# Volume Rendering auf irregulären Gittern

## Theorie und Implementierung

Diplomarbeit[1] Informatik (FB 13), Technische Universität Berlin

---

Tobias Höllerer,
Matr.Nr. 127028,
Kottbusser Damm 65,
D-10967 Berlin,
Tel. (030)693 24 95

Aufgabensteller: PD. Dr. K. Tönnies
In Zusammenarbeit mit:
Konrad-Zuse-Zentrum Berlin, *Abt. Visualisierung & paralleles Rechnen*

2. Januar 1996

[1] mit experimentellem Charakter

*'Take some more tea,' the March Hare said to Alice, very earnestly.*
*'I've had nothing yet,' Alice replied in an offended tone, 'so I can't take more.'*
*'You mean you can't take less,' said the Hatter:*
*'it's very easy to take more than nothing.'*

## Zusammenfassung

*Direktes Volume Rendering* ist eine Visualisierungsmethode, mit deren Hilfe man räumliche Datenverteilungen als farbige semitransparente Dichtewolken in einem zweidimensionalen Bild darstellen kann. Die vorliegende Arbeit beschäftigt sich mit *Volume Rendering* als einer Technik zur direkten Visualisierung von dreidimensionalen wissenschaftlichen Datensätzen, die über irregulären Gittern definiert sind.

Während für reguläre Gitter bereits sehr effiziente *Volume Rendering* Verfahren entwickelt werden konnten, gestaltet sich eine schnelle *und* genaue Bildberechnung für irreguläre Gitter weiterhin problematisch.

Mit der vorliegenden Arbeit wird ein allgemeines Programmsystem für Volume Rendering auf nichtstrukturierten Gittern (VoRANG) vorgestellt, das über verschiedene Approximationen der Bildberechnung ein kontrolliertes Abwägen von Geschwindigkeit und Bildgenauigkeit ermöglicht.

Das Programmsystem ist bewußt offen und erweiterbar gehalten, so daß es als Testumgebung für verschiedene Strategien in unterschiedlichen Bereichen genutzt werden kann, z.B. für

- Definition beliebig dimensionaler Feldverteilungen
- beliebige Interpolation in den Gitterzellen
- neue Abbildungsmethoden von Daten- nach Farbwerten
- kompliziertere Beleuchtungsmodelle
- neue Approximationsmethoden für die Integration

Das Design und die Implementierung des Programmsystems ist im objektorientierten Stil gehalten, so daß Erweiterungen in den angesprochenen Bereichen einfach durch die Implementierung neuer Unterklassen erfolgen können.

Besonderes Augenmerk wurde auf eine klare modulare Struktur des Systems gelegt.

# Inhaltsverzeichnis

# Kapitel 1

# Einleitung

## 1.1 Motivation des Projektes

*Direktes Volume Rendering* hat sich als eine eindrucksvolle Methode zur grafischen Darstellung höherdimensionaler Datenfelder einen festen Platz im Bereich der wissenschaftlichen Visualisierung gesichert. Ein großer Vorteil dieser Methode ist, daß man schon in einem einzigen Bild eine Gesamtübersicht über die Datenverteilung im dreidimensionalen Raum erhalten kann.

Bei den Volume-Rendering-Methoden, die heute bereits in der wissenschaftlichen Praxis Verwendung finden, sei es im medizinischen, im naturwissenschaftlichen oder im ingenieurswissenschaftlichen Bereich, sind die Volumendaten in den allermeisten Fällen als Verteilungen auf regulären oder zumindest rectilinearen Gittern definiert.

In vielen wissenschaftlichen Bereichen finden jedoch Rechnungen und Messungen statt, die Volumendaten auf irregulären Gittern produzieren. *Computational Fluid Dynamics (CFD)* und die in numerischen Simulationen verwendeten *Finite Elemente Methoden (FEM)* sind zwei der bekanntesten Anwendungsgebiete, in denen mit unterschiedlichen Arten von nicht regulären Gittern gearbeitet wird.

Im folgenden wird zunächst eine Übersicht über Methoden gegeben, die eine Visualisierung solcher irregulär definierten Datensätze mit Hilfe des *Volume Rendering* Ansatzes ermöglichen. Dann wird mit dem System VoRANG eine allgemeine Programmumgebung für Volume Rendering auf unstrukturierten Gittern vorgestellt.

Das Hauptproblem an der Visualisierungsmethode *Volume Rendering* ist die große Komplexität, sowohl den Rechenaufwand als auch die Wahl geeigneter Parameter für die Bildberechnung betreffend. Die Methode besitzt das Potential, mit einem Bild mehr über einen Datensatz auszusagen als eine umfangreiche Serie von Schnittbildern. Die Suche nach den hierzu günstigsten Einstellungen (Transferfunktionen für Farbe und Dichte, zugrundeliegendes Beleuchtungsmodell, Blickwinkel etc.) kann sich jedoch als sehr aufwendig erweisen. Im Normalfall sind viele Berechnungen von Probebildern vonnöten, bis eine Einstellung für aussagekräftige Bilder gefunden ist. Eine schnelle Bildberechnung ist von daher sehr wichtig.

Nun ist gerade die Geschwindigkeit ein Hauptproblem der direkten Visualisierung auf irregulären Gittern. Für reguläre Gittergeometrien gibt es mittlerweile recht schnelle Algorithmen, die viel Rechenaufwand einsparen, indem sie die regelmäßige Anord-

nung der Gitter geschickt nutzen. Für irreguläre Gitter ist es weitaus schwieriger, effiziente Volume Rendering Algorithmen zu entwerfen.

Mit der Zeit werden von der im Bereich der wissenschaftlichen Visualisierung angesiedelten Forschung immer mächtigere Darstellungsmethoden und bessere Grundeinstellungen für bewährte Verfahren entwickelt werden, um die beschriebene Komplexität in den Griff zu bekommen. Hierfür sind eine Menge von Experimenten mit den immer wieder leicht zu verändernden gegenwärtig aktuellen Visualisierungsverfahren anzustellen.

Eine hauptsächliche Motivation für den Entwurf des Programmsystems VoRANG war die Perspektive, ein Werkzeug bei der Hand zu haben, mit dessen Hilfe auf solche Weise mit neuen Beleuchtungsmodellen, Approximationsverfahren, Darstellungsmethoden etc. im Bereich des *Volume Renderings* auf unstrukturierten Gittern experimentiert werden kann.

Entstanden ist dabei zunächst einmal ein System, das die Visualisierung von beliebig auf Tetraedergittern definierten Feldverteilungen in verschieden Approximationsstufen mit dem *Volume Rendering* Ansatz ermöglicht. Es ist dabei bewußt offen gehalten und auf leichte Erweiterbarkeit ausgelegt. Es seien stellvertretend drei Bereiche genannt, in denen zukünftig Experimentierarbeit geleistet werden soll:

- Gradientenabhängige Beleuchtung, kombinierte Anwendung von verschiedenen Beleuchtungsstrategien für verschiedene Bildbereiche
- Visualisierung von höherdimensionalen Feldverteilungen
- Visualisierung von Daten, die mit Hilfe von Interpolationsmethoden höherer Ordnung auf dem Gitter definiert sind — mit tragbarer Geschwindigkeit.

## 1.2 Überblick über die Kapitel

An dieser Stelle erfolgt als Lesehinweis für die vorliegende Arbeit eine kurze Übersicht über die nachfolgenden Kapitel.

Nachdem bisher in dieser Einleitung kurz in die Thematik eingeführt und die Motivation für die Arbeit vorgestellt wurde, beschäftigt sich Kapitel 2 ausführlich mit dem gegenwärtigen Stand der Forschung im Bereich *Volume Rendering auf irregulären Gittern*. Es wird zunächst in ein theoretisches Modell für die Bildsynthese beim Volume Rendering eingeführt. Es wird diskutiert, welche Berechnungen durchgeführt werden müssen, um von einer durch irregulär angeordnete *Samples* definierten Feldverteilung zu einem *Volume Rendering* Bild zu gelangen. Der zweite Teil von Kapitel 2 liefert eine Taxonomie von verschiedenen Verfahrensansätzen für das Volume Rendering auf irregulären Gittern. Das Kapitel endet mit einem Ausblick auf weiteren Forschungsbedarf.

Kapitel 3 bringt eine Anforderungsdefinition an das im Rahmen dieser Arbeit abgehandelte Projekt VoRANG. Es spiegelt somit die Aufgabenstellung wieder, die der Auslöser für die vorliegende Arbeit und vor allem das dabei entstandene Programmpaket war.

In Kapitel 4 werden die hauptsächlichen Entwurfsentscheidungen für das System VoRANG vorgestellt. Es wird eine Entscheidung für eine bestimmte Verfahrensart ge-

troffen, gestützt auf die Ergebnisse aus Kapitel 1. Weiterhin werden die wichtigsten Fragen für das Design und den Funktionsumfang geklärt.

Kapitel 5 bringt daraufhin den softwaretechnischen Systementwurf für das Projekt VoRANG. Da ein objektorientierter Programmieransatz verfolgt wurde, wird die Klassenstruktur des Programmsystems erarbeitet.

In Kapitel 6 werden ausgewählte Fragen der Implementierung diskutiert. Dabei wird vor allen Dingen auf die Erweiterbarkeit des Programmsystems eingegangen. Themenschwerpunkte sind die Definition von Feldverteilungen, Mapping von Daten- auf Farb- und Dichtewerte, und Näherungsmethoden für die Bildberechnung.

Testläufe und Resultate werden in Kapitel 7 präsentiert. Schon in Abfolge auf die Entwurfsentscheidungen sind am Ende von Kapitel 4 die wichtigsten Tests spezifiziert worden. Hier werden nun die Ergebnisse diskutiert.

Nach einer Zusammenfassung der Arbeit in „Kapitel“ 8 folgen im Anhang die wohl wichtigsten Unterlagen für eine Benutzung des Programmpaketes, nämlich die Beschreibungen zu der Skriptsprache, dem Dateiformat für Gitter und Daten und dem Dateiformat für das Color- und Alpha-Mapping.

Abschließend wird ein Überblick über die entstandene Softwareinstallation gegeben.

# Kapitel 2

# Stand der Forschung

## 2.1 Lichttransfermodell, Integrationsmethoden

Beim *Volume Rendering* Ansatz wird das Datenvolumen als dreidimensionale *Dichtewolke* von Partikeln aufgefaßt. Man modelliert die zu visualisierende Datenverteilung über dem Volumen als die Dichte imaginärer z.B. in Luft verteilter Partikel, d.h. die Datenfunktion über dem Volumen wird (in nachvollziehbarer Weise) auf eine kontinuierliche Dichtefunktion $\rho(x, y, z)$ abgebildet.

Wenn man $\rho$ als das lokal definierte Verhältnis des von Partikeln ausgefüllten Volumens $Vp$ zum Gesamtvolumen $V$ definiert, also $\rho(x, y, z) = \frac{dVp}{dV}$, kann $\rho(x, y, z)$ als die Wahrscheinlichkeit aufgefaßt werden, in einem infinitesimal kleinen Volumen $dV$ auf ein Teilchen zu stoßen [Sab88].

Abhängig davon, wie man die Beleuchtung des Volumens modelliert, besitzen die Partikel bestimmte Eigenschaften. Für die nachfolgende Beschreibung eines Lichttransfermodells wird ein einfaches *Absorptions-/Emissionsmodell* angenommen. Bei diesem ist das Volumen aus selbstleuchtenden (*emittierenden*) Partikeln aufgebaut, wobei vorgelagerte Teilvolumina das Licht aus dahinterliegenden Regionen dämpfen (*Absorption*).

Vorstellbar ist neben diesem einfachen Modell auch die Modellierung von Einfach- oder Mehrfachstreuung von Licht, das von außerhalb des Volumens liegenden Lichtquellen ausgeht. Es sei darauf hingewiesen, daß das hier beschriebene Modell zwar am anschaulichsten als System von selbstleuchtenden Partikeln verstanden werden kann, daß es aber allgemein genug ist, um solche Effekte zu inkorporieren. Einfachstreuung von von außerhalb einstrahlendem Licht wird beispielsweise so modelliert, daß die „Emission“ der Partikel für jeden Ort im Volumen abhängig von außerhalb liegenden Lichtquellen festgelegt wird, entsprechend den physikalischen Gegebenheiten. Für die Berücksichtigung von Mehrfachstreuung muß ein erheblich komplizierterer Ansatz zur Bestimmung der Lichtintensität an jedem Volumenpunkt zugrunde gelegt werden, da hierbei die Intensität an *einem* Ort im Volumen von den Intensitäten an *allen übrigen* Stellen abhängt.

Das Partikelmodell als Grundlage der Dichtewolkenmodellierung ist sehr anschaulich. Es existieren umfassende Herleitungen der zentralen Volume Rendering Gleichungen aus diesem physikalischen Ansatz heraus [Sab88, WM92]. Jeder Partikel besitzt eine bestimmte Leuchtintensität und ein bestimmtes Absorptionsvermögen. Verschie-

dene Materialien des darzustellenden Volumens können durch Regionen mit Partikeln unterschiedlicher Eigenschaften und natürlich über die Dichte ihrer Anordnung repräsentiert werden. Die Emission und die Absorption von Licht über bestimmten Teilregionen des Volumens können in diesem Rahmen exakt aus den Partikeleigenschaften bestimmt werden. Um jedoch z.B. oben genannte Verallgemeinerungen des Modells vorzunehmen, ist es sinnvoll, sich von den physikalischen Vorstellungen der Partikelebene zu lösen. Im folgenden werden die Eigenschaften des Datenvolumens an jedem Punkt einfach durch kontinuierliche Funktionen $\eta$ und $\chi$ beschrieben.

$\eta(\boldsymbol{x}, \boldsymbol{n}, \nu)$ repräsentiert die Lichtintensität an einem Punkt $\boldsymbol{x}$ des Volumens für eine Blickrichtung $\boldsymbol{n}$ und eine Frequenz $\nu$.

$\chi(\boldsymbol{x})$ repräsentiert den Absorptionskoeffizienten an einem Punkt $\boldsymbol{x}$ im Volumen.

Die Abbildung der zu visualisierenden Datenverteilung auf diese zwei Funktionen findet oft über vier Transferfunktionen statt: drei für die Intensitäten im Bereich roter, grüner und blauer Lichtwellenlängen $i_r, i_g, i_b$ und eine für die Dichtewerte $d$. [1]

Mit $f(\boldsymbol{x})$ als Verteilungsfunktion der zu visualisierenden Originaldaten bestimmen sich $\chi$ und $\eta$ wie folgt:

$$\chi(\boldsymbol{x}) = d(f(\boldsymbol{x})) \tag{2.1}$$

$$\eta(\boldsymbol{x}, \nu_\alpha) = i_\alpha(f(\boldsymbol{x})), \quad \alpha = r, g, b \tag{2.2}$$

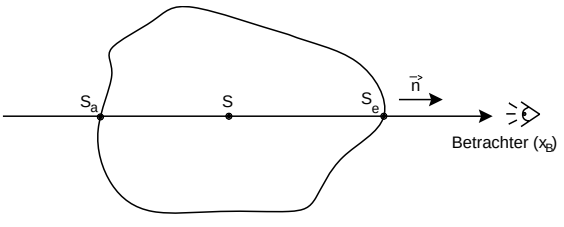
Man beachte, daß in diesem einfachen Fall $\eta$ unabhängig von der Blickrichtung ist. Für den abhängigen Fall gilt: $\eta(\boldsymbol{x}, \boldsymbol{n}, \nu_\alpha) = i_\alpha(f(\boldsymbol{x}), \boldsymbol{n})$.
Mitunter kann es auch erwünscht sein, ortsabhängige Informationen in die Transferfunktionen zu integrieren. Wenn man beispielsweise zusätzlich zur darzustellenden Skalarverteilung verdeutlichen möchte, daß verschiedene Volumenzellen aus verschiedenen Materialien bestehen, kann man für jede Volumenzelle „Materialkoeffizienten“ $C_\eta$ und $C_\chi$ definieren und als Gewichtungsfaktoren einbringen. Die Transferfunktionen können auch zusätzlich noch gradientenabhängig sein:

$$\chi(\boldsymbol{x}) = d(f,\ \nabla f,\ \boldsymbol{x}) \quad \text{und} \quad \eta(\boldsymbol{x}, \nu_\alpha) = i_\alpha(f,\ \nabla f,\ \boldsymbol{x}), \quad \alpha = r, g, b$$

Mit $\chi(\boldsymbol{x})$ als Funktion der Absorptionskoeffizienten ist nun

$$\tau(\boldsymbol{x}(s_1), \boldsymbol{x}(s_2)) = \int_{s_1}^{s_2} \chi(\boldsymbol{x}(s))\, ds \tag{2.3}$$

die optische Dichte entlang eines Strahles zwischen den Punkten $\boldsymbol{x}(s_1)$ und $\boldsymbol{x}(s_2)$[2]




Der Anteil Licht, der, emittiert an der Stelle s, das Auge des Betrachters erreicht (vgl. nebenstehende Abbildung), ist dann

$$\eta(\boldsymbol{x}(s), \boldsymbol{n}, \nu) \cdot e^{-\tau(\boldsymbol{x}(s), \boldsymbol{x}(s_e))} =$$
$$\eta(\boldsymbol{x}(s), \boldsymbol{n}, \nu) \cdot e^{-\int_s^{s_e} \chi(\boldsymbol{x}(s))\, ds}$$

Um nun die Gesamtintensität an Licht zu bestimmen, die entlang des Strahls

[1] Man kann auch die optische Dichte frequenzabhängig modellieren und muß dann entsprechend mehrere Transferfunktionen bereitstellen.

