1 Inhaltsverzeichnis

2 Einführung 2

2.1 Aufgabenstellung 3

3 Grundlagen 4

3.1 Molekulardynamikdaten 4

3.2 Ambient Occlusion 4

4 Related Work 6

4.1 Ray Tracing 6

4.2 Ambient Occlusion Fields 7

4.3 Shadow Maps 8

4.4 SSAO/SSDO 9

5 Objekt orientiertes Ambient Occlusion Verfahren 11

5.1 Volumen 12

5.2 AO-Berechnung 16

5.3 Steuerung 21

5.4 Einschränkungen 23

6 Bewertung 24

6.1 Grafikfehler 24

6.1.1 Theoretische Fehler 24

6.1.2 Praktische Fehler 26

6.2 Geschwindigkeit 28

6.2.1 nialout50_00599.crist 28

6.2.2 b2_cut.chkpt 29

6.2.3 exp2mill.f25.siff 30

7 Fazit 32

8 Literaturverzeichnis 34

2 Einführung

Molekulardynamiksimulationen im Bereich der Physik, Thermodynamik oder Materialforschung erzeugen Datensätze aus zehntausenden bis zu mehreren Millionen Atomen. Die Visualisierung dieser Daten ist mit einigen umfangreichen Optimierungen möglich. Allerdings erzeugen diese Datensätze Bilder mit sehr hoher visueller Komplexität, in denen Strukturen der Atome nur sehr schwer zu erkennen sind. Ansätze, um dieses Problem zu lösen, basieren sehr oft auf Verfahren, die die Tiefe in den Datensätzen hervorheben. Mögliche Verfahren sind hierbei Fogging, Silhouetten zeichnen oder Schatten darstellen. Diese Verfahren müssen jedoch bei sehr feinen Datenstrukturen sehr genau gesteuert werden damit die Strukturen sichtbar bleiben. Aus diesem Grund sollte in dieser Studienarbeit ein weiterer Ansatz implementiert werden: Ambient Occlusion.

Ambient Occlusion ist eine Beleuchtungstechnik, welche zum ersten Mal in der Arbeit von Zhukov et al. (Zhukov, Iones, & Kronin, 1998) vorgestellt wurde. Es wird verwendet, um mit relativ wenig Rechenaufwand eine realistische Beleuchtung zu erzeugen. Ambient Occlusion basiert auf der Annahme, dass in Spalten oder Ecken normalerweise eine geringere Beleuchtungsstärke vorherrscht. Dies ist auf die vielen diffusen Reflexionen in diesen Stellen zurückzuführen. Aufgrund dieser Annahme können entsprechende Stellen abgedunkelt werden, ohne Informationen über die gesamte Szene zu verwenden. Dadurch entsteht ein realistischer Tiefeneindruck. Allerdings ist das Ergebnis physikalisch nicht korrekt. Die Näherung reicht jedoch aus, um eine realistische Verschattung zu erzeugen, ohne auf rechenintensive globale Berechnungen zurückgreifen zu müssen.

Die erste Veröffentlichung dieser Technik war noch für große Renderingprojekte oder die Filmindustrie gedacht und demzufolge auch nicht echtzeitfähig. Seit damals wurde dieses Verfahren jedoch soweit verbessert, dass es in einer großen Anzahl an Echtzeitanwendungen Verwendung findet.

In dieser Studienarbeit wurde ein effizienter Algorithmus erstellt, der Ambient Occlusion auf sehr großen Molekulardynamikdatensätzen in Echtzeit ermöglicht. Hiermit soll ein realistischer Tiefeneindruck auf Bildern dieser Daten erzielt werden. Der Algorithmus muss ohne Vorberechnungen auskommen.

Das Verfahren ist für das Programm MegaMol (Grottel, Reina, Dachsbacher, & Ertl, 2010) erstellt worden. MegaMol ist ein Visualisierungstool für punktbasierte Molekulardatensätze. In Abbildung 2-1 ist das Programm mit einer einfachen Szene zu sehen. Bisherige Rendering Methoden für MegaMol hatten entweder keine Technik zur Verdeutlichung der Tiefe implementiert, oder der Tiefeneindruck war wie bei Fogging oder Screen Space Ambient Occlusion (SSAO) nicht ausreichend ausgeprägt. Deswegen wurde in diesem Verfahren ein Objekt orientiertes Ambient Occlusion (OOAO) verwendet, womit der Tiefeneindruck besser verdeutlicht wurde.

MegaMol wurde in dieser Studienarbeit um ein Rendering-Plug-in erweitert. In diesem Rendering-Plug-in muss das gesamte Rendering der Molekulardynamikdaten durchgeführt werden, wobei die Szene durch Ambient Occlusion einen realistischen Tiefeneindruck erhalten hat.

Die bisherige Ambient Occlusion Ansätze und Verfahren scheiden wegen der Komplexität und Größe der Daten aus, bzw. liefern unzureichende Ergebnisse. Da die Datensätze dynamisch sind, muss das Rendering in Echtzeit ausgeführt werden.

Es wurde ein neuer Ansatz zu Ambient Occlusion entwickelt, welcher auch für mehrere Millionen Atome ein ansehnliches Ergebnis erzeugt, ohne Vorberechnungen auskommt und noch echtzeitfähig ist.

2.1 Aufgabenstellung

In dieser Studienarbeit wurde untersucht, wie die Methode „Ambient Occlusion“ auf mittelgroße, zeitabhängige Datensätze angewandt werden kann, ohne vorberechnete Daten laden zu müssen. Ansätze hierzu bilden räumliche Datenstrukturen, welche eine Sichtbarkeits- bzw. Verschattungs-Untersuchung in der lokalen Nachbarschaft der Partikel beschleunigen. Auch der Einsatz von GPGPU-Schnittstellen, wie OpenCL, um diese Berechnungen oder Teile davon auf die Grafikkarte auszulagern, war denkbar. Zeitabhängige Daten sollten nur bis zu einer Größe hin untersucht werden, bei der die ganzen Daten im Hauptspeicher des Computers gehalten werden können. Das Laden vom Sekundärspeicher wurde damit aus dieser Studienarbeit ausgeklammert.

Der Bearbeiter der Studienarbeit sollte zunächst eine Literatur-Recherche durchführen, um bisherige Ansätze für die Berechnung „Ambient Occlusion“ zusammen zu tragen. Anschließend sollte eine Bewertung dieser Ansätze basierend auf der Anwendbarkeit auf Molekulardynamikdatensätze hin erfolgen und der vielversprechendste Ansatz unter Einbringung eigener Ideen im Visualisierungs-Tool „MegaMol“ implementiert werden.

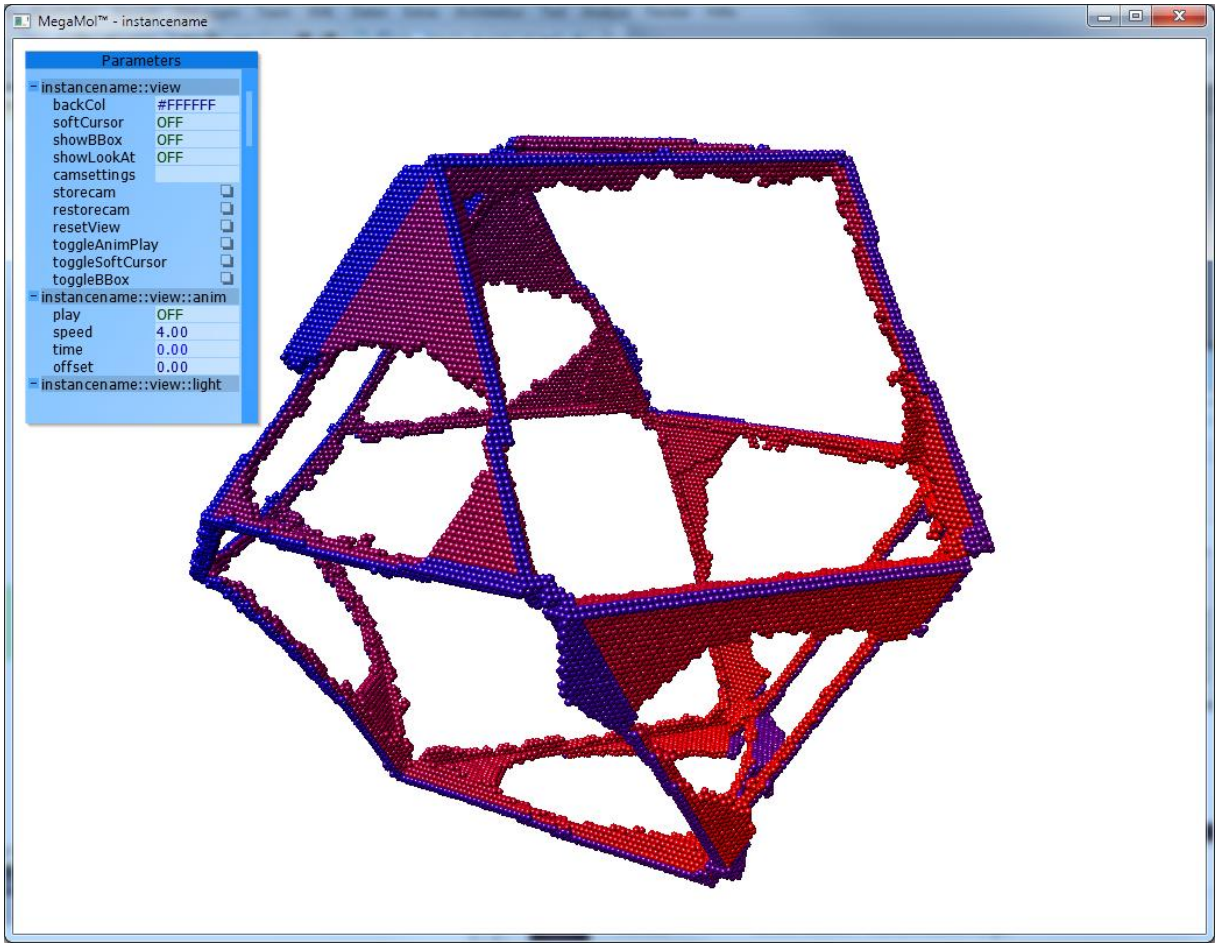

Abbildung 2-1: Programm „MegaMol“ mit dem Standart-Renderer. Datensatz: "dislocation.imd"

3 Grundlagen

3.1 Molekulardynamikdaten

Bei Molekulardynamikdatensätzen handelt es sich um Datensätze, die das Verhalten von Atomen oder Molekülen beschreiben. Diese Daten werden aus wissenschaftlichen Beobachtungen oder Simulationen gewonnen. Sie enthalten mindestens die Position der einzelnen Atome. Je nach Art der Daten können auch noch weitere Informationen wie beispielsweise Radius, Masse, elektrische Ladung oder die Orientierung eines Dipols des Atoms enthalten sein. Um die Animation der Atome zu speichern, werden pro Frame die geänderten Daten gespeichert. In der Regel bestehen die Daten aus zehntausenden bis mehreren Millionen Atomen. Sie sind also sehr groß.

Bei der Visualisierung werden normalerweise die einzelnen Moleküle durch Kugeln repräsentiert. Ein spezielles Visualisierungstool, das für die Darstellung von großen Datensätzen optimiert wurde, kann diese Datensätze darstellen.

3.2 Ambient Occlusion

Ambient Occlusion dient dazu in der Szene eine realistische Beleuchtung zu erzeugen. Speziell in Spalten und Ecken ist in der Realität die Beleuchtung geringer. Dies kommt von den vielen diffusen Reflexionen an den entsprechenden Stellen. In der Computergrafik wird dieses nachgebildet um einen realistischen Tiefeneindruck der Szene zu erzeugen.

Bei Ambient Occlusion wird berechnet, wie viel Licht aus jeder Richtung zu einem Atom kommen kann. Diese Berechnung muss für jedes Atom der Molekulardynamikdaten durchgeführt werden. Mit dieser Information kann bei der Beleuchtung der Szene entschieden werden, wie viel Licht zu einer bestimmten Stelle des Atoms kommen kann. Entsprechend kann diese Stelle abgedunkelt werden.

Die einfachste Methode Ambient Occlusion zu erzeugen ist, die Verschattungswerte für jedes Atom vor zu berechnen und zu speichern. Um diese Vorberechnung durchzuführen, gibt es verschiedene Techniken. Eine Implementierung dieser Technik, die ohne einen Vorberechnungsschritt auskommt, ist in (Bunnell, 2005) beschrieben. Eine Weiterentwicklung dieser Technik ist in (Hoberock & Jia, 2007) beschrieben. Diese Weiterentwicklung verbessert das Ergebnis und entfernt einige sichtbare Fehler in der Verschattung.

Beim Rendering kann man mit minimalem Aufwand auf diese vorberechneten Daten zugreifen und den Lichteinfall aus der entsprechenden Richtung entnehmen. Mit dieser Information kann während der Beleuchtung der Szene entschieden werden wie stark ein Pixel eines Atoms beleuchtet werden muss.

Diese Technik würde ein Bild mit perfektem Ambient Occlusion erzeugen, da an den Verschattungsfaktoren beliebig lange gerechnet werden kann. Das Rendering des Datensatzes würde bei statischen Daten in Echtzeit laufen, weil während des Renderings nur auf die vorberechneten Daten zugegriffen werden muss.

Bei dynamischen Daten ist die Sache komplizierter. Diese können zwar auch vorberechnet werden, aber dies benötigt viel mehr Zeit und Speicher, denn die Berechnung muss für jeden Frame durchgeführt werden. Zwischen zwei Frames wird normalerweise interpoliert. Hierdurch kann man einige Frames sparen, indem man auf exakte Werte verzichtet und die interpolierten Werte

verwendet. Allerdings muss die Anzahl der berechneten Frames dennoch hoch sein, damit diese Interpolation nicht sichtbar ist.

Bei der Darstellung des Datensatzes gibt es ebenfalls ein Problem: Die vorberechneten Verschattungsfaktoren müssen pro Frame zur Grafikkarte geschickt werden, bei Interpolation die Verschattungsfaktoren der zwei Frames, zwischen denen interpoliert wird. Das sind also doppelt so viele Daten. Wenn die Faktoren nur an sechs Stellen des Atoms berechnet und die restlichen Werte auf der Grafikkarte interpoliert werden, dann werden für die Übertragung der vorberechneten Faktoren mindestens sechs weitere Bytes pro Atom benötigt. Dies gilt allerdings nur, wenn für die Genauigkeit der Faktoren ein Byte pro Faktor ausreicht. Um ein Atom rendern zu können wird mindestens die Position des Atoms und der Radius benötigt. Diese werden in vier Floats übertragen, was 16 Bytes entspricht. Mit den Ambient Occlusion Faktoren müssen also 22 Bytes pro Atom übertragen werden. Dies entspricht mindestens 37,5 % mehr Daten, die übertragen werden müssen. Da bei größeren Datenmengen jedoch die Übertragung der Daten zur Grafikkarte die limitierende Größe ist, wird durch diesen Zuwachs an Daten dieses Problem weiter vergrößert.

4 Related Work

Um Ambient Occlusion mit Molekulardynamikdaten durchzuführen, gibt es schon verschiedene Ansätze. Einige dieser Ansätze werden hier vorgestellt.

4.1 Ray Tracing

Diese Möglichkeit Ambient Occlusion zu berechnen basiert auf einem Ray Tracer. Eine Möglichkeit eines einfachen Ray Tracers wurde in der Arbeit von Cook et al. (Cook, Porter, & Carpenter, 1984) vorgestellt.

