Visualisierungsinstitut der Universität Stuttgart
Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit Nr. 0005

Visualisierung von planetarischen Nebeln in Celestia

Florian Frieß

Studiengang:	Informatik
Prüfer:	Prof. Daniel Weiskopf
Betreuer:	Dipl.-Inf. Marco Ament
begonnen am:	01. Mai 2012
beendet am:	31. Oktober 2012
CR-Klassifikation:	I.3.7

Kurzfassung

Weltraumnebel sind Wolken aus Staub und Gas. Diese können in verschiedene Kategorien eingeteilt werden. Sie können Licht von benachbarten Sternen reflektieren (Reflexionsnebel), selbst Licht emittieren (Emissionsnebel) oder Licht absorbieren (Dunkelwolke).
Aufgrund ihrer Farbenvielfalt und Struktur besteht ein Interesse dreidimensionale Modelle dieser Nebel zu erstellen. Da nur 2D Aufnahmen dieser Gebilde vorliegen müssen die Modelle aus diesen Aufnahmen rekonstruiert werden. Dazu können die Modelle komplett von Hand modelliert werden, oder es können automatische Verfahren eingesetzt werden. Automatische Verfahren sind deutlich schneller, daher wurden die verwendeten Nebel mit einem solchen Verfahren rekonstruiert.
Die damit erstellten Modelle können mit einem Volumen Rendering Verfahren dargestellt werden. Hierzu kann entweder ein direktes oder indirektes Verfahren verwendet werden. Bei den indirekten Verfahren werden aus dem Modell Polygonflächen erzeugt. Das direkte Volumen Rendering Verfahren arbeitet direkt auf den Volumendaten. Mit Raycasting wird ein direktes Verfahren verwendet, das in das Programm Celestia eingebettet wurde.

Inhaltsverzeichnis

Abbildungsverzeichnis

Verzeichnis der Listings

1 Einleitung

Weltraumnebel sind aufgrund ihrer Struktur und ihrer Farben interessant. Jedoch ist ihre Darstellung als 3D-Modell nicht einfach zu realisieren, da nur 2D Daten dieser Gebilde vorliegen. Für die Betrachtung des Nebelinneren ist es notwendig die bereits vorhandenen Daten in dreidimensionale umzuwandeln. Es gibt Ansätze die Nebel komplett von Hand zu modellieren [NGN+01], auch mit speziell dafür entwickelten Tools [SKW+11]. Die Modellierung eines Nebels von Hand dauert oft Wochen oder Monate, folglich wird versucht die zur Rekonstruktion benötigte Zeit zu verringern. Eine Möglichkeit ist die Konstruktion zu automatisieren um die Erstellung des Nebels zu parallelisieren. Durch automatische Verfahren [MKHD04][MKMD05], die 3D Modelle erzeugen bis sie dem ursprünglichen Bild ähneln, kann diese Rekonstruktion deutlich schneller umgesetzt werden.
Die Modelle, die in dieser Arbeit verwendet werden, wurden mit dem Verfahren rekonstruiert, das in [WAG+12] vorgestellt wird. Dieses Verfahren basiert, wie [AFWMM08], auf der Verwendung eines tomographischen Verfahrens. Dabei werden die weit verbreitete Kugel- oder Achsensymmetrie der Nebel ausgenützt, aber auch andere Arten von Symmetrien können implementiert werden. Um kein komplett symmetrisches Modell des Nebels zu erhalten, muss der Nebel von der Erde aus betrachtet genauso aussehen wie das Originalbild. Ein so konstruiertes Modell basiert auf einem Emissionsnebel, einem Nebel der Licht emittiert. Der Nebel wird durch hochenergetische Photonen, von benachbarten heißen Sternen, zum Leuchten angeregt. Nachdem die dreidimensionalen Daten gewonnen wurden, müssen diese gerendert werden. Mit Volumen Rendering wird ein Verfahren vorgestellt, das die Darstellung von dreidimensionalen Daten ermöglicht. Volumen Rendering lässt sich in direktes und indirektes Rendering aufteilen. Mit Raycasting wird ein direktes Volumen Rendering Verfahren verwendet.

In dieser Arbeit werden die rekonstruierten Nebeldaten mit Hilfe eines Raycasters dargestellt. Dabei werden von der Kamera aus (Sicht)Strahlen durch das Volumen geschickt. In einer vorher festgelegten Schrittweite werden Farbwerte, aus der Voxelrepräsentation des Volumens, ausgelesen und zu einem Farbwert für den Pixel, des Bildschirms, verrechnet.
Celestia[1] ist eine OpenSource Software, die es ermöglicht durch große Teile des Universums zu navigieren. Der Raycaster wurde in dieses Programm eingebettet, um die dort bereits vorhandene Darstellung der Nebel zu verbessern. Diese wurden bisher als 3DS-Modelle umgesetzt. Hiermit werden die rekonstruierten Nebel für die breite Öffentlichkeit zugänglich. Zur Implementierung des Raycasters waren Veränderungen an dem bestehenden Quellcode notwendig. Abschließend werden noch Erweiterungen, zur bestehenden Arbeit vorgestellt.

[1] http://www.shatters.net/celestia/

Gliederung

Die Arbeit ist in folgender Weise gegliedert:

Kapitel 2 – Celestia: beschreibt das Programm das in dieser Arbeit erweitert wurde.

Kapitel 3 – Weltraumnebel: erläutert die Rekonstruktion der 3D Modelle.

Kapitel 4 – Volumen Rendering: erklärt die Grundlagen des Volumenrenderings, des Weiteren werden Optimierungsansätze vorgestellt.

Kapitel 5 – Implementierung: zeigt und beschreibt wichtige Teile des Quellcodes, die zur Darstellung der Nebel benötigt werden.

Kapitel 6 – Ergebnisse: stellt Ergebnisse dieser Arbeit vor.

Kapitel 7 – Zusammenfassung und Ausblick fasst die Ergebnisse der Arbeit zusammen und stellt Anknüpfungspunkte vor.

2 Celestia

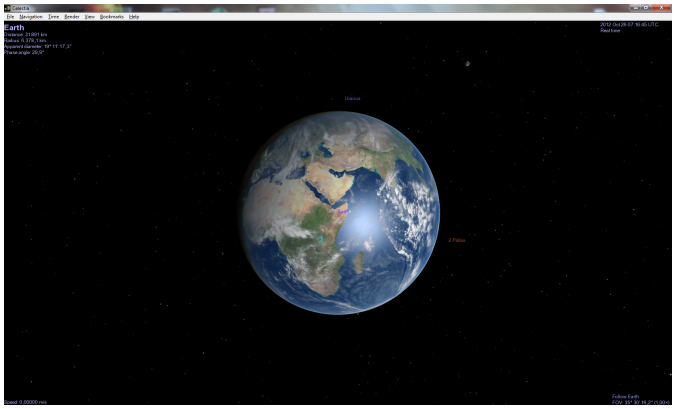

Abbildung 2.1: Erde in Celestia

Celestia[1] ist ein Open Source Projekt, das unter der *GNU Public License* steht, und kann daher von jedem, im Rahmen der Lizenzbedingungen, erweitert werden.
Es bietet die Möglichkeit unser Universum zu bereisen und Sterne und Planeten (2.1) zu betrachten. Dafür wird ein großer Datensatz an Galaxien, Sternen, Planeten, Asteroiden, etc. mitgeliefert. Zu jedem Objekt gibt es noch Hintergrundinformationen, wie z.B. die Leuchtkraft oder den Radius. Durch die Erstellung einer STC(STar Catalog), DSC(Deep Space Catalogs) oder SSC(Solar System Catalog) Datei[2], welche alle wichtigen Informationen enthält, und dem dazugehörenden 3DS Modell, ist es möglich das Universum beliebig zu erweitern. Eine STC Datei beschreibt einen Stern, eine DSC Datei charakterisiert eine Galaxie oder einen Nebel und mit einer SSC Datei wird ein Sonnensystem beschrieben.
Bisher wurden die Weltraumnebel auf genau diese Art und Weise implementiert. Dadurch wird die Möglichkeit den Nebel zu erkunden eingeschränkt, denn man kann nur in Bereiche vordringen die auch modelliert wurden. Um diese Limitierung zu umgehen wurde im Rahmen dieser Arbeit eine Erweiterung vorgenommen, die es ermöglicht vorher erstellte Volumendaten von Nebeln darzustellen. Da jeder Punkt innerhalb des Volumens definiert ist, ermöglicht dies eine uneingeschränkte Erkundung der Nebel.
Die Weltraumnebel haben keine scharfen, gleichmäßigen Kanten, sie verblassen an ihren Rändern (2.2). Mit Meshmodellen ist dieser Effekt kaum darstellbar (2.3), da immer eine

[1] http://www.shatters.net/celestia/
[2] http://en.wikibooks.org/wiki/Celestia/Catalog_File_Reference

Fläche dargestellt werden muss. Es muss also ein enormer Aufwand betrieben werden um die Ränder so zu modellieren. Die Transparenz der Nebel kann nicht nur mit den Volumenmodellen, sondern auch mit den 3DS Modellen umgesetzt werden. Celestia sortiert alle angezeigten Objekte von hinten nach vorne, in dieser Reihenfolge werden sie dann auch gerendert. Daher ist es mit halbtransparenten Meshmodellen möglich die Transparenz abzubilden.

Abbildung 2.2: Volumenmodell von M2-9

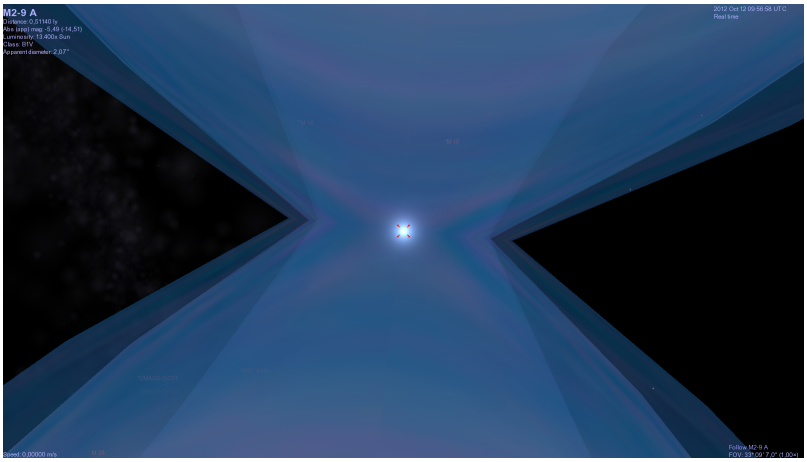

Abbildung 2.3: 3DS-Modell von M2-9

Der planetare Nebel M2-9[a] als 3DS Modell des Autors *jll*.

[a] http://www.celestiamotherlode.net/catalog/nonmessiernebulae.php

3 Weltraumnebel

Eine interstellare Wolke wird in der Astronomie als Nebel bezeichnet. Diese Wolke kann verschiedene Lichteigenschafen haben. Sie können Licht reflektieren (Reflexionsnebel), emittieren (Emissionsnebel) oder absorbieren (Dunkelwolke).
Mit **Reflexionsnebel** werden Nebel bezeichnet, die das Licht benachbarter Sterne reflektieren. Diese Nebel emittieren kein Licht, da die Sterne nicht heiß genug sind. Das einfallende Licht wird im Inneren gestreut, daher hat der Nebel das Farbspektrum der Sterne und ist sichtbar.
Emissionsnebel emittieren im Gegensatz zu Reflexionsnebel Licht, da die benachbarten Sterne heiß genug sind um hochenergetische Photonen zu produzieren. Diese Photonen regen den Nebel zum Leuchten an. Emissionsnebel können aus den Überresten einer Supernova entstehen. Bei einer Supernova breitet sich die Materie mit der Schockwelle aus und wird enorm erhitzt, auf über 10^7 Kelvin. Der dadurch entstandene Nebel wird als Supernovaüberrest bezeichnet und hat die Eigenschaften eines Emissionsnebels. Auch Planetarische Nebel gehören zum Typ der Emissionsnebel. Sie entstehen wenn ein Stern am Ende seines Lebenszyklusses ist. Dann stößt er Gas und Plasma ab, dadurch kann ein planetarischer Nebel entstehen. Dieser lebt mit einigen Zehntausend Jahren deutlich kürzer als die anderen Nebel, die mehrere Millionen Jahre alt werden können. Wenn der Stern im Zentrum des Nebels das Stadium des Weißen Zwerges erreicht, emittiert er, aufgrund seiner Hitze, hochenergetische Photonen, die den Nebel zum Leuchten bringen.
Eine **Dunkelwolke** absorbiert aufgrund ihrer hohen Staub- und Gasdichte das Licht von anderen Objekten und verdeckt diese. Aufgrund dieser Dichte können sich im Inneren des Nebels Moleküle ansammeln. Dadurch erhöht sich die Dichte immer weiter und es können neue Sterne entstehen.