[2] $\boldsymbol{x}(s)$ ist ein Punkt auf einem Strahl $\boldsymbol{x}_0 + s \cdot \vec{n}$ in Abhängigkeit vom Strahlparameter $s$. $\boldsymbol{x}(s)$ steht, wie im folgenden immer, vereinfacht für $\boldsymbol{x}(P(s))$, wobei $P(s)$ eine Strahlparametrisierung ist.

| | |
|---|---|
| $f(\boldsymbol{x})$: | Funktion der Originaldaten für Volumenpunkte $\boldsymbol{x}$ |
| $i_r, i_g, i_b, d$: | Transferfunktionen: Originaldaten $\rightarrow$ Emission, Absorption |
| $\chi(\boldsymbol{x})$: | Funktion der Absorptionskoeffizienten für Volumenpunkte $\boldsymbol{x}$ |
| $\eta(\boldsymbol{x}, \boldsymbol{n}, \nu)$ | Funktion der Emissionskoeffizienten (Volumenpunkt $\boldsymbol{x}$, Richtung zum Betrachter $\boldsymbol{n}$, Frequenz $\nu$) |
| $\tau(\boldsymbol{x}(s_1), \boldsymbol{x}(s_2))$: | optische Dichte $= \int_{s_1}^{s_2} \chi(\boldsymbol{x}(s))\, ds$ |
| $I_0 = I(\boldsymbol{x}(s_a), \boldsymbol{n}, \nu)$: | spezifische Intensität am Volumenrand (Stelle $\boldsymbol{x}(s_a)$) |

Abbildung 2.1: *Übersicht über verwendete Terme.*

emittiert und durch das zwischen Emissionsquelle und Auge liegende Volumen abgeschwächt wird, bildet man das Integral über alle solchen „Emissionsstellen“ $s$ von $s_a$ bis $s_e$:

$$\int_{s_a}^{s_e} \eta(\boldsymbol{x}(s), \boldsymbol{n}, \nu)\, e^{-\tau(x(s), x(s_e))}\, ds \tag{2.4}$$

In diesem Ausdruck fehlt noch der Lichtanteil, der vom Betrachter aus gesehen hinter dem Volumen emittiert wird, also als Hintergrundsintensität bereits am Punkt $\boldsymbol{x}(s_a)$ vorliegt. Sei $I_0 = I(\boldsymbol{x}(s_a), \boldsymbol{n}, \nu)$ die spezifische Intensität am Volumenrand, dann ergibt sich

$$I(\boldsymbol{x}, \boldsymbol{n}, \nu) = I_0\, e^{-\tau(\boldsymbol{x}(s_a), \boldsymbol{x}(s_e))} + \int_{s_a}^{s_e} \eta(\boldsymbol{x}(s), \boldsymbol{n}, \nu)\, e^{-\tau(x(s), x(s_e))}\, ds \tag{2.5}$$

Gleichung (2.5) ist die zentrale *Volume Rendering* Gleichung. Man erhält eine bildliche Darstellung des Originaldatenraums, indem die Lichtanteile berechnet werden, die das Auge eines beliebig positionierten Beobachters erreichen. Die Gleichung beschreibt genau den Lichtanteil, der je Frequenz $\nu$ an einem Punkt $\boldsymbol{x}$ für eine Betrachterrichtung $\boldsymbol{n}$ gemäß der Beleuchtung vorliegt, die durch Wahl der Emissions- und Absorptionsterme modelliert wird.

Williams und Max [WM92] entwickeln ein dem hier vorgestellten Modell sehr ähnlines *Continuous Volume Density Optical Model*. Der Unterschied ist, daß sie Emission und Absorption koppeln, indem sie als effektive Funktion der Emissionskoeffizienten ein Produkt von $\eta$ und $\chi$ annehmen:

$$I(\boldsymbol{x}, \boldsymbol{n}, \nu) = I_0\, e^{-\tau(\boldsymbol{x}(s_a), \boldsymbol{x}(s_e))} + \int_{s_a}^{s_e} \chi(\boldsymbol{x}) \cdot \eta(\boldsymbol{x}(s), \boldsymbol{n}, \nu) \cdot e^{-\int_{s_e}^{s} \chi(\boldsymbol{x}(s'))\, ds'}\, ds \tag{2.6}$$

Gute Darstellungen von mathematischen Modellen für Volume Rendering (incl. Herleitungen) finden sich in [Sab88, MHC90, WVG91, WM92, HHS94]

Um nun entsprechend dem hier vorgestellten Modell Bilder von dreidimensionalen Datenverteilungen zu produzieren, muß für jeden Pixel des Ergebnisbildes Gleichung

(2.5) gelöst werden. Die nachfolgenden Abschnitte beschäftigen sich mit (effizienten) Lösungsstrategien für die Gleichung.

### 2.1.1 Lösungen für die Lichttransfergleichung

Im Rahmen des hier zu betrachtenden Emissions-Absorptions Modelles läßt sich Gleichung (2.5) durch einfache Strahlenintegration lösen. Da das zu bestimmende Integral einer Lösung in geschlossener Form nur für ganz eingeschränkte Sonderfälle zugänglich ist (vgl. Abschnitt 2.1.1.2), muß man im allgemeinen mit einem numerischen Ansatz arbeiten und den Strahl seiner Länge nach diskretisieren.

Der Strahl wird solcherart unterteilt, daß die entstehenden Teilintegrale entweder exakt analytisch bestimmt oder durch vernünftige einschränkende Annahmen in Näherung berechnet werden können (s. Abschnitt 2.1.1.2).

Die ausgewerteten Teilintegrale werden schließlich geeignet zur Gesamtintensität entlang des ganzen Strahles zusammengefaßt. Dieser Schritt des Zusammenfügens von Einzelintensitäten läuft unter dem Begriff *Compositing* ab.

#### 2.1.1.1 Compositing von Teilintegralen

Wird der gesamte Integrationsbereich entlang eines Strahles vom Anfangspunkt $s_a$ bis zum Endpunkt $s_e$ entsprechend der Bennennnungskonventionen in Abb. 2.2 in $n$ Teilintervalle zwischen Stützstellen $s_0$ bis $s_n$ unterteilt, so verhält sich die akkumulierte Intensität an der Stelle $s_k$ zur Intensität an der Stelle $s_{k-1}$ folgendermaßen:

$$I(s_k) = I(s_{k-1}) \cdot e^{-\tau(\boldsymbol{x}(s_{k-1}),\boldsymbol{x}(s_k))} + \int_{s_{k-1}}^{s_k} \eta(\boldsymbol{x}(s), \boldsymbol{n}, \nu) \cdot e^{-\tau(\boldsymbol{x}(s),\boldsymbol{x}(s_k))}\, ds. \qquad (2.7)$$

Die akkumulierte Intensität an der Stelle $s_k$ entsteht, indem die Intensität, die als Beitrag des Intervalls $[s_0, s_{k-1}]$ bestimmt wurde, nämlich $I(s_{k-1})$, durch die Absorption im Intervall $[s_{k-1}, s_k]$ geschwächt wird und die aufintegrierte Intensität des neuen Intervalls hinzukommt.

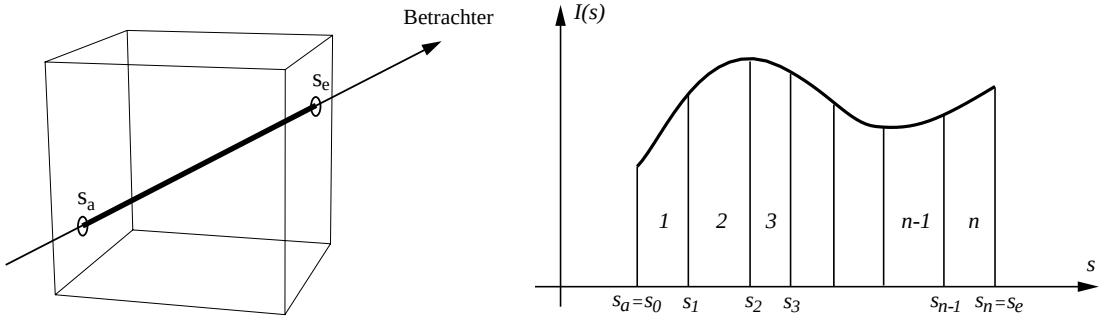



Abbildung 2.2: *Benennungskonventionen für die Strahlenintegration.*

Ist es für für den Fall regulärer Datengitter noch der Normalfall, daß die Stützstellen $s_k$ der Diskretisierung in gleichen Abständen gewählt werden, so ist dies im Falle

irregulärer Gitter fast nie der Fall. Meistens werden beim Volume Rendering auf irregulären Gittern die Schnittpunkte des Strahles mit den Volumenzellen als Stützstellen gewählt und sind demnach in unregelmäßigen Abständen angeordnet.

Für die nachfolgenden Betrachtungen seien folgende abkürzende Schreibweisen eingeführt:

$$\theta_k = e^{-\tau(\boldsymbol{x}(s_{k-1}),\boldsymbol{x}(s_k))} \qquad \text{und} \qquad b_k = \int_{s_{k-1}}^{s_k} \eta(\boldsymbol{x}(s),\boldsymbol{n},\nu)\cdot e^{-\tau(x(s),\boldsymbol{x}(s_k))}\,ds. \tag{2.8}$$

$b_0$ sei definiert als $b_0 := I(s_0) = I_0$, also als Intensität am Volumenrand $s_0$ (Hintergrundsintensität).

Die eingeführte Größe $\theta_k$ kann als *Transparenz* und $b_k$ als die *Leuchtintensität* des Materials zwischen $s_{k-1}$ und $s_k$ aufgefaßt werden. In der Literatur wird $b_k$ oft als *kumulative Zellenintensität* (*cumulative cell intensity*) und $\theta_k$ als *kumulative Zellentransparenz* bzw. $(1-\theta_k)$ als *cumulative cell opacity* bezeichnet. Man beachte, daß die $b_k$ frequenz- und richtungsabhängig sind, auch wenn dies im folgenden nicht ausdrücklich betont wird.

Die rekursive *Compositing* Gleichung (2.7) läßt sich direkt in einen Algorithmus umsetzen, der die Teilintegrale von hinten nach vorne zusammenfaßt (s. *Back-To-Front Compositing* in Abb. 2.3). Die Variable $I_{akk}$ enthält nach Schleifenende die gesuchte aufintegrierte Gesamtintensität entlang des Strahles.

```
I_akk = b_0;
for (k = 1; k ≤ n; k++)
{
    I_akk = θ_k · I_akk + b_k;
}
```

*Back-To-Front*

```
I_akk = b_n;
T_akk = θ_n;
for (k = n − 1; k ≥ 0 and T_akk > ε; k--)
{
    I_akk = I_akk + b_k · T_akk;
    T_akk = θ_k · T_akk;
}
```

*Front-To-Back*

Abbildung 2.3: *Zwei Verfahren zum Compositing.*

Das *Compositing* läßt sich auch von vorne nach hinten durchführen. Beim *Front-To-Back* Algorithmus muß man nur zusätzlich die Transparenz laufend aktualisieren (Variable $T_{akk}$ im Pseudocode in Abb. 2.3). Dieser Algorithmus hat den Vorteil, daß die Schleife abgebrochen werden kann, wenn die akkumulierte Transparenz so gering wird, daß dahinterliegende Volumenelemente keinen nennenswerten Beitrag mehr liefern können.

Beide Algorithmen lösen folgende mathematische Gleichung, die sich aus der rekursiven Gleichung (2.7) ergibt:

$$I(s_n) \;=\; I(s_{n-1})\theta_{n-1} + b_{n-1} = \Big(I(s_{n-2})\theta_{n-2} + b_{n-2}\Big)\theta_{n-1} + b_{n-1} = \ldots$$

$$= \sum_{k=0}^{n} b_k \prod_{j=k+1}^{n} \theta_j \tag{2.9}$$

Nachdem soeben die Synthese des Strahlenintegrals aus mehreren Teilintegralen besprochen wurde, soll im folgenden darauf eingegangen werden, wie die benötigten Größen $\theta_k$ und $b_k$ für die Intervalle zwischen den Stützstellen berechnet werden können. Insbesondere werden verschiedene Näherungen untersucht, die auf Spezialfällen oder Abschätzungen beruhen.

#### 2.1.1.2 Näherungen für die Teilintegrale

Für jedes Intervall zwischen den Stützstellen $s_k$ müssen nun also die beiden Größen

$$\theta_k = e^{-\int_{s_{k-1}}^{s_k} \chi(s)\,ds} \qquad \text{und} \qquad b_k = \int_{s_{k-1}}^{s_k} \eta(s) \cdot e^{-\int_{s}^{s_k} \chi(s')\,ds'}\,ds. \tag{2.10}$$

bestimmt werden. Dafür gibt es mehrere Möglichkeiten:

Eine exakte Berechnung in geschlossener Form ist leider nur für spezielle Annahmen bezüglich der Datenverteilung möglich. Alternativ arbeitet man mit Näherungen, die man durch Termabschätzung oder Mittelwertbildung über dem Strahlenintervall gewinnt.

Schließlich gibt es noch spezielle Möglichkeiten der Approximation, wenn man ein sogenanntes *Projektionsverfahren* zur Grundlage des *Volume Rendering* macht (s. Abschnitt 2.2.2.1). In diesem Fall werden nämlich alle Teilintervalle von Strahlen, die durch *eine* Volumenzelle hindurchführen, sozusagen im Paket abgearbeitet. Jedes dieser Teilintervalle ist genau einem Bildschirmpixel in der Projektion der Zelle zugeordnet. Es ergibt sich die Möglichkeit, nur für wenige geschickt gewählte Pixel in der Zellprojektion die genauen Farbintensitäten und Transparenzen zu bestimmen und für alle anderen Pixel die Werte zu interpolieren.

Eine Weiterführung dieser Idee, die allerdings eine sehr grobe Näherung der angenommenen physikalischen Phänomene darstellt, ist zu guter Letzt das sogenannte *Splatting*.

Im folgenden werden die einzelnen Näherungsverfahren genauer dargestellt:

##### Spezialfälle in geschlossener Form

Um Lösungen der Gleichungen (2.10) in geschlossener Form zu erhalten, müssen die Integrandenfunktionen einer analytischen Integration zugänglich sein. Die nachfolgenden drei Sonderfälle basieren alle auf speziellen Annahmen für die Funktionen $\chi$ und $\eta$. Es sei daran erinnert, daß diese Funktionen durch Anwendung von benutzerdefinierten Transferfunktionen auf die ursprüngliche Datenverteilung $f(\boldsymbol{x})$ entstehen ($\chi(\boldsymbol{x}) = d(f(\boldsymbol{x}))$, $\eta(\boldsymbol{x}) = i(f(x))$, vgl. Gleichungen (2.1)).

**1. Konstante Emission und Absorption** Annahme: Die Funktionen $\eta$ und $\chi$ sind innerhalb der zu betrachtenden Volumenzelle konstant:

$$\eta(\boldsymbol{x}(s)) = K_\eta \;\text{ und }\; \chi(\boldsymbol{x}(s)) = K_\chi, \qquad \text{für } s_{k-1} \le s \le s_k$$

Diese Annahme trifft insbesondere für beliebige Transferfunktionen $d(f)$ und $i(f)$ zu, wenn die Funktion $f$ innerhalb der Zelle konstant ist. Für konstante Emission und Absorption lassen sich die kumulative Zelltransparenz und -intensität wie folgt bestimmen:

$$\theta_k = e^{-\int_{s_{k-1}}^{s_k} \chi(s)\,ds} = e^{-K_\chi \cdot \Delta s_k} \quad \text{mit } \Delta s_k = (s_{k-1} - s_k) \tag{2.11}$$

$$b_k = \frac{K_\eta (1-\theta_k)}{K_\chi} \tag{2.12}$$

Wenn man dieses Näherungsverfahren auch in Fällen einsetzen will, in denen Emission und Absorption über einer Zelle eigentlich variieren, kann man zunächst jeweils den Mittelwert der Emissions- und Absorptionskoeffizienten an den Eintritts- und Austrittspunkten bestimmen und diese Mittelwerte als konstant über die Zelle annehmen.