Bei einem Ray Tracer werden mehrere Strahlen verschossen, um die Sichtbarkeit eines Pixels zu bestimmen. Diese Technik wurde entwickelt, um die gesamte Beleuchtung einer Szene zu berechnen. Allerdings kann man diesen Ansatz auch verwenden, um die Sichtbarkeit eines Pixels zu bestimmen. Wenn mehrere Strahlen pro Pixel in verschiedene Richtungen getestet werden, kann festgestellt werden, wie viel Licht auf den Pixel treffen kann. Eine Anwendung für Ambient Occlusion wurde in (Pharr, 2004) beschrieben. Hierbei wird in einem Vorberechnungsschritt für viele Punkte auf der Oberfläche des zu verschattenden Modells berechnet, wie stark der Punkt von anderer Geometrie verdeckt ist. Diese Information wird im anschließenden Renderschritt dazu verwendet, um die diffuse Beleuchtung zu berechnen. Ein Bild des Verfahrens ist in Abbildung 4-1 zu sehen.

Bei den Molekulardynamikdaten kann der gleiche Ansatz verwendet werden. Hier muss für jedes Atom berechnet werden, wie stark es verdeckt ist. Diese Berechnung kann mithilfe von mehreren Strahlen durchgeführt werden. Die Anzahl und Variation der Richtung der Strahlen sollte so gewählt werden, dass eine möglichst gleichmäßige und ausreichend dichte Abtastung des entsprechenden Abschnitts der Szene ermöglicht wird. Dies ist nötig, da nur Atome erkannt werden, die sich mit einem Atom schneiden. Atome, die zwischen Strahlen liegen, werden nicht berücksichtigt. Um die Anzahl an Strahlen zu reduzieren, kann man weitere Optimierungen verwenden. Eine dieser Optimierungen ist die Verwendung von „dicken“ Strahlen. Hierdurch wird die Anzahl der Strahlen reduziert, jedoch wird der Berechnungsaufwand für die Kollision erhöht.

Der Strahlentest sollte für mehre Stellen auf der Oberfläche des Atoms berechnet werden. Mithilfe einer Interpolation kann man beim Rendering die Verschattung an den fehlenden Punkten berechnen. Allerdings muss jeder Strahl, der verschossen wird, gegen jedes Atom auf Kollision getestet werden. Das ist das Problem an diesem Ansatz: Es gibt zu viele Atome.

Tests und Hochrechnungen haben ergeben, dass dieses Verfahren bei einer naiven Implementierung mehrere Stunden Rechenzeit pro Bild benötigen würde. Mithilfe von Objekt-Hierarchien, wie einem BSP-Tree oder einer Partitionierung des Volumens der Atome, kann man den Gesamtaufwand der Berechnungen für die Schnitttests der einzelnen Strahlen reduzieren. Jedoch ist es fast unmöglich echtzeitfähige Frameraten zu erreichen. Deswegen eignet sich dieses Verfahren besser zur Vorberechnung der Verschattungsfaktoren.

Die Qualität des Bildes variiert hierbei mit der Anzahl der verschossenen Strahlen. Je mehr Strahlen pro Verschattungsfaktor verschossen werden, desto besser ist das Bild. Die Wahrscheinlichkeit, dass die Strahlen ein Atom übersehen und somit die Verschattungsfaktoren näher an der wirklichen Verschattung liegen, ist geringer.

Abbildung 4-1: Bild, welches mit Ambient Occlusion gerendert wurde. Das Ambient Occlusion wurden mit einem Ray Tracer vorberechnet. Dieses Bild stammt aus dem Buch GPU Gems (Pharr, 2004).

4.2 Ambient Occlusion Fields

Diese Technik wurde in der Arbeit von Kontkanen et al. (Kontkanen & Laine, 2005) vorgestellt. Hierbei wird für jedes Objekt der Szene ein Feld um das Objekt berechnet. Dieses Feld ist ein Quader, der das Objekt umschließt. In diesem Feld wird gespeichert, an welchen Stellen das Objekt andere Objekte verdeckt. Das Feld wird in einem Vorberechnungsschritt für jedes Objekt aufgestellt und in einer Cube-Map gespeichert. Beim Rendering der Szene wird jedes Objekt, das ein anderes Objekt verdeckt, durch eine „spherical Cap“ repräsentiert. Dies ist ein Bereich, der das verschattende Objekt einschließt. Um diesen Cap zu berechnen, wird auf die vorberechnete Cube-Map, welche die Verdeckung des Objekts enthält, zugegriffen. Solche Caps werden für alle Objekte der Szene berechnet. Mit dem Wissen, dass ein Punkt auf einer Oberfläche nur von Licht aus der positiven Halbkugel beleuchtet werden kann, kann man berechnen, wie viel Prozent dieser Halbkugel von Caps verdeckt wird. Diesen Wert kann man als Ambient Occlusion-Faktor für diesen Punkt verwenden. Auf diese Weise ist es möglich, in Echtzeit ein Objekt orientiertes Ambient Occlusion zu implementieren. Abbildung 4-2 zeigt ein Ergebnis dieser Methode.

Für den Einsatz mit Molekulardynamikdaten ist diese Technik jedoch nicht geeignet. Ambient Occlusion-Fields sind für komplexere Objekte entwickelt worden. Bei Kugeln wäre es wahrscheinlich schneller, wenn man auf die Felder komplett verzichtet und direkt die Kugel auf die Cap abbildet.

Um ein Atom zu verschatten, muss ein Cap für jedes andere Atom berechnet werden. Allerdings gibt es bei Molekulardynamikdatensätzen sehr viele Atome. Deswegen ist dies in Echtzeit nicht mehr möglich.

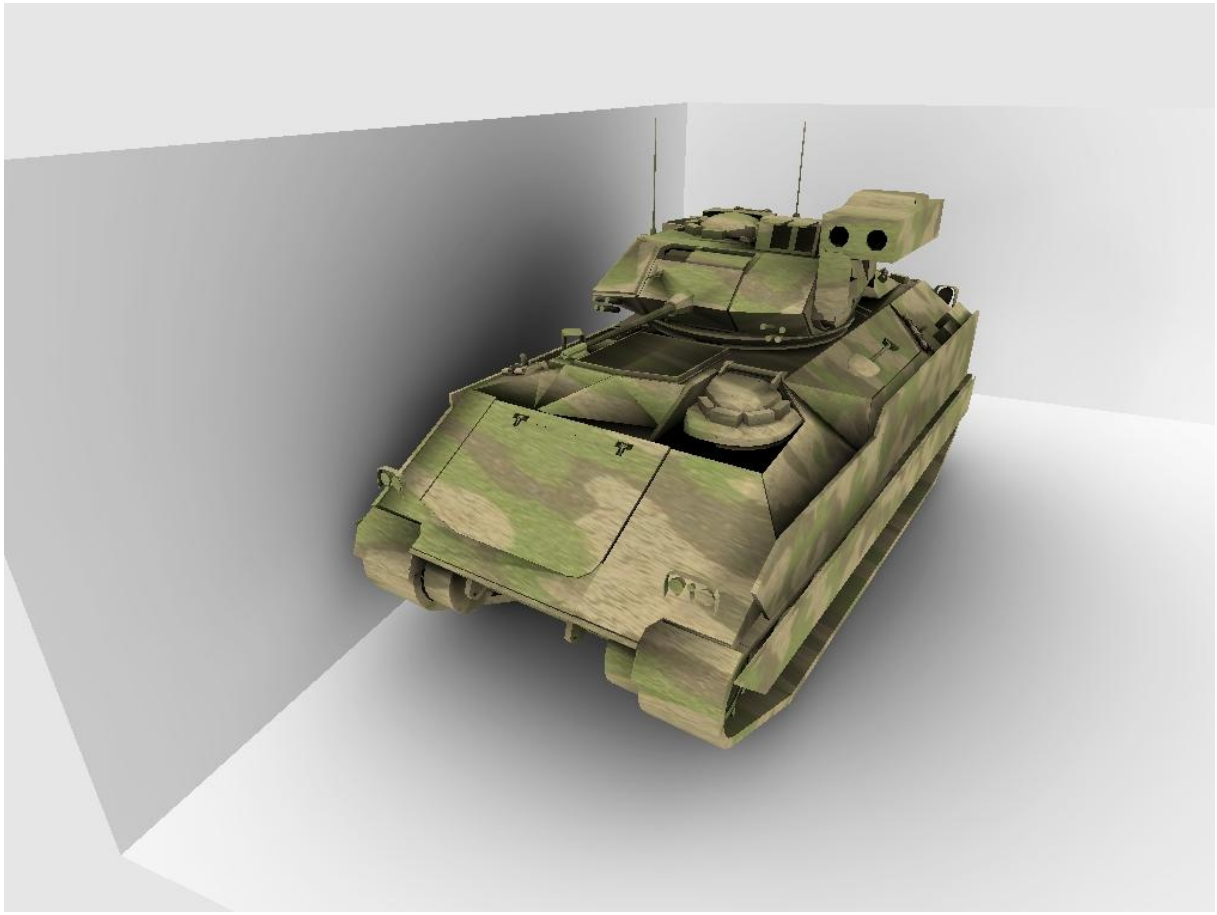

Abbildung 4-2: Ergebnis eines Verfahrens zur Berechnung von Ambient Occlusion. Hier wurde die Wand hinter dem Panzer abgedunkelt. Dieses Bild stammt aus der Arbeit (Kontkanen & Laine, 2005).

4.3 Shadow Maps

Mit Schatten kann man ebenfalls einen realistischen Tiefeneindruck vermitteln. Hierzu wird die gesamte Szene erst aus Sicht der Lichtquelle in eine ShadowMap gerendert. Anschließend wird mit dieser Shadow Map die Szene mit Schatten dargestellt. Bei diesem Verfahren benötigt man einen Renderpass, der die Shadow Map erzeugt und einen weiteren, der die Szene mit Schatten rendert. Welche Informationen die Shadow Map enthält, ist je nach Technik verschieden. Eine Zusammenfassung einiger Schattentechniken ist in der Arbeit von Hasenfratz et al. (Hasenfratz, Lapierre, Holzschuch, & Sillion, 2003) gegeben.

Eine herkömmliche Shadow Map enthält die geringste Entfernung eines Objekts zu einer Lichtquelle. Diese kann erzeugt werden, indem man die Szene aus Sicht der Lichtquelle rendert und die Tiefeninformation in die Shadow Map speichert. Hierfür ist ein Renderpass nötig. In einem weiteren Renderpass wird diese Shadow Map verwendet um Schatten in der Szene zu erzeugen. Hierzu muss für jeden gerenderten Pixel die Entfernung zur Lichtquelle berechnet werden. Ist diese Entfernung bekannt, kann diese mit dem Wert aus der Shadow Map verglichen werden. Ist die Entfernung kleiner oder gleich der aus der Shadow Map, wird der Pixel beleuchtet. Ist die Entfernung größer, liegt der Pixel im Schatten.

Eine Weiterentwicklung der einfachen Schadow Map Technik sind die Deep Shadow Maps. Diese Schattentechnik wurde in der Arbeit von Lokovic et al. (Lokovic & Veach, 2000) vorgestellt. Diese Technik speichert keine einfache Tiefe mehr in der Shadow Map, sondern eine Sichtbarkeitsfunktion.

Mit dieser Sichtbarkeitsfunktion ist es möglich, den Schatten von halbtransparenten Oberflächen zu berechnen. Ein Ergebnis der Schattenberechnung ist in Abbildung 4-3 zu sehen.

Abbildung 4-3: Ball aus 50 000 Haaren, welche mit Deep Shadow Maps gerendert wurden. Dieses Bild stammt aus der Arbeit (Lokovic & Veach, 2000).

Eine weitere Verbesserung der Schatten sind die Variance Shadow Maps. Diese wurden in der Arbeit von Donnelly et al. (Donnelly & Lauritzen, 2006) vorgestellt. Variance Shadow Maps stellen weiche Schatten dar. Dies wird erreicht, weil in der Variance Shadow Map neben der einfachen Entfernung auch die quadratische Entfernung gespeichert wird. Bei der Verwendung dieser Shadow Map kann mittels einer Formel entschieden werden, ob und wie stark ein Pixel im Schatten liegt. Neben der Verbesserung des Aussehens des Schattens gibt es eine weitere Verbesserung. Bei Variance Shadow Maps kann man ein Mip-Mapping anwenden, ohne die Information der Shadow Map zu verfälschen.

Zusätzlich zu den hier vorgestellten Techniken zu Shadow Maps gibt es noch weitere Verfahren, speziell zu Soft Shadow Maps.

Dieses Verfahren ist echtzeitfähig, denn es wird insgesamt nur eine zusätzliche Renderphase benötigt. Da eine Shadow Map jedoch nur eine begrenzte Auflösung hat, treten Probleme auf. Ein Atom wird auf mindestens einen Pixel in der Shadow Map gerendert (sofern kein Culling angewandt wird). Ist das Atom jedoch kleiner als ein Pixel, wirft es im finalen Renderpass einen zu großen Schatten. Die Folge ist ein fehlerhafter Schattenwurf. Bei Molekulardynamikdaten gibt es normalerweise sehr viele kleine Atome, welche über die Szene verteilt sind. Deswegen sind Shadow Maps für die Darstellung von Molekulardynamikdaten nicht gut geeignet.

4.4 SSAO/SSDO

Ambient Occlusion kann auch mittels Screen Space Ambient Occlusion (SSAO) erzeugt werden. In dieser Technik wird Ambient Occlusion im Screen Space berechnet. Das Verfahren wurde in der Arbeit (Shanmugam & Arikan, 2007) und (Mittring, 2007) vorgestellt.

Bei SSAO wird Ambient Occlusion im Bildraum berechnet. Hierzu werden die Tiefen der Szene und das finale Bild in einer Textur gespeichert. In einem anschließenden Renderpass wird Ambient Occlusion in das Bild eingerechnet. Hierzu wird für jeden Pixel die Nachbarschaft in der Tiefentextur

analysiert um Höhenunterschiede festzustellen. Je nach Größe des Höhenunterschieds eines Pixels zu seinen Nachbarpixeln wird dieser abgedunkelt.

Die Berechnungen sind einfach und schnell durchzuführen. Ein Vorteil bei diesem Verfahren ist es, dass die Berechnungen nicht auf Objektebene, sondern auf der Bildebene durchgeführt werden. Der Aufwand skaliert nicht mit der Anzahl der Objekte, sondern nur mit der Auflösung des Bildes. Abbildung 4-4 zeigt einen Screenshot von Crysis, welches diese Technik einsetzt.

Abbildung 4-4: Screenshot von Crysis, welcher SSAO demonstriert. Dieses Bild stammt aus der Arbeit von (Mittring, 2007).

Eine Verbesserung des Ergebnisses von SSAO bietet Screen Space Directional Occlusion (SSDO). Hier wird nicht nur die Tiefe eines Pixels berücksichtigt, sondern auch die Richtung des einfallenden Lichts. Dies wurde von Ritschel et al. (Ritschel, Grosch, & Seidel, 2009) publiziert. Hiermit kann neben Ambient Occlusion auch eine einfache Reflexion von indirektem Licht berechnet werden.

