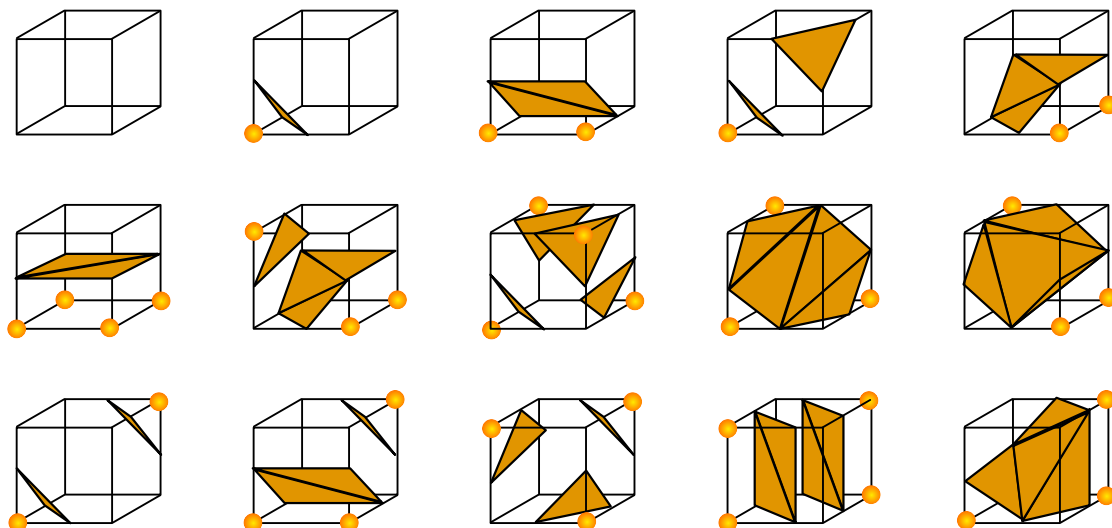


Abbildung 4: 15 Möglichkeiten, wie ein Voxel durch eine Oberfläche geschnitten werden kann [c09]

Im Gegensatz zum indirekten Rendern, werden beim direkten Rendern keine Polygone generiert, sondern der Farbwert wird für jedes Pixel direkt berechnet. Hierfür haben sich eine Reihe von Verfahren bewährt:

Volume Raycasting Beim Raycasting wird für jedes Pixel ein Strahl auf den Volumendatensatz geschossen, der dabei ein oder mehrere Voxel durchquert. Die Volumendaten werden dabei entlang des Strahls, in regelmäßigen Abständen, abgetastet (Abbildung 5). An den Abtastpunkten wird der Voxelwert ermittelt. Der Wert hängt dabei von der Position des Abtastpunktes im Voxel ab oder von der Strecke des Strahls innerhalb des Voxel. Die endgültige Farbe wird dann mit der Transferfunktion erzeugt, die entlang des Strahls akkumuliert wird. Dieser Vorgang kann dabei von vorne nach hinten oder umgedreht geschehen.

Splatting Splatting ist ein von Westover [Wes90] vorgestelltes Voxelprojektionsverfahren, bei dem eine Ebene, entlang der Sichtrichtung, durch den Volumendatensatz wandert und die Voxel auf die Ebene projiziert werden. Da ein Voxel in der Regel größer ist als ein Pixel muss bestimmt werden, welche Werte die Pixel auf der Ebene erhalten. Dies geschieht über eine dreidimensionale Filterung. Als Filter kann zum Beispiel ein Gausskern verwendet werden, der einen kreisrunden Abdruck auf der Bildebene erzeugt (Abbildung 6). Die projizierten Werte werden im Framebuffer akkumuliert. Der Vorgang kann von Vorne nach Hinten oder umgedreht durchgeführt werden. Bildlich lässt sich das Ganze so vorstellen, als ob man mit einem Auto durch einen Insektenschwarm fährt und diese gegen Windschutzscheibe klatschen (engl. to splat).

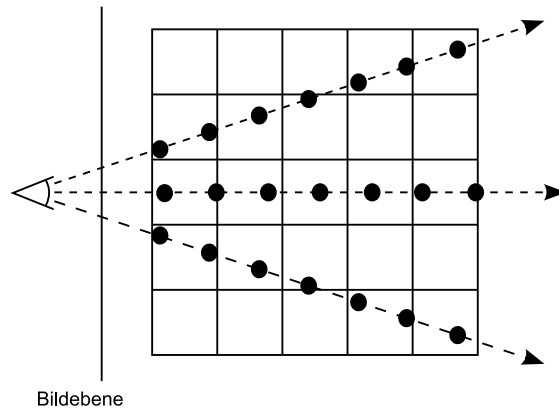


Abbildung 5: Raycasting

Texturbasierten Render Beim texturbasierten Render wird der Volumendatensatz in einer 3D Textur geladen. Dieser Datensatz wird dann mit n Ebenen geschnitten, die parallel zur Bildebene sind. Jede Ebene erzeugt mit den Schnittkanten der Bounding-Box des Datensatzes ein Polygon, die sogenannte Proxy-Geometrie. Diese wird dazu verwendet um den Volumendatensatz abzutasten (Abbildung 7).

Nun wird die Proxy-Geometrie sequentiell von vorne nach hinten oder umgedreht gezeichnet. Dabei wird für jedes Fragment die 3D-Textur bzw. der Volumendatensatz nun mit einer trilineare Interpolation abgetastet und mit der bisherigen Pixelfarbe vermischt. Ikits zeigt in [IKLH04] wie sich diese Verfahren auf einer Grafikkarte realisieren lässt.