**2. Emission und Absorption proportional** Annahme: Die Funktionen $\eta$ und $\chi$ sind zueinander proportional:

$$\eta(s) \sim \chi(s)$$

Wenn man dies für jede einzelne Volumenzelle sicherstellen will, kann man die Transferfunktionen $i(f)$ und $d(f)$ global für alle Zellen durch dieselbe Proportionalitätskonstante miteinander verknüpfen:

$$i(f) = K_0 \cdot d(f), \qquad K_0 : \text{ Konstante}$$

Will man hingegen für verschiedene Zellen des Volumens verschiedene Verhältnisse von Emission und Absorption erzielen, muß man ortsabhängige Transferfunktionen $i'(f, \boldsymbol{x})$ und $d'(f, \boldsymbol{x})$ benutzen, wie bereits in Abschnitt 2.1, Seite 5 beschrieben. Wenn man als Ortsabhängigkeit für jede Zelle *konstante Materialkoeffizienten* in das Modell einbringt, gilt obige Proportionalitätsbeziehung auch für $i'$ und $d'$.

Auf welchem Wege auch immer die Annahme erfüllt wird, so gilt schließlich innerhalb jeder einzelnen Zelle, daß $\eta$ und $\chi$ proportional *zueinander* und damit auch jeweils zu einer angenommenen Dichtefunktion $\rho(\boldsymbol{x}(s))$ sind:

$$\eta = K_\eta \cdot \rho \text{ und } \chi = K_\chi \cdot \rho, \qquad K_\eta, K_\chi : \text{ Konstanten}$$

Unter dieser Voraussetzung kann man $b_k$ direkt aus $\theta_k$ bestimmen[3]

$$b_k = \frac{K_\eta}{K_\chi} (1 - \theta_k) \tag{2.13}$$

Es fällt auf, daß diese Gleichung mit (2.12) identisch ist. Dies ist nachvollziehbar, wenn man bedenkt, daß der Fall konstanter Emissions- und Absorptionskoeffizienten ein Sonderfall von Proportionalität ist, wie sie hier vorliegt.

Für eine exakte Berechnung von $\theta_k$ in geschlossener Form muß man $\chi$ analytisch integrieren können. Für einen linearen Verlauf von $\chi$ zwischen $s_{k-1}$ und $s_k$ ergibt sich beispielsweise (nach Trapezregel):

[3] vgl. Herleitungen u.a. in [HHS94], S.15 f., und [MHC90], S. 29.

$$\theta_k = e^{-\frac{1}{2}(\chi(x(s_{k-1}))+\chi(x(s_k)))\,\Delta s_k}, \quad \text{mit } \Delta s_k=(s_k-s_{k-1}) \tag{2.14}$$

Ein linearer Verlauf von $\chi$ zwischen $s_{k-1}$ und $s_k$ liegt dann vor, wenn die Originaldatenfunktion $f$ linear über die Zelle interpoliert wird und gleichzeitig die Transferfunktion $d(f)$ über dem Intervall $[f(\boldsymbol{x}(s_{k-1})), f(\boldsymbol{x}(s_k))]$ linear ist.

**3. Emission und Absorption linear, aber unabhängig** Für den Fall linearer Transferfunktionen $d(f)$ und $i(f)$ — bei gleichzeitig linearem Verlauf der Originaldatenfunktion $f$ entlang des Strahles durch die Zelle — kann man auch dann eine exakte Lösung für $b_k$ bestimmen, wenn Emission und Absorption nicht proportional zueinander sind. Diese ist allerdings recht kompliziert und einigermaßen rechenaufwendig.

In [WM92] wird für den Fall, daß $\chi$ und $\eta$ lineare Funktionen des Strahlparameters s sind, eine Lösung für das in Gleichung 2.6 auf S.6 beschriebene optische Modell hergeleitet. Diese Lösung ist nicht in geschlossener Form angegeben, sondern erfordert die Tabularisierung oder Approximation der Integralfunktionen $\int_0^x e^{u^2} du$ und $\int_0^x e^{-u^2} du$. Die Berechnung ist insgesamt relativ aufwendig und wohl nicht dazu geeignet, für kleine Intervalle $[s_{k-1}, s_k]$ an jedem einzelnen Pixel ausgewertet zu werden, wenn Geschwindigkeit eine große Rolle spielt. Hingegen interessant ist eine solches Berechnungsschema zur Bestimmung genauer Referenzbilder. Immerhin erlaubt diese Methode ganz ohne numerische Approximation die Berechnung von Volume Rendering Bildern unter Einsatz stückweise linearer Transferfunktionen. In [SBM94] wird dieses Ergebnis im Rahmen eines Projektionsverfahrens dazu eingesetzt, für *einen* Pixel pro projizierter Volumenzelle eine exakte Integration vorzunehmen. Für die restlichen Pixel werden Ergebniswerte interpoliert (vgl. S. 14).

Für das in dieser Arbeit zugrunde gelegte optische Modell (Gleichung 2.5) wird in Abschnitt 6.3.3 eine Lösung für lineare Emission und Absorption entwickelt. Es zeigt sich, daß das Ergebnis einfacher zu formulieren ist als für den Fall in [WM92].

### Näherung durch Termabschätzung oder Mittelwertbildung

Nachdem im vorangegangenen Abschnitt die Gleichungen für die kumulative Zelltransparenz und -intensität $\theta_k$ und $b_k$ für Spezialfälle exakt gelöst wurden, seien im folgenden noch zwei einfache numerische Abschätzungen für die beiden Terme genannt.

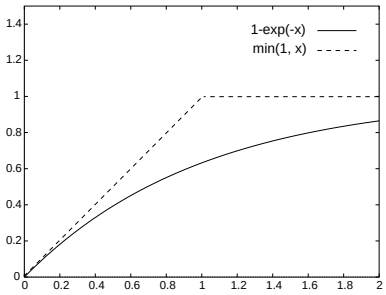
**1. Abschätzung durch Trapezregel** Schon in Gleichung 2.14 wurde die Trapezregel verwendet, um im Falle eines linear variierenden Absorptionskoeffizienten $\chi$ die kumulative Zelltransparenz $\theta_k$ zu bestimmen. Auf ähnliche Art und Weise und unter Verwendung eben dieser Zelltransparenz kann man als Näherung auch die kumulative Intensität $b_k$ bestimmen:

$$b_k = \frac{1}{2}(\eta(x(s_{k-1}))\,\theta_k + \eta(x(s_k)))\,\Delta s_k \tag{2.15}$$

Man bildet also den Mittelwert zwischen abgeschwächter Emission am Eintrittspunkt und der Emission am Austrittspunkt der Zelle und multipliziert ihn mit der

Entfernung dazwischen. Dadurch vermeidet man die weitere Berechnung eines exponentiellen Ausdrucks. Diese Näherung tendiert zu etwas stärkerer Intensität, als im theoretischen Modell begründet liegt.

**2. Abschätzung durch Mittelwertbildung** Eine der vorangegangenen nicht unähnliche Methode zur Bestimmung der beiden Terme, die allerdings ganz auf die Auswertung der $e$-Funktion verzichtet, wird von Wilhelms und Van Gelder unter dem Namen *Average C*D Integration* vorgeschlagen [WVG91]. Im Grunde ist diese Methode eine weitere Vereinfachung des Falles konstanter Koeffizienten, der in den Gleichungen (2.11) und (2.12) beschrieben wurde. Zunächst werden die Mittelwerte für Emissions- und Absorptionskoeffizienten an Ein- und Austrittspunkt der Zelle bestimmt ($\eta_\emptyset$ & $\chi_\emptyset$).




Diese werden als für die Zelle konstant angenommen und *zusätzlich* wird noch der Term $(1 - e^{-\chi_\emptyset \cdot \Delta s_k})$ durch den Ausdruck $min(1, \chi_\emptyset \cdot \Delta s_k)$ angenähert (vgl. Abbildung links).

$$\theta_k = 1 - min(1, \eta_\emptyset \cdot \Delta s_k) \tag{2.16}$$

$$b_k = (\tfrac{\eta_\emptyset}{\chi_\emptyset})\,(1 - \theta_k) \tag{2.17}$$

Diese Methode erfordert einen nur sehr geringen Rechenaufwand, ist aber vor allem für kleine Volumenzellen sehr „ungenau“. Die auf diesem Wege bestimmte Transparenz ist eher schwächer und die Intensität eher stärker als nach dem theoretischen Modell.

### Interpolation von Farbe und Transparenz über die Zellprojektion

Die Näherungsverfahren in diesem Abschnitt sind, wie weiter oben bereits erwähnt, im Rahmen eines *Projektionsverfahrens* anwendbar. Bei den „exakten“ Projektionsverfahren werden (im Gegensatz zum *Splatting*, das im nächsten Abschnitt beschrieben wird) die genauen Umrisse einer Volumenzelle in die Bildebene projiziert. Abb. 2.4 zeigt die beiden einzig möglichen nichtdegenerierten Projektionen für den Fall einer Tetraederzelle. An diesem Beispiel sollen die Interpolations-Näherungsverfahren veranschaulicht werden[4].

Für jedes projizierte Tetraeder kann auf einfache geometrische Weise ein Punkt T (vgl. Abb. 2.4) in der Bildebene bestimmt werden, an dem die Tiefe des Tetraeders maximal ist. An den Rändern der Projektion ist die Tiefe des projizierten Tetraeders hingegen null, d.h. die kumulative Transparenz $\theta_k$ an den Randpunkten ist 1 und die kumulative Intensität $b_k$ ist null. Die Idee ist nun, die Farbbeiträge und Tranzparenzen ausgehend von an Punkt T und den Eckpunkten bestimmten Werten über die gesamte Projektion der Zelle zu interpolieren. Die lineare Interpolation der Datenwerte findet während der Scankonvertierung eines Polygones statt, indem zunächst entlang der Polygonkanten von Scanline zu Scanline linear interpoliert wird, und innerhalb jeder Scanline wiederum linear zwischen den Schnittpunkten von Scanline und Kante.

[4]Für andere Zellgeometrien kann man auf ähnliche Weise verfahren, nur die möglichen Topologien der Projektion werden komplizierter und man muß entsprechend für mehrere „Stützpunkte“ die genaue Berechnung vornehmen.

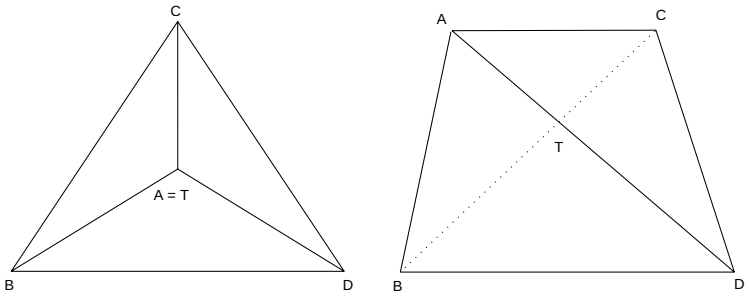



Abbildung 2.4: *Die beiden nichtdegenerierten Fälle einer Tetraederprojektion*

Im folgenden sollen zwei verschiedene Methoden genannt werden, um Farbbeitrag und Transparenz eines jeden Pixels in der Zellprojektion mit Hilfe von Interpolation zu bestimmen: Der *Gouraud Shading* Ansatz und *Exponentielle Interpolation*. Interpoliert werden dabei unterschiedliche Größen: Im ersten Fall direkt Farb- und Transparenzwerte und im zweiten Fall die Absorptions- und Emissionskoeffizienten und die Zellentiefe.

Zur Genauigkeit dieser Näherungsverfahren vgl. die Testläufe in Abschnitt 7.

**1. Gouraud Shading Interpolation** Dei diesem Ansatz werden direkt die beim *Compositing* verwendeten Farbbeiträge und Transparenzen linear über die Zellprojektion interpoliert. Für $\theta_k$, also die kumulative Transparenz, läßt sich das direkt durchführen: sie wird an Punkt T z.B. nach Gleichung 2.14 bestimmt und zum Rand hin nach 1 interpoliert.

Für die kumulativen Farbintensitäten $b_k$ ist der entsprechende Weg nicht gangbar, da es ein falsches Ergebnis bringt, wenn man die $b_k$ zum Rand hin einfach zu null hin (Farbe Schwarz) interpoliert. Das hieße nämlich, daß für die gesamte Zelle nur *ein* Farbwert (nämlich an T) bestimmt wird, der zum Rand hin immer mehr abgedunkelt würde. Die eigentlichen Farbwerte an den Eckpunkten der Zellprojektion würden vollkommen unter den Tisch fallen.

Stattdessen bestimmt man an den Eckpunkten einfach $\eta(x)$ und an Punkt T einen Mittelwert von $\eta_{front}$ und $\eta_{back}$[5] und ändert das *Compositing*-Verfahren (vgl. Gleichung 2.9) entsprechend, damit es statt der bisherigen $b_k$ nun die interpolierten $\eta$ Werte $\bar{b_k}$ verarbeitet:

$$I_{new} = I_{old} \cdot \theta_k + \bar{b_k} \cdot (1 - \theta_k)$$

Ein großer Vorteil bei diesem Vorgehen ist, daß diese *Compositing* Gleichung exakt dem *Alpha Compositing* entspricht, wie es von modernen Grafikworkstations direkt in Hardware unterstützt wird. Die Scankonvertierung von Polygonen mit *Gouraud-Shading* kann meist ebenfalls mit Hardwareunterstützung vorgenommen werden. Wichtig dabei ist jedoch, daß neben den Farbkanälen *rot*, *grün* und *blau* auch die Transparenz bzw. Opazität mitinterpoliert wird.

Damit ergibt sich insgesamt eine sehr schnelle hardwareunterstützte Bildberechnung.Der Einsatz von *Gouraud Shading* als Interpolationsmethode bei *Volume Rendering Projektionsverfahren* wurde das erste Mal in [ST90] vorgestellt.

[5] $\eta_{front}$ und $\eta_{back}$ sind die Emissionskoeffizienten am Zelleneintritts- und Zellenaustrittspunkt für den Strahl durch Punkt T. In der Implementierung dieser Approximation in VoRANG wird bei der Mittelung noch mit der aktuellen Transparenz gewichtet: $\eta_{\emptyset} = \eta_{back} \cdot \theta_k + \eta_{front} \cdot (1 - \theta_k)$

**2. Exponentielle Interpolation** Bei dieser Näherungsmethode werden die Teilintegrale für jeden Pixel der Zellenprojektion wieder einzeln nach einem der schon beschriebenen Näherungsverfahren (z.B. nach Gleichungen 2.11 und 2.12) bestimmt. Man verzichtet allerdings auf das Auswerten der Transferfunktionen $d$ und $i_\nu$ (vgl. S. 5) für jeden einzelnen Pixel und interpoliert stattdessen die Emissions- und Absorptionskoeffizienten und die Tiefe der Volumenzelle linear über die Zellprojektion. Das ist zumindest für die Zellentiefe realistisch ist, da diese zum Rand hin tatsächlich linear gegen null abfällt.

Dieses Verfahren kann man variieren, indem verschiedene Integrationsnäherungen zugrundegelegt werden und entsprechend andere Größen linear über die projizierte Fläche interpoliert werden. Z.B. kann man neben der Zellentiefe $l$ die *durchschnittlichen* Emissions- und Absorptionskoeffizienten $\chi_\varnothing$ und $\eta_\varnothing$ interpolieren und die Integration nach Gleichungen 2.11 und 2.12 vornehmen. Man kann auch die Koeffizienten an Vorder- und Rückseite der Volumenzelle interpolieren und nach einer etwas genaueren Methode integrieren (vgl. [WVG91]).

Es existieren mehrere Vorschläge, wie man den Vorteil der Hardware-Unterstützung bei der *Gouraud Shading* Methode beibehalten und gleichzeitig die Genauigkeit der Integralnäherungen durch einen exponentiellen oder zumindest quadratischen ([WVG91], S. 280) Interpolationsansatz verbessern kann. Z.B. läßt sich die Fähigkeit mancher Grafik-Hardware zum *Hardware Texture Mapping* benutzen, um eine zweidimensionale *Look-Up-Table* für den exponentiellen Ausdruck $\theta_k = e^{-\chi_\varnothing \cdot l}$ in Abhängigkeit der interpolierten Parameter $\chi_\varnothing$ und $l$ aufzubauen (vgl. [SBM94]).