Alle in dieser Arbeit verwendeten Weltraumnebel wurden mit dem von [WAG+12] vorgestellten Verfahren rekonstruiert. Bevor dieses Verfahren näher erläutert wird, wird ein kleiner Überblick über andere Verfahren zur Rekonstruktion von Nebeln gegeben.

3.1 Überblick über Rekonstruktionsmethoden

Weltraumnebel sind aufgrund ihrer Form und der Farbenvielfalt interessante Objekte im Weltall, von denen nur 2D Aufnahmen vorliegen. Daher versuchen Astronomen 3D Modelle mit Hilfe dieser Bilder zu konstruieren. Diese Modelle können komplett von Hand erstellt [NGN+01, WO95] werden.
Da es bereits Tools [DTM96, FM01, ZDPSS02] gibt um 3D Modelle aus 2D Bildern zu erzeugen, kann dieser Ansatz [SKW+11] auch für Weltraumnebel verwendet werden. Diese Tools verwenden vom Benutzer erzeugte Geometrien und verbessern diese durch automatische

Prozesse soweit, dass diese unter gewissen Einschränkungen dem 2D Bild entsprechen. Auch automatische Prozesse [MKHD04, MKMD05, Lea91] werden zur Rekonstruktion von 3D Modellen genutzt. Diese erzeugen auf einer Basis von unterschiedlichen Daten immer neue Modelle, die nach jedem Schritt weiter optimiert werden, bis ein Modell den Vorgaben genügt. Die Optimierung in jedem Schritt erfolgt über die Anpassung der Emissionswerte des neu gerenderten Volumenmodells. Alle diese Verfahren setzen eine Achsensymmetrie des Nebels voraus, erzeugen aber Modelle die wenig detailreich sind. Auch für [LLM+07] ist Achsensymmetrie eine Voraussetzung. Die Qualität der Ergebnisse hängt aber hier von Eingabebildern für die Staub- und Gasdaten ab. Diese müssen einige Anforderungen erfüllen, damit ein qualitativ hochwertiges Modell erzeugt werden kann. Die Eingabebilder müssen dieselbe Auflösung haben und gleichgerichtet sein. Denn die Absorption, Emission und die Lichtstreuung werden aus diesen Daten gewonnen. Unter der Annahme das die Geometrie zylindersymmetrisch ist, kann mit tomographischen Verfahren ein 3D Modell rekonstruiert werden[AFWMM08]. Kleine Abweichungen von dieser Symmetrie können korrigiert werden. Dadurch sind die korrigierten Nebel nicht mehr eindeutig, da die Korrektur der Asymmetrie nicht auf physikalischen Messungen beruht. Bei Reflexionsnebel wird das Licht von benachbarten Sternen ausgesendet und dann über Lichtbrechung im Nebel verteilt und absorbiert. Um ein Modell eines solchen Nebels zu rekonstruieren wird nur die Absorption und Lichtbrechung betrachtet, um den Lichttransport innerhalb des Nebels zu modellieren. Mit einer speziellen Konfiguration der Wege, die das Licht innerhalb der Reflexionsnebel nimmt, können plausible 3D Volumen rekonstruiert werden [LHM+07]. Ein iteratives Verfahren, das rundenbasiert Modelle erstellt, um diese dem Originalbild anzunähern bewerkstelligt diese Rekonstruktion. Vor jeder Runde wird die Annahme zur Staubdichte basierend auf dem vorherigen Modell angepasst.

3.2 Nebel-Rekonstruktion

Das Ziel der Rekonstruktion[WAG+12] war es ein detailreiches Modell eines Emissionsnebels zu erstellen. Dazu wird zuerst ein Bild eines einigermaßen symmetrischen Nebels ausgesucht, danach werden vom Benutzer Parameter, wie die Symmetrie und die gewünschte Auflösung, eingegeben und dann wird das Modell, mit Hilfe eines GPU-Clusters, automatisch erstellt. Nach der Erstellung kann das Modell interaktiv mit direktem Volumenrendering visualisiert werden (5.1).

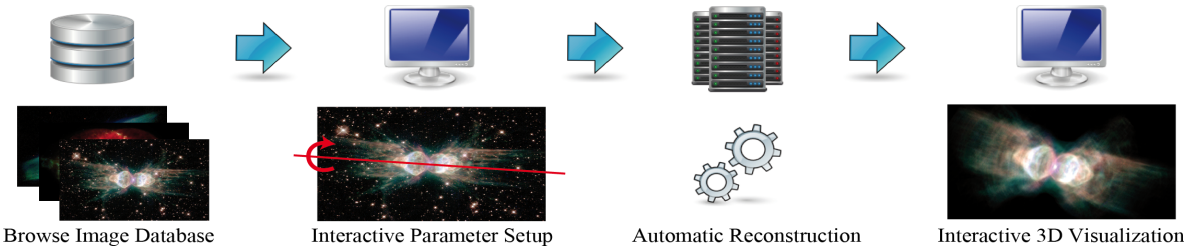

Abbildung 3.1: Programm Pipeline von [WAG+12]

Die automatische Rekonstruktion basiert auf einem tomographischen Algorithmus. Dieser Algorithmus setzt das Volumen aus vielen Bildern zusammen. Diese Bilder werden mit virtuellen Kameras aufgenommen, die um die Symmetrieachse platziert werden. Um Artefakte, wie Schlierenbildung, zu vermeiden werden diese dann geringfügig versetzt. Inwieweit die Kameras versetzt werden ist zufällig. Je mehr Kameras eingesetzt werden, desto akkurater wird die erzeugte Symmetrie. Je weniger Kameras eingesetzt werden, desto größer wird die Variation. Dadurch wird ein realistischerer Eindruck erzeugt.

Anhand eines optischen Modells wird eine Projektion des Volumens zu diesen Bildern durchgeführt. Da die Emissionsnebel keine Absorption haben wird bei dem optischen Modell nur die Emission betrachtet. Daher wird die Intensität bei der Visualisierung des Modells, eines Pixels mit $I = \sum_i e_i d_i$ bestimmt, wobei e_i die Emission in der $i$ten Gitterzelle des Volumens ist und d_i die Abtastrate angibt. Die Projektion $M_k v = b_k$ ergibt sich daraus, dass die Emissionswerte der einzelnen Gitterzellen in einem Vektor v und die Intensität der Pixel des $k$ten Bildes als Vektor b_k gespeichert werden. Die Rückprojektion M_k^T verteilt die Intensität eines Pixels proportional zwischen allen beitragenden Gitterzellen.
Da eine gewisse Anzahl (n_{views}) Kameras erzeugt wurden werden ebenso viele Bilder b_k erstellt, daraus ergibt sich das lineare Gleichungssystem

$$(M_0, ..., M_{n_{views}-1})^T v = (b_0, ..., bn_{views} - 1)^T \tag{3.1}$$

Wenn alle Projektionen M und alle Bilder b bekannt sind lässt sich das Volumenobjekt v durch Lösen dieses linearen Gleichungssystems bestimmen. Meistens sind jedoch nicht genügend Informationen in den Bildern b enthalten um v eindeutig zu bestimmen. Wenn dies der Fall ist muss die beste Lösung gewählt werden, dazu wird ein *Compressed Sensing* Algorithmus verwendet. Mit Hilfe dieses Algorithmus wird das Optimierungsproblem (Fig.3 aus [WAG+12]) gelöst. Die innere Schleife ist eine Zusatzbedingung die dafür sorgt, dass der Nebel aus Sicht der Erde wie das Originalbild aussieht. Diese Bedingung muss von der Form $Bx = c$, mit $BB^T = \mathbb{I}$ sein. Eine solche Einschränkung ist nützlich wenn ein Bild korrekt ist und die anderen Bilder rauschen oder Inkonsistenzen enthalten. Bei den Nebeln ist genau dies der Fall. Es gibt eine Aufnahme die korrekt ist, das Bild was von der Erde aus gemacht wurde. Die anderen Bilder, die über die virtuellen Kameras erstellt wurden, beruhen auf Vermutungen. Der Unterschied zwischen einem Ergebnis mit dieser Zusatzbedingung (3.2) und einem Ergebnis ohne diese Bedingung (3.3) ist deutlich. Das Ergebnis ohne Zusatzbedingung ist zu symmetrisch und sieht dadurch unrealistisch aus, ganz im Gegensatz zu dem Ergebnis mit der Bedingung.

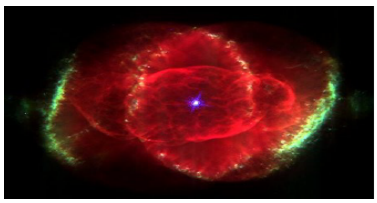

Abbildung 3.2: Mit Zusatzbedingung[WAG+12]

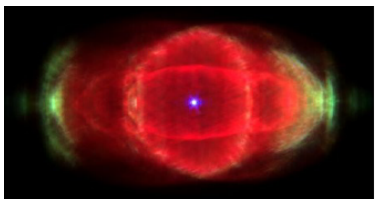

Abbildung 3.3: Ohne Zusatzbedingung[WAG+12]

Die Rekonstruktion wird durch wiederholtes projizieren der Bilder auf das Volumen und andersherum mit den Matrizen M_k und M_k^T durchgeführt. Aufgrund des benötigten Speicherplatzes für M_k werden $M_k v$ und $M_k^T b_k$ in jeder Iteration berechnet.
Unter der Annahme das die skalaren Voxelwerte des Volumens eine stückweise trillineare Funktion im Raum definieren, hinterlässt jeder Wert von v einen Fußabdruck der zweimal so groß ist wie der Abstand zwischen zwei Voxlen entlang aller drei Achsen. Da der Fußabdruck symmetrisch entlang der Achsen ist, kann eine gewöhnliche Formel zum Integrieren einer Gitterzelle, dem Raum zwischen acht benachbarten Abtastpunkten, verwendet werden. Die Integration kann auf einer per-Voxel Basis gelöst werden, da nur die Emission betrachtet wird. Somit werden die gewichteten Abtastwerte der acht benachbarten Voxel einfach addiert.

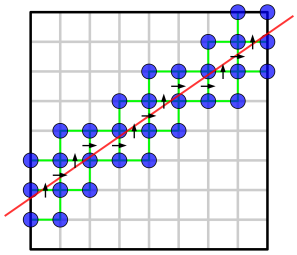

Abbildung 3.4: Projektion [WAG+12]

Alle Voxel (Blau) die aufgrund des Sichtstrahls (Rot) zu der Intensität eines Pixels beitragen.

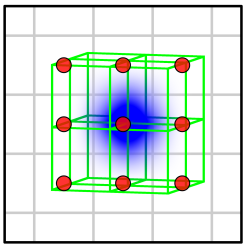

Abbildung 3.5: Rückprojektion [WAG+12]

Alle Pixel (Rot) die zu dem Wert des Voxels (Blau) etwas beitragen.

In beiden Fällen sind die betroffenen Gitterzellen grün markiert.