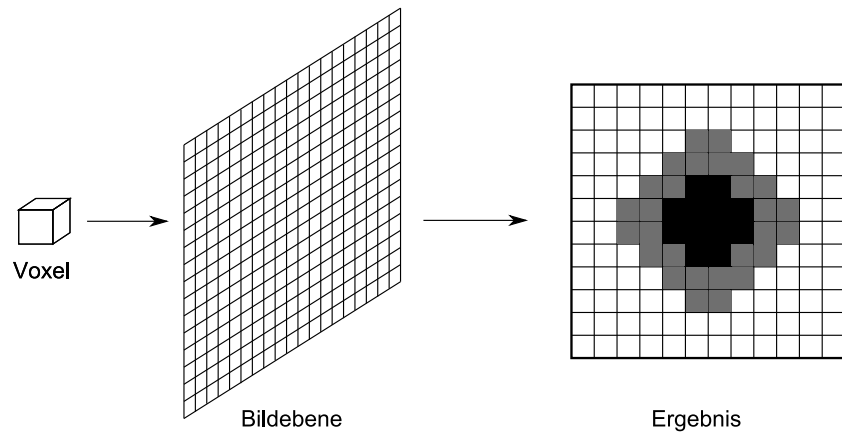


Abbildung 6: Projektion eines Voxel auf die Ebene mittels Gausskern

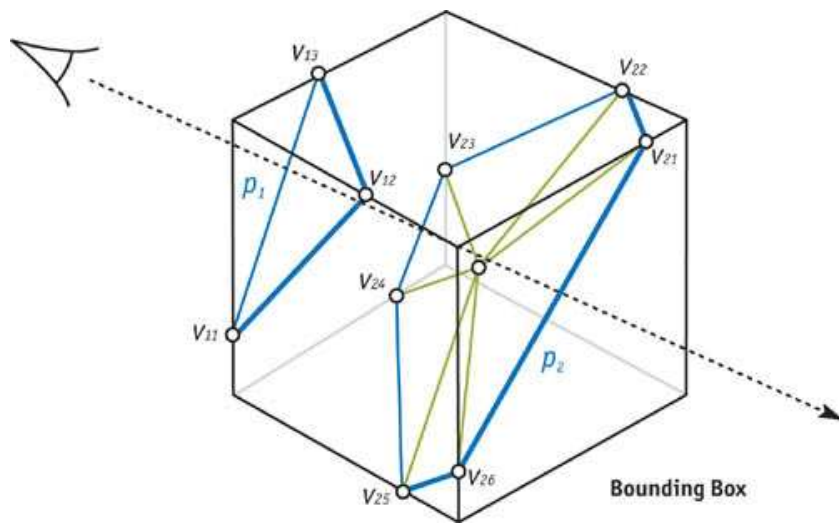


Abbildung 7: Proxy-Geometriy [IKLH04]

4 Rauch und Feuer mit Volumenrendering

Feuer und Rauch sind klassische Effekte in der Computergrafik. In diesem Abschnitt wird ein Verfahren vorgestellt, mit dem diese simuliert und mit einem volumenbasierten Renderverfahren visualisiert werden. Dieser Abschnitt basiert in erster Linie auf dem Artikel *Real-Time Simulation and Rendering of 3D Fluids* von Crane et al. [CLT08]. Darin wird ein Verfahren vorgestellt, das unter Verwendung von Computational Fluid Dynamics (CFD) Rauch, Feuer und Flüssigkeiten simuliert, die außerdem auf bewegliche und statische Hindernisse reagieren. Diese Arbeit beschränkt sich allerdings auf Feuer und Rauch. Die Wahl fiel auf dieses Verfahren, weil sowohl Simulation als auch das Rendern auf der Grafikkarte berechnet wird und weil es seine Praxistauglichkeit bereits in dem Computerspiel *Hellgate London* unter Beweis gestellt hat [CLT08].

Dieser Abschnitt ist in zwei Teile aufgeteilt. Im ersten Teil wird kurz auf das Simulationsverfahren eingegangen, durch das der Volumendatensatz für einen Zeitpunkt generiert wird. Danach wird im zweiten Teil auf das Renderverfahren eingegangen, das dazu verwendet wird um den Volumendaten zu visualisieren.

4.1 Simulation

Feuer und Rauch sind keine statischen Effekte, sondern sind ständig in Bewegung. Als Mensch ist man mit diesen Effekten vertraut und hat somit eine gewisse Erwartung an dessen Bewegungsverhalten. Diese Effekte werden durch Strömungen verursacht, die Teilchen durch ein Medium bewegen. Diese Strömung selbst wird dabei von verschiedenen Größen beeinflusst, wie externe Kräfte, Dichte, Hindernisse und/oder Druck.

Mit dieser Problematik setzt sich die Fluidodynamik auseinander. Diese befasst sich mit der Bewegung von Fluiden. Ein Fluid ist ein strömendes Medium, also Flüssigkeiten und Gase. Es gibt eine Reihe von Modellen mit denen sich diese beschreiben lassen. Zur algorithmischen Lösung dieser Problemstellung hat sich die numerische Strömungsmechanik (engl. computational fluid dynamics) etabliert. Ein Einblick in diese Thematik gibt Laurien und Oertel in [LOj09].

4.1.1 Modellierung

Zur Modellierung der Fluide haben sich eine Reihe von Gleichungen etabliert. Einen Überblick gibt [LOj09]. Eine der bekanntesten ist wohl die Navier-Stokes-Gleichung. Diese gibt es für kompressible und inkompressible Fluide. Ein Fluid ist kompressibel wenn durch Krafteinwirkung die Dichte und somit das Volumen verringert werden kann. Beispiel hierfür sind Gas, wohingegen Flüssigkeiten so gut wie inkompressibel sind [LOj09]. Harris verwendet in [Har04] die *Navier-Stokes-Gleichung für inkompressible Fluide*. Die

vereinfachte Annahme der Inkompressibilität ist auch bei kompressiblen Medien oft gerechtfertigt da, diese in Bereichen, wo sich die Dichte nicht ständig ändert keinen signifikanten Einfluss auf die Simulation hat [LOj09][Har04]. Gleichung 1 und 2 beschreiben Navier-Stokes-Gleichung für inkompressible Fluide.

$$\frac{\partial \mathbf{u}}{\partial t} = -(\mathbf{u} \cdot \nabla) \mathbf{u} - \frac{1}{\rho} \nabla p + \nu \nabla^2 \mathbf{u} + F \quad (1)$$

$$\nabla \cdot \mathbf{u} = 0 \quad (2)$$

Harris [Har04] beschreibt die vier Terme auf der rechten Seite von Gleichung 1 (von links nach rechts) als Advektion, den Druck, die Diffusion und eine externe Kraft.

Advektion Advektion beschreibt den Transport von Objekten oder Eigenschaften in Fluiden abhängig von der Geschwindigkeit. Dies ist durch den Advektionsoperator $-(\mathbf{u} \cdot \nabla)$ beschrieben. Der ganze Term beschreibt dabei die Selbstadvektion des Geschwindigkeitsfeldes \mathbf{u} .

Druck Der zweite Term beschreibt die Beschleunigung die sich durch den Druck ergibt. Dieser entsteht dadurch, dass Moleküle innerhalb eines Fluides eine Krafteinwirkung nicht sofort an das ganze Fluid weitergeben, sondern dass diese sich an einem Ort verdichten. ρ ist hierbei die Dichte, die im inkompressiblen Fall konstant ist. p steht für den Druck, also die Kraft pro Flächeninhalt.

Diffusion Dieser Term beschreibt die Zähflüssigkeit von Fluiden. Diese wird durch die Viskosität ν angegeben, die von Fluid zu Fluid unterschiedlich. So ist zum Beispiel Sirup zähflüssiger als Wasser.

Externe Kraft Dieser Term beschreibt die Beschleunigung die durch externe Krafteinwirkung stattfindet. Diese können lokal sein, d. h. sie wirken nur auf einen Bereich des Fluids (z.B. Windhauch) oder global und wirken auf das ganze Fluid (z. B. Erdanziehung).

Gleichung 2 ist die Kontinuitätsgleichung. Diese beschreibt Inkompressibilität des Fluids, durch die Divergenzfreiheit von \mathbf{u} . In [CLT08] wird auf den Diffusionsterm verzichtet, wodurch die Gleichung zur *Euler-Gleichung* vereinfacht wird.

$$\frac{\partial \mathbf{u}}{\partial t} = -(\mathbf{u} \cdot \nabla) \mathbf{u} - \frac{1}{\rho} \nabla p + F \quad (3)$$

Ein Grund hierfür wird nicht explizit genannt. Jedoch liegt der Verdacht nahe, dass er die Gleichung vereinfachen wollte, da die Viskosität für Feuer und Rauch vernachlässigbar ist und sie somit den Wert $\nu = 0$ hat, wodurch der Diffusionsterm weg fällt. Im Folgenden wird nur noch die Euler-Gleichung verwendet.