### Splatting

Beim sogenannten *Splatting* wird schließlich auch von der Form der zu projizierenden Volumenzelle abstrahiert. Ähnlich einem Schneeball, der gegen eine Wand klatscht (*to splat*), werden die Beiträge für jede Zelle bei der Projektion nach einem benutzerdefinierten Schema auf mehrere benachbarte Pixel verteilt. Oft werden für jede Zelle semitransparente Oberflächenprimitive in der ungefähren Form der Zellenumrisse ins Bild hineingefügt, wobei die Tiefenausdehnung der Zelle zumeist vernachlässigt wird. Alternativ versteht man unter *Splatting* ein Verfahren, bei dem für jeden Daten*punkt* im Volumen ein „Energiebeitrag“ in Form von Farbintensität und Transparenz bestimmt wird, und dieser über einen vom Benutzer bestimmbaren *Rekonstruktionsfilterkern* über die Umgebung des vom projizierten Datenpunkt getroffenen Pixels verteilt wird.

## Zusammenfassung

In diesem Abschnitt wurde ein allgemeines mathematisches Lichttransfermodell für direktes *Volume Rendering* beschrieben. Um ein *Volume Rendering* Bild einer dreidimensionalen Datenverteilung zu erstellen, muß für jeden Pixel die zentrale Lichttransportgleichung (2.5) gelöst werden. Es wurden Verfahren zur numerisch diskretisierten Integration dieser Gleichung sowie Näherungsmethoden auf verschiedenen Ebenen vorgestellt. Einige Approximationsverfahren zur Teilintegralbestimmung, die in den vorangehenden Absätzen vorgestellt wurden, werden in Abschnitt 6.3 wieder aufgegriffen,

wenn die im System VoRANG implementierten Näherungsverfahren beschrieben werden.

## 2.2 Taxonomie von VR-Verfahren auf irregulären Gittern

Die nachfolgende Übersicht gründet sich auf verschiedene aktuelle Veröffentlichungen seit 1990 ([Gar90, ST90, MHC90, WCAR90, Use91, Luc92, Wil92c, Gie92, Koy92, Wil92a, Wil92d, RW92, MBC93, VGW93, Cha93, TSM94, Frü94, SBM94] ), in denen Verfahren für Volume Rendering auf irregulären Gitter vorgestellt werden.

Diese Verfahren unterscheiden sich gewaltig in Hinsicht auf die Gitterarten, mit denen sie arbeiten können, ihr Laufzeitverhalten und die Art und Qualität der Bilder, die sie erstellen können – natürlich bedingt durch den jeweiligen Anwendungsfall, für den sie entworfen wurden.

### 2.2.1 Kriterien

Zunächst sollen die Merkmale besprochen werden, anhand derer im folgenden die Verfahren verglichen und beurteilt werden. Diese Merkmale decken viele verschiedene Bereiche ab, so z.B. das Eingabeformat (*Art der Gitter*), das bildliche Ergebnis (*Bildqualität*), Performance (*Geschwindigkeit, Speicheranforderungen*) sowie die Art der verwendeten (Teil-)Algorithmen (*Hardware-Anforderungen, Datenstrukturen, Beleuchtungsmodelle*) und ihre Eigenschaften (*Flexibilität, Parallelisierbarkeit*).

#### 2.2.1.1 Art der Gitter

Irreguläre Gitter treten in wissenschaftlichen Berechnungen in verschiedensten Formen auf. Grundsätzlich lassen sich Datengitter in zwei Kategorien unterteilen: in *strukturierte* und *unstrukturierte* Gitter.

Die **strukturierten Gitter** sind in einem Array angeordnet, besitzen also eine vorgegebenen Nachbarschaftsstruktur. Bei Volumendaten liegt ein dreidimensionales Feld von Zellen vor, auf die mit drei Indizes zugegriffen werden kann. Man spricht auch von *array-organisierten* Volumendaten. Im Fall *regulärer* Gitter liegen als Zellen regelmäßige Quader einheitlicher Größe vor. Bei den *rectilinearen* Gittern handelt es sich immer noch um quaderförmige Zellen, aber die Abstände zwischen Punkten auf einer Achse müssen nicht mehr konstant sein, so daß die Quader verschiedene Größen annehmen können. Die Zellen in einem krummlinigen oder *curvilinearen*[6] Gitter schließlich sind nicht mehr unbedingt Quader, sondern beliebig verformte Hexaeder. Die Eckpunkte der Zellen müssen nicht durch Geradenstücke sondern können auch durch Kurven verbunden sein. Somit sind in diesem Fall auch die Seiten einer Zelle möglicherweise nicht planar.

**Unstrukturierte Gitter** hingegen besitzen keine inhärente Anordnungsstruktur. Der Datenraum besteht aus einer Menge von räumlich positionierten Knoten und ei-

[6] Im nachfolgenden Text wird für krummlinige strukturierte Gitter durchgehend der Begriff *curvilineares Gitter* verwendet

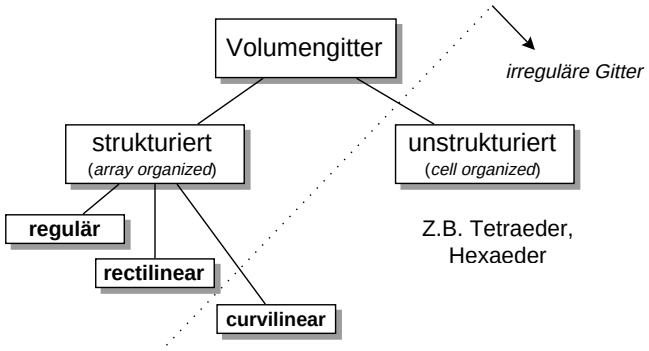



Abbildung 2.5: *Arten von Volumengittern*

ner Menge von Zellen, deren Eckpunkte eben diese Knoten darstellen. Gewöhnlich werden Gitter mit einheitlichen Zell-Topologien verwendet, wie Tetraedergitter oder Hexaedergitter. Es können aber auch verschiedene Zellformen in einem Gitter kombiniert werden. Unstrukturierte Gitter werden auch zellen-organisierte Gitter genannt. Ein direkter indizierter Zugriff auf einzelne Zellen wie bei den strukturierten Gittern ist nicht möglich. Allerdings kann auf einfache Weise Information über die Zellnachbarschaft in die Gitterdatenstrukturen aufgenommen werden, so daß dann für jede Zelle schnell alle angrenzenden Zellen bestimmt werden können.

Datenwerte, die als verstreute Punktverteilung im dreidimensionalen Raum gegeben sind und noch nicht durch ein Netz von Kanten miteinander verbunden sind, können zum Beispiel durch Delaunay-Triangulierung effizient in die Form von Tetraedergittern gebracht werden.

Die typischen Fälle **irregulärer Gitter** sind die (strukturierten) curvilinearen Gitter und aus dem Bereich unstrukturierter Gitter Tetraeder- oder Hexaeder-Gitter. Weder das gesamte Gitter noch die einzelnen Zellen sind notwendigerweise konvex (Tetraeder sind dies allerdings). Innerhalb der unstrukturierten Gitter können beliebige Hohlräume auftreten.

Eine wichtige Frage ist, ob die betrachteten Algorithmen auf bestimmte Gittertypen festgelegt sind, oder sich relativ einfach um das Verarbeiten anderer Gitterarten erweitern lassen. Interessant könnte die Fähigkeit sein, dynamische lokale Verfeinerungen des Ursprungsgitters zuzulassen und darauf bei der Bildberechnung durch lokale Nachbearbeitung schnell zu reagieren, ohne daß das gesamte Bild neu berechnet werden muß.

#### 2.2.1.2 Rekonstruktionsfilter, Interpolation

Eng an die Art der verwendeten Gitter sind die Methoden gekoppelt, die aus den diskret vorliegenden Datenwerten eine kontinuierliche Verteilungsfunktion rekonstruieren. Diese bildet dann die Grundlage für die zur Beleuchtung notwendigen Integralberechnungen.

Für das Innere von Tetraedern, an deren Knotenpunkten die Datenwerte definiert sind, bietet sich der einfachen Berechnung und Stetigkeit halber eine lineare Interpolation an:

$f(\boldsymbol{x})$ wird nach folgender Gleichung linear im Tetraederinneren interpoliert:

$$f(x, y, z) = c_1 + c_2\, x + c_3\, y + c_4\, z$$

Da die Feldverteilung an den 4 Eckpunkten des Tetraeders bekannt ist, kann man in einem linearen Gleichungssystem mit vier Gleichungen und vier Variablen nach den Parametern $c_1, c_2, c_3, c_4$ auflösen.

Schon für quaderförmige Zellen wird deutlich, daß trilineare Interpolation nur die einfachste von vielen Annäherungen einer kontinuierlichen Originalfunktion darstellt. Kubische Filter liefern für viele Datenvolumina genauere Rekonstruktionen der originalen Datenverteilung. ([Neu93], S. 56ff.)

Sind Computersimulationsrechnungen (z.B. mit Hilfe von *Finite Elemente Methoden*) die Quelle für das darzustellende Datenvolumen, ist es sinnvoll und erwünscht, die bereits für die Simulation verwendeten Interpolationsverfahren auch für die Visualisierung zu nutzen. Dadurch können direkt die in der Simulation bestimmten kontinuierlichen Verteilungen visualisiert werden.

#### 2.2.1.3 Bildqualität, Artefaktfreiheit

Die Bildqualität ist ein nicht immer leicht zu fassendes Kriterium, vor allem läßt sie sich nicht absolut beurteilen. Besonders bei Datenfeldern, die abstrakte Größen, wie z.B. Temperaturverteilungen oder Strömungsgeschwindigkeiten repräsentieren, hängt die Aussagekraft des Ergebnisbildes in erster Linie von einer geschickten Betonung und Heraushebung der interessanten Gebiete durch die Abbildungsfunktionen der Daten- zu Farb- und Transparenzwerten (*mapping functions*) und das Lichttransfermodell (s.u.) ab.

Die Art der Abbildung von den Datenwerten zu Partikeleigenschaften in semitransparenten Dichtewolken sollte für den Anwender nachvollziehbar bleiben, damit der Rückschluß auf die Struktur des Datenvolumens gewärleistet ist.

Ein wichtiges Kriterium der angewendeten Verfahren ist das Auftreten oder Fernbleiben von Artefakten im Ergebnisbild, also von Strukturen, die ihren Ursprung nicht im zu visualisierenden Datenvolumen haben, sondern aus Darstellungsfehlern resultieren. Artefakte können u.a. auch aus bewußt in Kauf genommenen Vereinfachungen des theoretischen Abbildungsmodells entstehen. Als Beispiel sei auf die verschiedenen Approximationsverfahren zur Lösung der Lichttransfergleichung verwiesen, bei denen das Auftreten bestimmter Artefakte durch einen Geschwindigkeitsvorteil eingehandelt wurde (vgl. Testläufe in Abschnitt 7).

#### 2.2.1.4 Geschwindigkeit

Geschwindigkeit ist eine wesentliche Eigenschaft der Darstellungsverfahren. Die höchste Aussagekraft hat ein durch *Volume Rendering* visualisierter Datensatz, wenn er schnell, im Optimalfall interaktiv, von verschiedenen Blickwinkeln aus betrachtet werden kann, wenn Ausschnittsänderungen schnell ausgeführt werden und Änderungen der *mapping functions* sich innerhalb von kürzester Zeit im Bild niederschlagen. Auf diese Weise läßt sich ein Datensatz tatsächlich intensiv erkunden, während ein einzelnes Standbild gerade einmal einen übersichtartigen Hinweis auf mögliche interessante Regionen im

Datensatz liefern kann. Und selbst für Standbilder ist es eine zu optimistische Annahme, daß man mit einem Visualisierungsvorgang auskommt, denn die optimale Einstellung der Bildparameter erfordert im allgemeinen mehrere „Zufallsschüsse".

Um einen Eindruck von der unterschiedlichen Ausrichtung der einzelnen Verfahren zu geben, sei angemerkt, daß für die ausgewerteten Algorithmen bezüglich der Geschwindigkeit auf einer Ein-Prozessor-Workstation eine Spannbreite von ungefähr einer halben Sekunde bis ca. einer halben Stunde als Bildberechnungszeit für ein Bild von einem größeren Datensatz auftrat.

#### 2.2.1.5 Beleuchtung

Wie bereits erwähnt kommt der Umwandlung der Daten in Farb- und Transparenzwerte entscheidende Bedeutung für das Verständnis des Datenvolumens zu.

In der Frage der Beleuchtung und Farbgebung des Datenvolumens sind zwei wichtige Entscheidungen zu treffen: Die erste betrifft das *Lichttransfermodell*, d.h. es muß überlegt werden, wie die Lichtintensität entlang eines Sehstrahles aufintegriert wird und welche physikalischen Effekte des Lichttransportes durch ein Medium variierender Dichte modelliert werden (z.B. Absorption, Einfachstreuung, Mehrfachstreuung, inelastische Streuung).

Die zweite Frage betrifft die Abbildung von Datenwerten nach Farb- und Tranzparenzwerten. Diese sogenannten *mapping functions* können in zwei verschiedenen Ausprägungen auftreten: Entweder ist die Farbwahl in das physikalische Lichttransfermodell integriert oder die Einfärbung des Datenvolumens erfolgt mehr oder weniger symbolisch.

Meistens wird entsprechend dem ersten Fall die Intensität entlang des Sehstrahles für endlich viele Lichtfrequenzen (meist drei, entsprechend dem RGB-Modell) unabhängig aufintegriert. In diesem Fall muß der Benutzer Abbildungsfunktionen von den Datenwerten zu Rot-, Grün- und Blau-Intensitäten und zur optischen Dichte (Opazität) bereitstellen [7]. Im Rahmen des physikalischen Modells heißt das: Die Datenwerte des Volumens werden als Dichtefunktion von Partikeln interpretiert oder in eine solche überführt, und der Benutzer des Systems modelliert über die *mapping functions* die optischen Eigenschaften der Partikel (meist in Abhängigkeit der Dichtefunktionswerte). Die Abbildungen werden für jeden Datenwert im Volumen vorgenommen und schließlich werden die Farbintensitäten entsprechend der Transparenzwerte aufintegriert.

Ein einfaches Beleuchtungsmodell, wie es in der Computergrafik für Oberflächendarstellungen verwendet wird (z.B. Phong'sches Modell), kann leicht durch die *mapping functions* inkorporiert werden, indem z.B. die Gradienten im Datenvolumen (als Normalenvektoren) und die Richtung zu beliebig positionierbaren Punktlichtquellen in die Abbildung zu den Intensitäten miteinbezogen werden. Dabei stellt sich bezüglich des Lichttransfermodells die Frage, ob man die Schwächung des Lichtes auf dem Weg von der externen Lichtquelle zum Datenpunkt durch dazwischenliegende Daten modelliert oder nicht (*self shadowing*, vgl. [HHS94], S.23 ff.)

Um Farbe mehr symbolisch einzusetzen, kann man zunächst die gesamte Lichtintensität frequenzunabhängig entlang der Sehstrahlen aufintegrieren und anschließend

[7] Für gewöhnlich reicht **eine** frequenzunabhängige Abbildung auf die Opazitätswerte $\alpha$.

ein Mapping von Intensität und anderen Strahlcharakteristika in ein Farbmodell vornehmen. Welche Größen man dabei berücksichtigt, bleibt dem eigenen Einfallsreichtum und der Experimentierfreudigkeit überlassen [8].

Allerdings unterscheiden sich die hier betrachteten Algorithmen kaum bis gar nicht in ihrer Entscheidung für eine bestimmte Beleuchtungsstrategie, d.h. ein bestimmtes Lichttransfermodell. Die allermeisten Verfahren implementieren ein einfaches Emissions-Absorptions-Modell, das keine Streuung des Lichtes im Datenvolumen berücksichtigt, sondern nur die Intensität des selbstleuchtenden (oder ohne *self shadowing* angestrahlten) Volumens entlang eines Sehstrahles aufintegriert, wobei allerdings Intensitäten durch das zu durchquerende Volumen auf dem Weg zum Betrachter abgeschwächt werden.

Realitätsnähe im Beleuchtungsmodell spielt für die Visualisierung im naturwissenschaftlichen Bereich – und diese ist eindeutig Adressat für Verfahren auf irregulären Gittern – eine nicht so bedeutende Rolle wie in anderen Anwendungsgebieten der Computergrafik.

Zwar ist die Entstehung des *Volume Rendering* eng mit den ersten Bemühungen verknüpft, eine fotorealistische Darstellung von natürlichen Phänomenen wie z.B. Wolken oder Nebel zu berechnen, aber bei der Übertragung des Dichtewolken-Konzeptes auf die hauptsächlich abstrakten Größen, die im Bereich *Scientific Visualization* auftreten, bestimmen andere Anforderungen als Fotorealismus die Darstellung der Datensätze.