Die Projektion $M_k v$ nutzt den *fast voxel traversal algorithm* [AW87] um alle Voxel Werte v zu finden die zu einem Pixel im Bild b_k beitragen (3.4). Die Rückprojektion $M_k^T b_k$ muss alle Pixel im Bild b_k finden, die zu einem Voxel v etwas beitragen. Dazu wird eine Bounding Box, bestehend aus $2 * 2 * 2$ Gitterzellen, auf das Eingabebild projiziert. Dann wird ein Strahl für jeden Pixel durch das Teil-Volumen geschickt (3.5).

Um die Berechnung des Modells zu beschleunigen wird ein GPU-Cluster verwendet. Hierzu werden die Volumendaten V in Teil-Volumen V_i aufgeteilt, um sie über die GPUs (Knoten) verteilen zu können. Jeder Knoten i berechnet jetzt für sein Teil-Volumen V_i die Projektion für alle Kamerabilder $k = 0, ..., n_{views} - 1$ und erhält die Bilder P_i^k. Diese müssen mit den Bildern P_i^k aller anderen Knoten zusammengesetzt werden.
Für die Rückprojektion braucht man das komplette Bild der Projektion einer Kamera. Dieses Ergebnis ist über die einzelnen Knoten verteilt. Deshalb werden die Teilergebnisse über alle Knoten verteilt. Die Rückprojektion benötigt nur einen $2 * 2 * 2$ Gitterzellen großen Bereich des Volumens und kann, auf jedem Knoten unabhängig, durchgeführt werden sobald der Knoten die benötigten Bilddaten hat.

4 Volumen Rendering

Die Darstellung dreidimensionaler Objekte über Polygonflächen, Meshes, Nurbs, etc. hat ihre Grenzen. Bei Strukturen, wie sie die Weltraumnebel aufweisen, wird es schwer eine geeignete Oberflächenrepräsentation mit den oben genannten Verfahren zu finden. Da diese Strukturen fließend ineinander übergehen und es so schwer machen einer Struktur eine eindeutige Oberfläche zuzuordnen.
Daher wurden Verfahren entwickelt, um diese Objekte darstellen zu können. Diese werden unter dem Begriff Volumen Rendering zusammengefasst. Volumen Rendering kann noch in direktes und indirektes Volumen Rendering aufgeteilt werden. Beim indirekten Rendering werden aus den Volumendaten Polygonflächen erzeugt, beim direkten Rendering wird nur mit den Volumendaten gearbeitet. Mit diesen Verfahren lassen sich Objekte darstellen, die keine scharfen Abgrenzungen haben oder halb transparent sind, wie z.B. die in dieser Arbeit verwendeten Weltraumnebel. Daher wird bei der Implementierung ein Raycaster verwendet, da dieser die beste Bildqualität liefert.

4.1 Volumen Rendering Integral

Raycasting ist ein direktes Volumen Rendering Verfahren, daher wird direkt auf den Volumendaten gearbeitet. Die Daten müssen dazu mit einer Skalarfunktion $f(\vec{x})$, die für alle Punkte $\vec{x}$ im Volumen definiert ist, zwischen den Abtaststellen, interpoliert werden [Nel95]. Dazu wird jeder Sichtstrahl $\vec{x}(t)$ mit seiner Distanz zur Kamera t parametrisiert und in das Volumen geschickt. Um das Bild aus dem Volumen zu berechnen werden optische Modelle angewandt. Die wichtigsten sind *Absorption only*, *Emission only* und *Absorption plus emission*. Das vollständige Volumen Rendering Integral (4.1) setzt sich aus Absorption, Emission und Streuung zusammen.

$$L(x, x') = g(x, x') * (L_e(x, x') + \int_S b(x, x', x'')L(x', x'')\, dx') \tag{4.1}$$

Der Term $g(x, x')$ gibt die Absorption an. Da die Absorption der Emissionsnebel sehr gering ist kann dieser Term vernachlässigt werden. Daraus resultiert das angepasste Volumen Rendering Integral (4.2).

$$L(x, x') = (L_e(x, x') + \int_S b(x, x', x'')L(x', x'')\, dx') \tag{4.2}$$

Der Energiefluss $L(x, x')$ gibt an wie viel Licht x von x' erreicht. Mit S wird die Gesamtheit aller Flächen der Szene angegeben. Der Term $b(x, x', x'')$ gibt die Streuung innerhalb des

Volumens an. Da es innerhalb der Emissionsnebel eine vernachlässigbare Streuung gibt, kann auch diese weggelassen werden. Auf diese Weise entsteht das Volumen Rendering Integral, das die Emissionsnebel berechnet (4.3).

$$L(x, x') = L_e(x, x') \tag{4.3}$$

Die Funktion $L_e(x, x'$ kann mit dem *Emission only* Integral [Nel95] ausgedrückt werden.

$$L_e(x, x') = I(D) = \int_0^D c(f(\vec{x}(t)))\, dt \tag{4.4}$$

Bei der Emission senden die Partikel des Nebels Licht aus. Die daraus resultierende Intensität $I(D)$ basiert auf den Farbwerten $c(\vec{x}(t))$ des Sichtstrahls zwischen 0 und D. Die Lösung dieses Integrals (4.4) wird numerisch berechnet, indem das Integral in einen Summenterm (4.5) umgewandelt wird.

4.2 Raycasting

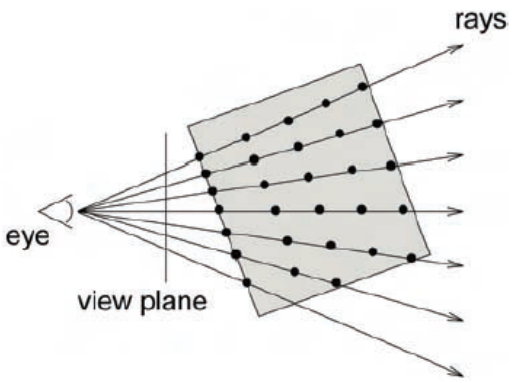

Abbildung 4.1: Ein Raycasting Schritt [EHK+04]

Ein Ansatz das Volumen Rendering Integral numerisch zu berechnen ist das Raycasting. Beim Raycasting wird, für jedes Pixel des Bildes, ein Strahl von der Kamera durch dieses Pixel in das Volumen geschickt (4.1). Dieser Strahl wird, solange er sich im Volumen befindet, abgetastet um das Signal (den Skalarwert) an dieser Stelle zu bekommen. Der Farbwert muss mit Hilfe einer Transferfunktion bestimmt werden. Bei den Emissionsnebel wird anstatt eines Signals direkt der Farbwert gespeichert, daher wird keine Transferfunktion benötigt. Die Abtastrate bleibt konstant und sollte für die Implementierung (5) kleiner gleich 0.01 gewählt werden, um Artefakte zu vermeiden. Wenn Optimierungstechniken eingesetzt werden, kann sich die Abtastrate entlang eines Strahls ändern. Bei [YS93] wird die Abtastrate zeitweise erhöht um leere Räume am Rand des Volumens zu überspringen.

Um an die Werte der Abtastpunkte zu gelangen wird das Volumen in ein Voxel Objekt umgewandelt. Dieses Objekt spannt einen dreidimensionalen Raum auf. Die Position eines Voxels kann daher mit einem Vektor ausgedrückt werden. Meistens sind die in einer gitterförmigen Struktur (4.2) angeordnet. An jedem Eckpunkt eines Würfels ist ein Voxel, der einen Skalarwert speichert. Die Werte der Abtastpunkte werden durch z.B. trilineare Interpolation gewonnen.

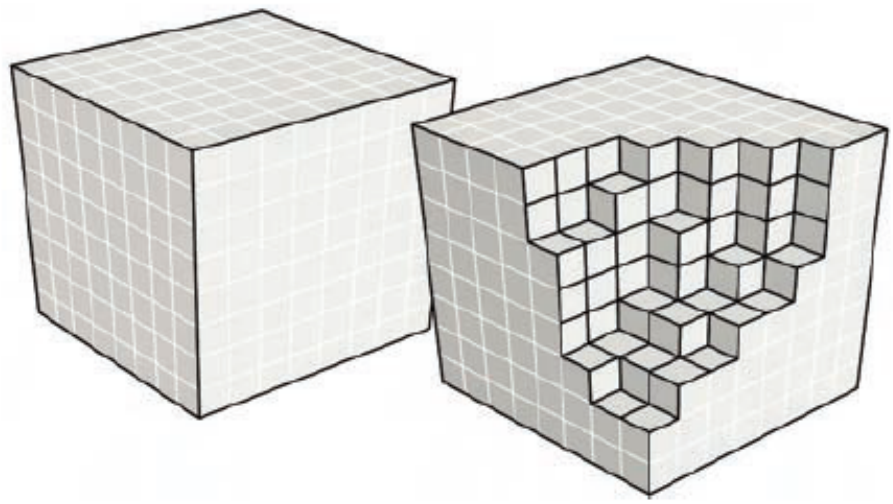

Abbildung 4.2: Ein Voxel Objekt [EHK+04]

Mit den Werten der Abtastpunkte wird die numerische Approximation des Volumen Rendering Integrals, oder eines anderen optischen Modells, berechnet. Diese Berechnung kann wie in [Nel95] beschrieben durchgeführt werden. Dabei wird das Integral (4.4) durch

$$\sum_{i=0}^{\lfloor D/\Delta d \rfloor} c(f(\vec{x}(i\Delta d)))\Delta d \tag{4.5}$$

approximiert, wobei Δd den Abstand zwischen zwei Abtastpunkten angibt. Die Emission kann auch für jedes Segment einzeln berechnet werden:

$$C_i = c(f(\vec{x}(i\Delta d)))\Delta d \tag{4.6}$$

Die Emission für einen Strahl ergibt sich dann aus:

$$\sum_{i=0}^{\lfloor D/\Delta d \rfloor} C_i \tag{4.7}$$

Die in Kapitel (3) beschriebene Rekonstruktion der Nebel liefert Volumendaten ohne Opazitätswerte. Daher wird bei der Implementierung nur die Emission (4.7) betrachtet. Die iterative Auswertung erfolgt dann über

$$C_i^{akkum} = C_{i-1}^{akkum} + (C_i^{akt} * \Delta d * intensity) \tag{4.8}$$

Der aktuelle Emissionswert C_i^{akt} wird mit der Abtastrate Δd und der *intensity* multipliziert. Die Multiplikation mit der Abtastrate verhindert, dass die Farbwerte den darstellbaren Bereich verlassen und mit der *intensity* kann die Leuchtkraft der Nebel verstärkt oder reduziert werden. Der Startwert C_0^{akkum} ist mit null belegt.

4.3 Optimierungen

Da Raycasting für große Volumen langsam sein kann, gibt es zahlreiche Versuche das Verfahren zu optimieren. Für jedes Verfahren muss ein zusätzlicher Aufwand betrieben werden. Eine Optimierung macht daher nur Sinn wenn das Verfahren trotz des zusätzlichen Aufwands beschleunigt wird.

4.3.1 Empty space skipping

Beim *Empty space skipping* werden Teile des Volumens übersprungen, die nicht zu Emission oder Absorption beitragen, also alle Voxel deren Werte gleich null sind. In [YS93] wird dazu ein *C-buffer* eingeführt, der für jeden Pixel die Objektkoordinaten des ersten sichtbaren Voxels speichert. Dadurch werden aber keine leeren Räume im Inneren des Volumens übersprungen. Um auch diese zu überspringen, müssen andere Verfahren angewendet werden. Eine Möglichkeit ist das Volumen in kleinere Volumen aufzuteilen [LMK03]. Jedes dieser Teil-Volumen enthält Voxel mit ähnlichen Werten. Teil-Volumen, die als leerer Raum identifiziert werden, werden übersprungen. Da bei der Aufteilung die Struktur des Volumens nicht erhalten bleibt werden die Teil-Volumen in einen BSP-Baum eingefügt, um die Teil-Volumen in der richtigen Reihenfolge zu rendern.