4.1.2 Lösung

Gleichung 3 ist ein nichtlineares partielles Differentialgleichungssystem 1. Ordnung mit vier Gleichungen und vier Unbekannten u_x , u_y , u_z und p .

$$\begin{aligned}\frac{\partial u_x}{\partial t} &= -(\mathbf{u} \cdot \nabla)u_x - \frac{1}{\rho}\nabla p + F \\ \frac{\partial u_y}{\partial t} &= -(\mathbf{u} \cdot \nabla)u_y - \frac{1}{\rho}\nabla p + F \\ \frac{\partial u_z}{\partial t} &= -(\mathbf{u} \cdot \nabla)u_z - \frac{1}{\rho}\nabla p + F\end{aligned}$$

Das hier gezeigte Verfahren basiert auf dem von Stam [Sta99] vorgestellten Verfahren, dass von Harris [Har04] auf der GPU realisiert wurde und von Crane et al. [CLT08] auf den dreidimensionalen Fall erweitert wurde. Im Folgenden wird nur die grundsätzliche Vorgehensweise und die verwendeten Verfahren kurz vorgestellt um ein Gefühl dafür zu bekommen. Für eine genaue mathematische Beschreibung sei auf [Sta99] verwiesen. Eine Shader-Implementierung wird in [Har04] und [CLT08] gezeigt.

Die Lösung der Gleichung 3 wird in drei Schritten durchgeführt. Der erste Schritt besteht darin, die Advektion zu lösen. Hieraus resultiert ein Geschwindigkeitsfeld auf das nun eine externe Kraft addiert wird. Daraus ergibt sich das divergente Geschwindigkeitsfeld \mathbf{w} , woraus sich nun p bestimmen lässt [Har04]. Aus \mathbf{w} und p wird nun \mathbf{u} bestimmt werden.

Advektion Zur Lösung der Advektion kann das *Semi-Lagrange-Verfahren* verwendet. Dies ist bei großen Zeitschritten stabil [Sta99] und lässt sich auf Fragment-Shadern implementieren [Har04]. Bei diesem Verfahren wird die Geschwindigkeit eines Fluidelements an der neuen Position zum Zeitpunkt $t + \delta t$, aus den Geschwindigkeiten in der Umgebung des Fluidelements der alten Position zum Zeitpunkt $t - \delta t$ berechnet. Abbildung 8 verdeutlicht diesen Sachverhalt noch einmal.

Crane et al. [CLT08] schlagen eine Alternative zum Semi-Lagrange-Verfahren vor: das *MacCormack-Verfahren*. Hierbei wird, vereinfacht gesagt, das Semi-Lagrange-Verfahren zweimal pro Advektionsschritt angewendet. Einmal für eine Vorhersage der Advektion und einmal für eine Korrekturrechnung dieser Vorhersage. Aus diesen beiden Ergebnissen wird dann die endgültige Advektion für den Schritt bestimmt. Das MacCormack-Verfahren führt zu besseren Ergebnissen bei gleicher Gittergröße, da es eine höhere Genauigkeit hat (Abbildung 9). Dies ist zwar rechenintensiver als das Semi-Lagrange-Verfahren jedoch ist Rechenzeit auf der GPU günstig im Vergleich zur Speicherbandbreite.

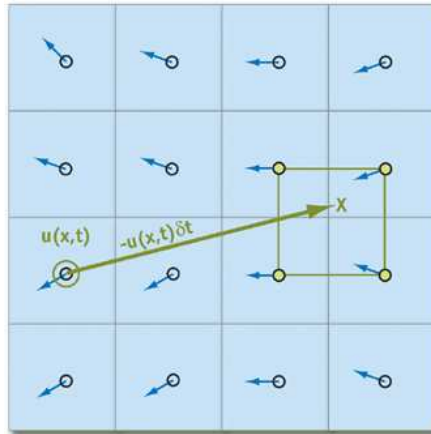


Abbildung 8: Semi-Lagrange-Verfahren [Har04]

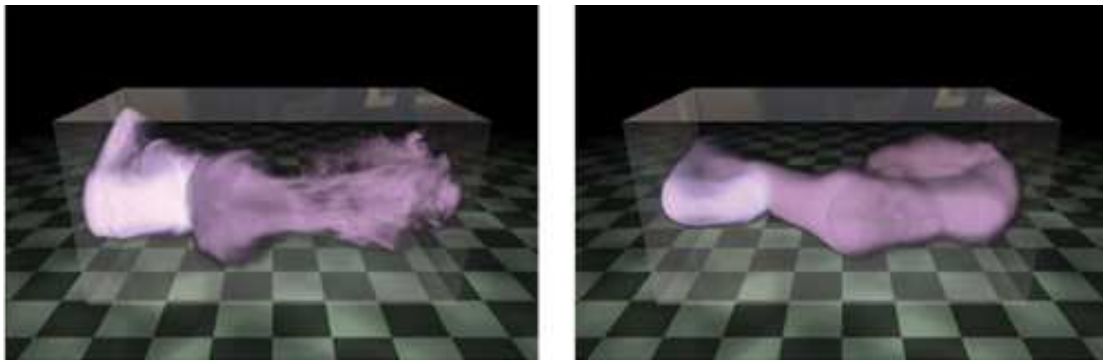


Abbildung 9: Links MacCormack-Verfahren (128x64x64),
Rechts: Semi-Lagrangian-Verfahren (256x128x128) [CLT08]

Externe Kraft Die externe Krafteinwirkung wird einfach durch eine Addition auf das bisherige Geschwindigkeitsfeld der Advektion aufaddiert. Daraus ergibt sich nun das divergente Geschwindigkeitsfeld \mathbf{w} .

Druck Durch anwenden des Helmholtz-Hodge-Zerlegung und der Berücksichtigung der Kontinuitätsgleichung (Gleichung 2), ergibt sich die Poisson-Gleichung

$$\nabla^2 p = \nabla \cdot \mathbf{w} . \quad (4)$$

Eine Herleitung hierfür ist [Har04] zu entnehmen. Mit dem Jacobi-Verfahren lässt sich dieses System lösen, wodurch man den p erhält.

Gemäß dem Helmholtz-Hodge-Zerlegung [Har04]

$$\mathbf{u} = \mathbf{w} - \nabla p, \quad (5)$$

lässt sich nun \mathbf{u} als Differenz von \mathbf{w} und ∇p darstellen, womit alle Unbekannten bestimmt sind.

4.1.3 Berücksichtigung von Objekten

Die bisherige Betrachtung geht davon aus, dass sich das Fluid alleine in einem Volumen befindet, in dem es sich bewegen kann. Oft ist es aber der Fall, dass ein festes Objekt von dem Fluid umgeben ist, dass dessen Bewegung beeinflusst. Des weiteren könnte sich das Objekt auch noch bewegen, wodurch die Bewegung des Fluids auf unvorhergesehen Weise beeinflusst wird. Abbildung 10 zeigt dies am Beispiels einer Figur, die den Rauch mit seinen Flügel weg wedelt.