Auch im Vergleich mit der medizinischen Bildverarbeitung, wo in größerem Maße ausgeprägte Oberflächen auftreten, die zur besseren Orientierung realistisch dargestellt werden müssen, sind in der Wahl des Beleuchtungsmodells hier größere Freiheiten möglich. Wenn man, wie z.B. im Fall von CFD-Daten, mitunter im vornherein noch keine Vorstellung davon hat, was man eigentlich sehen will [Use91], verwundert es nicht, daß erst einmal ein sehr einfaches Beleuchtungsmodell gewählt wird, das wenig Verwirrung stiftet und mit Absicht grobe symbolische Farbdarstellungen vornimmt.

Bei alledem darf jedoch nicht vergessen werden, daß natürlich auch in der wissenschaftlichen Visualisierung auf die Erfahrung einer realen Welt zurückgegriffen wird. Der Betrachter eines Volume Rendering Bildes erkennt ja gerade deshalb Strukturen darin, weil er eine Vorstellung davon hat, wie sich in einer teilweise transparenten Wolke Lichtereignisse optisch überlagern können.

Die Transportgleichung, die das in den Verfahren verwendete Beleuchtungsmodell beinhaltet, ist im Normalfall um beliebige Terme erweiterbar, d.h. die Beleuchtungsstrategie kann (auf Kosten des Rechenaufwands) verkompliziert werden. Es könnte interessant sein zu untersuchen, inwieweit manche Algorithmen die Verwendung bestimmter Beleuchtungstrategien auf effizienterem Wege ermöglichen als andere.

#### 2.2.1.6 Flexibilität

Ein Verfahren, das dem Benutzer Einfluß auf die Ablaufsteuerung zugesteht (z.B. Angebot mehrerer Bildgenauigkeiten, freie Wahl von *mapping functions*, etc.) und das auf

[8] Sabella [Sab88] bildet beispielsweise die Intensität, den Maximaldatenwert entlang des Strahls und die Streckenlänge zum Maximalwert oder zum gewichteten Mittelpunkt des Strahles auf das HSV-Farbmodell ab (Intensität → V, Maximalwert → H, Streckenlänge → S (depth cueing))

dynamische Anderungen schnell reagiert, hat sicherlich seine Vorteile gegenüber einem starren Verfahren, das auf eine Einstellung festgelegt ist und bleibt.

Im einzelnen ist Flexibilität u.a. in folgenden Bereichen erwünscht:

- *Trading* zwischen Geschwindigkeit und Bildtreue
- Unterstützte Kamera (perspektivisch/orthogonal)
- Inkorporierte Beleuchtungsmodelle
- Gleichzeitige Darstellung von nicht-volumenhaften Geometrien (Oberflächen, Linien)
- Dynamische Transformationen des Blickwinkels, Transferfunktions-, Skalarfeld- und Gitteränderungen, auf die möglichst lokal reagiert wird.

#### 2.2.1.7 Hardware-Anforderungen

Einige der betrachteten Verfahren nutzen serienmäßig verfügbare Grafikhardware (z.B. zum *Rendering* von Polygonen mit Gouraud-Shading), um interaktive Antwortzeiten bei der Bildberechnung zu erreichen. Natürlich lassen sich diese Algorithmen auch rein softwaretechnisch implementieren, aber sie sind teilweise recht trickreich auf die Ausnutzung der (bis heute leider nicht auf volumenorientierte Darstellungsmethoden ausgelegten) Grafikhardware getrimmt, so daß sie ihre Daseinsberechtigung hauptsächlich aus diesem Geschwindigkeitsgewinn ziehen.

#### 2.2.1.8 Datenstrukturen, Speicheranforderungen

Die Datenstrukturen zur Beschreibung irregulärer Gitter können schnell eine sehr unhandliche Größe annehmen. Über 50 MB für einen Datensatz sind keine Seltenheit (vgl. [Use91], S. 4, [Gar90], S. 40). Allein die Zeiten, um eine solche Datenmenge einfach im Speicher umzukopieren, sind nicht zu vernachlässigen. Die Größe der zusätzlich vom Algorithmus benötigten Datenstrukturen spielt sicherlich nur dann eine Rolle, wenn sie proportional zur Größe des Eingabedatensatzes ist.

#### 2.2.1.9 Parallelisierbarkeit

Um tatsächlich Bildberechnungszeiten im interaktiven Bereich zu erzielen, muß man, wenn man dabei keine Kompromisse bezüglich der Realitätstreue und Artefaktfreiheit der Bilder eingehen will, seinen Algorithmus auf einem eminent schnellen Rechner ablaufen lassen.

Die einzigen Rechner, die eine derartige Geschwindigkeit bieten können sind solche mit (massiv) paralleler Architektur. Damit der Algorithmus aber tatsächlich den gewünschten Gewinn an Geschwindigkeit erzielt, muß er geeignet parallelisiert werden. Dabei kommt es vor allem darauf an, die Kommunikation unter den Prozessoren gering zu halten und den Algorithmus gut skalierbar zu machen, d.h. für beliebige Zahlen von Prozessoren die Effizienzsteigerung des Algorithmus gegenüber dem sequentiellen Fall hoch zu halten.

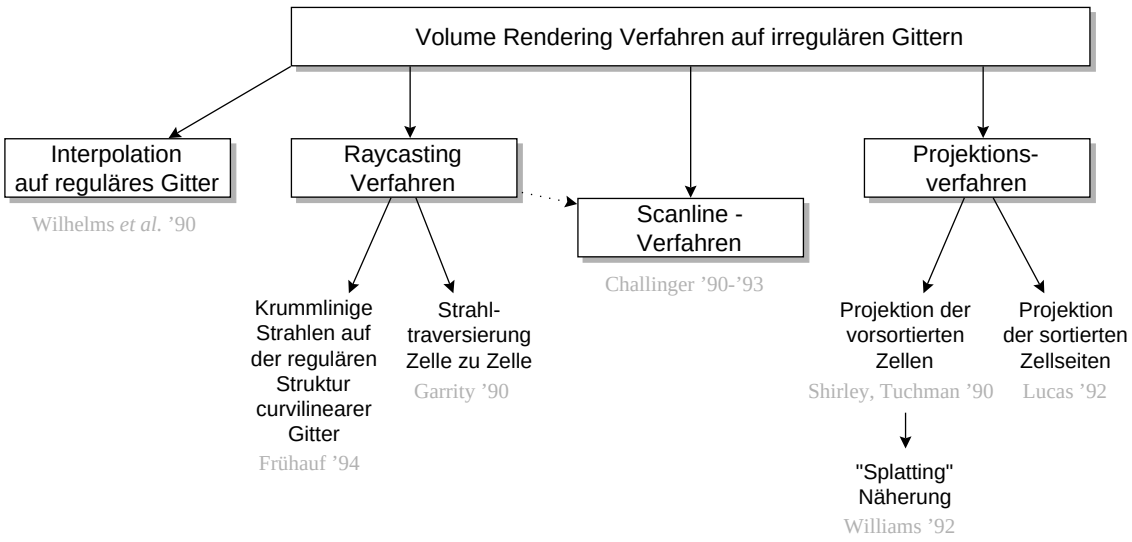



Abbildung 2.6: *Klassifizierung von VR-Verfahren auf irregulären Gittern*

Unter den begutachteten Verfahren sind drei zu finden, die bereits auf einem Parallelrechner implementiert wurden.

### 2.2.2 Klassifizierung der Verfahren

Eine grundlegende Einordnung der bestehenden Verfahren zum Volume Rendering für irregulär definierte Daten läßt sich am sinnvollsten nach der Funktionsweise der Verfahren vornehmen. Die Funktionsweise des Verfahrens beeinflußt maßgeblich die oben genannten Kriterien. Innerhalb jeder Gruppe von Verfahren liegen meist ähnliche Verhaltensweisen vor.

Jede der Vorgehensweisen besitzt bezüglich oben genannter Kriterien Vor- und Nachteile. Im folgenden soll kurz ein Überblick gegeben werden, bevor die Verfahren jeder Klasse im einzelnen betrachtet werden.

#### 2.2.2.1 Überblick

Als erste und einfachste Methode, um mittels *Volume Rendering* ein Bild eines irregulär definierten Datensatzes zu erhalten, besteht die Möglichkeit der *Interpolation auf ein reguläres Gitter* und anschließender Nutzung von Volume Rendering Verfahren für reguläre Gitter. Dieses Vorgehen sei hier aber nur am Rande erwähnt, weil es kein eigenes Verfahren zum direkten Volume Rendering impliziert, sondern von bereits vorhandenen (schnellen) Verfahren zur Darstellung regulär definierter Volumendaten Gebrauch macht.

**Strahlverfolgungs- und Projektionsverfahren** Die direkten Verfahren zum Volume Rendering auf irregulären Gittern gliedern sich zunächst einmal in zwei unterschiedliche Vorgehensweisen: Bei den *Strahlverfolgungsverfahren* (**Raycasting**) werden die Pixel der Bildebene nacheinander abgearbeitet und pro Pixel ein oder mehrere

Strahlen in das Datenvolumen geschickt, entlang derer dann die Beiträge der getroffenen Zellen der Beleuchtungsstrategie entsprechend aufintegriert werden. Bei den *Projektionsverfahren* arbeitet man nacheinander die Volumenelemente im Datenvolumen ab und berechnet ihren Beitrag an den Integralen für die Bildschirmpixel, auf die ihr Umriß projiziert wird. Um das *Compositing* der einzelnen Beiträge entsprechend den physikalischen Gesetzmäßigkeiten des Lichttransportes durchzuführen, muß man die Datenzellen in einer *back-to-front* oder *front-to-back* Ordnung abarbeiten. Nur wenn man das Lichttransfermodell sehr stark vereinfacht, kann man auf diese Vorsortierung verzichten[9]

Die Projektionsmethoden nutzen Datenkohärenz innerhalb des Volumens wesentlich besser als das *Raycasting*: Bei letzterem müssen für gewöhnlich viele Berechnungen mehrfach ausgeführt werden. Strahlen aus benachbarten Pixeln treffen oft gleiche Volumenelemente, aber es ist sehr schwer, hierdurch tatsächlich Berechnungen einzusparen. Man kann dies versuchen, indem man beispielsweise für jede Zelle die Parameter für die Seitenflächengleichungen und für die Interpolation in einem *Cache-Speicher* hält.

Die Projektionsverfahren haben hier den Vorteil, daß sie alle Arbeiten, die für ein Volumenelement anfallen, in einem Zug hintereinander weg abarbeiten können. Hierbei können auf einfache Weise Zwischenberechnungen für verschiedene Pixel, auf die das Voxel projiziert wird, wiederverwendet werden. Schnittpunktberechnungen von Sehstrahl und Seitenflächen der Volumenzelle werden hier mit Hilfe eines Scanline-Algorithmus sehr effizient bestimmt.

Projektionsverfahren bieten sich in viel größerem Maße als Strahlverfolgungsverfahren dazu an, aus Vereinfachungen im Lichttransfermodell Geschwindigkeitssteigerungen zu erzielen. Bei einer Untergruppe der Projektionsmethoden, dem sogenannten *Splatting*, wird z.B. das Volumen einfach Knoten für Knoten traversiert und die Zellengeometrie wird vernachlässigt. Der Datenwert in jedem Knoten wird auf Farb- und Transparenzwerte gemappt und mit einem Faltungskern, der die „Energie“ dieses Knotens über mehrere Pixel in der Nachbarschaft verteilt (z.B. auf einem 5x5 Pixel-Fenster), auf die Bildebene projiziert. Da sich diese Nachbarschaftsregionen überlappen, entsteht ein glattes Bild mit fließenden Farbverläufen.

Bei den Strahlverfolgungsmethoden wird jeder in das Volumen hineingesschickte Strahl zum Zwecke der Berechnung des Integrals aus dem Lichttransfermodell auf seiner gesamten Länge in kleinere Abschnitte zerlegt. Auf diesen Abschnitten erfolgt die Integration entsprechend dem gewählten Modell und die Ergebnisse werden zu einem Gesamtbeitrag entlang des Strahls geeignet akkumuliert.

Die Verfahren zum *Raycasting* unterscheiden sich vor allem dadurch, wie die Unterteilung des Strahls bestimmt wird.

Wenn man den Strahl in äquidistanten Abständen abtastet, kann man die Integration auf das *compositing* der an den *sample points* abgegriffenen Werte reduzieren (vgl. [Lev88]). Dies ist zu einer Standardvorgehensweise für regelmäßige Gitter geworden.

Für einen solchen Ansatz muß für jeden dieser in gleichen Abständen auf dem Strahl angeordneten *sample points* die getroffene Volumenzelle bestimmt werden (*point*

[9] z.B. bei proportionalen Absorptions- und Emissionskoeffizienten (vgl. [HHS94], S.15 f.) oder bei uniformer Einfärbung des Datenvolumens (s. [MBC93], S. 20 f.)

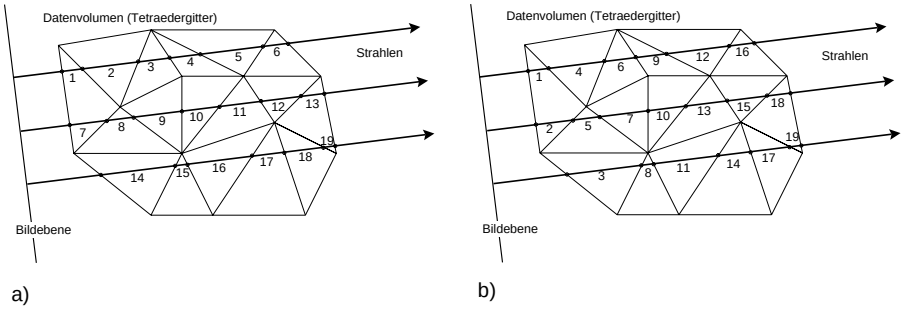



Abbildung 2.7: *Integrationsreihenfolge bei Raycasting (a) und Projektion (b)*

*location*)[10], in der dann durch Interpolation der Datenwerte und durch Anwendung der benutzerdefinierten Abbildungsfunktionen die zur Akkumulation benötigten Farb- und Transparenzwerte bestimmt werden.

Für irreguläre Gitter ist das Auffinden der fraglichen Volumenzelle für jeden *sample point* jedoch zu kostenintensiv. Daher wird oft ein anderer Ansatz verfolgt: man berechnet die Schnittpunkte der Strahlen mit den Datenzellen und bestimmt die Integrale zwischen den Schnittpunkten im Innern jeder Zelle analytisch oder angenähert. Um den gesamten Strahl vom ersten Eintritt in das Volumen bis zum endgültigen Verlassen desselben abzuarbeiten, kann man entweder von Zelle zu Zelle wandern, wobei sich die nachfolgende Zelle automatisch durch den Austritt-Schnittpunkt des Strahls mit einer Wand der aktuellen Zelle und durch die Nachbarschaftsbeziehung der Zellen ergibt. Oder man berechnet tatsächlich im vorhinein alle Schnittpunkte eines Strahles mit allen Zellenwänden, wobei man vorher die Auswahl der für einen Schnitt in Frage kommenden Zellen durch Vorsortierung beträchtlich einengt.

Wenn die Interpolationsart innerhalb einer Datenzelle und die Integrationsmethode gemeinsam vorgegeben sind, sind ein Projektions- und ein Raycastingverfahren, bei dem zwischen dem Eintritts- und Austrittspunkt jeder Zelle integriert wird, ihrem Ergebnis nach äquivalent. In beiden Methoden werden die das Volumen durchdringenden Sehstrahlen in Abschnitte aufgeteilt, für die eine Integration exakt oder genähert möglich ist. Die Integrale für diese Einzelabschnitte werden schließlich zur Berechnung des Lichtbeitrages über den gesamten Strahl entsprechend dem Lichttransfermodell (vgl. 2.1.1.1) zusammengefügt.

Wie eben ausgeführt, ist es ein für irreguläre Gitter üblicher Ansatz, beim *Raycasting* als Einzelabschnitte die zwischen Eintritts- und Austrittspunkt jeder Zelle liegenden Strecken zu wählen. In diesem Fall unterscheiden sich Projektions- und Strahlverfolgungsverfahren nur durch die Reihenfolge, in der die Teilintegrale bestimmt werden. Die Rechenmethode zur Lösung dieser Teilintegrale ist beliebig und kann in beiden Fällen gleich gewählt werden. Abbildung 2.7 zeigt die unterschiedliche Reihenfolge der Teilstrahlabarbeitung für die beiden Verfahren.

Die im nächsten Abschnitt beschriebenen *Scanline-Verfahren* verhalten sich bezüglich dieser Reihenfolge wie das *Raycasting*, da die Schnittpunkte von Strahl und Zellenwänden sortiert Strahl für Strahl abgearbeitet werden.