4.3.2 Early ray termination

Um zu bestimmen, ob die Auswertung des Strahls abgebrochen werden kann wird die Opazität verwendet. Hierzu muss geprüft werden, ab wann die kommenden Teile des Volumens nicht mehr sichtbar sind. Dazu eignet sich der *front-to-back* (4.12) Ansatz, da dort die Opazität akkumuliert wird. Sobald diese den Wert 1.0 übersteigt, sind die dahinter liegenden Teile des Volumens nicht mehr sichtbar. Die Berechnung kann also abgebrochen werden.
Wenn keine Absorption vorhanden ist, kann die Berechnung über die akkumulierte Emission abgebrochen werden. Es gibt dafür keine Bindung an ein Verfahren, sobald bei *back-to-front*

(4.10) oder *front-to-back* (4.12) alle Farbwerte (Rot, Grün und Blau) größer gleich eins sind kann die Berechnung abgebrochen werden.

Für das Volumen Rendering Integral, das Absorption und Emission enthält, lautet die numerische Approximation [Nel95]

$$\sum_{i=0}^{\lfloor D/\Delta d \rfloor} C_i \prod_{j=0}^{i-1}(1-\alpha_i) \tag{4.9}$$

Diese numerische Approximation kann iterativ ausgewertet werden in dem die Abtastpunkte von hinten nach vorne (*back-to-front*) oder von vorne nach hinten (*front-to-back*) abgearbeitet werden.

***back-to-front*:**

$$C_i^{akkum} = C_i^{akt}\alpha_i^{akt} + (1-\alpha_i^{akt})C_{i+1}^{akkum} \tag{4.10}$$

In jedem Schritt, von $n-1$ bis 0 wird der aktuelle Emissionswert C_i^{akt} sowie der aktuelle Absorptionswert α_i^{akt} ausgelesen und mit dem bereits bekannten akkumulierten Emissionswert verrechnet. Der erste akkumulierte Emissionswert C_n^{akkum} ist gleich null.

***front-to-back*:**

$$C_i^{akkum} = C_{i-1}^{akkum}\alpha_{i-1}^{akkum} + (1-\alpha_{i-1}^{akkum})C_i^{akt}\alpha_i^{akt} \tag{4.11}$$

$$\alpha_i^{akkum} = \alpha_{i-1}^{akkum} + (1-\alpha_{i-1}^{akkum})\alpha_i^{akt} \tag{4.12}$$

In jedem Schritt, von 1 bis n werden die akkumulierte Absorption und Emission berechnet. Dazu werden zu den bereits akkumulierten Werte die aktuellen Emissions- und Absorptionswerte addiert. Die Startwerte C_0^{akkum} und α_0^{akkum} werden mit null initialisiert.

4.3.3 Occlusion culling

Beim Occlusion Culling werden Objekte, die von undurchsichtigen Objekten verdeckt werden, nicht gerendert. Dieser Test wird normal über die Opazität durchgeführt. Ist die Opazität des näheren Objekts eins, dann wird das verdeckte Objekte nicht gezeichnet. Wenn keine Opazitätswerte vorliegen muss anhand der Farbwerte geprüft werden ob ein Objekt verdeckt wird. Sobald die Farbwerte (Rot, Grün und Blau) größer gleich eins sind, ist das verdeckte Objekt nicht mehr sichtbar.

5 Implementierung

Um die Weltraumnebel in Celestia darstellen zu können, musste das Programm um einige Funktionen erweitert werden. Der Aufbau ist sehr modular gehalten und daher einfach zu erweitern.

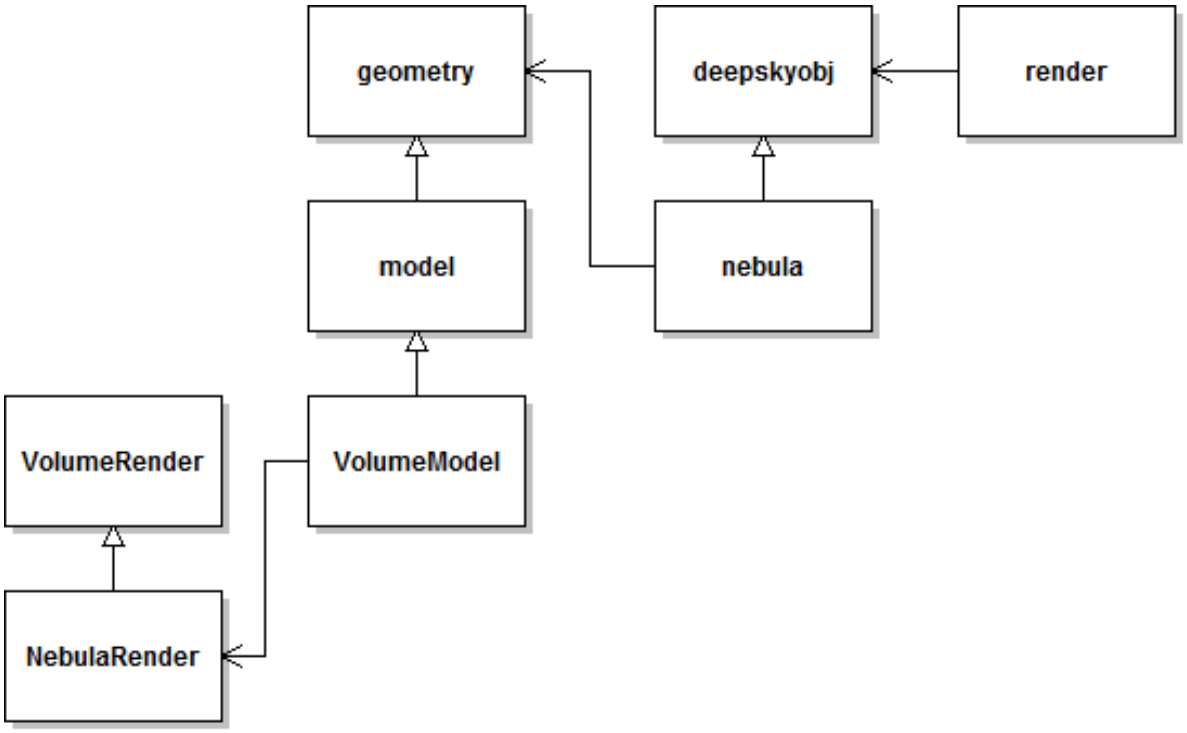

Abbildung 5.1: UML Klassendiagramm der Implementierung

Die Klassen *geometry, model, deepskyobj, nebula* und *render* sind bereits im Code von Celestia 1.6.1 enthalten und wurden erweitert. Die Klassen *VolumeModel, VolumeRender* und *Nebula-Render* wurden neu erstellt. *VolumeModel* erbt von *model* und erhält dadurch Zugriff auf alle, von Celestia, verwendete Methoden zum Laden und Rendern der Modelle. Der *VolumeRender* ist für das Rendern der Nebel zuständig. Er liefert bereits alle dafür nötigen Methoden. Um eigene Volumendaten zu rendern kann, einfach eine Klasse erstellt werden, die von diesem erbt und Methoden überschreibt. Dann muss im *VolumenModel* der Render angepasst werden und der eigene Datensatz wird angezeigt.
Die Implementierung wurde mit einer *NVIDIA GeForce GTX 460* und der Treiberversion

306.89, sowie mit einer *ATI Mobility Radeon HD 4300* und der Treiberversion *8.902-111012a-127490C-ATI* getestet.

5.1 Importieren der Daten

Zunächst wird der Vorgang beschrieben wie in Celestia 3DS-Modelle importiert werden. An den entsprechenden Stellen werden die benötigten Änderungen beschrieben, um die Volumendaten zu importieren. Um ein 3DS-Model zu importieren, muss zunächst eine ssc oder dsc Datei erstellt werden. In dieser werden alle bekannten Daten zu diesem Objekt eingetragen. Da wir uns auf das Importieren von Weltraumnebeln beschränken, betrachten wir nur die dsc Dateien.

Listing 5.1 M2-9.dsc

```
# http://en.wikipedia.org/wiki/Planetary_Nebula_M2-9
Nebula "M2-9"
{
        Mesh "M2-9.vol"
        Radius 0.7
        RA         17.0938
        Dec       -10.1423
        Distance 2100
        Axis     [-0.1045 0.7799 0.6171]
        Angle    173.587
}
```

In der obersten Zeile steht als Kommentar meistens der Link zum englischen wikipedia Eintrag. In der zweiten Zeile steht der Typ (Nebula) des Objekts und danach sein Name (M2-9). Über diesen Namen kann man das Objekt später selektieren. Danach kommt, in geschweiften Klammern, ein Verweis auf das zu ladende Modell. Dann der Radius des Objektes in Lichtjahren, danach die *Right ascension*[1] und die *Declination*[2]. Des Weiteren wird noch die Entfernung zur Erde angegeben, ebenfalls in Lichtjahren. Die letzten beiden Einträge, Axis und Angle, sind optional und geben eine Rotation um die angegebene Achse mit dem festgelegten Rotationswinkel an.

Celestia lädt beim Starten alle ssc und dsc Dateien und erzeugt eine Liste mit allen daraus resultierenden Objekten. Zu jedem Objekt wird auch der Pfad zu seinem Modell gespeichert. Wenn eines dieser Objekte dargestellt werden muss, wird der Klasse *meshmanager* der Pfad zu der Meshdatei dieses Objektes übergeben. Diese entscheidet aufgrund des Dateityps der zu ladenden Meshdatei, welche Methode zum Laden verwendet wird. Der Dateityp wird in der Klasse *filetype* bestimmt. Dort wird ein Vergleich mit Dateitypen durchgeführt die Celestia

[1] http://en.wikipedia.org/wiki/Right_ascension
[2] http://en.wikipedia.org/wiki/Declination

kennt. Um die Nebel laden zu können, mussten diese beiden Klassen erweitert werden. Die Liste von *filetype* wurde um die Dateiendung ".vol" ergänzt und im *meshmanager* wurde eine Abfrage auf diese Dateiendung eingefügt. Für ein Mesh mit dem Dateityp ".vol" wird jetzt die statische VolumeLoader-Methode der *VolumeModel* Klasse ausgeführt. Diese erzeugt ein neues *VolumeModel* Objekt das die Daten lädt. Es können alle Volumendaten geladen werden die byteweise gespeichert sind und deren ersten 12 Bytes die Länge, Breite und Höhe des Modells angeben. Für die Länge, Breite und Höhe werden jeweils vier Byte verwendet, also die Länge eines Integers. Die Daten werden über fopen und fread geladen, dabei wird zuerst die Dimension eingelesen und gespeichert. Danach folgen die Farbwerte, die in einen std::vector gespeichert werden, dessen Werte vom Typ unsigned char sind. Für fopen wird der Pfad zur Datei und der Modus "rb" angegeben. Mit "r" wird die Datei zum Lesen geöffnet und mit "b" wird die Datei binär ausgelesen. Wird das "b" weggelassen werden die Volumen nicht mehr richtig eingelesen und es fehlt ein Großteil der Farbwerte, da beim Einlesen nicht mehr jedes Byte betrachtet wird.