Beide Varianten des Verfahrens benötigen einen sehr geringen Aufwand um Ambient Occlusion zu berechnen, sind also echtzeitfähig. SSAO/SSDO ist für Molekulardynamikdaten einsetzbar. SSAO wurde auch für MegaMol implementiert. Da SSAO/SSDO jedoch auf der Bildebene arbeitet, kann kein so gutes Ergebnis erzielt werden wie bei einem Verfahren, welches direkt auf der Objektebene arbeitet. Dies resultiert daraus, dass bei einem Verfahren im Bildraum Informationen über die direkte Nachbarschaft verloren gehen können, welche jedoch im Objektraum berücksichtigt werden können.

5 Objekt orientiertes Ambient Occlusion Verfahren

Das hier vorgestellte Verfahren verwendet eine einfache Technik, um die Molekulardynamikdaten mit Ambient Occlusion darzustellen.

Das Verfahren verwendet zwei Renderpasses, um die Molekulardynamikdaten darzustellen. Der erste Pass stellt das Volumen auf und speichert die Ergebnisse in einer Textur. Der zweite Renderpass rendert das Bild. Die Atome der Molekulardynamikdaten werden als Kugeln dargestellt. Die Darstellung erfolgt mit Ambient Occlusion, welches die zuvor berechnete Volumentextur verwendet, um die Ambient Occlusion-Faktoren direkt im Shader zu berechnen. Abbildung 5-1 verdeutlicht diese Schritte.

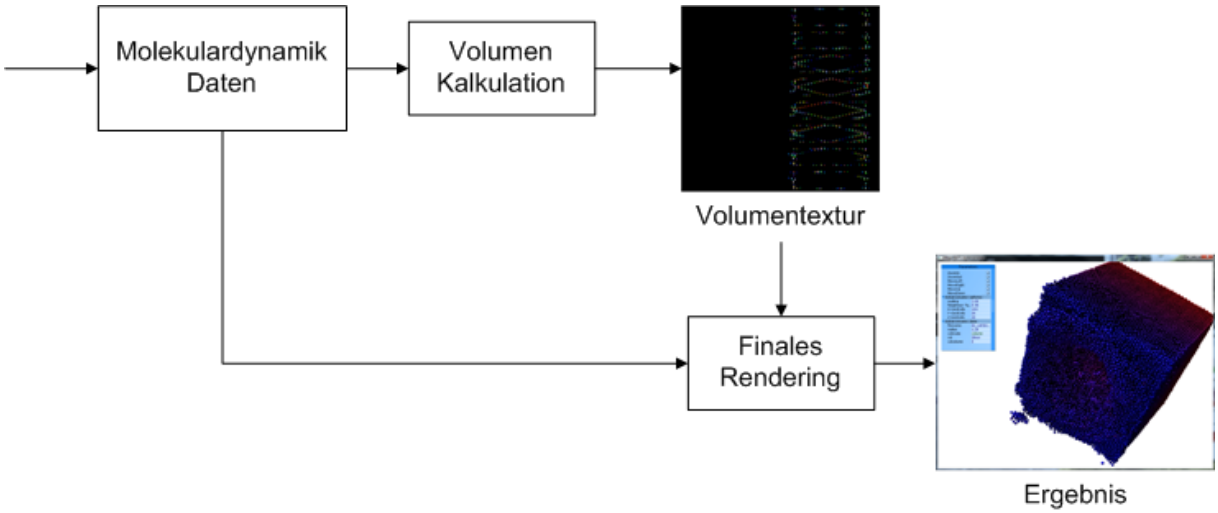

Abbildung 5-1: Darstellung der einzelnen Berechnungsschritte des Verfahrens.

Da sehr viele Atome dargestellt werden müssen, wird für das Ambient Occlusion eine Annäherung verwendet. Die Ambient Occlusion-Faktoren werden nur an sechs Punkten der Sphäre berechnet. Auf der restlichen Oberfläche der Kugel werden die Ambient Occlusion-Werte interpoliert. Hierdurch sind keine Kanten in der Verschattung auf einer einzelnen Kugel sichtbar. Diese Interpolation kann direkt in einem Shader durchgeführt werden. Die einzelnen Faktoren eines Atoms bestimmen, wie viel Licht aus der entsprechenden Richtung auf das Atom fällt.

Die sechs berechneten Ambient Occlusion Faktoren befinden sich genau auf den einzelnen Achsen. Also an +X, -X, +Y, usw.. Um die sechs Verschattungsfaktoren in Echtzeit zu berechnen, wird der gesamte Raum, in dem sich die Molekulardynamikdaten befinden, in Volumenzellen aufgeteilt. Für jedes Atom wird berechnet, in welcher Volumenzelle es sich befindet. Mit dieser Information kann die prozentuale Füllung in den einzelnen Volumenzellen berechnet werden. Dieser Wert wird als Verschattungsfaktor für die Atome in den Nachbargitterzellen verwendet. Hierzu wird angenommen, dass die Atome in einer Volumenzelle homogen verteilt sind und deswegen auch die Menge des Lichts, das durch eine Volumenzelle dringt, proportional zur Füllung der Volumenzelle ist.

Bei der Berechnung der Ambient Occlusion-Werte für ein Atom wird auf die Nachbargitterzellen der Gitterzelle des Atoms zugegriffen um den Verschattungsfaktor zu erhalten. In Abbildung 5-2 sind die Volumenzellen zu sehen, die von einer Volumenzelle beeinflusst werden.

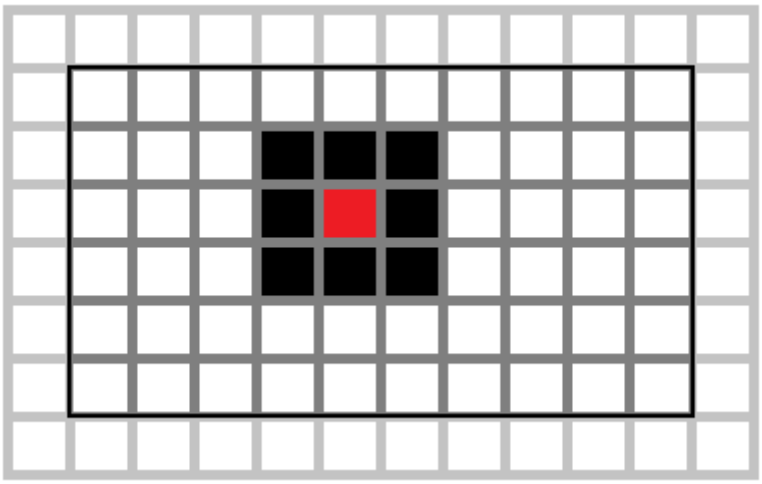

Abbildung 5-2: 2D Volumen mit den Volumenzellen (schwarz), die von einer Volumenzelle (rot) aus beeinflusst werden.

Mit dieser Technik lassen sich die Ambient Occlusion- Faktoren der Atome in zwei Renderschritten berechnen: Die Aufstellung des Volumens und das Rendern des Bildes.

Das Verfahren wurde als Rendering-Plug-in für MegaMol implementiert. Hierfür wurde der Standard Renderer „SimpleSphere“ um die Ambient Occlusion Berechnung aus diesem Verfahren erweitert.

5.1 Volumen

Das Volumen dient dazu, die Molekulardynamikdaten in einzelne Zellen aufzuteilen. Dies ist die Berechnungsgrundlage für dieses Verfahren.

Das Volumen wird in einer Textur gespeichert, in welcher jeder Pixel eine Volumenzelle repräsentiert. Diese Textur kann anschießend verwendet werden, um auf die einzelnen Volumenzellen zuzugreifen.

Da bei diesem Verfahren auf die Nachbarzellen von jedem Atom zugegriffen werden muss, hat man das Problem, dass bei Randzellen auf eine falsche Zelle zugegriffen werden würde. Dieser Fehler entsteht, weil das Volumen auf ein 1D Array abgebildet wird und somit die einzelnen Volumenzellen in dem Array hintereinander liegen. Dies kann man durch eine Abfrage bei der späteren Berechnung abfangen. Jedoch würde dies in vielen if-Abfragen in einem Shader enden und die Performance des Verfahrens würde darunter leiden. Alternativ kann eine leere Zelle, also eine Zelle, die außerhalb der maximalen Ausdehnung der Molekulardynamikdaten liegt und deswegen kein Atom enthält, am Rand eingefügt werden. Hierbei erhöht sich die Anzahl der Zellen auf jeder Achse um zwei Zellen (eine am Anfang und eine am Ende der Achse). In Abbildung 5-3 ist der Fehler zu sehen, wenn auf die zusätzlichen Randzellen (hellgraue Zellen) verzichtet wird. Die blaue Zelle ist die Zelle, von der aus auf die Nachbarzellen zugegriffen wird. Auf die gelben Zellen wird mit und ohne die zusätzlichen, leeren Randzellen korrekt zugegriffen. Wenn jedoch auf die zusätzlichen Randzellen verzichtet wird, wird fälschlicherweise auf die roten Zellen zugegriffen. Mit den zusätzlichen Randzellen wird auf die Randzellen (die grünen Zellen) zugegriffen.

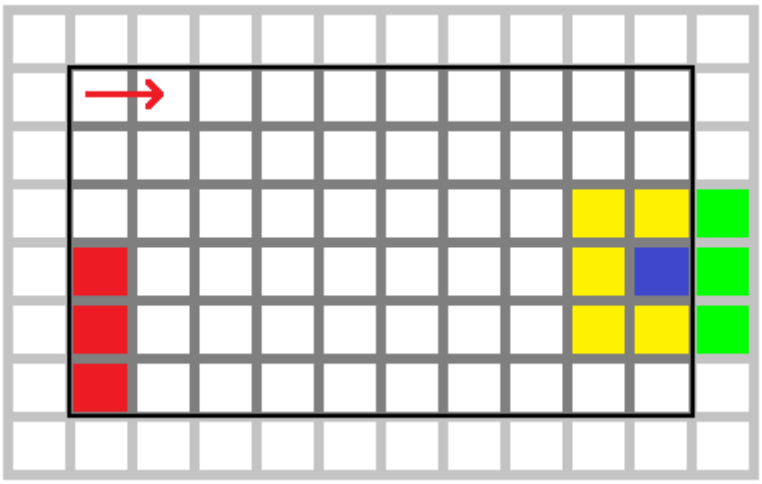

Abbildung 5-3: 2D Volumen in einem Raum. Das schwarze Rechteck ist der Raum und die dunkelgrauen Rechtecke das erzeugte Volumen. Die hellgrauen Rechtecke sind leere Volumenzellen, die in das Volumen eingefügt wurden um für jede gefüllte Volumenzelle Nachbarzellen zu haben. Zusätzlich ist der Fehler zu sehen, der bei einem Zugriff mit und ohne Randzellen entsteht.

Die Textur, in welcher das Volumen gespeichert wird, muss mindestens so viele Pixel enthalten wie es Volumenzellen gibt. Dabei werden in jeder Achse zwei zusätzliche Volumenzellen gebraucht. Die Anzahl der benötigten Pixel kann mit Formel 1 berechnet werden.

Formel 1: Berechnung der Anzahl der Volumenzellen die benötigt werden.

$$AnzGitterzellen = (AnzahlZellen.x + 2) * (AnzahlZellen.y + 2) * (AnzahlZellen.z + 2)$$

Um das Volumen aufzustellen, wird für jedes Atom aus den Molekulardynamikdaten berechnet, in welcher Volumenzelle es sich befindet. Diese Berechnung wird aufgrund der Position des Atoms im Raum durchgeführt. Für die Berechnung wird der gesamte Raum so verschoben, dass der Minimalwert auf den Nullpunkt fällt. Diese so erhaltene Position wird durch die Größe der Volumenzellen geteilt. Die Volumenzellen erhalten also eine normierte Größe. Der Ganzzahlenwert des Ergebnisses gibt an, in welcher Zelle sich das Atom befindet.

Formel 2: Berechnung der Volumenzelle für ein Atom

$$Gitterzelle = \left\lfloor \frac{Atom - Min}{Zellgröße} \right\rfloor$$

Hier ist der GLSL-Code, der die Volumenzelle für ein Atom berechnet:

```
vec3 calcGirdPos( vec3 fAtom,
                  vec3 fMin,
                  vec3 fSeperation)
{
      // Identify the Region
      vec3 pos = fAtom - fMin;
      vec3 sep = pos / fSeperation;
      return trunc(sep);
}
```

fAtom ist hierbei die Position des Atoms, fMin ist die minimale Position des Volumens und fSeperation ist die Größe einer Gitterzelle.

Nachdem die Volumenzelle des Atoms berechnet wurde, muss berechnet werden, welchem Pixel in der Textur diese Volumenzelle entspricht. Hierzu wird erst die Position des Atoms in einem eindimensionalen Array berechnet. Diese Position wird anschließend auf eine 2D Textur umgerechnet.

Eine eindeutige Abbildung der 3D Volumenposition auf den 1D Array erhält man, indem erst die Position der Y-Z-Ebene berechnet wird. Anschließend wird die Position der Y-Linie in dieser Ebene berechnet und zum Schluss die Position des Z-Punktes auf der Linie. Da jedoch eine zusätzliche Volumenzelle am Anfang und Ende jeder Achse benötigt wird, muss dies hier berücksichtigt werden.

Formel 3: Berechnung der Array-Position einer Volumenzelle mit einer leeren Volumenzelle am Anfang und Ende einer Achse

$$arrayPos = (AnzahlZellen.y + 2) * (AnzahlZellen.z + 2) * (Gitterzelle.x + 1) + (AnzahlZellen.z + 2) * (Gitterzelle.y + 1) + (Gitterzelle.z + 1)$$

Hier ist der GLSL-Code, der diese Berechnung durchführt, wobei die Anzahl der Volumenzellen schon um zwei pro Achse erhöht wurde.

```
float calcGirdArrayPos( vec3 gird,
                        vec3 nGirdCells)
{
      return      nGirdCells.y*nGirdCells.z*(gird.x+1)+
                  nGirdCells.z*(gird.y+1)+
                  (gird.z+1);
}
```

Da nun bekannt ist, auf welcher Position in einem 1D Array sich die Volumenzelle befindet, muss diese noch in die Pixelposition auf der Textur umgerechnet werden. Dies geschieht mit Formel 4.

Formel 4: Formeln zur Berechnung der Pixelposition

$$Y = \left\lfloor \frac{arrayPos}{texDim.x} \right\rfloor$$

$$X = arrayPos - texDim.x * Y$$

Der entsprechende GLSL-Code sieht so aus:

```
vec2 calcGirdTexCoords( float nArrayPos,
                        vec2 texDim)
{
      float y = trunc(nArrayPos / texDim.x);
      float x = nArrayPos - texDim.x * y;
      return vec2(x,y);
}
```

Um auf diese Texturposition zu rendern, muss genau ein Pixel an der errechneten Position gerendert werden. Der entsprechende GLSL-Code ist hier gegeben:

```
gl_Position = vec4(tex.x*texDim.z-1.0,tex.y*texDim.w-1.0,0.0,1.0);

gl_PointSize = 1.0;
```

texDim.z und texDim.w sind hier eins dividiert durch die Größe der Textur in X- und Y-Richtung.

Es fehlt jedoch noch, was an dieser Position gespeichert werden soll. Dieses Verfahren benötigt pro Volumenzelle die prozentuale Füllung dieser Zelle. Dies kann erreicht werden, indem jedes Atom berechnet, wie viel Prozent es von der Volumenzelle füllen würde. Anschließend wird dies mit additivem Blending an die errechnete Position der Volumenzelle in der Textur gerendert, also addiert. Initial muss jeder Wert in der Textur null sein. Auf diese Weise wird direkt in der Textur die gesamte Füllung der einzelnen Volumenzellen berechnet.