Abbildung 10: Figur, die mit ihrem Flügelschlag den Rauch wegwedelt [CLT08].

Der Effekt der hierbei entstehen soll, ist dass das Fluid an der Objekt Oberfläche entlang fließt. Dabei soll das Fluid keinesfalls in das Objekt hinein fließen. Dies muss bei der Lösung des Druckterms berücksichtigt werden, da Voxel, die innerhalb eines Objektes sind, keinen Einfluss auf den Druck haben sollen. Wie dies mathematisch berücksichtigt wird, wird in [CLT08] beschrieben.

Für diese Berücksichtigung muss bekannt sein, welche Voxel innerhalb des Objektes sind, und welche Geschwindigkeit das Objekts hat. Um diese Daten zu bestimmen, stellen Crane et al. ein Verfahren in [CLT08] vor, bei dem das Objekt schnell voxelisiert werden kann.

Hierbei werden zunächst mehrere Querschnitte des Objektes erzeugt. Das Verfahren das Crane et al. verwenden ist an den Stencil-Shadow-Volume-Algorithmus angelehnt. Das Ergebnis dieses Prozesses sind mehrere Schwarz-Weiß-Texturen, die den Querschnitt des Objektes an verschiedenen Stellen zeigen (*inside-outside texture*). Abbildung 11 (oben rechts) verdeutlicht dies. Weiße Bereiche sind dabei außerhalb und schwarze innerhalb des Objekts.

Danach muss die Geschwindigkeit für das Objekt bestimmt werden. Hierbei wird zunächst die Geschwindigkeit an den Vertices des Objekts bestimmt. Danach kann durch Interpolierung die Geschwindigkeit von den Vertices auf die Voxel übertragen werden. Das Ergebnis ist in Abbildung 11 (unten rechts) dargestellt. Es ist ein Vektorfeld in Form einer Textur, das die interpolierten Geschwindigkeiten am Objektrand speichert (*velocity texture*). Hierfür kann auch ein vereinfachtes Modell mit gröberer Geometrie verwendet werden, um den Prozess zu beschleunigen.

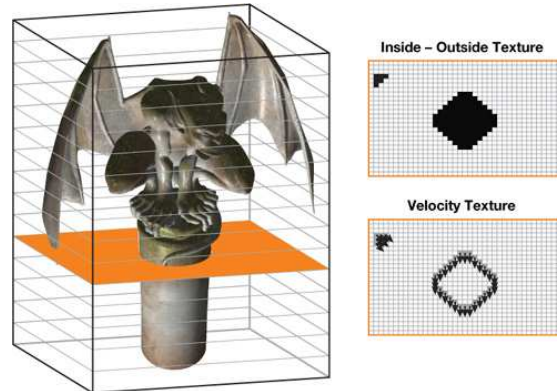


Abbildung 11: Voxelesierung der Figur [CLT08].

4.1.4 Effektspezifische Einflussgröße

Je nach verwendetem Effekt müssen noch weitere Größen gespeichert werden, die das Fluid transportieren soll oder die Einfluss auf die Bewegung des Fluides haben. Jeder dieser Größen wird durch ein Skalarfeld beschrieben. Die Advektion der einzelnen Werte wird dabei gemäß

$$\frac{\partial \phi}{\partial t} = -(\mathbf{u} \cdot \nabla) \phi, \quad (6)$$

vorgenommen. ϕ ist hierbei die Größe, die transportiert werden soll.

Rauch hat beispielsweise in [CLT08] zwei zusätzliche Größen für Dichte und Temperatur. Daher werden zwei weitere Skalarfelder — letztendlich Texturen — benötigt, um diese zu speichern. Soll nur eine Größe transportiert werden, so reicht es aus, sie mit Gleichung 6 zu berücksichtigen. Hat die Größe jedoch Einfluss auf die Bewegung des Fluids, wie die Temperatur in diesem Beispiel, so muss diese noch als externe Kraft berücksichtigt werden. Dies wird hier durch die Auftriebskraft beschrieben, die als externe Kraft aufaddiert wird. Der Effekt ist, dass kalter Rauch ab- und warmer aufsteigt.

Für Feuer wird noch eine Reaktionskoordinate transportiert. Diese beschreibt einfach gesagt den Brennwert des Feuer an einer Position und wird in jedem Simulationsschritt reduziert. Ist der Wert 0 erreicht, so „erlischt“ das Feuer an dieser Position.

4.2 Rendern

Nach jedem Simulationsschritt steht nun ein veränderter Volumendatensatz zur Verfügung. Aus diesen Daten soll nun ein Bild generiert werden. Im zweiten Kapitel wurden bereits mehrere Verfahren zum Volume-Rendern kurz vorgestellt. Crane et al. verwendeten in [CLT08] zum Rendern ein Raycasting-Verfahren, das sich an dem von Westermann und Krüger in [KW03] vorgestellten Verfahren orientiert. Im Folgenden wird dieses Verfahren vorgestellt.

Hierbei wird ausgehend vom Augpunkt mehrere Strahlen auf den Volumendatensatz geschossen. Entlang des Strahls werden die Werte innerhalb des Datensatzes akkumuliert um so die Farbe für den Pixel zu bekommen. Hierfür muss bekannt sein, wo der Strahl auf das Volumendatensatz auftrifft, in welche Richtung er diesen durchquert und wie lang die Strecke des Strahls innerhalb des Volumens ist.

Dieser Prozess läuft in zwei Schritten ab. Der erste Schritt ist ein Vorverarbeitungsschritt, um dem Eintrittspunkt und die Länge für jeden Strahl zu bestimmen. Diese Werte werden in einer 2D-Textur abgespeichert, hier bezeichnet mit *RayData*. *RayData* hat dabei die Ausmaße des Viewports. Der zweite Schritt ist der eigentliche Raycasting-Prozess, bei dem der Volumendatensatz abgetastet wird.

4.2.1 Vorverarbeitung

Der Volumendatensatz wird zunächst in eine 3D-Textur geladen. Zu dieser 3D-Textur wird nun ein Bounding-Volume erstellt, das in der Szene platziert wird. Die Strahlänge ist die Strecke zwischen Eintritts- und Austrittspunkt des Strahl im Bounding-Volume (Abbildung 12). Der Eintrittspunkt ergibt sich aus der Texturkoordinate der Vorderseite des Bounding-Volumens.