5.2 Rendern der Daten

Bevor das Modell gerendert werden kann müssen noch ein paar Probleme gelöst werden. Das Universum von Celestia wird mittels eines Octrees eingeteilt. Dadurch werden nur Objekte in Betracht gezogen, die in der Nähe des Betrachters sind. Danach wird nochmals ein lokales Culling durchgeführt. Alle Objekte die beide Culling Verfahren überleben werden dann geladen, da sie unter Umständen ins Blickfeld des Betrachters geraten und dann dargestellt werden müssen. Für die Objekte die bisher in Celestia dargestellt wurden ist das kein Problem, da sie meistens unter zehn mb groß sind. Bei den Nebeln führt dieses Verfahren zu Problemen, da jeder Nebel mehr als 400 mb groß ist. Wenn jetzt mehrere Nebel gleichzeitig geladen werden kann es schnell dazu kommen, dass der Speicher der Grafikkarte ausgeht und es zu einem Absturz von Celestia kommt. Daher musste noch ein dritter Culling Schritt eingebaut werden, der nur für Nebel gilt. Diese werden erst geladen wenn die Distanz kleiner als 250 Lichtjahre ist, da die projizierte Fläche der Nebel ab dieser Distanz größer als ein Pixel ist. Dieses Culling wird, wie das lokale Culling auch, in der Klasse *render*, für jedes *deepskyobj* Objekt durchgeführt. Um die Objekte herauszufiltern, die keine Nebel sind, wird der Typ des aktuellen *deepskyobj* Objekts abgefragt. Wenn es sich um einen Nebel handelt dann wird das Culling durchgeführt, ansonsten wird kein Culling durchgeführt.
Diese Methode löst aber nur einen Teil des Problems, denn aus Performancegründen werden einmal geladene Objekte nicht mehr vergessen, um sie bei weiteren Besuchen ohne Verzögerung darstellen zu können. Auch hier ist das für die bisherigen Objekte kein Problem. Aufgrund ihrer Größe müssen die Nebel aus dem Speicher der Grafikkarte gelöscht werden, sobald diese nicht mehr angezeigt werden. Daher muss es möglich sein die Nebel erneut zu laden. Um die Nebel löschen und erneut laden zu können musste die Klasse *deepskyobj* um drei Methoden erweitert werden: uninit3D, init3D und getActive.
Diese Methoden müssen in allen Klassen die von *deepskyobj* erben implementiert werden, damit sie von der *render* Klasse aufgerufen werden können. Eine solche Klasse ist *nebula*,

diese erstellt ein Objekt der Klasse *geometry* und bietet daher Zugriff auf Methoden, die von der Klasse *VolumeModel* geerbt werden. Die Klasse *geometry* wurde ebenfalls um die Methoden uninit3D und init3D erweitert, damit diese der Klasse *VolumeModel* zur Verfügung stehen.
Wenn ein Nebel weiter als 250 Lichtjahre von der Kamera entfernt ist, wird geprüft, ob dieser im letzten Bild noch dargestellt wurde. Dazu wird die getActive Methode des *nebula* Objekts aufgerufen. Wenn diese true zurückgibt wurde der Nebel gerendert. Gibt sie false zurück wurde der Nebel nicht dargestellt. Der Nebel muss gelöscht werden, wenn er im letzten Bild dargestellt wurde und jetzt weiter als 250 Lichtjahre entfernt ist. Dazu wird die Methode uninit3D der *nebula* Klasse aufgerufen, die den Befehl an das betreffende *VolumeModel* Objekt weitergibt. Der Nebel wird dann von dem Grafikspeicher gelöscht. Um ihn erneut zu laden wird geprüft, ob er im letzten gerenderten Bild nicht aktiv war, sonst wird ein Nebel mehrfach geladen. Für einen nicht aktiven Nebel wird die init3D Methode des *nebula* Objekts aufgerufen, die den Befehl wieder an das entsprechende *VolumeModel* Objekt weitergibt.

Sobald ein Nebel dargestellt werden soll wird ein neues *VolumeModel* Objekt erstellt. Dieses lädt den Datensatz und erzeugt ein *NebulaRender* Objekt. Diesem Objekt werden die Dimensionen des Volumens und ein Zeiger auf die Daten übergeben. Mit Hilfe der Dimensionen wird eine Bounding Box erzeugt, in der später der Nebel dargestellt wird. Die Bounding Box wird für zwei Dinge verwendet. Zum einen, um zu überprüfen ob sich die Kamera innerhalb des Volumens befindet oder außerhalb. Zum anderen, um die Richtung der Sichtstrahlen zu bestimmen die das Volumen durchdringen. Die Eckpunkte der Bounding Box sind mit einer bestimmten Farbe belegt. Die Farbe der Außenflächen wird dann zwischen den Farben der Eckpunkte interpoliert. Um diese Daten abzuspeichern wird ein struct verwendet das sechs Werte speichern kann. Die ersten drei Werte geben die Position in x, y und z an, die letzten drei Werte geben die Farbwerte in $RGB \in \{0.0, ..., 1.0\}$ an. Über die Angaben zur Distanz, der *Right ascension* und der *Declination* in der dsc Datei wird eine Position für das Objekt festgelegt. Diese Position ist später im Zentrum des Objekts. Um diese Bedingung für die Volumen zu erfüllen, müssen die acht Eckpunkte der Bounding Box so liegen, dass ihr Mittelpunkt mit diesem Zentrum übereinstimmt (5.2). Die Eckpunkte der Bounding Box liegen zwischen $(0.0, 0.0, 0.0)$ und $(1.0, 1.0, 1.0)$. Daher ist ein Eckpunkt im Zentrum und nicht der Mittelpunkt. Zur Korrektur dieses Fehlers wird eine Translation um -0.5 in x-,y- und z-Richtung durchgeführt.

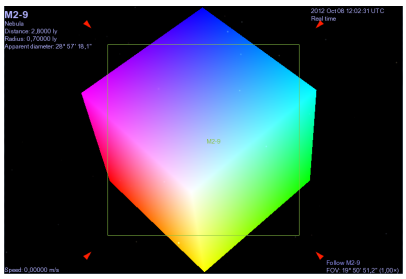

Abbildung 5.2: Bounding Box

Dieses struct wird dann als VertexArray in einen Buffer der Grafikkarte geladen, um beim Rendern einen schnellen Zugriff auf die Daten zu bekommen. Danach wird das struct gelöscht. Wenn beim allokieren des Buffers oder beim Schreiben der Daten in den Buffer Probleme auftreten wird die Erstellung der Bounding Box abgebrochen und es wird kein Nebel angezeigt. Wenn die Bounding Box erfolgreich erstellt wurde wird das 3D-Model des Nebels auf die Grafikkarte geladen und aus dem Arbeitsspeicher gelöscht. Somit existieren nie zwei Kopien des Models zeitgleich. Wie bei der Bounding Box wird der Rendervorgang nicht durchgeführt wenn hier ein Fehler auftritt.

Um zu überprüfen, ob die Kamera innerhalb der Bounding Box ist, muss zunächst die Position der Kamera festgestellt werden. Da Celestia die Kameraposition nicht zu Verfügung stellt, muss diese mit Hilfe der modelview Matrix berechnet werden. Nach der Transformation mit der modelview Matrix ist die Kamera immer im Ursprung. Daher kann mit der Inversen der modelview Matrix die Kameraposition vor der Transformation berechnet werden.
Dazu wird der Vektor (0.0,0.0,0.0) erstellt, der die Kameraposition nach der Transformation mit der modelview Matrix angibt. Dieser Vektor wird zu einem 4D Vektor (0.0,0.0,0.0,1.0) erweitert und mit der Inversen der modelview Matrix multipliziert. Die x-,y- und z-Komponenten des Ergebnisvektors ergeben die Kameraposition vor der Transformation.
Diese wird dann in die Texturkoordinaten transformiert. Wenn die Kamera sich dann zwischen (0.0,0.0,0.0) und (1.0,1.0,1.0) befindet, ist sie innerhalb der Bounding Box, wenn das Volumen die gleiche Länge, Breite und Höhe hat. Wenn wir außerhalb der Bounding Box sind dann werden die Frontfaces der Bounding Box gerendert. Wenn wir innerhalb der Bounding Box sind werden die Backfaces gerendert.

Da die Nebel alle eine andere Helligkeit haben wurde der Parameter intensity eingefügt den der Benutzer mit "shift" erhöhen und mit "strg" verringern kann. Dadurch werden die Nebel heller, bzw. dunkler und man kann die Intensität auf seine Bedürfnisse einstellen. Um die Helligkeit nicht bei jedem neuen Start von Celestia neu einstellen zu müssen ist hier eine Erweiterung um eine Speicherfunktion denkbar. Hierzu wird der aktuelle Intensitätswert mit den RGB-Werten multipliziert. Das Ergebnis muss entweder direkt in die Volumendaten oder in eine Extradatei gespeichert werden. Dazu bekommt die *VolumenModel* Klasse den aktuellen Intensitätswert übergeben. Diese liest dann die Daten ein, verrechnet diese mit der Intensität und speichert sie wieder.

Die Schrittweite kann zwischen 0.01 und 0.0001 variiert werden. Mit "Bild-Hoch" kann die Schrittweite vergrößert, über "Bild-Runter" kann die Schrittweite verringert werden. Werte größer als 0.01 verursachen Artefakte aufgrund von Unterabtastung und Werte kleiner als 0.0001 können zu einem Absturz des OpenGL Treibers führen.

5.2.1 Frontfaces

Im Fragmentshader wird dann zuerst die Richtung des Sichtstrahls berechnet. Der vom Vertexshader übergebene Farbwert entspricht der Position, des Vertex, auf der Außenfläche der Bounding Box. Um die Richtung zu berechnen, wird der normalisierte Differenzvektor von der Kameraposition und dem Farbwert gebildet. Danach werden die Schnittpunkte mit

der Bounding Box berechnet. Diese geben den Bereich des Sichtstrahls an der innerhalb des Volumens ist. Dafür wird die Methode Intersect verwendet, die einen Vektor *intersections* liefert, der die beiden angesprochenen Werte beinhaltet. Um zu berechnen, ob ein Strahl die Bounding Box schneidet werden nur zwei der acht Eckpunkte benötigt, mit diesen lassen sich alle anderen Eckpunkte ausdrücken. Zuerst werden die Schnittpunkte mit allen sechs Flächen der Bounding Box berechnet (5.2). Die Schnittpunkte werden dann sortiert um die minimalen (*tmin*) und maximalen (*tmax*) Werte zu finden. In *tmin* sind die Flächen gespeichert, die am nächsten zur Kamera sind. Die entfernten Flächen werden in *tmax* gespeichert. Die letzte Fläche die in *tmin* eingetragen wurde, ist der Eintrittspunkt. Also wird das maximum von *tmin* berechnet. Um den Austrittspunkt zu bestimmen wird die erste Fläche gesucht die in *tmax* eingetragen wurde, also wird das minimum von *tmax* berechnet.

Listing 5.2 Intersect Methode

```
vec2 intersect(vec3 rayDirection, vec3 rayOrigin) {
    vec3 invR = 1.0 / rayDirection;
    vec3 tbot = invR * (vec3(0.0) - rayOrigin);
    vec3 ttop = invR * (vec3(1.0) - rayOrigin);

    vec3 tmin;
    tmin.x = min(ttop.x, tbot.x);
    tmin.y = min(ttop.y, tbot.y);
    tmin.z = min(ttop.z, tbot.z);
    vec3 tmax;
    tmax.x = max(ttop.x, tbot.x);
    tmax.y = max(ttop.y, tbot.y);
    tmax.z = max(ttop.z, tbot.z);

    float largest_tmin = max(max(tmin.x, tmin.y), max(tmin.x, tmin.z));
    float smallest_tmax = min(min(tmax.x, tmax.y), min(tmax.x, tmax.z));
    float tnear = largest_tmin;
    float tfar = smallest_tmax;
    return vec2(tnear,tfar);
};
```

Mit Hilfe dieses Vektors können jetzt die Farbwerte aus der Textur ausgelesen werden. Da die Daten nur RGB-Werte enthalten kann nur nur ein Teil der Rendergleichung ausgerechnet werden, der Emissionsterm. Für diese Rechnung wird einen Vektor *out_ color* in dem wir alle Farbwerte aufaddieren, einen Vektor *texColor* in den wir die Farbwerte speichern, einen Vektor *ci* in dem die behandelten Farbwerte gespeichert sind, eine Schrittweite *step* und eine Zählvariable *i* benötigt. Vom Startpunkt, der Farbwert des Vertex, aus entlang der berechneten Richtung wird der Strahl durch die 3D-Textur geschickt. In jedem Schritt, der Größe *step*, wird der RGB-Wert der Textur ausgelesen und in *texColor* gespeichert. Danach wird der Wert mit der Schrittweite, *step*, multipliziert und in *ci* gespeichert. Um die Helligkeit anzupassen wird der Wert noch mit der Intensität verrechnet. Danach wird der Wert zu dem bisherigen Farbwert *out_ color* addiert. Um an den nächsten Farbwert zu gelangen wird *i* um einen Schritt, *step*, erhöht.