Die einzelnen Atome werden mit Kugeln repräsentiert. Mithilfe der Volumenformel für eine Kugel (Formel 5) und des Gesamtvolumens der Volumenzelle (Formel 6) wird berechnet, zu wie viel Prozent die Kugel die Volumenzelle füllt (Formel 7). Dieser Wert wird anschließend zur bisherigen Füllung der Volumenzelle addiert.

Formel 5: Volumen einer Kugel

$$VolumenKugel = \frac{4}{3}\pi r^3$$

Formel 6: Volumen einer Volumenzelle

$$VolumenZelle = Zellgröße.x * Zellgröße.y * Zellgröße.z$$

Formel 7: Prozentuale Füllung einer Volumenzelle mit einer Kugel

$$Füllung = \frac{VolumenKugel}{VolumenZelle}$$

Nach dem Rendern aller Atome erhält man eine Textur, welche für jede Volumenzelle die prozentuale Füllung dieser Zelle enthält. Aus dieser Textur können im folgenden Renderschritt die Ambient-Occlusion-Werte gewonnen werden. Abbildung 5-4 zeigt ein Beispiel einer so erzeugten Volumentextur.

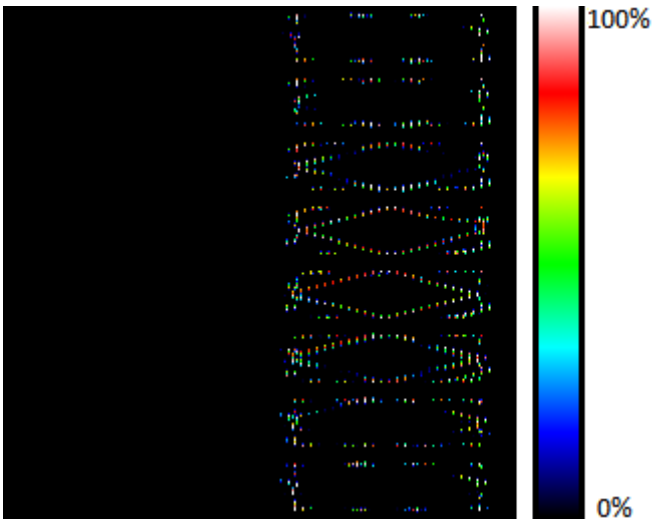

Abbildung 5-4: Volumentextur vom Datensatz "dislocation.imd" mit 30 Zellen pro Achse

5.2 AO-Berechnung

Als Berechnungsgrundlage hierfür wird die Füllung der Nachbargitterzelle des Atoms genommen. In einem früheren Renderschritt wurde berechnet, zu wie viel Prozent die einzelnen Volumenzellen gefüllt sind. Das Ergebnis wurde in einer Textur gespeichert. Diese Information wird nun verwendet, um zu berechnen, wie viel Licht auf das Atom fällt.

Für jedes Atom wird also erst berechnet, in welcher Volumenzelle es sich befindet. Anschließend wird für jeden der sechs Faktoren die Nachbargitterzelle gesucht und daraus die Lichtmenge, die aus dieser Richtung kommt, bestimmt. Diese errechnet sich aus $1 - Füllung$ der Volumenzelle in der entsprechenden Richtung. Abbildung 5-5 stellt dies in 2D dar. Die Füllung der Volumenzellen wird aus der vorher berechneten Textur ausgelesen.

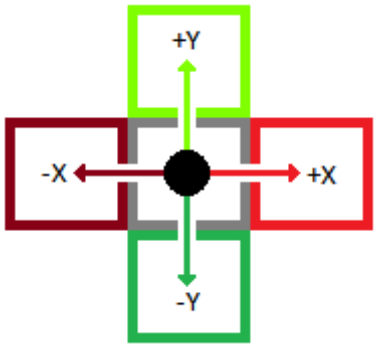

Abbildung 5-5: 2D Darstellung der Nachbargitterzellen, die zur Berechnung der Ambient Occlusion Faktoren eines Atoms verwendet werden.

Wird die Berechnung so durchgeführt, wird der gleiche Faktor für ein Atom berechnet. Der Ort eines Atoms in der Volumenzelle spielt dabei keine Rolle. Im Gesamtbild wäre dann die Gitterstruktur in der Verschattung der Atome zu erkennen. Um dies zu vermeiden, werden eine bilineare und eine lineare Interpolation durchgeführt.

Zunächst wird eine Interpolation zwischen den vier nächstliegenden Nachbargitterzellen des Atoms durchgeführt. Diese Interpolation wird sowohl für die Ebene, in der sich die Volumenzelle befindet, als auch für eine weitere Ebene in Richtung des gesuchten Faktors berechnet. Anschließend wird zwischen den beiden Werten, welche aus der Interpolation in den beiden Ebenen kommen, interpoliert.

Als Interpolationsgrundlage für die erste Interpolation, die zwischen den vier nächstliegenden Volumenzellen in einer Ebene berechnet wird, wird die Position des Atoms in der Volumenzelle genommen.

Für die Interpolation muss erst entschieden werden, zwischen welchen Volumenzellen interpoliert wird. Diese Entscheidung wird aufgrund der Position des Atoms zum Mittelpunkt der Volumenzelle getroffen. Hierfür muss die Position des Atoms innerhalb der Volumenzelle berechnet werden. Dies geschieht, indem die Position der Volumenzelle des Atoms multipliziert mit der Ausdehnung einer Volumenzelle von der Position des Atoms subtrahiert wird. In welcher Volumenzelle sich das Atom befindet, kann mit Formel 2 berechnet werden. Die Berechnung der Position eines Atoms in einer Volumenzelle ist in Formel 8 gegeben.

Formel 8: Berechnung der Position eines Atoms in einer Volumenzelle

$$AtomPos = Atom - Min - Gitterzelle * Zellgröße$$

Nachdem diese Position mit der inversen Volumenausdehnung skaliert wurde, also im Bereich von null bis eins ist, kann mit einem Test, ob die Position größer oder kleiner 0.5 ist, entschieden werden, in welchem Bereich der Volumenzelle sich das Atom befindet.

Die Berechnung der Position und die anschließende Skalierung werden mit folgendem GLSL-Code durchgeführt, wobei invGirdSize eins dividiert durch die Größe einer Gitterzelle ist:

```
vec3 calcGirdFactor(    vec3 fAtom,
                        vec3 fMin,
                        vec3 fSeperation,
                        vec3 invGirdSize)
{
      // Identify the Region
      vec3 pos = fAtom - fMin;
      vec3 sep = pos / fSeperation;
      return (pos-trunc(sep)*fSeperation)*invGirdSize;
}
```

Diese Entscheidung wird für die zwei freien Achsen durchgeführt. Die dritte Achse ist durch die Richtung des Faktors festgelegt. Abbildung 5-6 zeigt, welche Nachbargitterzellen zur Interpolation herangezogen werden. Die X-Achse ist festgelegt, also muss nur auf der Y-Achse entschieden werden zwischen welchen zwei Nachbarzellen interpoliert werden muss. In der linken Zelle ist ein Atom in seiner Volumenzelle dargestellt. In der rechten Ebene sind die Nachbarzellen markiert, zwischen denen interpoliert werden muss.

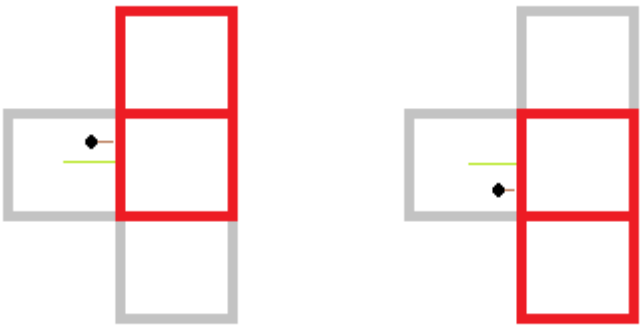

Abbildung 5-6: 2D Darstellung zwischen welchen Volumenzellen interpoliert werden muss (rote Volumenzellen). Als Entscheidungsgrundlage dient hierbei die Position des Atoms (schwarzer Punkt).

Nachdem nun bekannt ist, welche Zellen für die Interpolation verwendet werden, fehlt noch die Gewichtung der Zellen. Hierfür wird die Entfernung des Atoms zum Rand und zum Mittelpunkt auf jeder der zwei freien Achsen berechnet. Die Entfernung vom Rand zum Mittelpunkt wird als eins angenommen. In Abbildung 5-7 ist dies für zwei freie Achsen zu sehen, wobei x und y, bzw. 1-x und 1-y die verwendeten Interpolationsfaktoren sind.

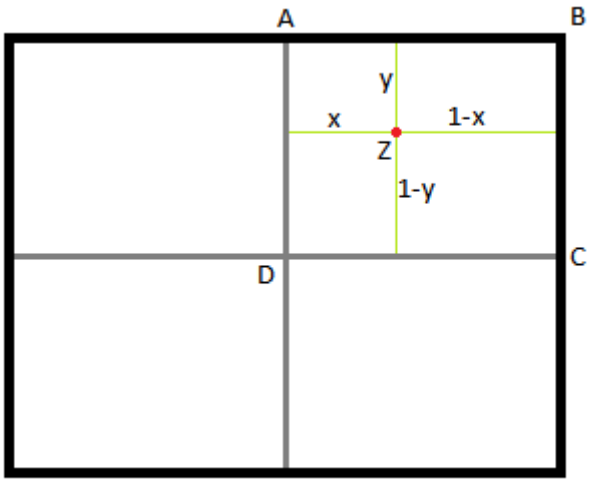

Abbildung 5-7: Interpolationsfaktoren eines Atoms (Roter Punkt) in einer Volumenzelle.

Interpoliert wird mit der bekannten bilinearen Interpolationsformel für vier Gewichte. Abbildung 5-7 und Formel 9 stellen diese bilineare Interpolationsformel dar.

Formel 9: Interpolationsformel für vier Werte in einem Viereck.

$$Z = (1 - x) * (1 - y) * A + x * (1 - y) * B + x * y * C + (1 - x) * y * D$$

Werden nur die interpolierten Faktoren als Ambient Occlusion Faktoren verwendet, kann es passieren, dass Kanten in der Verschattung auftreten. Diese entstehen, weil am Rand einer Zelle die direkte Nachbarzelle kaum noch Einfluss auf die Faktoren ausübt.

Um das zu verhindern, wird eine lineare Interpolation aus der Füllung der direkten Nachbarzelle und den interpolierten Werten verwendet. Formel 10 beschreibt die Kombination dieser zwei Faktoren. Diese Interpolation ist eine einfache Interpolation im eindimensionalen Fall. In Formel 11 und Abbildung 5-8 ist die lineare Interpolationsformel zu sehen.

Formel 10: Berechnung des Faktors. Hierbei wird der interpolierte Faktor mit dem Nachbarzellenfaktor kombiniert. Über das Gewicht kann eingestellt werden, wie stark die einzelnen Anteile gewichtet werden sollen.

$$Faktor = (1 - Gewicht) * InterpolFaktor + Gewicht * Nachbarzellenfaktor$$

Abbildung 5-8: Lineare Interpolation.

Formel 11: Lineare Interpolationsformel.

$$C = (1 - x) * A + x * B$$

Ein korrektes Ergebnis würde man bei einem Gewicht von 0.5, also eine Mittelung aus interpoliertem Faktor und dem Faktor der direkten Nachbarzelle, erhalten. Hier sind die Volumenzellen in der Verschattung nicht mehr sichtbar. Tests haben jedoch ergeben, dass bei manchen Szenen ein leicht abweichender Wert ein optisch besseres Ergebnis liefert. Abbildung 5-9 zeigt einen Vergleich zweier Bilder, bei denen ein unterschiedliches Gewicht verwendet wurde. Hier ist deutlich zu sehen, dass bei

dem Bild mit einem Gewicht von 0.5 (links) Artefakte zu erkennen sind, während diese bei einem Gewicht von 0.2 (rechts) nicht mehr so stark zu erkennen sind.

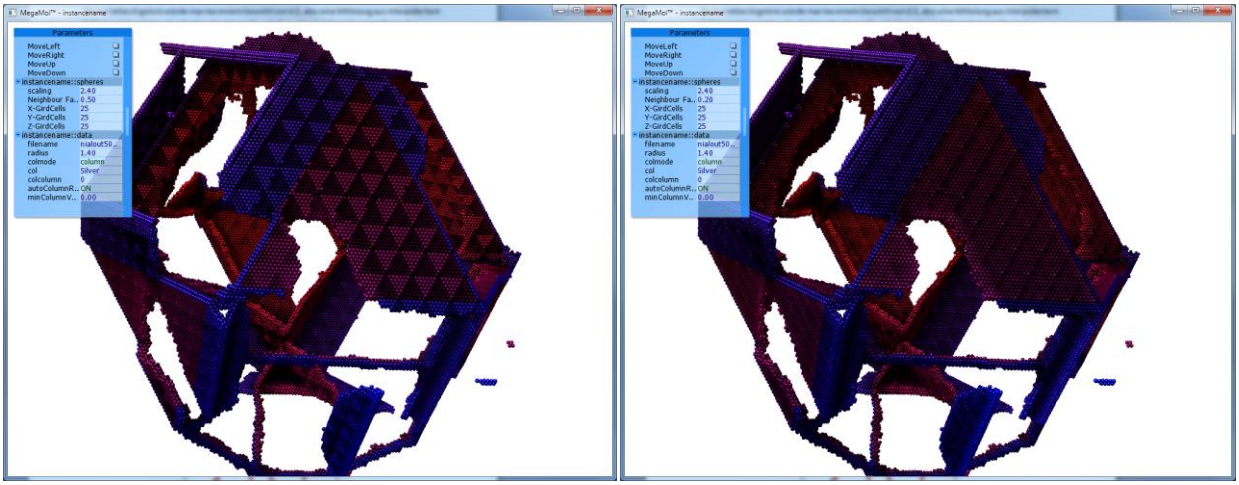

Abbildung 5-9: Vergleich der Ergebnisse mit unterschiedlichen Gewichten auf dem Datensatz "nialout50_00599.crist". Links: Gewicht 0.5. Rechts: Gewicht 0.2.

Nachdem diese Interpolation für die beiden Ebenen durchgeführt wurde, kann jetzt die zweite Interpolation durchgeführt werden. Mit dieser Interpolation wird die Position des Atoms auf der dritten Achse, also der Achse für die der Ambient Occlusion-Faktor berechnet werden soll, berücksichtigt. Hierzu wird die Position innerhalb der Volumenzelle als Grundlage genommen. Mit diesem Wert wird eine Interpolation zwischen den berechneten Werten der beiden Ebenen durchgeführt. Abbildung 5-10 zeigt die beiden Extremfälle, in denen jeweils nur eine einzige Ebene den Faktor bestimmt.

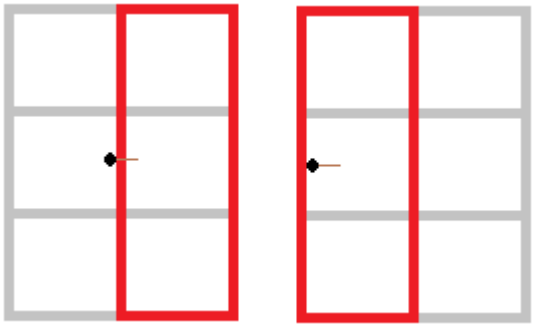

Abbildung 5-10: Darstellung, welche Ebene für die Verschattung in den Extrempunkten genommen wird.