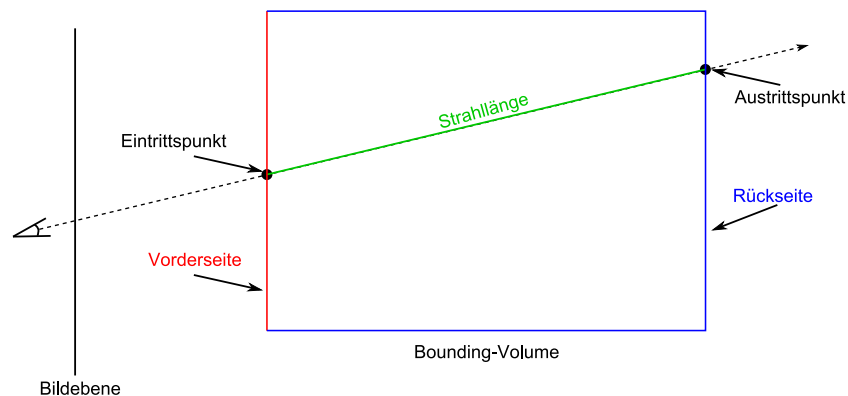


Abbildung 12: Querschnitt des Bounding-Volumens

Nun wird als erstes die Rückseite des Bounding-Volumes gezeichnet. Ein Fragmentprogramm bestimme nun den Abstand (im View-Space) zwischen Augpunkt und Rückseite und speichere diesen Wert im Alphakanal von *RayData*

$$RayData.alpha = length(Austrittspunkt - Augpunkt)$$

Danach wird die Vorderseite des Bounding-Volumes gezeichnet. Ein anderes Fragmentprogramm bestimmt nun den Abstand zwischen Augpunkt und Vorderseite sowie den Eintrittspunkt. Der Eintrittspunkt ist dabei die Texturkoordinate des Bounding-Volumes. Diese Werte werden auch in *RayData* geschrieben, bei aktivierter subtraktiver Farbmischung. Dies hat den folgenden Effekt:

$$RayData.alpha = RayData.alpha - length(Eintrittspunkt - Augpunkt)$$

$$RayData.RGB = Vorderseite.RGB$$

Nun befindet sich im RGBKanal von *Raydata* der Eintrittspunkt und im Alphakanal die Länge des Strahls.

4.2.2 Raycasting

RayData enthält nun alle notwendigen Daten. Nun kann das Volumen mit dem Strahl abgetastet werden. Die Richtung des Strahls ergibt sich dabei aus der Differenz zwischen Eintrittspunkt und Augpunkt:

$$Strahlrichtung = RayData.RGB - Augpunkt$$

Ausgehend vom Eintrittspunkt entlang des Strahls werden nun die Volumendaten abgetastet, bis die maximale Anzahl der Abtastpunkte für diesen Strahl erreicht ist. In [Crane2008] wurde die Anzahl so gewählt, dass der Abstand zwischen zwei Punkten die halbe Voxelgröße hat.

Der Farbwert wird nun nach folgenden Schema entlang der Abtastpunkte akkumuliert:

$$RGB_{final+} = RGB_{Sample} \cdot Alpha_{Sample} \cdot (1 - Alpha_{final})$$

$$Alpha_{final+} = Alpha_{Sample} \cdot (1 - Alpha_{final})$$

Hierbei handelt es sich um die Formel für Farbmischung von Vorne nach Hinten. Überschreitet $Alpha_{final}$ einen bestimmten Wert (z. B. $Alpha_{final} > 0.99$), kann das Abtasten für diesen Strahl abgebrochen werden, da dahinter liegende Objekte nicht mehr sichtbar sind. Dieser Vorgang wird für jeden Strahls wiederholt.

4.3 Rauch und Feuer

Wie bei der Simulation bereits gesagt wurde, wird für den Raucheffect ein Dichte- sowie ein Temperaturwert benötigt. Die Temperatur ist jedoch für die Farbgebung uninteressant. Die Farbgebung basiert ausschließlich auf der Dichte. Dieser wird bei der Generierung der Fluidelemente durch eine zeit- und zufallsabhängigen sinusförmige Funktion bestimmt. Außerdem wird noch ein Gaussfilter verwendet. Diese Prozedur ist nicht physikalisch begründet, sondern sie liefert nur einen ausreichend guten Effekt von herumwirbelnden Rauch (Abbildung 10). Eine Transferfunktion wird in diesem Fall nicht verwendet, da die Farbe direkt aus der Dichte bestimmt werden kann.

Auch Feuer verfügt über eine Dichte die ähnlich wie beim Rauch gesetzt wird aber mit Parametern. Der eigentliche Unterschied ist aber, dass für Feuer eine Reaktionskoordinate verwendet wird dessen Bewegung durch

$$\frac{\partial Y}{\partial t} = -(\mathbf{u} \cdot \nabla)Y - k, \quad (7)$$

beschrieben wird [NFJ02]. Y ist hierbei die Reaktionskoordinate und k der Wert, um dem sie in jedem Simulationsschritt reduziert wird. Y wird nun über Transferfunktion (Lookup in einer 1D Textur) ein Farbwert zugewiesen. Dieser wird noch mit der Feuerdichte vermischt und man erhält die endgültige Farbe. Solange Y einen Wert größer 0 hat, wird Feuer gezeichnet. Hat der Wert jedoch 0 erreicht wird nur Rauch gezeichnet. Dadurch erhält man einen feinen Übergang vom Feuer zum Rauch (Abbildung 13).



Abbildung 13: Gargoyle im brennenden Feuer. An der „Feuergrenze“ geht Feuer in dunklen Rauch über

Natürlich ist abbilden von Y auf einen Farbwert eine starke Vereinfachung. Ein stärker physikorientiertes Verfahren ist [NFJ02] zu entnehmen. Auch für den Rauch gibt es auf-

wenigere Verfahren [FSJ01]. Darin werden auch noch Self-Shadowing und Beleuchtung besprochen.

4.3.1 Weitere Problemstellungen

Die bisherige Ausführung diente dazu um den groben Ablauf des Renderns zu verstehen. Es gibt aber noch weitere Problemstellungen, auf die noch nicht eingegangen wurde.

Zunächst einmal sind beim Renderprozess noch keine Objekte berücksichtigt worden, die innerhalb des Effekts sind oder ihn ganz oder teilweise verdecken. Abbildung 14 illustriert die Situation.

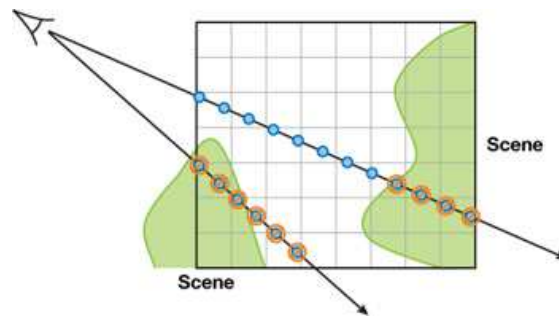


Abbildung 14: Zwei Objekte (grün) im Effekt. Linkes Objekt verdeckt die Vorderseite, Rechtes ist vom Effekt umschlossen [CLT08].