[10] [Nee90] stellt eine vergleichsweise effiziente *point location* Methode für irreguläre Gitter vor. Auch in [Wil92d], S.140-144, werden solche Methoden untersucht

**Scanline Verfahren** Die sogenannten *Scanline-Algorithmen* verfolgen den Ansatz, alle Schnittpunkte zwischen Zellenwänden und Strahl von vornherein unter Ausnutzung von Kohärenz zwischen und innerhalb sogenannter *Scanlines*, also verschiedener Pixelzeilen der Bildebene, zu bestimmen. Die für einen Schnitt in Frage kommenden Zellenwände werden durch Vorsortierung nach y- und x-Koordinaten eingegrenzt.

Scanline-Algorithmen wurden schon seit Anfang der 70er Jahre im *Surface Rendering* zur Bestimmung der sichtbaren Polygone entlang einer Scanline verwendet (vgl. z.B. [FvDFH90]). Ein solcher Algorithmus macht nichts anderes, als alle Polygone einer Szene auf die Bildfläche zu projizieren, die aktiven Elemente durch Sortiererung nach y- und x-Werten einzugrenzen und schließlich mit Hilfe von Integer-Arithmetik (und Ausnutzung der Kohärenz innerhalb eines Polygons) die Schnittpunkte von Polygonkanten mit der aktuellen Scanline zu bestimmen und bei sich verdeckenden Polygonen das oberste zu bestimmen. Diese Methode wird auf das Volume Rendering übertragen, indem statt Oberflächenpolygonen die Zellwände der Voxel auf die Bildebene projiziert und statt Bestimmung nur des obersten Polygones für jeden Pixel *alle* Schnittpunkte zwischen Pixelstrahl und Zellenwänden bestimmt und in einem Buffer notiert werden. Auf diesem Buffer kann dann entlang jeden Pixelstrahls die Intensität im Datenvolumen von Schnittpunkt zu Schnittpunkt aufintegriert werden. Das Vorgehen entspricht in dieser Hinsicht vollkommen den anderen Strahlverfolgungsverfahren. Mit der *Scan-Konvertierung* der Zellengeometrie jedoch übernimmt diese Methode ein wesentliches Element aus den Projektionsverfahren, weshalb manche Autoren Scanline Verfahren als *hybride Projektions-/Strahlverfolgungsmethode* kennzeichnen.

Im folgenden Abschnitt wird die Untergliederung der Verfahren jeder Klasse (Strahlverfolgungs-, Scanline- und Projektionsverfahren) in einzelne Teilalgorithmen untersucht. Danach folgt eine tabellarische Übersicht der hier behandelten Veröffentlichungen und schließlich werden die allgemeinen Vor- und Nachteile der verschiedenen Verfahren – bezogen auf die oben angeführten Kriterien – diskutiert.

#### 2.2.2.2 Verschiedene Ansätze innerhalb der Verfahrenklassen

Bevor die weitere Unterteilung der Strahlverfolgungs-, Projektions- und Scanlineverfahren in einzelne Ansätze untersucht wird, soll kurz Sinn und Zweck einer Interpolation der Datenwerte auf reguläre Gitter besprochen werden:

**Interpolation auf reguläres Gitter** Interpolation der irregulär definierten Datenverteilung auf ein reguläres Gitter zum Zwecke der Visualisierung ist für viele Wissenschaftler, die mit Standardvisualisierungssoftware arbeiten, heute (noch) der Normalfall, aber in vielen Fällen ist diese Methode für eine adäquate Darstellung der Daten ungeeignet.

Hierbei können bezüglich der Datenbeschaffung und -anordnung vier Fälle unterschieden werden, die die Methode mehr oder weniger sinnvoll erscheinen lassen:

1. Die Datenverteilung liegt als kontinuierliche Funktion vor und es gibt keine Regionen mit besonders hoher Variation der Funktionswerte.

2. Die Datenverteilung liegt (nur) in der Form irregulär angeordneter Samples vor, es gibt aber keine Regionen mit besonders hoher Variation der Funktionswerte.

3. Die Datenverteilung liegt als kontinuierliche Funktion vor und ist ziemlich ungleichmäßig, d.h. neben großflächigen konstanten Regionen existieren Gebiete mit großer Variation der Funktionswerte.

4. Die Datenverteilung liegt (nur) in der Form irregulär angeordneter Samples vor und ist ziemlich ungleichmäßig.

Im ersten Fall spricht nichts dagegen, die Funktion auf einem regulären Gitter abzutasten (*Sampling*), bzw., wenn bereits eine Verteilung auf einem irregulären Gitter vorliegt, ein reguläres *Resampling* durchzuführen. Der Geschwindigkeitsvorteil bei der Bildberechnung wiegt den einmaligen Neuberechnungsaufwand sofort auf.

Im zweiten Fall kann man kein Resampling vornehmen, sondern muß die vorhandenen Werte auf ein reguläres Gitter interpolieren. Man kann aufgrund der gleichmäßigen Datenverteilung allzu große Interpolationsartefakte vermeiden, wenn man die kleinste irreguläre Gitterzelle als Maßstab für die Größe der regulären Zellen nimmt. Ob die Artefakte hier im erträglichen Rahmen bleiben, hängt vom Grad der Regelmäßigkeit ab.

In den letzten beiden Fällen hat direktes Volume Rendering auf irregulären Gittern eindeutige Vorteile gegenüber dem Resampling-/Interpolationsansatz. Ein Datenfeld mit großer Variation der Funktionswerte läßt sich auf einem regulären Gitter nur sehr unzureichend repräsentieren. Würde man hier die kleinste irreguläre Zelle als Maßstab für die reguläre Zellengröße verwenden, wüchsen die Datenmengen bei solchen Feldern in untragbare Bereiche. Wählt man die Gitterabstände hingegen größer, bekommt man kaum tolerierbare Artefakte durch Unterabtastung an den Stellen hoher Variation[11]. Gerade bei der Visualisierung wissenschaftlicher Daten, von deren Anordnung man bisher keine Vorstellung hat, ist es jedoch besonders wichtig, genau die Werte zu visualisieren, die auch tatsächlich in den zu untersuchenden Rechnungen auftraten, und nicht nur eine Annäherung.

Ein weiteres Argument gegen ein reguläres *Resampling* von auf irregulären Gittern definierten Datenverteilungen ist, daß die Topologie des Gitters bei der Untersuchung von Simulationsergebnissen oft genauso wichtig ist wie die Ergebnisdaten selbst. Jede Art von Interaktion des Betrachters mit dem Ergebnisdatensatz (Wertabfragen, Schnittpositionierungen, etc.), sollte unter Verwendung der ursprünglichen Gittertopologie und -koordinaten erfolgen.

**Verschiedene Strahlverfolgungsansätze (Raycasting)** Wenn man die *Scan Line Verfahren* separat betrachtet, wie das oben geschehen ist, so basieren im Rahmen der hier untersuchten Implementierungen – mit einer Ausnahme – sämtliche *Ray Casting* Ansätze auf der Methode, auf dem Strahl von Zelle zu Zelle zu wandern, indem man über den Austritts-Schnittpunkt des Strahles mit einer Wand der aktuellen Zelle in die angrenzende Zelle übergeht. [Gar90, Use91, RW92, Koy92, TSM94]. Das in [Frü94] beschriebene Verfahren verfolgt einen anderen Ansatz. Es wird am Ende dieses Abschnittes näher beschrieben.

[11]Eine Kompromißlösung wäre evtl. die Interpolation auf hierarchische rectilineare Gitter. Aber Volume Rendering Verfahren auf solchen Gittern sind im Gegensatz zu denen auf regulären Gittern nur noch geringfügig effizienter als die Visualisierung auf den irregulären Gittern, so daß sich im Endeffekt der Aufwand nicht mehr lohnt.

Vorstellbar wäre auch noch, daß man über die irregulären Zellen ein Gitter von regulären *bounding boxes* legt und zunächst mit einem schnellen Verfahren alle Schnittpunkte der Strahlen mit diesen *bounding boxes* bestimmt und dann entscheidet, welche in der *box* liegende(n) irreguläre(n) Zelle(n) getroffen wurde(n). Ein solches Verfahren wird von Garrity ([Gar90]) zur Bestimmung des *Eintrittspunktes* eines Strahls in das Volumen vorgeschlagen. Für die anschließenden Schnittberechnungen ist diese Methode dem *cell-to-cell* Ansatz allerdings vom Aufwand her deutlich unterlegen.

Die wichtigen Entscheidungen speziell bei den Strahlverfolgungsverfahren sind:

- Wie wird, ausgehend von der aktuellen Zelle, die nachfolgende Zelle bestimmt ?
- Wie wird der *Eintrittspunkt* in das Volumen (erste vom Strahl getroffene Zelle) bestimmt ?
- Welche innerhalb einer Zelle berechneten Informationen werden zum Zwecke der Wiederverwendung mit abgespeichert ?

Garrity [Gar90] stellt 1990 das grundlegende Verfahren zur Traversierung irregulärer Zellen entlang eines Sehstrahles vor. Uselton [Use91] wendet dieses Verfahren auf Daten aus dem Bereich *Computational Fluid Dynamics* an und stellt eine verbesserte Methode zur Bestimmung des Eintrittspunktes vor. In [RW92] wird ebendasselbe Verfahren mit einigen Variationen in einer Übersicht genau untersucht. Koyamada [Koy92] schließlich stellt ein eigenes, etwas schnelleres Verfahren vor, um auf dem Strahl die jeweils nachfolgende Zelle zu bestimmen.

Der Ansatz von Garrity ist für alle konvexen Gitterzellen durchführbar: Der Strahl tritt durch eine Zellenwand in die Zelle ein. Nun berechnet man für die Ebenen durch die verbleibenden (im Falle von Hexaedern: fünf) Zellenwände die Schnittpunkte mit dem Strahl, sofern sie existieren. Die Zellenwand, deren Schnittpunkt mit dem Strahl dem Eintrittspunkt in die Zelle am nächsten liegt, ist logischerweise die Wand, durch die der Strahl die Zelle wieder verläßt. Die Folgezelle ergibt sich automatisch durch die Zellnachbarschaft.

Koyamada verfolgt einen etwas anderen Ansatz: Er setzt ein Tetraedergitter voraus (Konvexität der Zellen ist aber auch bei ihm im Prinzip hinreichend). Es sei angenommen, der Strahl ist durch eine Seitenfläche in ein Tetraeder eingetreten. Die Berechnung des Austrittspunktes wird in 2D gelöst, indem die restlichen drei Seitenflächen nacheinander in die zweidimensionale Bildebene projiziert werden, bis der als Pixel repräsentierte Strahl in einem der entstehenden Dreiecke liegt, was durch einen schnellen *point in polygon* Algorithmus leicht feststellbar ist. Diese Vorgehensweise hat den (Geschwindigkeits-)vorteil, daß statt dreier Schnittberechnungen im dreidimensionalen Raum hierbei im Durchschnitt nur zwei Projektionen plus einfache 2D-Schnittberechnungen durchzuführen sind. Es hat den Nachteil, daß das Verfahren auf Sehstrahlen festgelegt ist.

Zum Zwecke der Bestimmung der ersten von einem Strahl getroffenen Zellenwand sind im Normalfall alle Außenflächen der Gitterzellen als solche in den Datenstrukturen markiert und aufzählbar gehalten. Man kann auch noch anhand der Flächennormalen die nach vorne und nach hinten zeigenden Außenflächen unterscheiden.

Das einfachste Verfahren ist nun natürlich der Test eines Strahls auf Schnitt mit jeder nach vorne zeigenden Außenfläche. Garrity schlägt zusätzlich den Einsatz regulärer quaderförmiger *bounding boxes* vor.

Koyamada setzt auch für diesen ersten Test *scan conversion* ein. Zunächst sortiert er alle nach vorne weisenden Außenflächen nach ihrer Nähe zur Bildebene von vorne nach hinten durch. Nun führt er nacheinander eine Projektion dieser Außenflächen in die Bildebene durch und schickt für alle Pixel, die von der Projektion erfaßt wurden, einen Strahl in das Volumen, den er am berechneten Schnittpunkt mit der projizierten Zellenwand beginnen läßt. Dann erst fährt er mit der Projektion der nächsten Außenfläche fort. Für den Fall konvexer Datenvolumen, kann er sich die Sortierung der Außenflächen sogar sparen.

Ein sehr effektives Vorgehen zur Bestimmung des Volumeneintrittspunktes für jeden Strahl, ohne dabei die Abarbeitungsreihenfolge der Pixel zu ändern, ist durch die sog. *item buffer*-Technik möglich, wie sie von Uselton als Anwendung einer schon 1984 veröffentlichten Technik vorgestellt wird ([Use91], S. 7 f.). Dieses Verfahren nutzt für seine Zwecke das (evtl. durch Hardwareunterstützung beschleunigte) *Z-Buffer Rendering* von Polygonen.

Als *item buffer* wird der *frame buffer* verwendet, also der Bildspeicher, in dem das Ergebnisbild Pixel für Pixel aufgebaut wird. Das Auffüllen des *item buffers* läuft als *Preprocessing* vor dem Absenden des ersten Strahls ab. Die Außenflächen der Volumenzellen werden durch einen eindeutigen umkehrbaren *Isomorphismus* auf Farben abgebildet (Wären die Zellen in einem x-/y-/z-Array angeordnet, wäre z.B. (x $\rightarrow$ rot, y $\rightarrow$ grün, z $\rightarrow$ blau) eine solchermaßen geeignete Abbildung). *Einer* Farbe, der Hintergrundsfarbe, wird keine Fläche zugewiesen. Nun wird zunächst das gesamte Bild in dieser Hintergrundsfarbe eingefärbt (d.h. der *item buffer* mit derselben gefüllt). Dann folgt das *Rendering* aller Außenflächen mit *Z-Buffer*-Unterstützung, wobei jede Fläche mit der ihr durch den Isomorphismus zugeordneten Farbe gezeichnet wird. Ist dieser Schritt ausgeführt, kann aus der Farbe jedes Pixels über den invertierten Isomorphismus direkt auf die erste getroffene Außenfläche geschlossen werden.

Ramamoorthy und Wilhelms [RW92] geben zu Bedenken, daß diese Technik nur für den ersten Eintritt in das Volumen anwendbar ist, und nicht für Wiedereintritte der Strahlen ins Volumen bei konkaven Volumengittern. Solche Wiedereintritte ins Datenvolumen versuchen sie effizient mit rechtwinkligen *bounding boxes* zu bestimmen.

Im gleichen Artikel wird des weiteren der Vorschlag gemacht, die Flächengleichungen für jede Zellenfläche in einem *Preprocessing* Schritt zu bestimmen und mit dem Volumengitter abzuspeichern. Der Platzbedarf für das Gitter werde dadurch zwar stark erhöht[12], aber da jede Fläche von mehreren Strahlen geschnitten wird, wird die Schnittberechnung durch die gespeicherten Werte stark beschleunigt (die Autoren sprechen von ca. 75%).

Hingegen kaum Auswirkungen auf die Geschwindigkeit hatten nach ihren Angaben leichte Änderungen in der Strategie zum Auffinden der jeweiligen Folgezelle, wie z.B. gesondertes vorgezogenes Testen der „dem Eintrittspunkt gegenüber“ liegenden Zellenwand auf Schnitt mit dem Strahl.

[12]Bei den von [RW92] verwendeten Datenstrukturen für curvilineare (also strukturierte) Gitter wurde die Datensatzgröße nahezu verdreifacht

Ein anderes Raycasting-Verfahren als das oben beschriebene iterative Übergehen zur nächsten Volumenzelle wird von Frühauf ([Frü94]) vorgeschlagen. Es arbeitet auf *curvilinearen* Gittern und nutzt die inhärente Struktur dieser Gitter. Wie auf Seite 15 bereits angemerkt, sind curvilineare Gitter *array*-organisiert. Somit besteht eine eineindeutige Abbildung zwischen jeder curvilinearen Zelle und einer Zelle in einem topologisch äquivalenten regulären Referenzgitter, das von Frühauf als *computational space* bezeichnet wird. Das Raycasting wird nun auf diesem Referenzgitter ausgeführt, wobei nun die Strahlen natürlich potentiell krummlinig sind und mit Stromlinien über einem Vektorfeld verglichen werden können. Integration erfolgt nun wie bei vielen Verfahren auf regulären Gittern: Auf den berechneten Stromlinien werden in gleichen Abständen Datenwerte abgegriffen, auf Farb- und Transparenzwerte abgebildet und in einem *Compositing* Schritt akkumuliert.

**Verschiedene Scanline-orientierte Ansätze** Die bisher beschriebenen *Raycasting* Methoden bestimmen die Schnittpunkte des aktuellen Strahls mit den Volumenzellen bzw. die jeweils nächsten Samplepunkte lokal entlang des Strahles unter Ausnutzung der Nachbarschaftsstrukturen im Gitter. Scanline-Verfahren hingegen bestimmen die Schnittpunkte von Strahlen und Zellen entlang einer ganzen Pixelzeile oder *Scanline* des Ergebnisbildes und machen keinen Gebrauch von Nachbarschaftsinformation, dafür aber von einer Einordnung der Zellen, bzw. ihrer Kanten, nach Bildschirmkoordinaten.