Auf diese Weise wird berücksichtigt, dass die Ebene, in der sich die Volumenzelle des Atoms befindet, auch zur Verschattung beiträgt.

Mit dieser Methode werden die sechs Ambient Occlusion Faktoren jedes Atoms berechnet. Diese Berechnung kann direkt in einem Vertex Shader durchgeführt werden.

Um mit diesen sechs berechneten Faktoren einen gleichmäßigen Helligkeitsverlauf auf der Kugeloberfläche zu erzielen, muss für jeden gerenderten Pixel zwischen den drei Faktoren interpoliert werden, die dem Pixel auf der Kugel am Nächsten sind.

Um diese Interpolation durchführen zu können, muss entschieden werden, welche der berechneten Faktoren als Interpolationsgrundlage genommen werden. Einer dieser Faktoren ist immer auf der X-Achse, also der Faktor bei +X oder der bei –X, der Zweite immer auf der Y-Achse und der Dritte immer auf der Z-Achse. Um zu entscheiden, ob der Faktor bei +X/+Y/+Z oder bei -X/-Y/-Z genommen wird, wird die Normale des Pixels, für den der Ambient Occlusion-Faktor berechnet wird, herangezogen. Wenn die entsprechende Komponente der Normalen positiv ist, wird der Faktor bei +X/+Y/+Z gerechnet. Ist sie negativ, wird der Faktor bei -X/-Y/-Z gerechnet. Der folgende GLSL-Code führt diese Entscheidung für alle Achsen durch und gibt die entsprechenden Faktoren aus.

```
vec3 getAOFactor( const in vec3 normal,
                  const in vec3 AOFactPos,
                  const in vec3 AOFactNeg)
{
      vec3 AOFact;
      if(normal.x > 0.0)
            AOFact.x = AOFactPos.x;
      else
            AOFact.x = AOFactNeg.x;
      if(normal.y > 0.0)
            AOFact.y = AOFactPos.y;
      else
            AOFact.y = AOFactNeg.y;
      if(normal.z > 0.0)
            AOFact.z = AOFactPos.z;
      else
            AOFact.z = AOFactNeg.z;

      return AOFact;
}
```

Nachdem nun bekannt ist, zwischen welchen Faktoren interpoliert werden muss, kann die Interpolation durchgeführt werden. Hierfür wird wieder die Normale herangezogen.

Normierten Normalen sind eine Partition der Eins. Es gilt also Formel 12.

Formel 12: Pythagoras Formel bei normierten Normalen in 3D

$$1 = (N.x)^2 + (N.y)^2 + (N.z)^2$$

Dies kann verwendet werden um die Ambient Occlusion-Faktoren zu interpolieren. Die Ambient Occlusion-Faktoren der einzelnen Achsen können also mithilfe der Normalen so partitioniert werden, dass die einzelnen Komponenten korrekt gewichtet werden. Formel 12 kann leicht abgewandelt zur Berechnung der Interpolation der Ambient Occlusion-Faktoren herangezogen werden.

Formel 13: Berechnung des Ambient Occlusion-Faktors eines Pixels

$$Fact = AOFact.x * (N.x)^2 + AOFact.y * (N.y)^2 + AOFact.z * (N.z)^2$$

Formel 13 zeigt, wie die Interpolation der Ambient Occlusion-Faktoren der einzelnen Pixel auf der Kugel durchgeführt wird.

Der Faktor liegt im Bereich [0,1] und repräsentiert die Verschattung des Pixels. Dieser so berechnete Faktor wird mit der errechneten Farbe des Pixels multipliziert. Abbildung 5-11 zeigt das Ergebnis dieses Verfahrens.

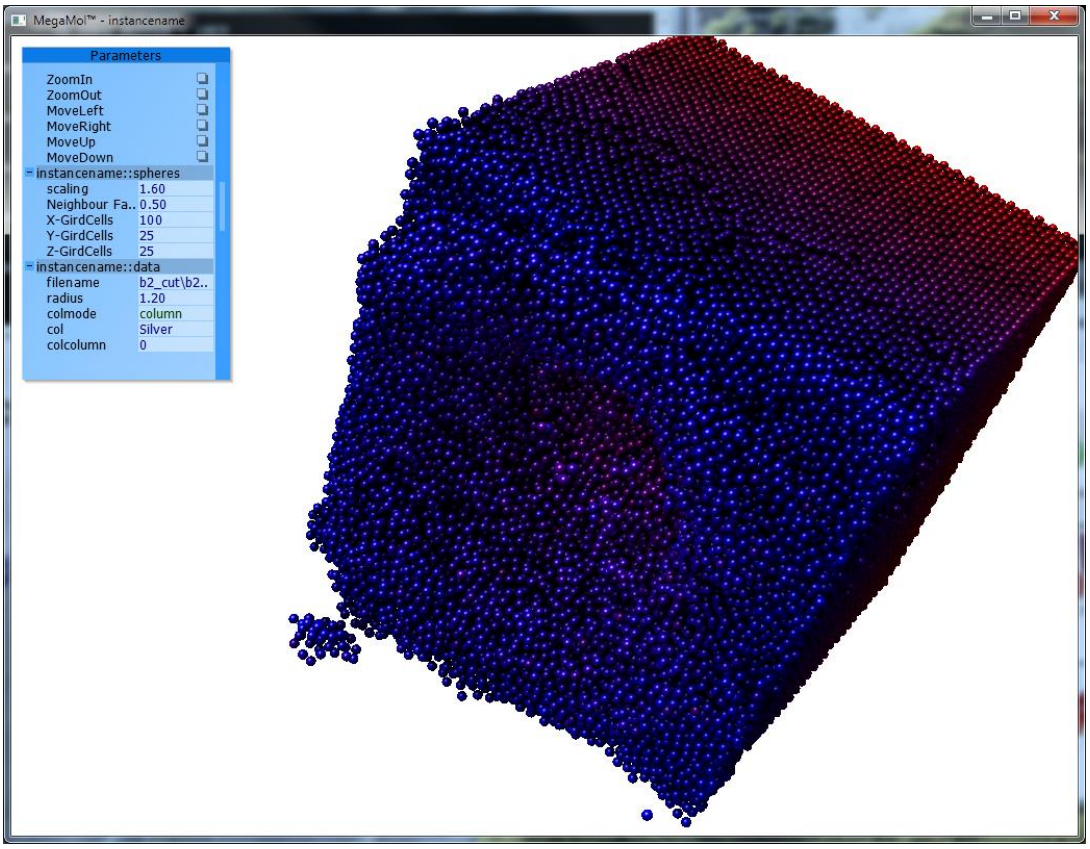

Abbildung 5-11: MegaMol Daten dargestellt mit Ambient Occlusion. Datensatz "b2_cut.imd" mit 25*25*100 Volumenzellen.

5.3 Steuerung

Zu den Steuermöglichkeiten von MegaMol sind drei weitere hinzugekommen. In Abbildung 5-12 sind die neuen Einstellungsmöglichkeiten markiert.

Abbildung 5-12: Neue Einstellmöglichkeiten für dieses PlugIn. Die Skallierung (Rot) skaliert den Radius der Kugeln für die Berechnung des Volumens. Der Faktor, wie stark die direkten Nachbarzellen Einfluss nehmen, wird mit dem Feld „Neighbour Factor“ (Gelb) eingestellt. Die Anzahl der Volumenzellen wird mittels der drei Parameter (Grün) für jede Achse eingestellt.

1. Für die Berechnung der Füllung der Volumenzellen kann der Radius der Sphären mit einem Faktor skaliert werden. In Abbildung 5-12 ist dies in Rot markiert. Hiermit kann die Stärke des Ambient Occlusion-Effekts gesteuert werden. In Abbildung 5-13 sind die Auswirkungen dieser Steuermöglichkeit zu sehen. Wie in dem linken, oberen Bild zu sehen ist schaltet null den Ambient Occlusion-Effekt aus, da alle Gitterzellen leer bleiben. Der Renderer verhält sich hierbei wie der Standard Renderer von MegaMol „SimpleSphere“. Je höher der Wert ist, desto stärker wirkt sich die Verdeckung der einzelnen Atome aus.

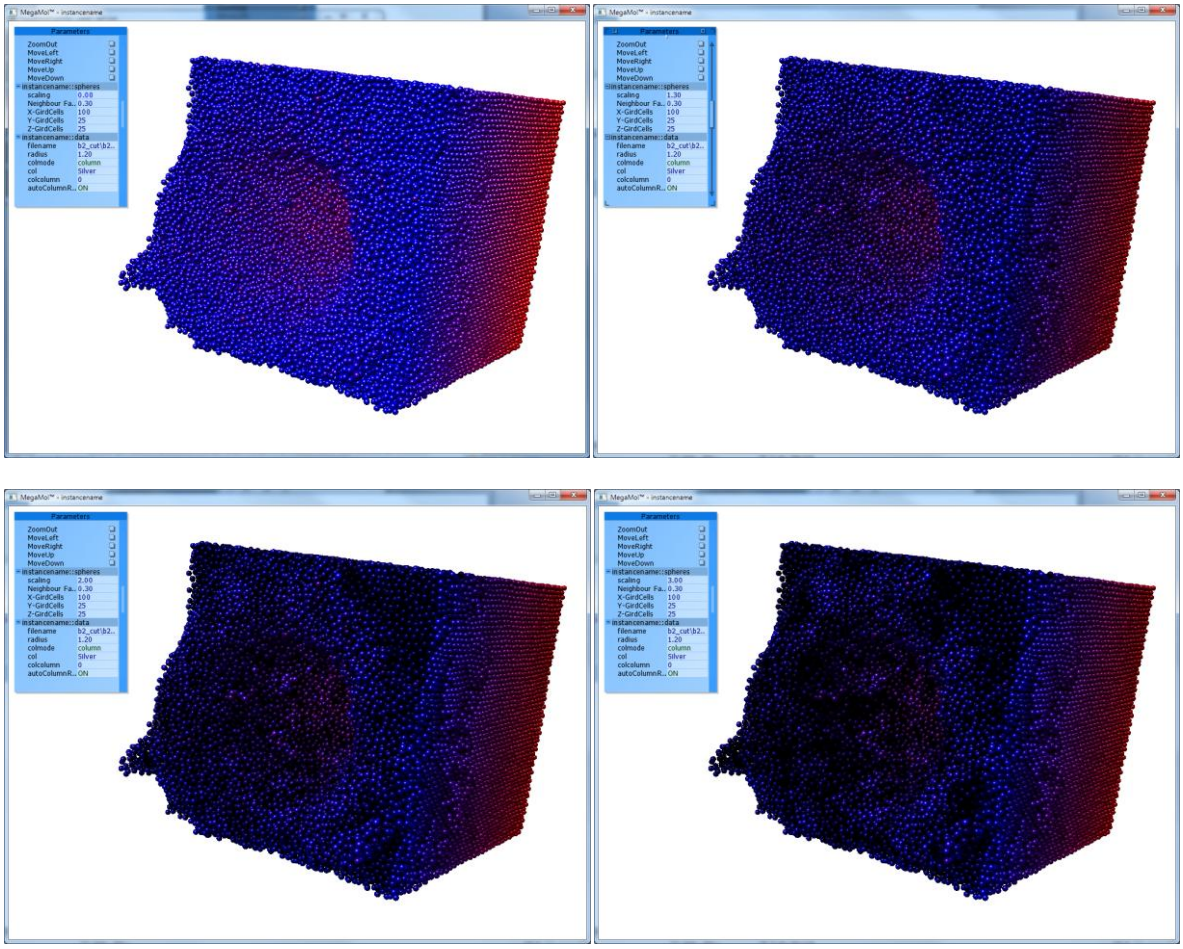

Abbildung 5-13: Auswirkungen bei verändern der Skalierung der Atomradien für die Ambient Occlusion Berechnung auf dem Datensatz „b2_cut.chkpt“. Links oben: Skalierung 0. Rechts oben: Skalierung 1.3. Links unten: Skalierung 2. Rechts unten: Skalierung 3.

2. Das Gewicht, mit dem die direkten Nachbarzellen in die Berechnung der Ambient Occlusion-Faktoren eingehen, kann geändert werden. Dies ist in Abbildung 5-12 in Gelb markiert. Abbildung 5-9 zeigt ein Beispiel der Auswirkungen dieses Parameters. Der Standartwert ist hier 0.5, da dieser Wert theoretisch korrekt ist und in den meisten Fällen ein gutes Ergebnis liefert.
3. Die Anzahl der Volumenzellen auf jeder Achse kann eingestellt werden. In Abbildung 5-12 ist dies Grün markiert. Je mehr Volumenzellen vorhanden sind, desto genauer wird Ambient Occlusion berechnet. Allerdings ist die Berechnung der Faktoren räumlich begrenzt, da nur die Nachbarzellen für die Berechnung herangezogen werden. Hier muss ein Kompromiss zwischen gewünschter Genauigkeit und Gesamteindruck bei der Einstellung gefunden werden. In Abbildung 5-14 sind die Auswirkungen beim Verändern der Anzahl an Volumenzellen zu sehen. Bei einer Zelle in X-Richtung (links oben) ist kein starker Ambient

Occlusion-Effekt mehr zu erkennen, da die Ambient Occlusion-Werte auf der X-Achse immer voll beleuchtet sind. Bei zehn Zellen (rechts oben) ist noch ein Absatz in der Verschattung zu erkennen. Hier ist die Anzahl an Zellen nicht ausreichend hoch. Bei 50 (links unten) und 100 Zellen (rechts unten) sieht man hingegen keine Fehler mehr. Hier ist die Auflösung ausreichend hoch gewählt.