Listing 5.3 Farbwertberechnung

```
vec3 out_color = vec3(0.0); vec3 texColor = vec3(0.0); vec3 ci = vec3(0.0);
float i = 0.0;
while(i < (intersections.y-intersections.x)) {
        texColor = (texture(Tex3D,(color + dir*i))).xyz;
        ci = (texColor * (step * intensity));
        out_color = out_color + ci;
        i = i + step;
}
out_FragColor = vec4(out_color,1.0);
```

5.2.2 Backfaces

Beim Rendern der Backfaces befindet sich die Kamera innerhalb der Bounding Box und damit auch potentiell innerhalb des Nebels. Daher werden im Fragmentshader zunächst die Eingabewerte *objPos* und *camera_ pos* in die Texturkoordinaten der Bounding Box transformiert. Zusätzlich wird die Blickrichtung und die Länge des Vektors von der Kameraposition zur Position des Vertex in Texturkoordinaten, also zur Fläche der Bounding Box auf dem der Vertex liegt berechnet. Um die Farbe zu bestimmen wird dasselbe Verfahren wie bei den Frontfaces angewendet. Der Startpunkt ist diesmal jedoch die Kamera.

6 Ergebnisse

Um die Implementierung zu testen wurden acht Nebel bereitgestellt. Im Folgenden wird ein kleiner Überblick über diese Nebel gegeben. Dazu wird von jedem Nebel ein Steckbrief erstellt, in dem Angaben zur Größe, zur Entfernung zur Erde, etc. gemacht werden. Jeder Steckbrief enthält eine 2D Aufnahme des Nebels und einige Bilder der 3D Modelle die mit der Implementierung (5) visualisiert wurden.

6.1 Calabash

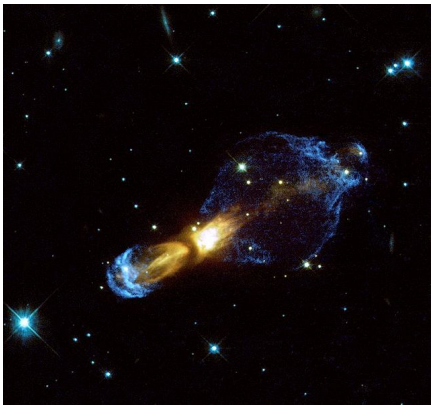

Abbildung 6.1: Calabash

Calabash[1] (6.1) ist ein präplanetarischer Nebel, ein Vorstadium zum planetarischen Nebel. Er ist rund 5000 Lichtjahre von der Erde entfernt, hat eine Länge von 1,4 Lichtjahren und sein Radius beträgt 0,7 Lichtjahre.

[1] http://en.wikipedia.org/wiki/Calabash_Nebula

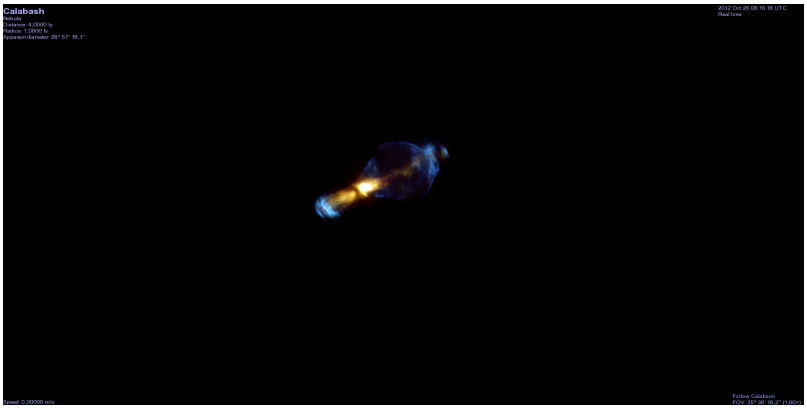

Abbildung 6.2: Calabash aus der Ferne

Abbildung 6.3: Calabash aus der Nähe

Abbildung 6.4: Aus dem Inneren von Calabash

6.2 Red Rectangle

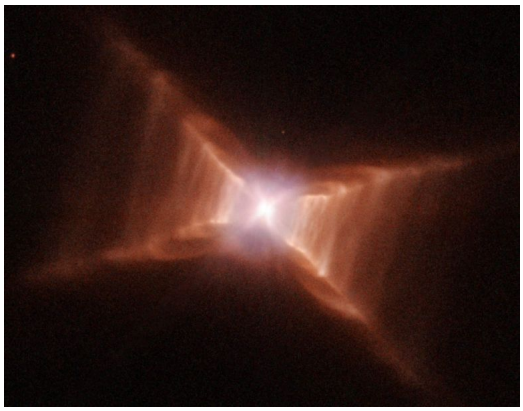

Abbildung 6.5: Red Rectangle

Der Red Rectangle[2] (6.5) ist, wie Calabash, ein präplanetarischer Nebel. Er ist rund 2300 Lichtjahre von der Erde entfernt.

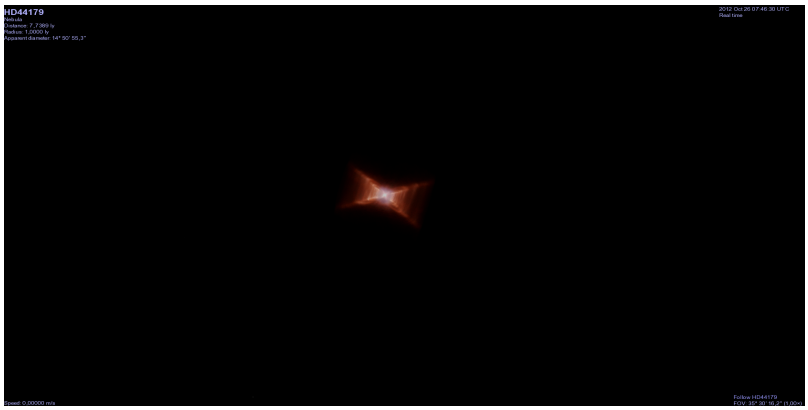

Abbildung 6.6: Red Rectangle aus der Ferne

[2] http://en.wikipedia.org/wiki/Red_Rectangle_Nebula

Abbildung 6.7: Red Rectangle aus der Nähe

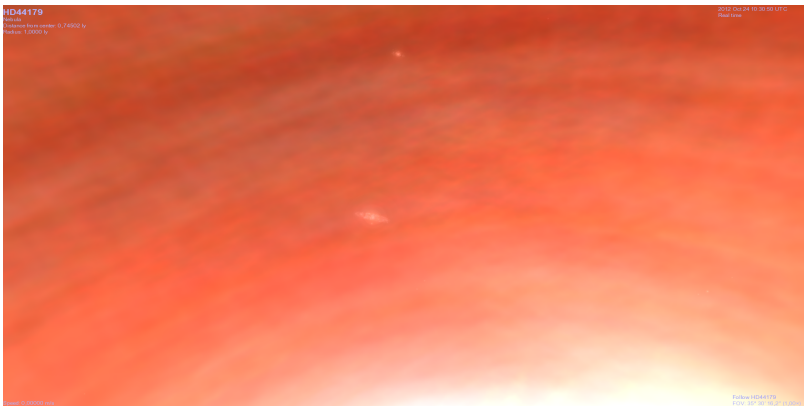

Abbildung 6.8: Aus dem Inneren von Red Rectangle

6.3 M2-9

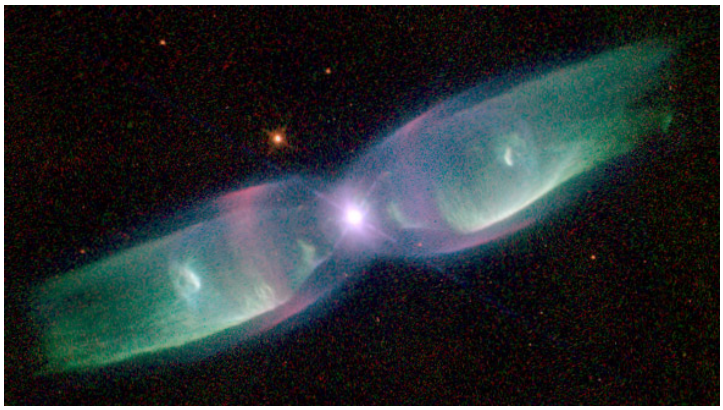

Abbildung 6.9: M2-9

M2-9[3] (6.9) ist ein planetarischer Nebel. Er ist rund 2100 Lichtjahre von der Erde entfernt und sein Radius beträgt 0,7 Lichtjahre.

Abbildung 6.10: M2-9 aus der Ferne

[3] http://en.wikipedia.org/wiki/Planetary_Nebula_M2-9

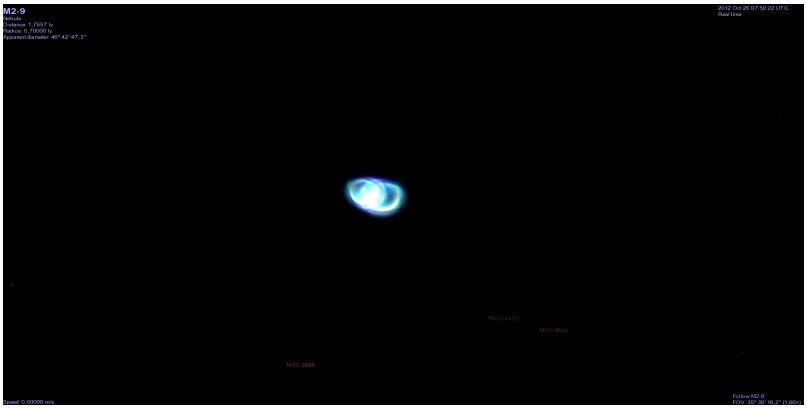

Abbildung 6.11: M2-9 aus der Nähe

Abbildung 6.12: Aus dem Inneren von M2-9

6.4 MZ3

Abbildung 6.13: MZ3

MZ3[4] (6.13) ist ein junger planetarischer Nebel und expandiert mit rund $50 km/s$. Er ist rund 8000 Lichtjahre von der Erde entfernt und sein Radius beträgt 1,0 Lichtjahre.

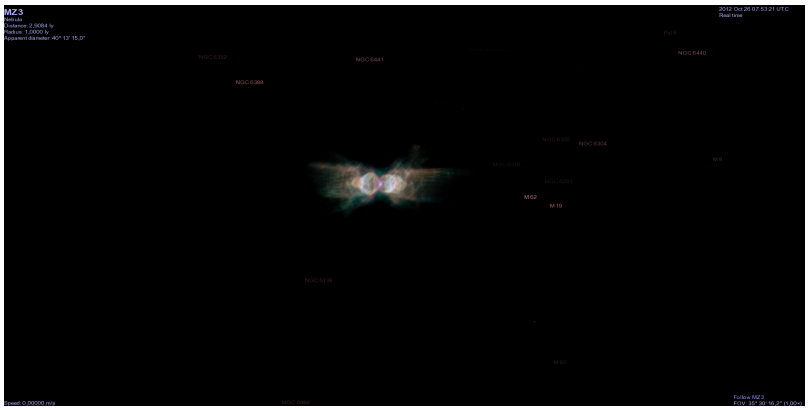

Abbildung 6.14: MZ3 aus der Ferne

[4] http://en.wikipedia.org/wiki/Mz_3

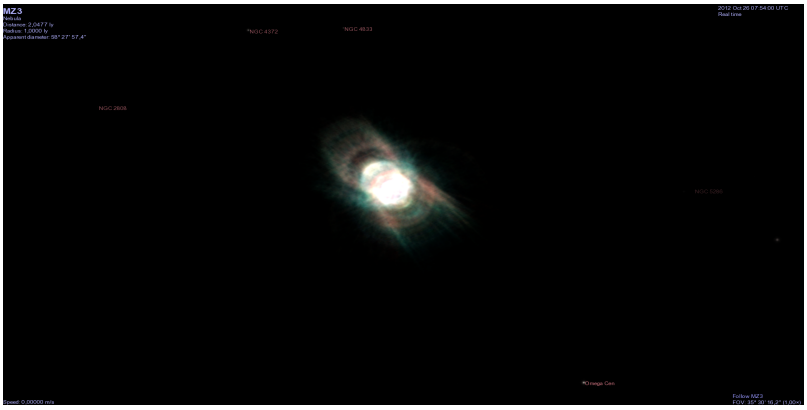

Abbildung 6.15: MZ3 aus der Nähe

Abbildung 6.16: Aus dem Inneren von MZ3

6.5 NGC6302

Abbildung 6.17: NGC6302

NGC6302[5] (6.17) ist ein planetarischer Nebel mit einem der heißesten Sterne im Zentrum mit Temperaturen um 200,000 Kelvin. Er ist rund 3100 Lichtjahre von der Erde entfernt und sein Radius beträgt 1,6 Lichtjahre.