Ein großer Vorteil von Scanline Verfahren ist ihre Flexibilität den darzustellenden Elementen gegenüber. Sie stellen keine Anforderungen an das den Daten zugrunde liegende Gitter — Struktur, Ausrichtung der Zellen zueinander und Zellenform sind frei wählbar — und erlauben die gleichzeitige Verarbeitung von Elementen unterschiedlicher Art (Volumendaten, Oberflächendaten, geometrische Primitive etc.), was unter dem Begriff *Integrated Rendering* geführt wird.

Als Implementierungen von Scanline-orientierten Volume Rendering Verfahren für irreguläre Gittern sind die Arbeiten von Wilhelms und Challinger ([WCAR90, Cha90, Cha93]) und von Giertsen ([Gie92]) zu nennen.

Giertsens Motivation für die Entwicklung seines Algorithmus war die Anforderung, Daten auf Gittern zu visualisieren, deren Elemente weder an gemeinsamen Flächen ausgerichtet, noch notwendigerweise überhaupt verbunden sein mußten (*sparse irregular meshes*). Er schränkt die Zellenform auf konvexe Hexaeder ein, u.a. um die Interpolation innerhalb der Zellen einigermaßen effizient vornehmen zu können. Kernstück des Algorithmus ist der sogenannte *scan-plane buffer*, den man sich als in das Datenvolumen hineinragende horizontale Ebene auf der Höhe der Scanline vorstellen kann. In diesem Buffer werden für jeden Pixel auf der Scanline die Integrationsanteile, die entlang des durch den Pixel führenden Strahls anfallen, *front-to-back* protokolliert. Wie bei Scanline-Algorithmen üblich, wird für die aktuelle Pixelzeile eine Liste „aktiver" Elemente geführt, diese werden gegen die *scan-plane* geschnitten und die entstehenden Polygone in Dreiecke zerlegt. In diesen erfolgt die Interpolation der Datenwerte und die Bestimmung der Integrationsanteile für jeden Pixel. Schließlich wird der gefüllte *scan-plane buffer* für jeden Pixel traversiert und die Scanline gezeichnet.

An der University of California, Santa Cruz, entstand im Rahmen einer Masters- und Doktorarbeit ([Cha90, Cha93]) ein Scanline-orientierter Volume Renderer, der auch als paralleler Algorithmus implementiert wurde. War das Projekt ursprünglich

auf die Anforderung curvilinearer Gitter ausgelegt, nutzen spätere Implementierungen die Allgemeinheit des Algorithmus und bieten größtmögliche Freiheit in der Wahl der behandelbaren Gitter.

Es existieren drei Ausbaustufen des Systems: Der erste Algorithmus arbeitet auf curvilinearen Gittern und speichert für jede Volumenzelle neben den Koordinaten der Eckpunkte noch die Koordinaten einer rechtwinkligen Bounding-Box ab, über die die „aktiven" Zellen bestimmt werden. Mittels eines Scanline-Algorithmus mit nach y- und x-Koordinaten sortierten Listen der aktiven Zellen werden nacheinander für jeden Pixel auf der Scanline die Schnittpunkte von Pixelstrahlen und Volumenzellen bestimmt und die *Sampling, Mapping* und *Compositing* Schritte ausgeführt.

Im zweiten Algorithmus werden die Schritte *Sampling* und *Mapping* in zwei verschieden Phasen ausgeführt, wodurch eine schnelle Bildkorrektur bei Änderung der *mapping functions* ermöglicht wird. Erkauft wird dies durch die Notwendigkeit eines großen globalen Buffers zum Speichern der an den *sample points* ermittelten Daten. Dies macht Sinn, wenn man im Rahmen einer Parallelisierung den Speicher lokal auf viele Prozessoren verteilen kann.

Der dritte Algorithmus arbeitet nicht mehr auf (curvilinearen) Zellen, sondern auf Zellflächen als Hauptelementen und kann Objekte verschiedensten Typs in einer gemeinsamen Szene rendern. Auch dynamische Gitteränderungen können behandelt werden.

Es hat sich herausgestellt, daß sich der grundsätzlich eher langsame Scanline-Algorithmus gut zur Parallelisierung eignet, da er ein gutes Skalierungsverhalten mit steigender Anzahl der Prozessoren an den Tag legt.

**Verschiedene Ansätze bei den Projektionsverfahren** Genau wie bei den *Raycasting*-Methoden ergibt sich auch bei den Projektionsmethoden das Bild einiger weniger sehr erfolgreicher Basisverfahren und schrittweiser Verbesserung und Erweiterung dieser Ansätze.

Max, Hanrahan und Crawfis ([MHC90]) und Shirley und Tuchman ([ST90]) stellen 1990 zwei sehr ähnliche Verfahren zur Projektion unstrukturierter Gitterzellen vor. Mehrere Autoren greifen die grundlegenden Ideen dieser Ansätze auf und erweitern sie in verschiedene Richtungen, indem sie z.B. neue Näherungsverfahren entwickeln ([Wil92c, Wil92d]), den Algorithmus für andere Gitterarten adaptieren ([VGW93]) oder auf neue Anwendungsfälle hin spezialisieren ([MBC93]). Lucas ([Luc92]) schlägt ein etwas anderes Verfahren vor, das auf der Projektion von Zellen*seiten* beruht.

Die grundlegende Vorgehensweise des Ansatzes aus [MHC90, ST90] ist folgende: Erst müssen die Gitterzellen ihrer Lage zum Betrachter nach sortiert werden (*back-to-front* oder *front-to-back*), dann wird für jede Zelle eine Scan-Konvertierung der Zellenumrisse vorgenommen. Die Intensitäts- und Dichtebeiträge für jeden betroffenen Pixel werden durch Integration und/oder Approximation bestimmt und die resultierenden Werte werden im *Compositing* Schritt ins Bild eingefügt.

Der Hauptunterschied zwischen beiden Verfahren ist, daß in [MHC90] eine genaue Integrationsmethode für konvexe Polyederzellen vorgeschlagen und das Potential von Näherungen und Hardwarenutzung nur angemerkt wird, während der Schwerpunkt von dem in [ST90] vorgestellten *Projected Tetrahedra* (PT) Algorithmus auf einem schnellen Approximationsverfahren für Tetraedergitter liegt. Die hohe Geschwindigkeit des PT-

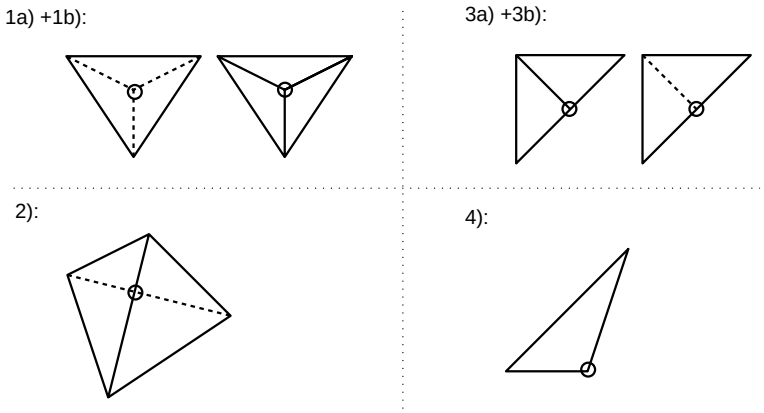



Abbildung 2.8: *Mögliche Projektionen eines Tetraeders*

Algorithmus, u.a. durch effektiven Einsatz moderner Grafik-Hardware, hat zu seiner weiten Verbreitung (wenn man für den Anwendungsfall irregulärer Gitter überhaupt von einer solchen sprechen kann) entscheidend beigetragen.

Die Grundzüge des Verfahrens wurden bereits im Rahmen der Approximationsmethoden für die Intensitätsintegration dargestellt (S. 12 f.). Wenn man die degenerierten Projektionen mitzählt, gibt es sechs verschiedene Projektionsmöglichkeiten für ein Tetraeder. Für jede Projektion läßt sich auf einfache Weise ein Punkt T größter Tiefe ermitteln. Abb. 2.8 zeigt alle möglichen Projektionen, wobei die Punkte T durch Kreise markiert sind. Die Projektionen werden nun in Dreiecke zerlegt (drei in Fall 1a und 1b, vier in Fall 2, zwei in Fall 3a und 3b und eines in Fall 4). Für die Eckpunkte des Tetraeders und für Punkt T werden die anliegenden Datenwerte bestimmt und auf Farbe und Dichte abgebildet (*Mapping*). Über die Tetraeder werden, wie in Abschnitt 2.1.1.2 auf S. 13 beschrieben, die farbgebenden Größen interpoliert, wobei *Hardware-Gouraud-Shading* eingesetzt werden kann.

Williams ([Wil92c, Wil92d]) greift den PT-Algorithmus mit dem Ziel auf, interaktive Bildberechnungszeiten zu erreichen. Er stellt vier Näherungsstufen vor, die allesamt die über die Projektion variierende Tiefe des Tetraeders außer acht lassen und statt dessen konstante Tiefe mit einem (empirisch ermittelten) Wichtungsfaktor annehmen. Dadurch müssen weniger Dreiecke verarbeitet werden (z.B. reicht es, in Projektionsfall 2 die beiden nach vorne zeigenden Flächen zu behandeln). Außerdem verlagert Williams das *Mapping* der Skalarwerte in einen Preprocessing Schritt, indem er für jede Zelle konstante Durchschnittswerte für Farbe und Transparenz ermittelt und abspeichert.

Neben diesen Methoden zur weiteren Approximation der Intensitätsintegrale erörtert Williams kontrollierte Ausdünnungen des Datenvolumens, sogenannte *reduced resolution meshes*. Dafür werden entweder nach einem bestimmten Schema ausgewählte Zellen einfach unberücksichtigt gelassen oder Teile des Volumens unter Aussparung bestimmter Knoten *re-trianguliert*. Weiterhin stellt er parallele Versionen des Renderingalgorithmus sowie des Algorithmus zum Sortieren der Zellen vor. Die Skalierung mit der Anzahl der Prozessoren ist jedoch nicht unbedingt zufriedenstellend.

Insgesamt entstehen durch diese vielfältigen Beschleunigungsverfahren gute Möglichkeiten, zwischen Geschwindigkeit und Genauigkeit abzuwägen. Es ist jedoch ein ungelöstes Problem, welchen theoretischen Rahmen und welche Parameter man dem Be-

nutzer für diese Aufgabe an die Hand gibt.

Interaktive Antwortzeiten sind auch das Ziel des in [MBC93] beschriebenen Ansatzes. Dieser ist ganz und gar auf die interaktive Erkundung von Strömungsmodellen ausgerichtet und setzt hierzu den PT-Algorithmus ein. Da die meisten Strömungsmodelle auf curvilinearen Gittern definiert sind, wird ein adaptiver Algorithmus präsentiert, der die Unterteilung in Tetraeder vornimmt. Visualisiert wird der Fluß von (Rauch)partikeln in einem Strömungsfeld. Der Benutzer positioniert mit einem 3D-Cursor ein „Quellpolygon", das sozusagen Rauch ausstößt, der in den folgenden Zeitschritten ein sich immer weiter ausbreitendes Flußvolumen füllt. Um die nahezu interaktiven Antwortzeiten für die einzelnen Zeitschritte zu erreichen, wird eine uniforme Farbe für das Rauchvolumen definiert, was angeblich für den gewünschten visuellen Effekt ausreicht. Dadurch entfällt bei Rotationen der Szenerie die ansonsten notwendige Neusortierung der Zellen. Als Verbesserung des Gouraud-Shading Ansatzes aus dem PT-Algorithmus wird vorgeschlagen, die *texture mapping* Fähigkeiten mancher Grafik-Hardware zur Realisierung einer *look-up-table* einzusetzen, um eine exponentielle Interpolation über die einzelnen Dreiecke zu erreichen (vgl. auch S. 14 und ausführliche Darstellung in [SBM94]).

Van Gelder und Wilhelms ([VGW93]) stellen einen Projektionsalgorithmus für curvilineare Gitter vor, der einige Ideen des PT-Algorithmus und der Erweiterungen von [Wil92c] aufgreift. Krummlinige Hexaeder haben natürlich sehr viel kompliziertere Projektionen als Tetraeder. Die Autoren bieten mehrere Approximationsstufen an, die teilweise wie bei Williams' Näherungen zum PT-Algorithmus die Tiefeninformation über der Zellprojektion vernachlässigen.

Bevor zum Abschluß dieses Abschnitts ein Verfahren beschrieben wird, das sich von der bisher zugrundegelegten Vorgehensweise, Volumenzellen zu projizieren leicht unterscheidet, soll noch auf die Sortierungsverfahren für die Zellen eingegangen werden:

Williams ([Wil92b]) gibt eine detaillierte Abhandlung von Sortierungsverfahren für konvexe Polyeder, für die innerhalb eines Gitters die benachbarten Zellen bekannt sind. Das grundlegende Verfahren ist der MPVO-Algorithmus (*Meshed Polyhedra Visibility Ordering*) Für eine vorgegebene Betrachterposition findet der Algorithmus bei Gittern ohne Löcher und Konkavitäten immer eine korrekte Sortierung, solange keine Zyklen, also wechselseitige oder ringförmige Verdeckungen von Volumenzellen auftreten. Es ist gesichert, daß solche Zyklen bei Tetrahedrisierungen durch *Delaunay-Triangulierung* nicht auftreten. Für Gitter mit Löchern oder Konkavitäten an der Außenfläche wird eine Heuristik angegeben, die in den allermeisten Fällen eine korrekte Sortierung findet und nur in exakt definierbaren „anomalen" Ausnahmefällen kleine lokale Fehler macht.

Dieser Sortierungsalgorithmus ist sehr schnell, sein Aufwand ist linear zur Anzahl der Zellflächen im Gitter. Williams berichtet von einer Verarbeitungsrate von ca. 60.000 Tetraedern in der Sekunde.

In [SBM94] wird ein alternatives Sortierverfahren vorgeschlagen, das eine fest definierte Nachbarschaft zwischen Volumenzellen *nicht* zur Voraussetzung macht. Mit diesem sehr viel langsameren Verfahren können auch Datengitter sortiert werden, bei denen die Zellen nicht an gemeinsamen Seitenflächen ausgerichtet sind.

Alle bisher beschriebenen Projektionsverfahren arbeiten nacheinander alle Volumenzellen ab und bestimmen für alle Pixel in deren Projektion Intensitätsbeiträge.

Das von Lucas ([Luc92]) beschriebene Verfahren unterscheidet sich insofern, daß hier die Zellen selbst von vornherein vernachlässigt werden und nur die Zellen*seiten* sortiert, projiziert und zur Integration herangezogen werden. Das Verfahren ist Teil eines sehr kompletten und durchdachten Visualisierungssystems, das als Rendering-Modul Einsatz in der Visualisierungsumgebung *IBM Data Explorer* gefunden hat. Das Volume Rendering Verfahren arbeitet wie folgt: Die Seitenflächen der Volumenzellen (beliebige Polyeder) werden *back-to-front* sortiert. Jede Fläche wird nun scan-konvertiert, wobei für jeden Pixel die aktuelle z-Koordinate in einem Buffer gespeichert wird. Somit steht bei der Projektion der nächsten Fläche im Buffer der z-Wert der nächst weiter hinten liegenden Seitenfläche, auf die ein Strahl durch den aktuellen Pixel treffen würde. Somit steht auch die Länge des Integrals fest, über das für diesen Pixel integriert werden muß, um den nächsten Beitrag für das *Compositing* zu bestimmen.

Dieses Verfahren ermöglicht in ähnlicher Allgemeinheit wie die *Scanline Algorithmen* eine gleichzeitige Verarbeitung von verschiedenen geometrischen Elementen (*Integrated Rendering*). Lucas stellt außerdem eine Methode zum schnellen *render-time clipping*, also interaktiven Entfernen von Bildbereichen vor, die auf dem Einsatz Z-Buffer-ähnlicher Speicherstrukturen basiert. Schließlich wird ein Konzept zur Parallelisierung sämtlicher in diesem *Renderer* verwirklichten Funktionalität vorgestellt.

### 2.2.3 Die Verfahren im Vergleich

#### 2.2.3.1 Tabellarische Übersicht

Auf den folgenden Seiten soll eine tabellarische Übersicht über diejenigen Veröffentlichungen gegeben werden, die eigenständige Verfahren zum *Volume Rendering auf irregulären Gittern* etabliert oder Verbesserungen zu bestehenden Verfahren durchgesetzt haben.

Es wird die Art des Verfahrens charakterisiert, Bezüge zu früheren Veröffentlichungen werden herausgestellt und die Art der behandelbaren irregulären Gitter wird beschrieben.