Um dies zu berücksichtigen, muss der Renderprozess um zwei Schritte ergänzt werden. Zunächst einmal wird bei der Bestimmung der Länge des Strahls nicht mehr der Abstand vom Augpunkt zur Rückseite des Bounding-Volumes genommen, sondern das Minimum von Szenedistanz und Abstand zur Rückseite (Gleichung 8).

$$RayData.alpha = \min(\text{Abstand}_{Szene}, \text{Abstand}_{Rueckseite}) \quad (8)$$

Um festzustellen, ob ein Objekt vor dem Effekt ist, wird überprüft, ob Szenedistanz kleiner ist als der Abstand zur Vorderseite. Ist dies der Fall, so wird ein Teil der Vorderseite durch ein Objekt verdeckt. Um dies in der *RayData* zu vermerken, wird ein negativer Wert in den roten Farbkanal (x-Koordinate Eintrittspunkt) geschrieben. Beim Raycasting wird dann überprüft ob die x-Koordinate des Eintrittspunktes negativ ist. Sollte dies der Fall sein, wird der Vorgang für diesen Strahl abgebrochen.

Ein weiteres Problem, das auch noch nicht berücksichtigt wird, ist wenn der Vorderseite des Bounding-Volumes nicht oder nur teilweise gezeichnet wird. Dies ist dann der Fall, wenn der Augpunkt innerhalb des Effektes liegt oder so nah an ihm dran ist, dass die Near-Clipping-Plane ein Teil der Vorderseite abschneidet (Abbildung 15). Mit dem bisherigen Verfahren ist es so nicht mehr möglich, den Eintrittspunkt und somit auch nicht die Strahlrichtung zu bestimmen.

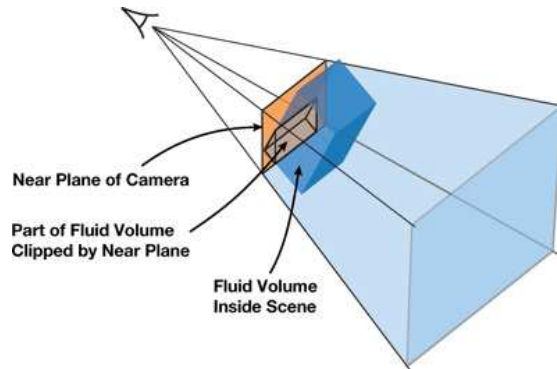


Abbildung 15: Near-Clipping-Plane schneide das Bounding-Volume so, dass der mittlere Bereich im Inneren und der Randbereich außerhalb liegt [CLT08].

Um das zu berücksichtigen, wird für jedes Pixel, bei dem nur die Rückseite aber nicht die Vorderseite gezeichnet wird, ein negativer Grünwert an der entsprechenden Position in *RayData* geschrieben. Dies geschieht beim Rendern der Rückseite im Vorverarbeitungsschritt, wo bisher nur der Abstand zur Rückseite gesetzt wurde. Beim Setzen der Eintrittspunkte in *RayData* wird der negative Grünwert nur an Stellen überschrieben, wo auch die Vorderseite gezeichnet wird. Beim Raycasting wird nun geprüft, ob der Grünwert negativ ist. Ist dies der Fall, dann wird der RGB-Wert für diesen Pixel durch die Position der Near-Clipping-Plane im Texturraum des Bounding-Volumen ersetzt. Außerdem wird der Distanzwert um den Abstand von Augpunkt und Near-Clipping-Plane verringert werden.

4.3.2 Artefakte

Es können eine Reihe von Artefakten beim Rendern von Volumen-Daten entstehen. Das bekannteste ist wohl der „Lego-Effekt“ (Abbildung 16). Ein weiterer ist das Banding, ein Treppeneffekt im Farbverlauf.

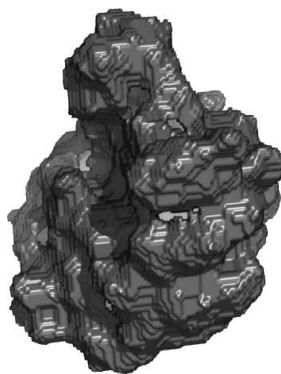


Abbildung 16: Lego-Effekt durch die Würfelstruktur [htt09].

Es gibt eine Reihe von Ansätzen die diese Probleme behandeln. Hier gehören unter anderem Filterung, Jittering, Erhöhung der Abtastfrequenz oder eine Kombination daraus [CLT08].

5 Vergleich zu Partikelsystemen

Wie bereits weiter oben erwähnt, stehen Partikelsysteme in Konkurrenz zu den volumenbasierten Ansätzen. In diesem Abschnitt werden nun die Partikelsysteme mit den volumenbasierten Ansatz verglichen. Bevor dies allerdings gemacht wird, wird zunächst noch das grundlegende Prinzip der Partikelsysteme erklärt.

Partikelsysteme basieren bis heute im Wesentlichen alle auf dem von Reeves in [Reeves1983] vorgestellten Verfahren. Hierbei wird von einem logischen Partikel ausgegangen, das ein Platzhalter für ein zu zeichnendes Objekt ist. Das Objekt kann dabei sehr einfach sein (z.B. Punkt mit Farbe) oder auch ein komplexes Modell. Die Erzeugung, Bewegung und Lebensdauer dieser Partikel wird dabei vom einem Partikelsystem bestimmt.

Ein Partikel verfügt über mehrere Attribute. Diese können prinzipiell beliebig festgelegt werden. Überlicherweise verfügen diese jedoch über eine Position, Geschwindigkeit, Alter/Lebensdauer, Farbe, Größe und Form [Reeves1983]. Außerdem können Partikel auch Partikelsysteme repräsentieren (hierarchische Partikelsysteme).

Für die Erzeugung der Partikel und Positionierung des Partikelsystems ist der Emitter zuständig. Diese markieren den Ursprung der Partikel. In der Regel wird dieser eine einfache Form sein, wie zum Beispiel ein Rechteck, das den Bereich festlegt in dem die Partikel entstehen können. Nach der Erzeugung müssen die Partikel in jedem Simulationsschritt bewegt werden. Das Schema hierfür kann beliebig einfach oder komplex sein, angefangen von einem zufälligen Bewegungsmuster bis hin zu einer physikbasierten Simulation. Ist die maximale Lebensdauer erreicht oder hat sich das Partikel zu weit vom Emitter entfernt, so wird das Partikel aus dem System wieder entfernt. Die Visualisierung der Partikel wird mit dem klassischen oberflächenbasierten Rendern realisiert. Die Partikel werden durch Grafikprimitive oder Billboards ersetzt und an die Grafikkarte zum Zeichnen übergeben. Je nachdem was das Partikel repräsentiert und wie es sich bewegt können unterschiedliche Effekte erzeugt werden (Abbildung 17).

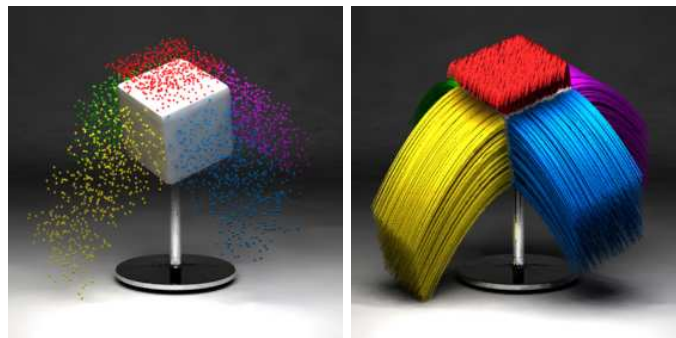


Abbildung 17: Visualisierung der Partikel als Partikel (links) und als Fäden (rechts) [Wik09a].