Abbildung 6.18: NGC6302 aus der Ferne

[5] http://en.wikipedia.org/wiki/NGC_6302

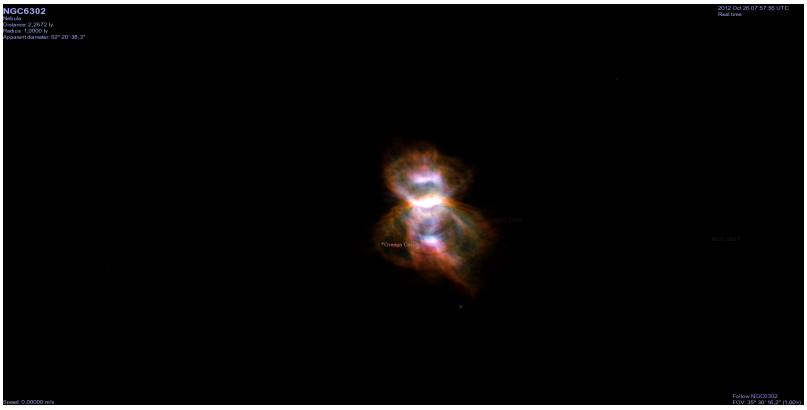

Abbildung 6.19: NGC6302 aus der Nähe

Abbildung 6.20: Aus dem Inneren von NGC6302

6.6 Cat's Eye

Abbildung 6.21: Cat's Eye

Cat's Eye[6] (6.21) ist ein planetarischer Nebel rund 1000 Jahre alt. Er ist rund 3300 Lichtjahre von der Erde entfernt und sein Radius beträgt 0,2 Lichtjahre.

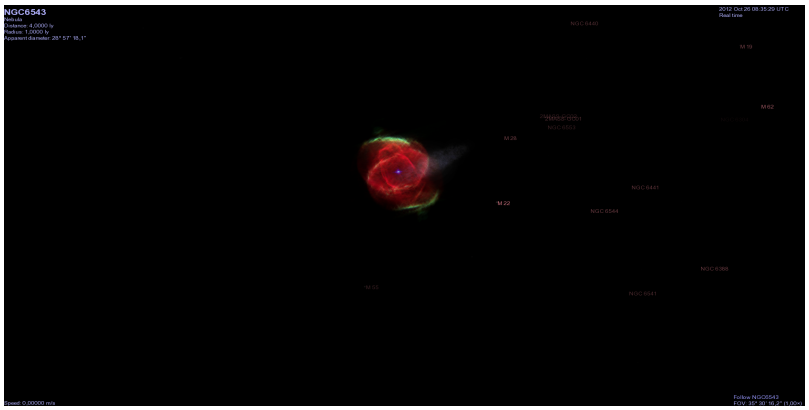

Abbildung 6.22: Cat's Eye aus der Ferne

[6] http://en.wikipedia.org/wiki/Cat's_Eye_Nebula

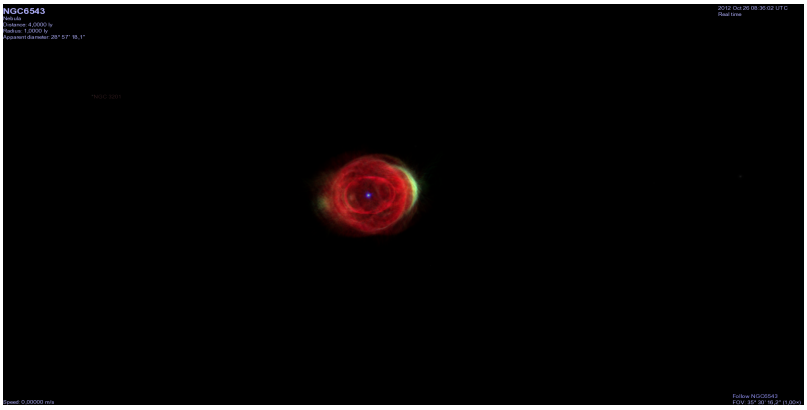

Abbildung 6.23: Cat's Eye aus der Nähe

Abbildung 6.24: Aus dem Inneren von Cat's Eye

6.7 NGC6826

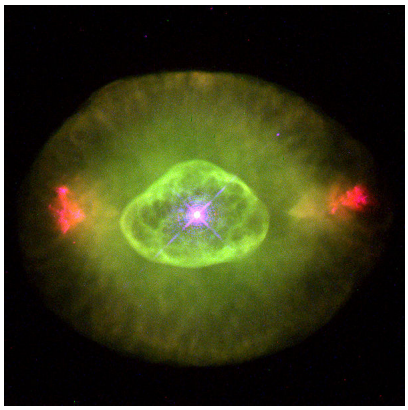

Abbildung 6.25: NGC6826

NGC6826[7] (6.25) ist ein planetarischer Nebel. Er ist rund 2000 Lichtjahre von der Erde entfernt und sein Radius beträgt 0,2 Lichtjahre.

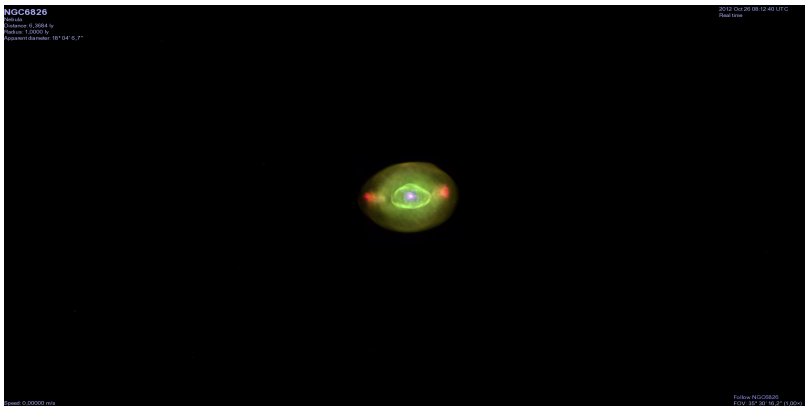

Abbildung 6.26: NGC6826 aus der Ferne

[7] http://en.wikipedia.org/wiki/NGC_6826

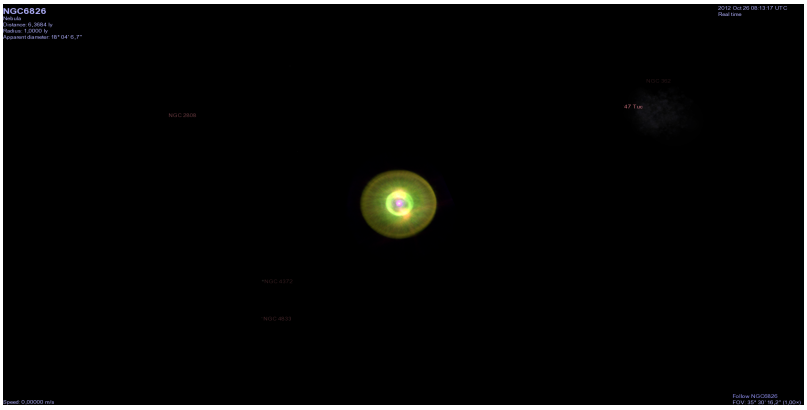

Abbildung 6.27: NGC6826 aus der Nähe

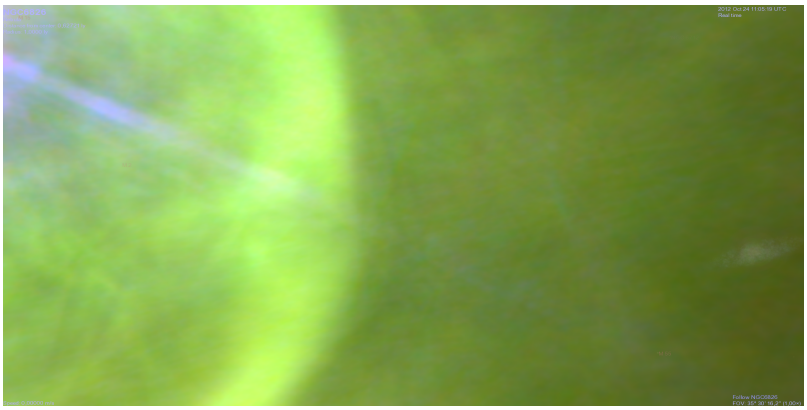

Abbildung 6.28: Aus dem Inneren von NGC6826

6.8 NGC7009

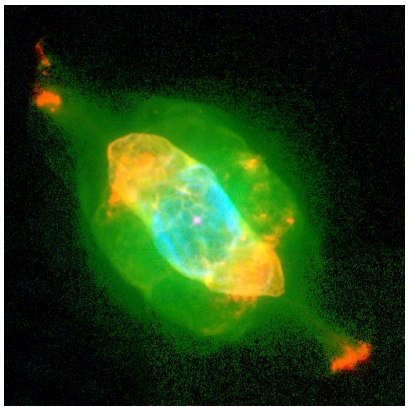

Abbildung 6.29: NGC7009

NGC7009[8] (6.29) ist ein planetarischer Nebel. Er ist rund 3000 Lichtjahre von der Erde entfernt und sein Radius beträgt 0,4 Lichtjahre.