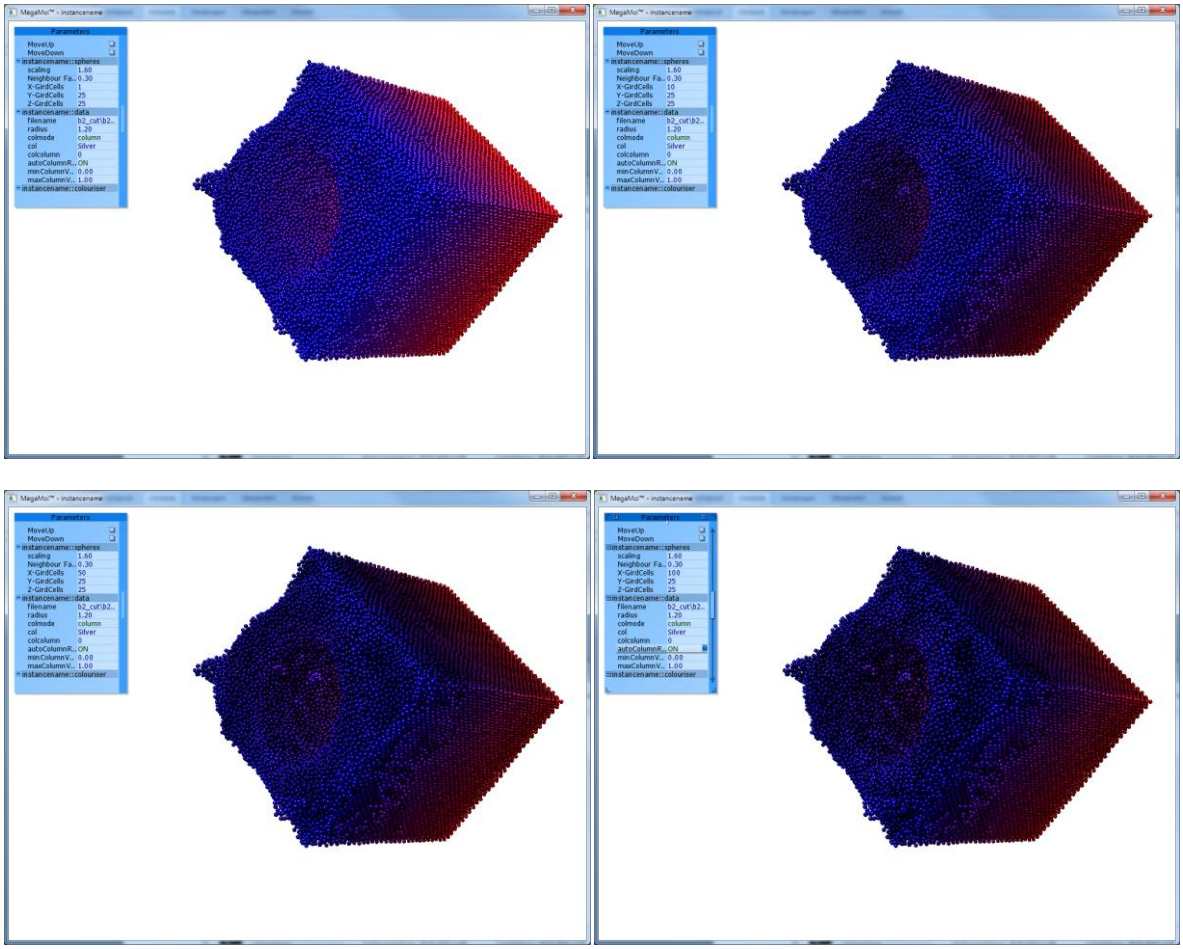

Abbildung 5-14: Auswirkungen der Veränderung der Anzahl an Volumenzellen auf dem Datensatz „b2_cut.chkpt“. In Y- und Z-Richtung wurde in je 25 Zellen geteilt. Die Anzahl der Zellen in X-Richtung wird Variiert. Links oben: X-GirdCells = 1. Rechts oben: X-GirdCells = 10. Links unten: X-GirdCells = 50. Rechts unten: X-GirdCells = 100.

5.4 Einschränkungen

Eine Einschränkung ist, dass die Textur für die Volumenberechnung nur eine begrenzte Auflösung besitzt. Bei aktueller Hardware sind dies $8192 * 8192 = 67108864 = 2^{26} = 2^2(2^8)^3$ Pixel. Dies limitiert die maximale Anzahl an Volumenzellen. Das Ergebnis von Formel 1 muss also kleiner oder gleich diesem Wert sein, da sonst nicht mehr alle Zellen untergebracht werden können. Allerdings reicht diese Auflösung aus, um in jeder Achse $2^8 = 256$ Zellen zu verwenden. Zusätzlich ist noch ein Faktor von $2^2 = 4$ übrig. Es können also noch mehr Zellen verwendet werden. Diese Einschränkung ist also nicht besonders wichtig, da die meisten Szenen mit weit weniger Zellen auskommen.

Eine weitere Einschränkung ist, dass für die Shader eine Grafikkarte, welche GLSL 1.3 unterstützt benötigt wird. Dies ist auf die trunc-Funktion zurückzuführen, welche bei der Berechnung der Volumenzelle eines Atoms benötigt wird. Die trunc-Funktion ist erst in GLSL 1.3 verfügbar.

6 Bewertung

Die Bewertung des Verfahrens wird nach zwei verschiedenen Kriterien durchgeführt: Qualität des Ergebnisses und Performance. Beide sind ausschlaggebend für die Leistungsfähigkeit des Verfahrens.

6.1 Grafikfehler

Obwohl das Verfahren in den meisten Fällen ein sehr gutes Ergebnis liefert, gibt es einige Ausnahmen. Diese Fehler lassen sich in zwei Kategorien einteilen: Theoretische- und praktische Fehler.

6.1.1 Theoretische Fehler

In diese Fehlerkategorie befinden sich alle Fehler, die zwar theoretisch falsche Werte liefern, sich aber auf die Qualität des Bildes nicht stark auswirken.

Einer dieser Fehler ist, dass Volumenzellen bei diesem Verfahren zu mehr als 100% gefüllt sein können. Dies kommt daher, dass die Radien der Atome vom Benutzer skaliert werden können, um ein qualitativ passendes Bild zu erzeugen. Hierdurch können sich einzelne Atome überschneiden, wodurch die maximal mögliche Füllung der Volumenzelle überschritten werden kann. Abbildung 6-1 verdeutlicht dies. Der Extremfall ist, dass sich viele Atome an der gleichen Position befinden. Hierdurch würde eine Volumenzelle sehr stark gefüllt werden, obwohl optisch nur eine einzelne Kugel sichtbar ist. Dieser Fall kann bei Molekulardynamikdaten jedoch nicht auftreten, da die einzelnen Atome in der Realität einen Mindestabstand zueinander haben. Es kann also nicht passieren, dass sich zwei Atome sehr nahe beieinander befinden.

Die dichteste Kugelpackung, also eine Packung, in der sich einzelne Atome nicht überschneiden, ist bei ungefähr 74% Füllung einer Volumenzelle. Formel 14 stellt die Berechnung dieses Wertes dar.

Formel 14: Berechnung der Füllung der dichtesten Kugelpackung ohne dass sich Kugeln überschneiden.

$$\frac{\pi}{3\sqrt{2}} \approx 0{,}74048 = 74\%$$

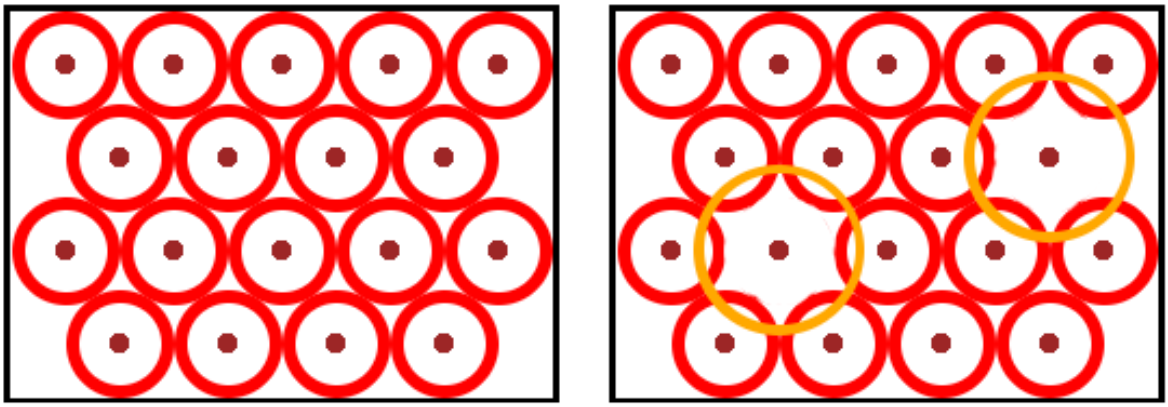

Abbildung 6-1: 2D Darstellung der Atome in einer Volumenzelle. Links: Dichteste Packung der Atome, ohne Überlappung der Sphären. Rechts: Atome mit Überlappung der Sphären.

Bei diesem Verfahren ist die maximale Füllung einer Volumenzelle jedoch auf 100% begrenzt. Werte darüber werden auf 100% festgesetzt. Dies entspricht keinem Lichtdurchlass durch die Zelle.

Wenn Atome sich nahe am Rand einer Volumenzelle befinden, kann es passieren, dass die Sphäre des Atoms sich mit der Volumenzelle schneidet. Wie in Abbildung 6-2 befindet sich ein Teil des Atoms in der Nachbarzelle. Dies wird bei diesem Verfahren nicht berücksichtigt. Ein Atom wird immer vollständig zu der Volumenzelle gerechnet, in der es sich befindet. Eventuelle Schnitte mit der Volumenzelle sind allerdings nicht wichtig, da zwischen zwei Nachbargitterzellen interpoliert wird. Hierdurch spielt der Fehler in der Berechnung der Füllung der Volumenzelle durch die Atomschnitte keine große Rolle mehr.

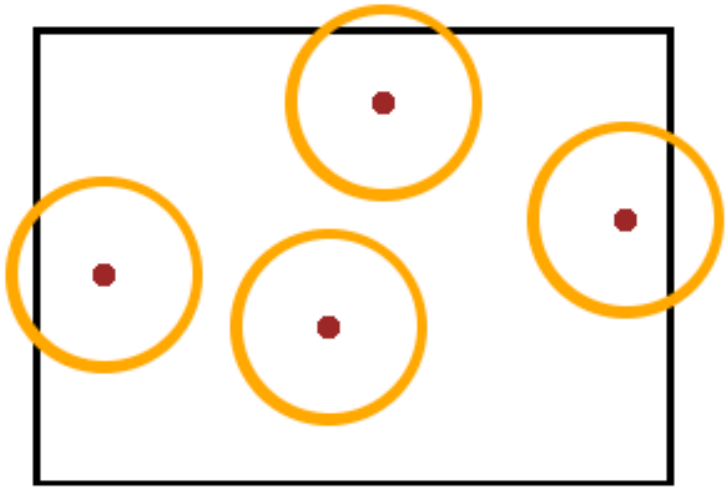

Abbildung 6-2: Atome schneiden sich mit der Volumenzelle.

Ein weiteres Problem ist, dass bei der Berechnung der Ambient Occlusion-Faktoren nur die direkten Nachbarzellen der Volumenzelle, in der sich das Atom befindet, berücksichtigt. Dadurch entstehen Fehler in der Verschattung. Um diese Fehler zu beheben, müssten immer alle Volumenzellen berücksichtigt werden. Das ist jedoch aus Performancegründen nicht möglich. Abbildung 6-3 zeigt ein Beispiel, in dem das Atom voll beleuchtet wird, obwohl theoretisch kein Licht zu ihm dringen kann.

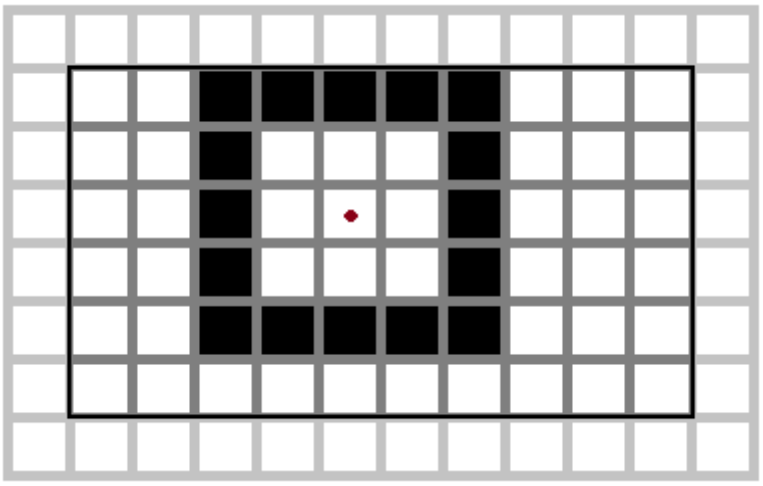

Abbildung 6-3: Atom (Roter Punkt) wird voll beleuchtet, obwohl theoretisch kein Licht zu ihm dringen kann, da es von vollständig gefüllten Volumenzellen eingehüllt ist.

6.1.2 Praktische Fehler

In dieser Kategorie sind Fehler, die man im Ergebnis des Verfahrens bemerkt.

Bei dem Verfahren wird davon ausgegangen, dass die Atome in einer Volumenzelle homogen verteilt sind. Ist dies nicht der Fall, können Fehler in der Berechnung der Ambient Occlusion-Faktoren entstehen. Ein Beispiel von nicht homogen gefüllten Volumenzellen ist in Abbildung 6-6 zu sehen. Hier sind die Atome in Ebenen angeordnet. Die Volumenzellen sind also nicht homogen gefüllt. Dies führt zu Fehlern in der Berechnung der Ambient Occlusion-Faktoren und ist an den dunklen Ecken in den Ebenen der Atome zu sehen (grün eingekreist).

Dieser Fehler entsteht, da Atome, welche sich im Zentrum einer Zelle befinden, nur von der Zelle selbst und den direkten Nachbarn beeinflusst werden. Atome, die sich jedoch am Rand einer Zelle befinden, werden durch die diagonale Zelle beeinflusst. In Abbildung 6-4 ist dies zu sehen. Hier bilden die Atome eine Linie, die Zellen sind also nicht homogen gefüllt. Das rote Atom wird nur von der Füllung seiner Zelle und den leeren Nachbarzellen beeinflusst. Es ist also heller. Das grüne Atom wird hingegen nur von der diagonalen Nachbarzelle beeinflusst. Hierdurch ist es dunkler.

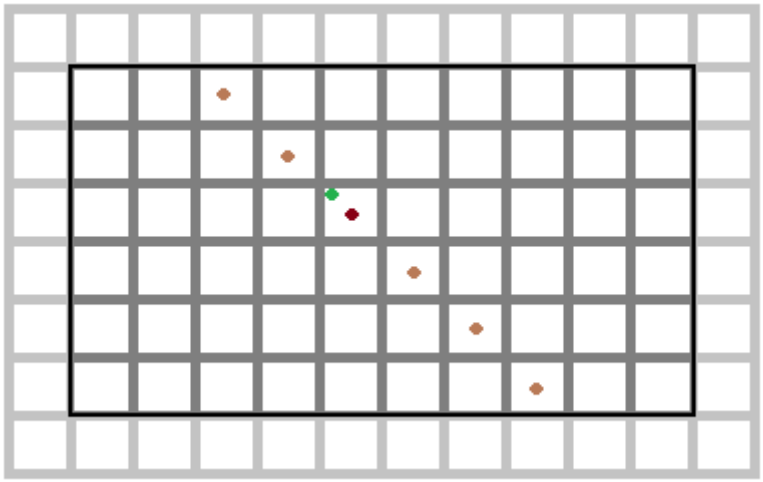

Abbildung 6-4: Erklärung des Fehlers bei nicht homogen gefüllten Zellen.

Abbildung 6-5 ist ein Beispiel, welches dies verdeutlicht. Es soll der Ambient Occlusion-Wert auf der negative X-Achse für das rote und grüne Atom berechnet werden. Es wird angenommen, dass alle Zellen mit einem Atom eine Füllung von 50% haben. Die Zellen Z_1 und Z_4 haben also einen Wert von 0.5. Die Zellen Z_2 und Z_3 sind leer, haben also einen Wert von 0. Das rote Atom befindet sich genau im Zentrum der Zelle an der Position $\begin{pmatrix}0.5\\0.5\end{pmatrix}$. Das grüne Atom befindet sich in der linken oberen Ecke an der Position $\begin{pmatrix}0\\1\end{pmatrix}$. Bei den Y-Gewichten G_{1y} und G_{2y} der Atome ist zu beachten, dass die Werte nur bis zur Mitte der Zelle genommen werden. Ansonsten würde zwischen anderen Zellen interpoliert werden.