Bis zur Simulation scheint der volumenbasierte Ansatz keinen Vorteil gegenüber dem Partikelsystem zu haben. Die Simulation der Partikel eines Partikelsystems kann auch in einem Voxelgitter stattfinden. Im Gegensatz zu den Partikelsystemen ist der volumenbasierte Ansatz jedoch auf die dreidimensionale Gitterstruktur angewiesen. Partikelsysteme sind zunächst einmal unabhängig von einer derartigen Struktur. Partikel können so theoretisch mit Objekten der Umwelt, an beliebiger Stelle, interagieren. Das Fluid hingegen nur mit Objekten, die sich innerhalb des Voxelgitters befinden. Jedoch sind viele physikbasierten Simulationen — wie auch die oben vorgestellte Strömungssimulation — auf eine zwei oder dreidimensionale Gitterstruktur angewiesen [LOj09]. Nutzt ein Partikelsystem eine derartige Simulation, so gilt diese Einschränkung auch für diese.

Die Hauptunterschiede zeigen sich zweifellos beim Rendern, da hier zwei grundsätzlich verschiedene Verfahren verwendet werden. Partikelsysteme greifen auf das klassische oberflächenbasiertes Rendern zurück, wohingegen der volumenbasierte Ansatz auf Verfahren des Volume-Renderings zurückgreift. Letztere werden in der Regel nicht nativ von der Grafikkarte unterstützt. Diese können jedoch inzwischen oft mit Shader-Programmen realisiert werden, die diesen Prozess unterstützen. Es gibt mehrere Visualisierungsmethoden beim Volume-Rendering (siehe Abschnitt 2). Je nach Anwendung kann somit ein Verfahren gewählt werden, dass das beste Ergebnis liefert.

Die optische Qualität zwischen den beiden Ansätzen ist schwer zu beurteilen. Beide Ansätze sind in der Lage sehr gute wie auch sehr schlechte Ergebnisse zu liefern. Das ganze ist von der Anwendung abhängig. Der voxelbasierte Ansatz hat jedoch große Vorteile, wenn es darum geht physikalische Größen bei der Farbbildung zu berücksichtigen. Feuer zum Beispiel wirkt bei Partikelsystemen immer wie brennender Rauch (Abbildung 18 c)). Auch wenn die Bewegung selbst gut aussieht so fehlt irgendwie die Flammenbildung. Diese kann zwar recht gut mit Videotexturen verbessert werden (Abbildung 18 a) und b) [Ngu04], jedoch ist es nicht oder nur schwer möglich, Größen wie Temperatur, Dichte, Brennwert die direkt Einfluss auf das Erscheinungsbild einer Flamme haben können zu berücksichtigen. Der voxelbasierte Ansatz kann dies, durch den Schritt über die Transferfunktion (Abbildung 18 d)).

Volumenbasierte Ansätze haben, bedingt durch die Volumenstruktur, einen hohen Speicherbedarf ($O(n^3)$). Zwar lässt sich dies durch entsprechende Datenstrukturen verbessern, dennoch bleibt die kubische Komplexität ein Problem. Tabelle 1 zeigt den Speicherbedarf aus dem oben vorgestellten Verfahren. Eine Aufstellung, wie diese Werte genau zustande kommen, ist aus [CLT08] zu entnehmen.

Fluidsimulation	Voxelisierung	Rendering
32 bytes / Voxel	9 Bytes / Voxel	20 Bytes / Pixel

Tabelle 1: Speicherbedarf

Ausgehend von dem Beispiel in Abbildung 9 links, mit einer Gittergröße von 128x64x64 (= 524.288 Voxel) bei einer Auflösung von 1920 x 1080 Pixel kommt man so auf einen

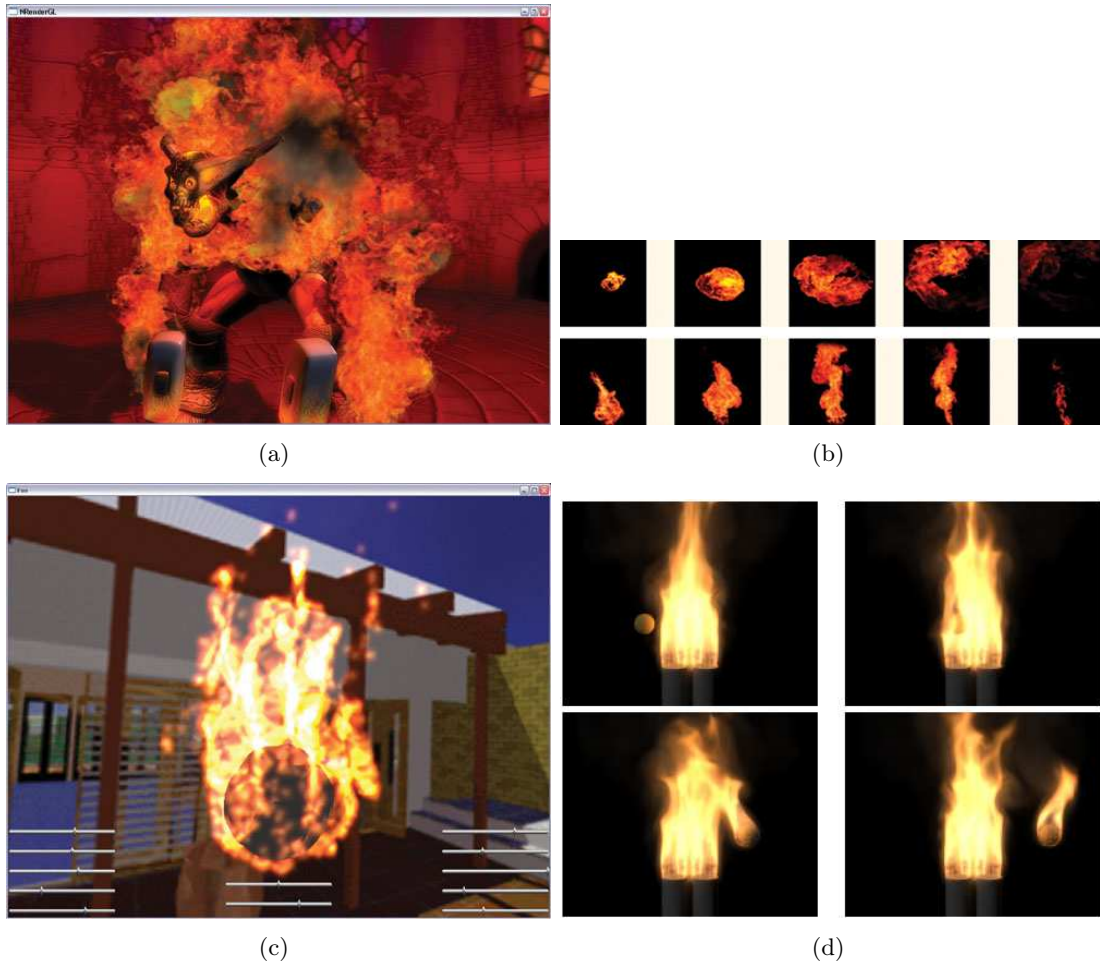


Abbildung 18: a) zeigt Feuer das mit der in b) gezeigten Textursequenz erzeugt wurde [Ngu04]. c) Feuer mit 5000 Partikeln [Ngu04]. d) Feuer mit einem physik-basierten Ansatz. Gittergröße: 160x160x160 [NFJ02]

Speicherbedarf von

$$(32\text{Bytes} + 9\text{Bytes}) \cdot 524288 + (1920 \cdot 1080) \cdot 20\text{Bytes} = 62967808\text{Bytes} \approx 63\text{MB}.$$

Bei einer Verdoppelung der Gittergröße auf 256x128x128 (= 4.194.304 Voxel) sind es schon

$$(32\text{Bytes} + 9\text{Bytes}) \cdot 4194304 + (1920 \cdot 1080) \cdot 20\text{Bytes} = 213438464\text{Bytes} \approx 214\text{MB}.$$

Der Speicherbedarf bei Partikelsystemen wächst hier hingegen nur linear mit Anzahl der Partikel.