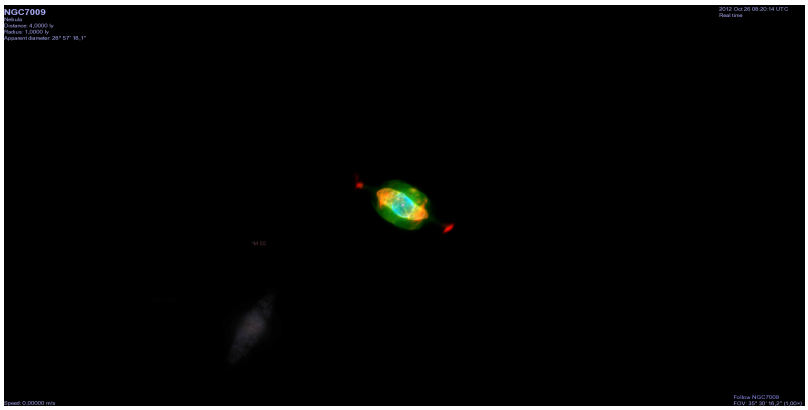

Abbildung 6.30: NGC7009 aus der Ferne

[8] http://en.wikipedia.org/wiki/Saturn_Nebula

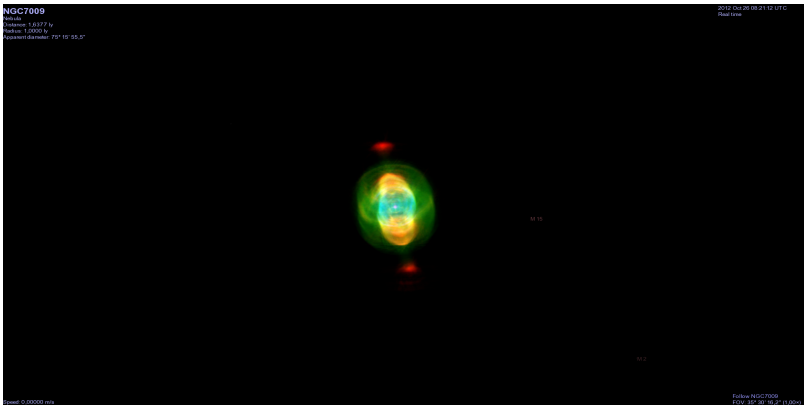

Abbildung 6.31: NGC7009aus der Nähe

Abbildung 6.32: Aus dem Inneren von NGC7009

7 Zusammenfassung und Ausblick

Der bisherige Ansatz Weltraumnebel, in Celestia, mit 3DS-Modellen darzustellen war begrenzt. So war es nicht möglich in Bereiche des Nebels vorzudringen, die nicht modelliert waren. Mit der neuen Implementierung ist dies möglich. Dafür müssen die Nebel nach dem Verfahren von [WAG+12] rekonstruiert werden, um danach mit einen Raycaster dargestellt werden zu können. Da das Ergebnis des rekonstruierten Weltraumnebels Volumendaten liefert die nur die Emissionswerte haben stehen keine Opazitätswerte zur Verfügung. Somit wird vom Volumen Rendering Integral nur der Emissionsteil, wie er in [Nel95] beschrieben wird, ausgewertet.
Die Implementierung wurde zunächst als Erweiterung zur bereits vorhandenen Umsetzung der 3DS-Modelle geplant, was zu Problemen führte. Aufgrund des groben Cullings wurden mehrere Nebel gleichzeitig geladen. Dies führte zu Fehlern und Abstürzen. Eine Anpassung der Culling Methode von Celestia löst das Problem. Nebel werden nur geladen, wenn die Distanz zur Kamera weniger als 250 Lichtjahre beträgt, ansonsten werden, bereits geladene Nebel, vom Grafikspeicher gelöscht.
Visualisiert werden die Nebel mit einem Raycaster, der zweigeteilt ist. Der erste Teil wird zum Rendern verwendet wenn die Kamera außerhalb des Volumens ist, der zweite Teil wenn die Kamera innerhalb des Volumens ist. Um zu bestimmen, ob die Kamera außer- oder innerhalb ist wird eine Bounding Box erzeugt, die sich den Dimensionen des Volumens anpasst. Diese wird auch zur Berechnung der Abtastpunkte verwendet, indem die Schnittpunkte eines Sichtstrahls mit der Bounding Box berechnet werden.
Mit dieser Implementierung ist eine deutlich detailliertere Visualisierung eines Weltraumnebels möglich, als es über 3DS Modelle möglich ist. Des Weiteren ist die Implementierung erweiterbar. Es können ohne Probleme eigene Renderklassen geschrieben werden und mit den Volumenmodellen verknüpft werden.

Ausblick

In Celestia hat jedes selektierbare Objekt einen Mittelpunkt. Dieser Mittelpunkt stimmt bei Planeten und Sternen mit dem physischen Mittelpunkt dieser Objekte überein. Die Weltraumnebel haben oft einen zentralen Stern, dieser befindet sich aber nicht immer im Zentrum der Volumendaten. Somit stimmt die Repräsentation nicht ganz, da der Mittelpunkt des selektierten Objekts nicht mit der Position des Stern übereinstimmt.
Um dieses Problem zu lösen müsste nach dem Laden des Nebels die Position des Sterns

bestimmt werden. Mit der Abweichung zum Zentrum des Volumens kann eine Translations-Matrix erstellt werden, mit der die Bounding Box verschoben wird. Danach muss dann die Schnittpunktberechnung im Fragmentshader ebenfalls angepasst werden.

Über Skripte[1] gibt es die Möglichkeit eine Kamerafahrt zu erstellen, ähnlich dem bereits vorhandenen "Demo-Skript" . Somit kann eine Key-Frame Interpolation erstellt werden. Key-Frames können mit den Befehlen: *goto* ,*rotate* und *move* erstellt werden. Jeder dieser Befehle hat einen Parameter *time* über den sich ein Zeitraum, in Sekunden, einstellen lässt. Innerhalb dieses Zeitraums wird die Aktion ausgeführt. Mit *goto* bewegt sich die Kamera auf das selektierte Objekt zu, bis die angegebene Entfernung erreicht ist. Über *rotate* kann eine Rotation um die x-,y- oder z-Achse visualisiert werden. Bewegungen in eine Richtung werden mit *move* umgesetzt, dabei wird die Geschwindigkeit mit einem Vektor angegeben. Die x-,y- und z-Komponente geben die Geschwindigkeit in diese Richtung an, mit negativen Werte entfernt sich die Kamera vom Objekt.

Da Reflexionsnebel keine Emission haben unterscheiden sich die dafür benötigten Volumendaten von denen die für die Emissionsnebel verwendet wurden. Um Reflexionsnebel darstellen zu können, müssen daher kleine Erweiterungen in der *VolumeModel* Klasse und in Celestia gemacht werden. Zum einem wird eine neue Methode zum Laden der Volumen benötigt. Und zum anderen muss die *filetype* Klasse um den neuen Dateityp erweitert werden. Die *mashmanager* Klasse muss dann bei dem neuen Dateityp die neue Methode zum Laden der Volumen aufrufen. Diese Methode muss dann auch ein Objekt der unten beschriebenen Klasse erzeugen.
Reflexionsnebel leuchten weil sie von Sternen angestrahlt werden und dieses Licht reflektieren. Daher muss vor jedem gerenderten Bild der Einfluss der nahen Sterne berechnet werden. Um an den Einfluss naher Sterne zu kommen, muss ihre Leuchtkraft und ihre Distanz bekannt sein. Die *render* Klasse besitzt eine Liste aller Sterne die von der aktuellen Position aus zu sehen sind. Aus dieser Liste kann jetzt der Einfluss auf den Nebel berechnet und gespeichert werden. Hierfür werden die Entfernung zur Position, die Leuchtkraft und die Farbe jedes Sterns in eine Liste eingetragen und an das betreffende *VolumeModel* Objekt übergeben, das diese dann an die Renderklasse übergibt.
Für die Darstellung wird eine eigene Renderklasse geschrieben die von *VolumeRender* erbt und z.B. nach der Methode von [MHLH05] arbeitet. Aus der Liste mit den Daten zu den Sternen wird für jeden Stern eine 3D Textur erstellt, die die Brechungstiefe für jeden Voxel, in Abhängigkeit zu diesem Stern, speichert.
Dazu muss die Entfernung noch angepasst werden. Um die Entfernung von einen Stern S zu dem Nebel N zu bekommen, wird der Vektor $\vec{SK}$ von der Position des Sterns S zur Kameraposition K gebildet. Dazu kommt der Vektor $\vec{NK}$ von der Nebelposition N zur Kameraposition K. Die Entfernung vom Stern S zum Nebel N ist dann die Länge des Vektors $\vec{SN} = \vec{SK} - \vec{NK}$.
Nachdem alle Texturen erstellt wurden kann der Nebel gerendert werden. Diese Texturen müssen bei jeder Änderung der Position neu berechnet werden, genauso wie die Liste der Sterne die Einfluss auf den Nebel haben.

[1] http://en.wikibooks.org/wiki/Celestia/Cel_Scripting

Literaturverzeichnis

[AFWMM08] J. Aja Fernández, S. Wenger, C. Morisset, M. Magnor. Algebraic 3D Reconstruction of Planetary Nebulae. Inst. f. Computergraphik, TU Braunschweig, 2008. (Zitiert auf den Seiten 9 und 14)

[AW87] J. Amanatides, A. Woo. A Fast Voxel Traversal Algorithm for Ray Tracing. In *Eurographics*, S. 3–10. 1987. (Zitiert auf Seite 16)

[DTM96] P. E. Debevec, C. J. Taylor, J. Malik. Modeling and Rendering Architecture from Photographs: A Hybrid Geometry- and Image-Based Approach. Technischer Bericht UCB/CSD-96-893, EECS Department, University of California, Berkeley, 1996. (Zitiert auf Seite 13)

[EHK+04] K. Engel, M. Hadwiger, J. Kniss, A. Lefohn, C. Rezk-Salama, D. Weiskopf. Real-Time Volume Graphics, SIGGRAPH Course Notes 28. ACM SIGGRAPH, 2004. (Zitiert auf den Seiten 6, 18 und 19)

[FM01] R. R. J. François, G. G. Medioni. Interactive 3D model extraction from a single image. *Image and Vision Computing*, 19:317–328, 2001. (Zitiert auf Seite 13)

[Lea91] D. A. Leahy. Deprojection of emission in axially symmetric transparent systems. *Astronomy and Astrophysics*, 247(2):584–589, 1991. (Zitiert auf Seite 14)

[LHM+07] A. Linţu, L. Hoffmann, M. Magnor, H. P. A. Lensch, H.-P. Seidel. 3D Reconstruction of Reflection Nebulae from a Single Image. In *VMV*, S. 109–116. 2007. (Zitiert auf Seite 14)

[LLM+07] A. Linţu, H. Lensch, M. Magnor, S. El-Abed, H.-P. Seidel. 3D Reconstruction of Emission and Absorption in Planetary Nebulae. In *Proc.of IEEE International Symposium on Volume Graphics*, S. 9–16. 2007. (Zitiert auf Seite 14)

[LMK03] W. Li, K. Mueller, A. Kaufman. Empty space skipping and occlusion clipping for texture-based volume rendering. In *Proc.of IEEE Visualization*, S. 317–324. 2003. (Zitiert auf Seite 20)

[MHLH05] M. A. Magnor, K. Hildebrand, A. Lintu, A. Hanson. Reflection Nebula Visualization. In *Proc.of IEEE Visualization*, S. 255–262. 2005. (Zitiert auf Seite 48)

[MKHD04] M. Magnor, G. Kindlmann, C. Hansen, N. Duric. Constrained Inverse Volume Rendering for Planetary Nebulae. In *Proc.of IEEE Visualization 2004*, S. 83–90. 2004. (Zitiert auf den Seiten 9 und 14)

[MKMD05] M. Magnor, G. Kindlmann, C. H. Member, N. Duric. Reconstruction and Visualization of Planetary Nebulae. *IEEE Transactions on Visualization and Computer Graphics*, 11:485–496, 2005. (Zitiert auf den Seiten 9 und 14)

[Nel95] M. Nelson. Optical Models for Direct Volume Rendering. *IEEE Transactions on Visualization and Computer Graphics*, 18:99–108, 1995. (Zitiert auf den Seiten 17, 18, 19, 21 und 47)

[NGN+01] D. R. Nadeau, J. D. Genetti, S. Napear, B. Pailthorpe, C. Emmart, E. Wesselak, D. Davidson. Visualizing Stars and Emission Nebulae. *Computer Graphics Forum*, 19:27–33, 2001. (Zitiert auf den Seiten 9 und 13)

[SKW+11] W. Steffen, N. Koning, S. Wenger, C. Morisset, M. Magnor. Shape: A 3D Modeling Tool for Astrophysics. *IEEE Transactions on Visualization and Computer Graphics*, 17(4):454–465, 2011. (Zitiert auf den Seiten 9 und 13)

[WAG+12] S. Wenger, M. Ament, S. Guthe, D. Lorenz, A. Tillmann, D. Weiskopf, M. Magnor. Visualization of Astronomical Nebulae via Distributed Multi-GPU Compressed Sensing Tomography. *IEEE Transactions on Visualization and Computer Graphics*, 18:2188–2197, 2012. (Zitiert auf den Seiten 6, 9, 13, 14, 15, 16 und 47)

[WO95] Z. Wen, C. R. O'dell. A three-dimensional model of the Orion Nebula. *Astrophysical Journal*, 438(2):784–793, 1995. (Zitiert auf Seite 13)

[YS93] R. Yagel, Z. Shi. Accelerating Volume Animation by Space-Leaping. *IEEE Conference on Visualization*, S. 62–69, 1993. (Zitiert auf den Seiten 18 und 20)

[ZDPSS02] L. Zhang, G. Dugas-Phocion, J.-S. Samson, S. M. Seitz. Single View Modeling of Free-Form Scenes. *IEEE Computer Vision and Pattern Recognition*, S. 990–997, 2002. (Zitiert auf Seite 13)