Rotes Atom:

1. Berechnung der Nachbarfaktoren NF_1 und NF_2:

$$G_{1y} = 1$$

$$NF_1 = G_{1y} \cdot Z_3 + (1 - G_{1y}) \cdot Z_1 = 1 \cdot 0 + 0 \cdot 0.5 = 0$$

$$NF_2 = G_{1y} \cdot Z_4 + (1 - G_{1y}) \cdot Z_2 = 1 \cdot 0.5 + 0 \cdot 0 = 0.5$$

2. Berechnung des Ambient Occlusion-Faktors AOF_1:

$$G_{1x} = 0.5$$

$$AOF_1 = G_{1x} \cdot NF_2 + (1 - G_{1x}) \cdot NF_1 = 0.5 \cdot 0.5 + 0.5 \cdot 0 = 0.25$$

Grünes Atom:

1. Berechnung der Nachbarfaktoren NF_1 und NF_2:

$$G_{2y} = 0$$

$$NF_1 = G_{2y} \cdot Z_3 + (1 - G_{2y}) \cdot Z_1 = 0 \cdot 0 + 1 \cdot 0.5 = 0.5$$

$$NF_2 = G_{2y} \cdot Z_4 + (1 - G_{2y}) \cdot Z_2 = 0 \cdot 0.5 + 1 \cdot 0 = 0$$

1. Berechnung des Ambient Occlusion-Faktors AOF_2:

$$G_{2x} = 0$$

$$AOF_2 = G_{2x} \cdot NF_2 + (1 - G_{2x}) \cdot NF_1 = 0 \cdot 0 + 1 \cdot 0.5 = 0.5$$

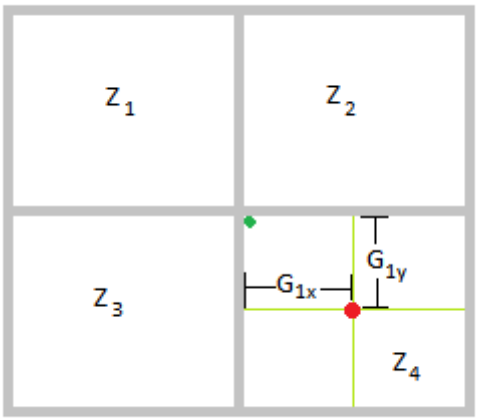

Abbildung 6-5: Beispiel zur Erzeugung von Artefakten bei homogen gefüllten Zellen.

An den Werten AOF_1 und AOF_2 sieht man, dass das rote Atom dunkler dargestellt werden würde als das grüne Atom. Für die anderen Ambient Occlusion-Werte gilt das Gleiche. Hieraus entstehen die Artefakte bei nicht homogenen Datensätzen.

Über die Einstellung des Gewichts für die Nachbarzellen kann dieser Effekt jedoch verringert werden.

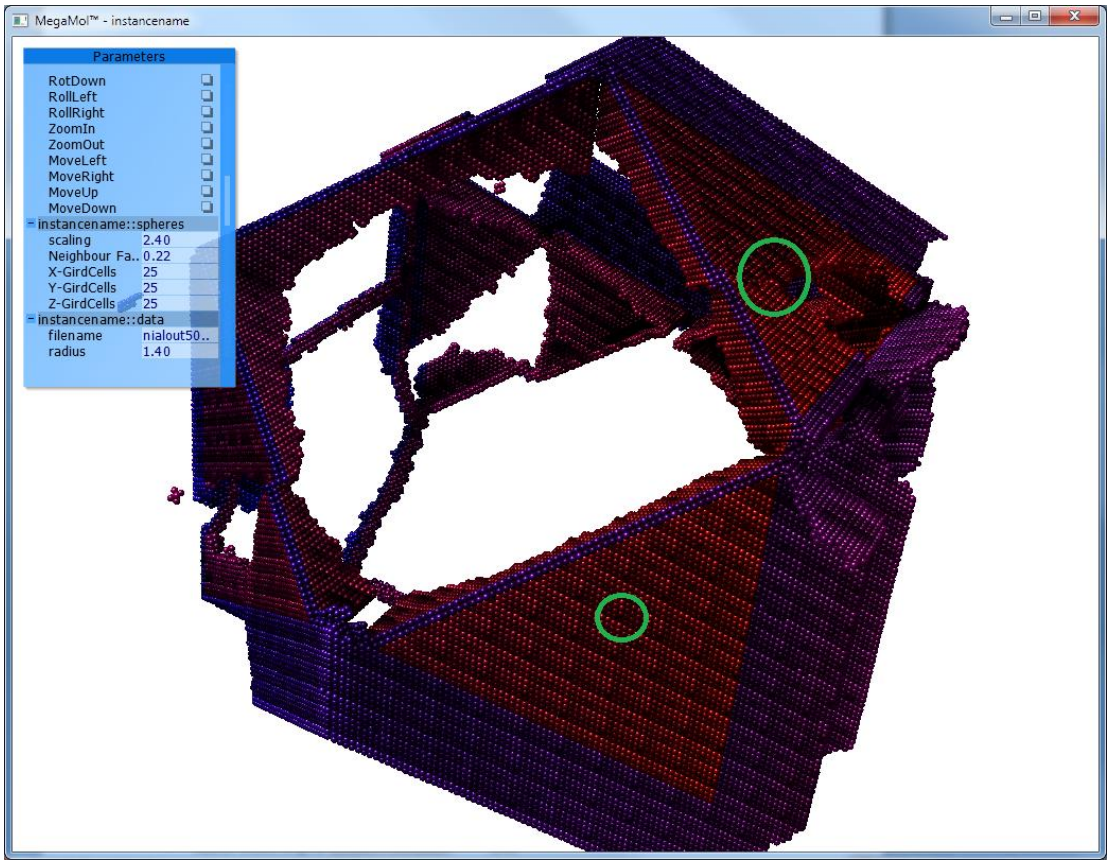

Abbildung 6-6: Fehler bei nicht homogen gefüllten Volumenzellen (Grüner Kreis). Als Datensatz wurde hier "nialout50_00599.crist" verwendet.

6.2 Geschwindigkeit

Um die Performance dieses Verfahrens zu bestimmen, wird das Plug-in mit verschiedenen Datensätzen auf verschiedenen Plattformen ausgeführt. Die Messung wird unter Windows 7 jeweils in 32 und 64 Bit durchgeführt. Als Vergleichswert wird die Messung auch für den Standard MegaMol-Renderer „SimpleSphere“ durchgeführt.

6.2.1 nialout50_00599.crist

- Anzahl Atome: 86.315
- Atomradius: 1.27
- Volumenzellen: 25*25*25=15.625
- Radius-Skalierung für Ambient Occlusion: 2.45
- Neighbour Faktor: 0.3

System		Performance in FPS		Performanceverhältnis
		Mit Ambient Occlusion	Ohne Ambient Occlusion	
4*2,4 GHz CPU, 4 GB RAM, GeForce 8800 GTX	32 Bit	76.81	107.17	71.67 %
	64 Bit	76.53	107.19	71.40 %
4*2,7 GHz CPU, 4 GB RAM, GeForce 280 GTX	32 Bit	149.52	215.18	69.49 %
	64 Bit	149.60	225.27	66.40 %
4*3.0 GHz CPU, 4 GB RAM, GeForce 480 GTX	32 Bit	321.99	531.69	60.56 %
	64 Bit	321.34	519.78	61.82 %

Anhand dieser Werte sieht man, dass das Objekt orientierte Ambient Occlusion-Verfahren bei diesem Datensatz mit einem recht konstanten Verlust auf den verschiedenen Plattformen läuft. Hierbei wird auf einem stärkeren System kein besseres Leistungsverhältnis erzielt. Ein starkes System bringt also keine relativen Vorteile, obwohl die absolute Performance natürlich steigt. Ein Unterschied zwischen 32 und 64 Bit ist nicht festzustellen.

Abbildung 6-7 zeigt einen Vergleich der Ergebnisse mit und ohne Ambient Occlusion, auf denen die Messung der Performance durchgeführt wurde.

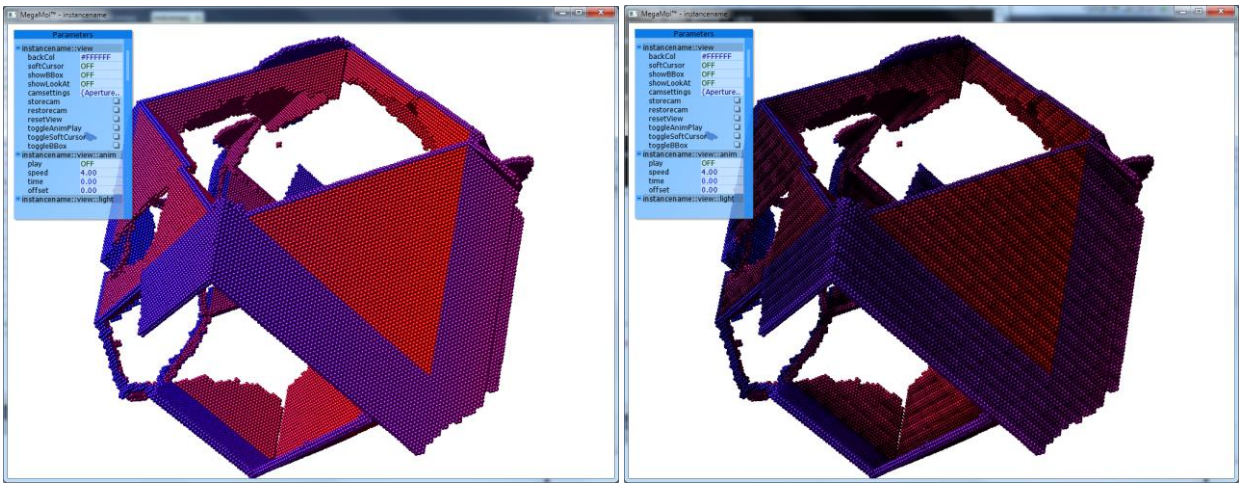

Abbildung 6-7: Vergleich der Ergebnisse mit und ohne Ambient Occlusion auf dem Datensatz "nialout50_00599.crist". Links: Rendering ohne Ambient Occlusion. Rechts: Rendering mit Ambient Occlusion.

6.2.2 b2_cut.chkpt

- Anzahl Atome: 107.391
- Atomradius: 1.2
- Volumenzellen: 100*25*25=62.500
- Radius-Skalierung für Ambient Occlusion: 1.6
- Neighbour Faktor: 0.3

System		Performance in FPS		Performanceverhältnis
		Mit Ambient Occlusion	Ohne Ambient Occlusion	
4*2,4 GHz CPU, 4 GB RAM, GeForce 8800 GTX	32 Bit	59.59	82.52	72.21 %
	64 Bit	59.59	83.93	70.98 %
4*2,7 GHz CPU, 4 GB RAM, GeForce 280 GTX	32 Bit	106.98	150.50	71.08 %
	64 Bit	105.14	151.85	69.24 %
4*3.0 GHz CPU, 4 GB RAM, GeForce 480 GTX	32 Bit	176.37	260.44	67.72 %
	64 Bit	176.73	267.44	66.08 %

Die Ergebnisse bei diesem Datensatz sind ähnlich zu den Ergebnissen bei dem Datensatz “nialout50_00599.crist”. Auch hier ist der Verlust an Performance relativ konstant und ein stärkeres System bringt keine großen Vorteile. Allerdings ist die FPS-Rate geringer. Dies ist auf den größeren Datensatz zurückzuführen. Auch hier ist kein Unterschied zwischen 32 und 64 Bit festzustellen.

Abbildung 6-8 zeigt einen Vergleich der Ergebnisse mit und ohne Ambient Occlusion, auf denen die Messung der Performance durchgeführt wurde.

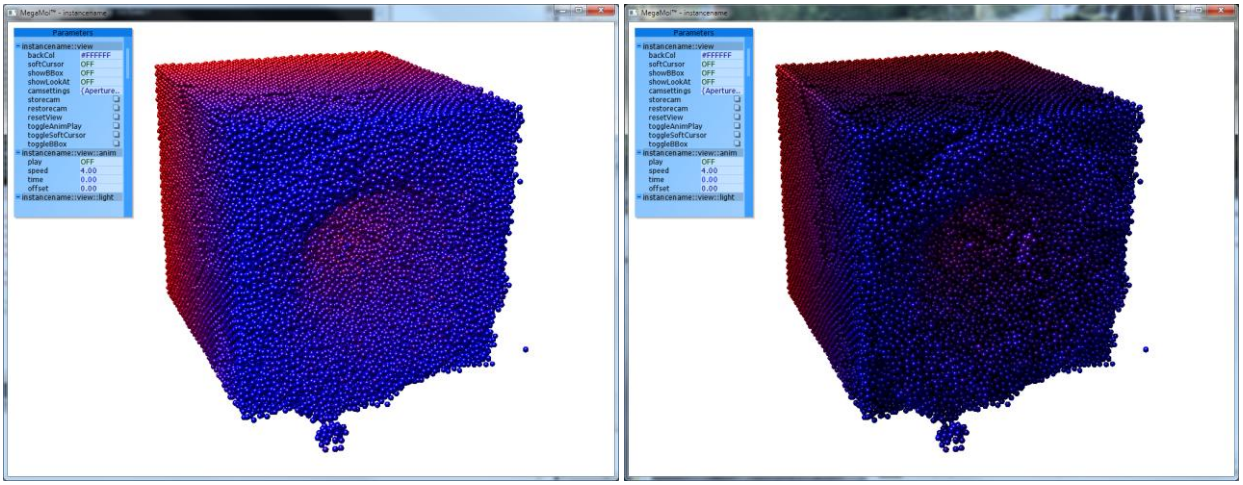

Abbildung 6-8: Vergleich der Ergebnisse mit und ohne Ambient Occlusion auf dem Datensatz "b2_cut.chkpt". Links: Rendering ohne Ambient Occlusion. Rechts: Rendering mit Ambient Occlusion.

6.2.3 exp2mill.f25.siff

- Anzahl Atome: 2.000.000
- Atomradius: 0.5
- Volumenzellen: 25*100*25=62.500
- Radius-Skalierung für Ambient Occlusion: 2.0
- Neighbour Faktor: 0.5

System		Performance in FPS		Performanceverhältnis
		Mit Ambient Occlusion	Ohne Ambient Occlusion	
4*2,4 GHz CPU, 4 GB RAM, GeForce 8800 GTX	32 Bit	6.09	22.64	26.90 %
	64 Bit	6.11	22.75	26.86 %
4*2,7 GHz CPU, 4 GB RAM, GeForce 280 GTX	32 Bit	8.19	30.42	26.92 %
	64 Bit	8.28	30.47	27.17 %
4*3.0 GHz CPU, 4 GB RAM, GeForce 480 GTX	32 Bit	26.71	65.50	40.77 %
	64 Bit	26.61	66.53	40.00 %

Bei diesem Datensatz verhält sich der Geschwindigkeitsverlust jedoch anders. Je stärker das System ist, desto besser ist das Performanceverhältnis. Dies ist auf die Größe des Datensatzes zurückzuführen. Die schwächeren Grafikkarten sind der Anzahl an Atomen nicht gewachsen. Hier hat man also Vorteile von einer starken Grafikkarte, da neben der FPS-Zahl auch das Leistungsverhältnis des Verfahrens steigt. Wie bei den anderen beiden Datensätzen ist auch hier kein Unterschied zwischen 32 und 64 Bit festzustellen.

Abbildung 6-9 zeigt einen Vergleich der Ergebnisse mit und ohne Ambient Occlusion, auf denen die Messung der Performance durchgeführt wurde.