Generell scheinen Anwendungsmöglichkeiten von Partikelsystem wesentlich größer zu sein. Sie beschränken sich nicht nur auf formverändernde Effekte wie Feuer oder Rauch, sondern können auch zur Simulation von Haaren, Pflanzen oder Textilien eingesetzt werden (Abbildung 17 rechts). Auch bei Effekten, bei denen die Partikel nicht nur als Gesamteffekt, sondern auch einzeln wahrnehmbar sein sollen, wie zum Beispiel herumfliegende Trümmer bei einer Explosion oder Regentropfen bei einem Regenschauer, eignen sich die Partikelsysteme besser. Volumenbasierte Ansätze eignen sich für Anwendungen, bei denen physikorientiertes Rendern wichtig ist.

6 Zusammenfassung

In dieser Arbeit wurde ein Verfahren zum Visualisieren von volumetrischen Effekten durch Volumen-Rendering vorgestellt. Hierbei wurde eine Reihe von gängigen Verfahren vorgestellt, mit denen Volumendaten visualisiert werden können. Der Hauptteil dieser Arbeit beschäftigte sich damit ein Verfahren von Crane et al. [CLT08] vorzustellen. Hierbei wurde ein Strömungsmodell verwendet, um die Simulation des Effektes in einem Voxelgitter durchzuführen. Zum Render des Volumendatensatzes wurde ein Raycasting Ansatz von [KW03] verwendet. Sowohl die Simulation als auch das Raycasting können komplett auf der Grafikkarte durchgeführt werden. Zum Schluss wurde noch ein Vergleich mit dem Partikelsystem gemacht und versucht, so die Vor- und Nachteile des volumenbasierten Ansatzes herauszuarbeiten.

Es setzte sich der Eindruck durch, dass wenn ein Effekt benötigt wird, bei dem Partikel als einzelne Objekte wahrnehmbar sein sollen (z. B. Regentropfen), dann ist das Partikelsystem die bessere Wahl. Soll jedoch eine Simulation verwendet werden, die eine Gitterstruktur benötigt, ist der volumenbasierte Ansatz sinnvoller. Natürlich sind auch hybride Ansätze denkbar. Durch immer leistungsfähigere Computer und Grafikkarten, sowie der steigenden Menge an Arbeitsspeicher, scheint in der Spieleindustrie das Interesse an Voxel-Engines wieder zu steigen [Ber09], was auch für den Einsatz der volumenbasierten Verfahren spricht.

Literatur

- [AS09] AMENT, MARCO und WOLFGANG STRASSER: *Dynamic Grid Refinement for Fluid Simulations on Parallel Graphics Architectures*. In: *Eurographics 2009*, 2009.
- [Ber09] BERTUCH, MANFRED: *Klötzchenwelten*. c't, (4), 2009.
- [c09] C: *Marching Cubes*. http://de.wikipedia.org/wiki/Marching_Cubes, 2009. (Letzter Zugriff 27.06.2009).
- [CLT08] CRANE, KEENAN, IGNACIO LLAMAS und SARAH TARIQ: *Real-Time Simulation and Rendering of 3D Fluids*. In: *GPU Gems 3*. Addison-Wesley, 2008.
- [FSJ01] FEDKIW, RONALD, JOS STAM und HENRIK W. JENSEN: *Visual Simulation of Smoke*. In: *SIGGRAPH 2001*, 2001.
- [Har04] HARRIS, MARK J.: *Fast Fluid Dynamics Simulation on the GPU*. In: *GPU Gems 1*. Addison-Wesley, 2004.
- [htt09] [HTTP://EN.WIKIPEDIA.ORG/WIKI/VOXEL](http://en.wikipedia.org/wiki/Voxel): *Voxel*. <http://en.wikipedia.org/wiki/Voxel>, 2009. (Letzter Zugriff 29.06.2009).
- [IKLH04] IKITS, MILAN, JOE KNISS, AARON LEFOHN und CHARLES HANSEN: *Volume Rendering Techniques*. In: *GPU Gems 1*. Addison-Wesley, 2004.
- [KW03] KRÜGER, JENS und RÜDIGER WESTERMANN: *Acceleration Techniques for GPU-based Volume Rendering*. In: *IEEE Visualization 2003*, 2003.
- [LOj09] LAURIEN, ECKART und HERBERT ORTEL JR.: *Numerische Strömungsmechanik*. Vieweg + Teubner, 2009.
- [NFJ02] NGUYEN, DUC Q., RONALD FEDKIW und HENRIK W. JENSEN: *Physically Based Modeling and Animation of Fire*. In: *SIGGRAPH 2002*, 2002.
- [Ngu04] NGUYEN, HUBERT: *Fire in the "Vulcan" Demo*. In: *GPU Gems 1*. Addison-Wesley, 2004.
- [Nvi] NVIDIA: *CUDA Zone – The resource for CUDA developers*. http://www.nvidia.com/object/cuda_home.html. (Letzter Zugriff 27.06.2009).
- [Sta99] STAM, JOS: *Stable Fluids*. In: *SIGGRAPH 99*, 1999.
- [Wat00] WATT, ALAN: *3D Computer Graphics*, Kapitel 13. Addison-Wesley, 2000.
- [Wes90] WESTOVER, LEE: *Footprint evaluation for volume rendering*. In: *SIGGRAPH 90*. ACM, 1990.
- [Wik09a] WIKIPEDIA: *Particle system*. http://en.wikipedia.org/wiki/Particle_system, 2009. (Letzter Zugriff 24.06.2009).

- [Wik09b] WIKIPEDIA: *Partikelsystem*. <http://de.wikipedia.org/wiki/Partikelsystem>, 2009. (Letzter Zugriff 29.06.2009).
- [Wik09c] WIKIPEDIA: *Schwarzer Körper*. http://de.wikipedia.org/wiki/Schwarzer_K%C3%B6rper, 2009. (Letzter Zugriff 29.06.2009).
- [WW92] WATT, ALAN und MARK WATT: *Advanced Animation and Rendering Techniques – Theory and Practice*, Kapitel 13. Addison-Wesley, 1992.