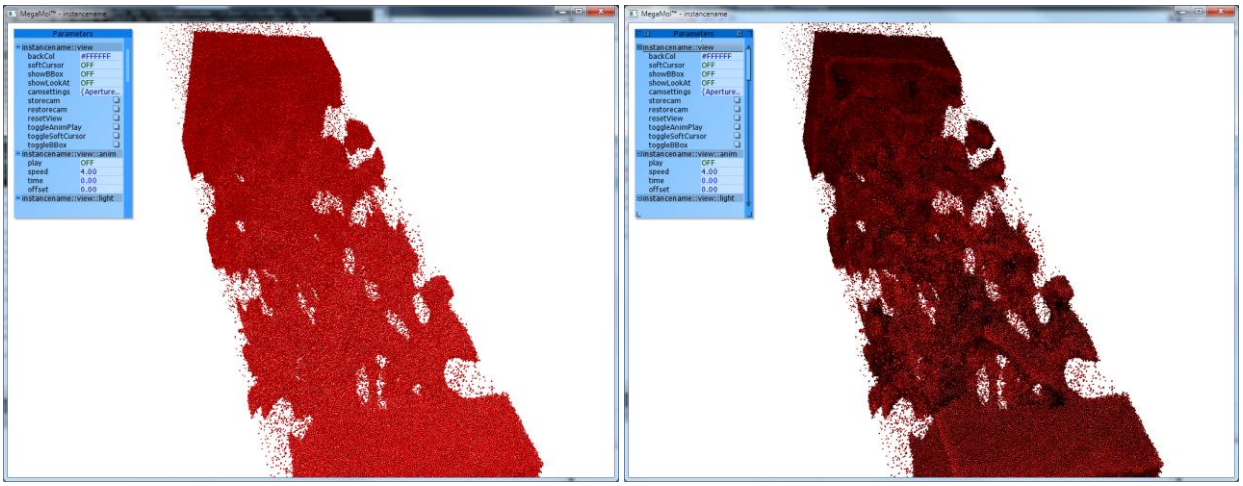

Abbildung 6-9: Vergleich der Ergebnisse mit und ohne Ambient Occlusion auf dem Datensatz "exp2mill.f25.siff". Links: Rendering ohne Ambient Occlusion. Rechts: Rendering mit Ambient Occlusion.

7 Fazit

Die Bildqualität wird durch das Objekt orientierte Ambient Occlusion-Verfahren stark verbessert. Hierbei werden Details sichtbar, die ohne Ambient Occlusion nicht wahrnehmbar wären. Abbildung 7-2 und Abbildung 7-3 vergleichen die Ergebnisse mit und ohne Ambient Occlusion. Hier kann man sehr gut erkennen, dass bei dem Ergebnis mit Ambient Occlusion die Struktur der Daten sehr gut sichtbar wird, während bei dem Bild ohne Ambient Occlusion die Struktur praktisch nicht sichtbar ist.

In Abbildung 7-1 ist ein Vergleich des Ergebnisses mit Ambient Occlusion und des Renderings ohne Ambient Occlusion zu sehen. An den markierten Stellen kann man deutlich erkennen, dass bei dem Bild mit Ambient Occlusion (rechts) ein Tiefeneindruck der Szene entsteht, während man bei dem Bild ohne Ambient Occlusion (links) keinerlei Tiefen wahrnehmen kann.

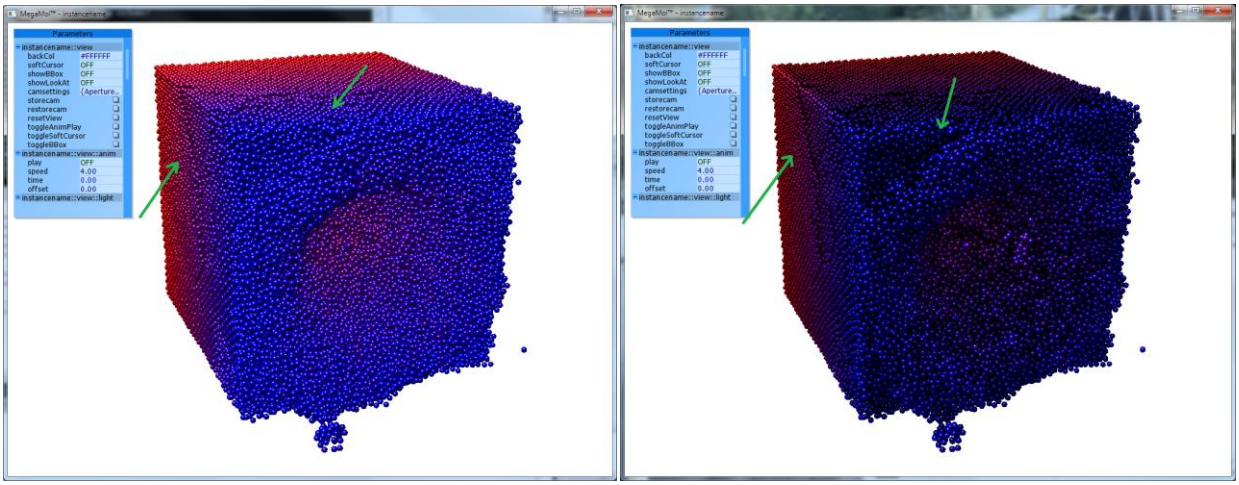

Abbildung 7-1: Vergleich der Ergebnisse mit und ohne Ambient Occlusion auf dem Datensatz "b2_cut.chkpt". Hier wurden Stellen markiert, welche eine Verbesserung durch Ambient Occlusion sind. Links: Rendering ohne Ambient Occlusion. Rechts: Rendering mit Ambient Occlusion.

Aus den Performancemessungen kann man ablesen, dass das Verfahren auch bei großen Datensätzen immer noch echtzeitfähig ist.

Man sieht auch, dass es bei kleinen Datensätzen keinen großen Unterschied macht, wie gut die Grafikkarte ist. Bei großen Datensätzen ist dagegen der Unterschied stark wahrnehmbar. Hierdurch ist klar, dass bei kleinen Datensätzen das gesamte System die Leistung limitiert, bei großen Datensätzen jedoch die Grafikkarte der limitierende Faktor ist.

Abschließend kann man sagen, dass trotz der verringerten Performance das Verfahren einen entscheidenden Vorteil bringt: Die verbesserte Tiefenwahrnehmung.

Problematisch sind jedoch noch Datensätze, bei denen die Atome nicht homogen in den Zellen verteilt sind. Bei solchen Daten entstehen Fehler im Ambient Occlusion, die im Bild erkennbar sind. Um dies zu lösen, müsste man die Geometrie der Daten erkennen können und einen Ansatz verwenden, welcher nicht auf einem Volumen basiert. Dieser müsste jedoch auch auf allgemeinen Datensätzen funktionieren. Vielleicht wäre ein Hybrid zwischen dem Objekt orientierten Ambient Occlusion-Verfahren und ein paar berechneten Ebenen ausreichend.

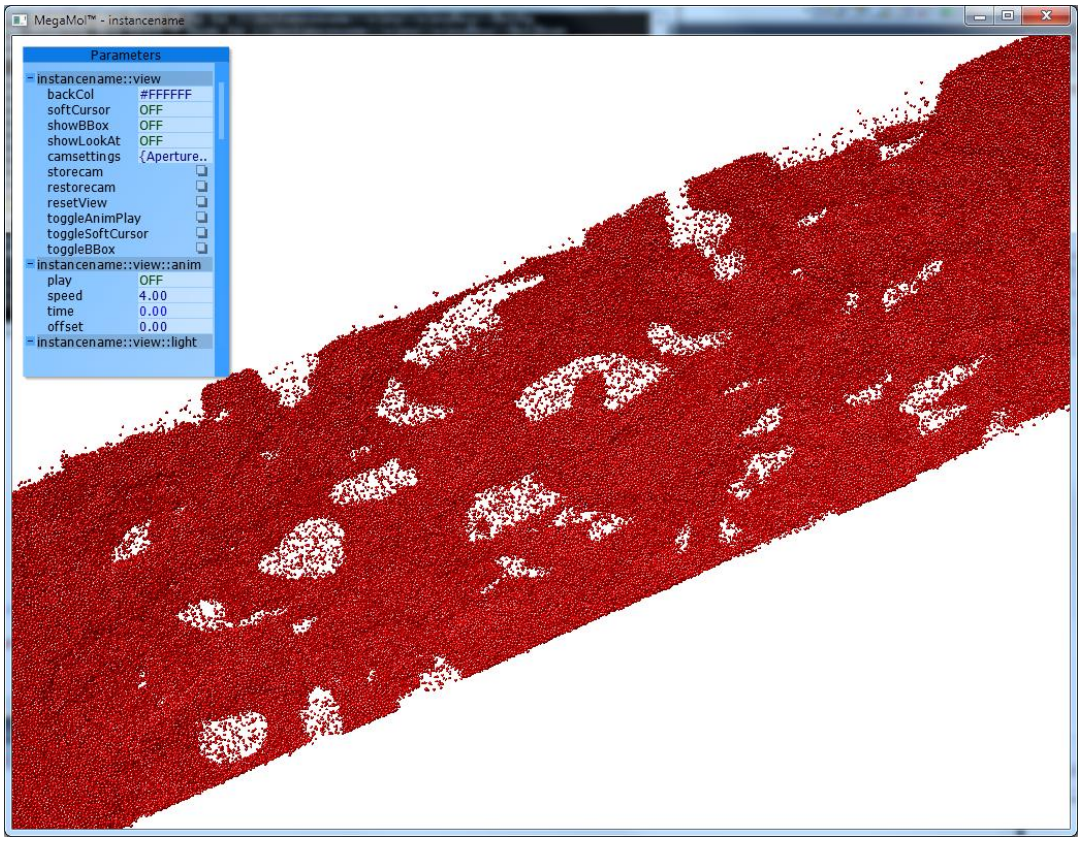

Abbildung 7-2: Ergebnis ohne Ambient Occlusion. Datensatz "exp2mill.f25.siff".

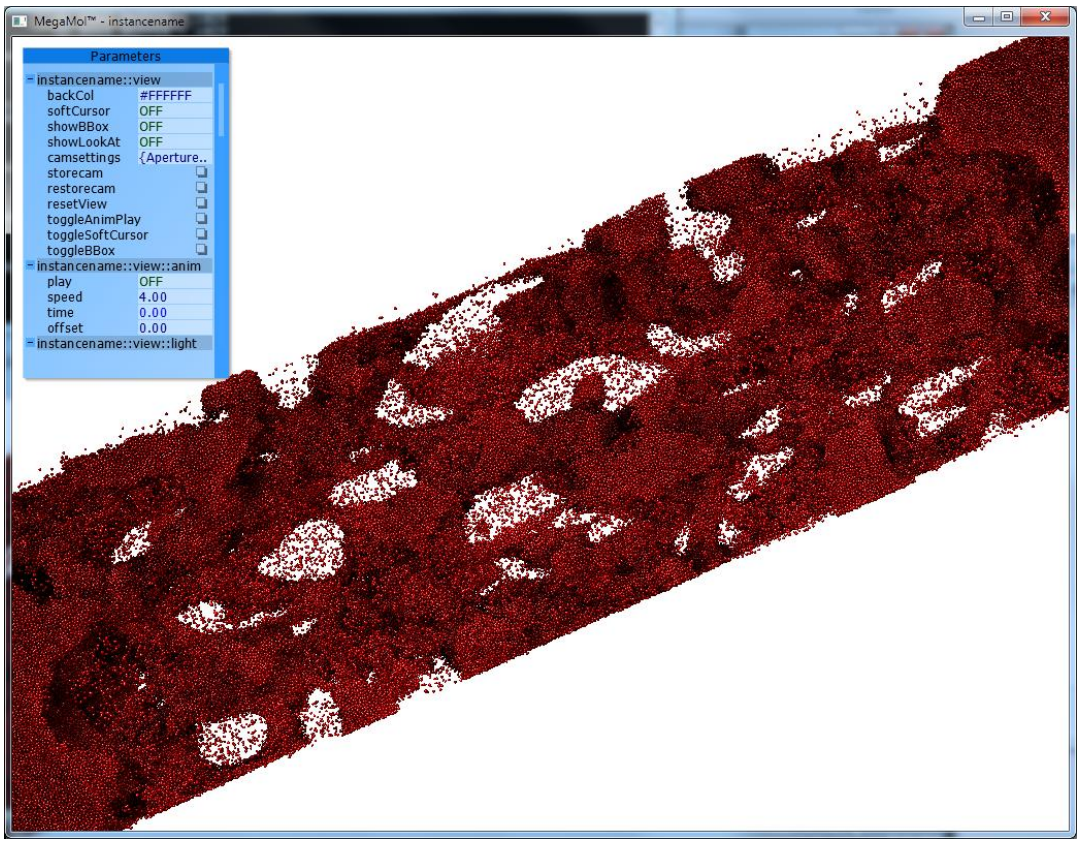

Abbildung 7-3: Ergebnis mit Ambient Occlusion. Datensatz "exp2mill.f25.siff".

8 Literaturverzeichnis

Bunnell, M. (2005). Dynamic Ambient Occlusion and Indirect. In M. Pharr, E. Addison, & M. Wesley, *GPU Gems 2* (S. 223-233). Nvidia Corporation.

Cook, R. L., Porter, T., & Carpenter, L. (1984). Distributed ray tracing. *Proceedings of the 11th annual conference on Computer graphics and interactive techniques* (S. 137-145). New York: ACM.

Donnelly, W., & Lauritzen, A. (2006). Variance Shadow Maps. *In Proceedings of the 2006 ACM SIGGRAPH Symposium on Interactive 3D graphics and games* (S. 161-165). Redwood City: ACM.

Grottel, S., Reina, G., Dachsbacher, C., & Ertl, T. (2010). Coherent Culling and Shading for Large Molecular Dynamics Visualization. *Computer Graphics Forum (Proceedings of EUROVIS 2010), 29(3)*, (S. 953-962).

Hasenfratz, J.-M., Lapierre, M., Holzschuch, N., & Sillion, F. (2003). A Survey of Real-time Soft Shadows Algorithms. *Computer Graphics Forum 22*, (S. 753–774).

Hoberock, J., & Jia, Y. (2007). High-Quality Ambient Occlusion. In H. Nguyen, *GPU Gems 3.* Nvidia Corporation.

Kontkanen, J., & Laine, S. (2005). Ambient Occlusion Fields. *Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games* (S. 41-48). ACM.

Lokovic, T., & Veach, E. (2000). Deep Shadow Maps. *Proceedings of ACM SIGGRAPH 2000* (S. 385-392). ACM.

Mittring, M. (2007). Finding Next Gen – CryEngine 2. *SIGGRAPH'07: ACM SIGGRAPH 2007 courses* (S. 97-121). New York: ACM.

Pharr, M. (2004). Ambient occlusion. In R. Fernando, *GPU Gems* (S. 667–692). Nvidia Corporation.

Ritschel, T., Grosch, T., & Seidel, H.-P. (2009). Approximating Dynamic Global Illumination in Image Space. *Proceedings of the 2009 symposium on Interactive 3D graphics and games* (S. 75-82). Boston, Massachusetts: ACM.

Shanmugam, P., & Arikan, O. (2007). Hardware Accelerated Ambient Occlusion Techniques on GPUs. *In Proceedings of ACM Symposium in Interactive 3D Graphics and Games* (S. 73-80). ACM.

Zhukov, S., Iones, A., & Kronin, G. (1998). An ambient light illumination model. *Rendering Techniques '98 (Proceedings of the Eurographics Workshop on Rendering)*, (S. 45-55).