Die Geschwindigkeit der einzelnen Verfahren wird in recht grober Einteilung abgeschätzt:

| | |
|---|---|
| 1 - Interaktive Antwortzeit | (bis $\sim$ 1 s.) |
| 2 - Schnell | (bis $\sim \frac{1}{2}$ min.) |
| 3 - Durchschnitt | (bis $\sim$ 5 min.) |
| 4 - Langsam | (bis $\sim$ 15 min.) |
| 5 - Remote | (über 15 min.) |

Die Zeitangaben mögen dabei als Bildberechnungszeiten eines irregulären Datensatzes durchschnittlicher Grösse (10.000 - 20.000 Zellen) bei einer mittleren Bildgröße (z.B. 512x512 Pixel) auf einer modernen Workstation verstanden werden. Es ist klar, daß eine Einordnung der Verfahren in solch ein Raster insbesondere bei den unterschiedlichen und teilweise recht mager dokumentierten Testläufen nur sehr bedingt möglich ist. Dennoch kann eventuell ein erster Eindruck von der Ausrichtung des jeweiligen Algorithmus' vermittelt werden. Es sollte bedacht werden, daß schnelle Bildberechnungen meist mit Qualitätseinbußen erkauft werden müssen. Dort, wo Geschwindigkeit durch Parallelisierung erreicht wurde, ist dies ausdrücklich vermerkt.

Schließlich werden besondere Charakteristika des im jeweiligen Artikel beschriebenen Verfahrens erwähnt und in der abschließenden Zeile wird von seiten des Diplomanden eine subjektive Beurteilung und Einordnung der Veröffentlichung vorgenommen.

Die Übersicht ist chronologisch geordnet.

| [WCAR90] | J. Wilhelms, J. Challinger, N. Alper and S. Ramamoorthy<br>*Direct Volume Rendering of Curvilinear Volumes* |
|---|---|
| Artikeltyp: | Proceedings: San Diego Workshop on Volume Visualization |
| Verfahren: | a) Scanline Verfahren, b) Interpolation auf reg. Gitter |
| Referenz: | [Cha90], sequentieller Vorläufer von [Cha93] |
| Gitterart: | *curvilinear*, deformierte Hexaeder (Konvexität angenommen) |
| R'zeit(1-5): | a) 4-5, b) Interpolation zeitaufwendig (aber nur einmalig) |
| Besonderes: | Verschiedene Strategien der Interpolation auf reg. Gitter. |
| Die Überlegenheit von nativem Volume Rendering auf irreg. Gittern gegenüber einem *Resampling* auf ein reguläres Gitter ist heute sehr viel deutlicher als damals | |

| [Gar90] | Michael P. Garrity<br>*Raytracing Irregular Volume Data* |
|---|---|
| Artikeltyp: | Proceedings: San Diego Workshop on Volume Visualization |
| Verfahren: | Raycasting Verfahren |
| Referenz: | |
| Gitterart: | konvexe Polyeder mit fester Nachbarschaft |
| R'zeit(1-5): | 3 |
| Besonderes: | Erste Veröffentlichung des klassischen Cell-to-Cell Raycastings |
| Gute Darstellung eines - sehr einfachen - Algorithmus | |

| [MHC90] | Nelson Max, Pat Hanrahan, and Roger Crawfis<br>*Area and Volume Coherence for Efficient Visualization of 3D Scalar Functions* |
|---|---|
| Artikeltyp: | Proceedings: San Diego Workshop on Volume Visualization |
| Verfahren: | Projektionsverfahren |
| Referenz: | Ähnlichkeit mit [ST90] |
| Gitterart: | konvexe Polyeder, Sortierung über feste Nachbarschaft |
| R'zeit(1-5): | 3 |
| Besonderes: | einfache Berechnung der Intensität, wenn Emission $\sim$ Absorption |
| Zusammen mit [ST90] *der* Pionier-Artikel für Projektionsverfahren auf irreg. Daten | |

| [ST90] | Peter Shirley and Allan Tuchman<br>*A Polygonal Approximation to Direct Scalar Volume Rendering* |
|---|---|
| Artikeltyp: | Proceedings: San Diego Workshop on Volume Visualization |
| Verfahren: | Projektionsverfahren |
| Referenz: | Ähnlichkeit mit [MHC90] |
| Gitterart: | Tetraedergitter, Sortierung über feste Nachbarschaft |
| R'zeit(1-5): | 2 |
| Besonderes: | hohe Geschwindigkeit durch Einsatz von Grafik-Hardware |
| Klassisches Verfahren für schnelles VR auf unstrukturierten Gittern. Immer wieder aufgegriffen | |

| [Use91] | Sam Uselton<br>*Volume Rendering for Computational Fluid Dynamics: Initial Results* |
|---|---|
| Artikeltyp: | TR, WWW: http://techreports.larc.nasa.gov/cgi-bin/NTRS |
| Verfahren: | Raycasting Verfahren |
| Referenz: | Weiterführung des Ansatzes von [Gar90] |
| Gitterart: | konvexe Polyeder mit fester Nachbarschaft |
| R'zeit(1-5): | 4 |
| Besonderes: | Darstellung von Vektorfeldern über 'naives' Vektor $\longrightarrow$ RGB Mapping |
| Ohne großen Innovationsanspruch. Einblick in die Anforderungen von CFD | |

| [Koy92] | Koji Koyamada<br>*Fast Traverse of Irregular Volumes* |
|---|---|
| Artikeltyp: | Proceedings, Sammelband: Visual Computing, Springer Verlag |
| Verfahren: | Raycasting Verfahren |
| Referenz: | Ähnlicher Ansatz wie [Gar90] |
| Gitterart: | Tetraedergitter mit fester Nachbarschaft |
| R'zeit(1-5): | 3 (leichte Beschleunigung ggü. [Gar90]) |
| Besonderes: | schnellere Berechnung des Austrittspunktes aus der aktuellen Zelle |
| Verbesserungen für das Cell-to-Cell Raycasting Verfahren | |

| [Gie92] | Christopher Giertsen<br>*Volume Visualization of Sparse Irregular Meshes* |
|---|---|
| Artikeltyp: | Zeitschrift (IEEE CG & Applications, März 92) |
| Verfahren: | Scanline Verfahren |
| Referenz: | |
| Gitterart: | bel. deformierte Hexaeder ohne feste Nachbarschaft und Ausrichtung |
| R'zeit(1-5): | 3 |
| Besonderes: | Sehr allg. Gitter behandelbar, HLS-basiertes Color-Mapping |
| Gliederung des Artikels etwas unübersichtlich. Sehr eigenständiges Verfahren | |

| [Wil92a] | Jane Wilhelms<br>*Pursuing Interactive Visualization of Irregular Grids* |
|---|---|
| Artikeltyp: | Proceedings: Workshop Visualisierung '92, Sankt Augustin |
| Verfahren: | Allgemeiner Vergleich |
| Referenz: | Präsentation einiger Ergebnisse aus [RW92] |
| Gitterart: | vgl. [RW92] |
| R'zeit(1-5): | " |
| Besonderes: | --- |
| Gut gegliederter Übersichtsartikel | |

| [RW92] | Shankar Ramamoorthy and Jane Wilhelms<br>*An Analysis of Approaches to Ray-Tracing Curvilinear Grids* |
|---|---|
| Artikeltyp: | Technical Report |
| Verfahren: | Raycasting Verfahren |
| Referenz: | Zusammenfassende Analyse von Verfahren aus [Use91] und [Gar90] |
| Gitterart: | *curvilinear*, Ausgleichsflächen bei Nichtplanaritäten |
| R'zeit(1-5): | 4-5 |
| Besonderes: | Aufwandsuntersuchungen für die einzelnen Schritte des Algorithmus |
| Ziemlich komplette Diskussion des Cell-to-Cell Raycasting Ansatzes | |

| [Luc92] | Bruce Lucas<br>*A Scientific Visualization Renderer* |
|---|---|
| Artikeltyp: | Proceedings: Visualization '92 |
| Verfahren: | Projektion von Voxel*seiten* |
| Referenz: | |
| Gitterart: | beliebige Polyeder ohne feste Nachbarschaft und Ausrichtung |
| R'zeit(1-5): | 2-3 (geschätzt, da nur sehr vage Angaben) |
| Besonderes: | *Integrated Rendering*[13]. Schnelles Rendertime-Clipping |
| Beschreibung eines sehr kompletten Systems (Renderer des IBM Data Explorers) | |

| [Wil92c] | Peter L. Williams<br>*Interactive Splatting of Nonrectilinear Volumes* |
|---|---|
| Artikeltyp: | Proceedings: Visualization '92 |
| Verfahren: | Projektionsverfahren |
| Referenz: | Weiterführung von [ST90] |
| Gitterart: | Tetraedergitter, Sortierung über feste Nachbarschaft |
| R'zeit(1-5): | 1-2 (incl. Zeit für Sortierung: 2) |
| Besonderes: | Beinahe Interaktivität (Näherungsverfahren, *Reduced Resolution Meshes*) |
| Die abgestuften Approximationen zu [ST90] ermöglichen ein Abwägen zwischen Qualität und Geschwindigkeit | |

| [Wil92d] | Peter L. Williams<br>*Interactive Direct Volume Rendering of Curvilinear and Unstructured Data* |
|---|---|
| Artikeltyp: | Ph.D. Arbeit,<br>FTP: a.cs.uiuc.edu:pub/dcs/Tech_Reports/UIUCDCS-R-93-1841.ps |
| Verfahren: | Projektionsverfahren |
| Referenz: | Nähere Betrachtung der Verfahren aus [Wil92c] |
| Gitterart: | vgl. [Wil92c] |
| R'zeit(1-5): | " |
| Besonderes: | zusätzliche Parallelisierung von Sortierung und Rendering |
| Etwas theoretischere Betrachtung der Algorithmen aus Williams' anderen Artikeln | |

| [Cha93] | Judith Ann Challinger<br>*Scalable Parallel Direct Volume Rendering for Nonrectilinear Computational Grids* |
|---|---|
| Artikeltyp: | Ph.D. Arbeit, FTP: ftp.cse.ucsc.edu:/pub/tr/ucsc-crl-93-47.ps |
| Verfahren: | Scanline Verfahren |
| Referenz: | Bezugnahme auf [WCAR90] |
| Gitterart: | bel. deformierte Hexaeder ohne feste Nachbarschaft und Ausrichtung |
| R'zeit(1-5): | 2-5 je nach Anzahl der Prozessoren (100..1) |
| Besonderes: | *Integrated Rendering*[13], Dynamische Gitteränderungen |
| Langsamer Basisalgorithmus, aber sehr allgemein und gut parallelisierbar. Die Doktorarbeit ist leider recht unübersichtlich strukturiert | |

| [VGW93] | Allen Van Gelder and Jane Wilhelms<br>*Rapid Exploration of Curvilinear Grids Using Direct Volume Rendering* |
|---|---|
| Artikeltyp: | Proceedings Visualization '93<br>FTP: ftp.cse.ucsc.edu:/pub/tr/ucsc-crl-93-02.ps.Z |
| Verfahren: | Scanline Verfahren |
| Referenz: | Ansatz aus [ST90] für curvilineare Gitter |
| Gitterart: | *curvilinear*, deformierte Hexaeder (Konvexität angenommen) |
| R'zeit(1-5): | 2 |
| Besonderes: | Mehrere Näherungsverfahren für die Zellenintensitäten |
| Gute Übertragung von Ideen aus [ST90],[Wil92c] auf den Fall curvilinearer Gitter | |

| [MBC93] | Nelson Max, Barry Becker, and Roger Crawfis<br>*Flow Volumes for Interactive Vector Field Visualization* |
|---|---|
| Artikeltyp: | Proceedings: Visualization '93 |
| Verfahren: | Projektionsverfahren |
| Referenz: | Anwendung von [ST90] auf *Computational Fluid Dynamics* |
| Gitterart: | Tetraedergitter, (Aufspaltung *curvilinearer* Zellen in Tetraeder) |
| R'zeit(1-5): | 1 (bei recht kleinen Datensätzen) |
| Besonderes: | Umgehen der Vorsortierung durch Annahme uniformer Farbverteilung |
| Vektorfeldvisualisierung durch interaktive Ausrichtung der Flußvolumina | |

| [SBM94] | Clifford Stein, Barry Becker, and Nelson Max<br>*Sorting and Hardware Assisted Rendering for Volume Visualization* |
|---|---|
| Artikeltyp: | Proceedings: 1994 Symposium on Volume Visualization |
| Verfahren: | Projektionsverfahren |
| Referenz: | genauere Integration und allgemeinere Sortierung für [ST90] |
| Gitterart: | Tetraedergitter, feste Nachbarschaft nicht Voraussetzung |
| R'zeit(1-5): | vgl. [ST90], vergleichsweise langsames Sortieren |
| Besonderes: | — |
| Das genauere Integrationsverfahren wurde bereits in [MBC93] vorgestellt. | |

| [Frü94] | Thomas Frühauf<br>*Raycasting of Nonregularly Structured Volume Data* |
|---|---|
| Artikeltyp: | Proceedings: Eurographics '94 |
| Verfahren: | Raycasting Verfahren |
| Referenz: | |
| Gitterart: | *curvilinear*, irregulär *strukturierte* Hexaedergitter |
| R'zeit(1-5): | leider keine Angabe |
| Besonderes: | Raycasting auf dem regulären *computational space* curvilinearer Gitter |
| Innovativ. Erstes Verfahren, das die Feldstruktur curvilinearer Gitter nutzt. | |

| [TSM94] | Behnam Tabatabai, Emanuela Sessarego, and Harald Mayer<br>*Volume Rendering on Non-regular Grids* |
|---|---|
| Artikeltyp: | Proceedings: Eurographics '94 |
| Verfahren: | Raycasting Verfahren |
| Referenz: | Raycasting, ähnlich [Gar90], unter Einsatz von FE-Methodik |
| Gitterart: | Hexaeder oder Tetraedergitter mit fester Nachbarschaft |
| R'zeit(1-5): | 3-4 |
| Besonderes: | Einsatz von *shape functions* zur Interpolation (Geometrie & Daten) |
| Vorteil: Mathematische Gleichbehandlung von Simulation und Visualisierung | |

[13]Unter *Integrated Rendering* versteht man die gleichzeitige Darstellung von geometrischen Objekten verschiedener Art (z.B. Volumenzellen, polygonal definierte Objekte, Linien etc.)

#### 2.2.3.2 Auswertung

In diesem Abschnitt werden die beschriebenen Verfahrensklassen nach den in Abschnitt 2.2.1 aufgeführten Kriterien miteinander verglichen. Somit wird noch einmal ein abschließender Überblick über die Leistungsfähigkeit der verschiedenen Verfahren gegeben. Eine klare Einordnung der Stärken und Schwächen ist wichtiger als eine erschöpfende Diskussion der Unterschiede zwischen den Verfahren.

**Art der Gitter** Was die Art der behandelbaren Gitterarten betrifft, bilden die Scanline-Verfahren die vielseitigste und allgemeinste Gruppe. Es werden von seiten des grundlegenden Algorithmus kaum Bedingungen an die Struktur der Datengitter gestellt. Diese Verfahren lassen somit auch am einfachsten hybride Gitter zu, also Kombinationen von z.B. curvilinearen und unstrukturierten Teilgittern. Auch ein sogenanntes *Integrated Rendering* ist mit dieser Algorithmenklasse ohne große Probleme zu bewerkstelligen. Der spezielle Projektionsansatz aus [Luc92], bei dem die Zellen*seiten* projiziert werden, ist allerdings ähnlich vielseitig in bezug auf verarbeitbare Gitterstrukturen und geometrische Objekte.

Bei den zellenorientierten Projektionsverfahren, bei denen die Zellprojektionen entsprechend ihrer Topologie klassifiziert werden müssen, können keine nicht-konvexen Zellen verarbeitet werden. Tetraeder sind am besten geeignet, da diese nur ganz wenige verschiedene Projektionstopologien besitzen. Curvilineare oder beliebige Hexaeder-Zellen erfordern mehr Aufwand. Bei der Projektion dieser Zellenarten wird im Gegensatz zu Tetraedern meist rigoros genähert (Splatting). Für curvilineare Zellen und unregelmäßige Hexaeder gibt es auch mehr Probleme mit der Vorsortierung der Zellen [Wil92a]. Gesichert korrekt kann die Vorsortierung auf effizientem Wege für Delaunaytetraedisierte Gitter vorgenommen werden ([Wil92b]).

Für die Raycasting Verfahren sollte das Gitter zusammenhängend sein. Nur über eine feste Nachbarschaftsstruktur kann man sich von Zelle zu Zelle hangeln. Die Komplexität der Schnittpunktberechnung zwischen Strahl und Seitenflächen hängt direkt von der zugrunde liegenden Form der Gitterzellen ab: Bei mehr Flächen pro Zelle oder nichtplanaren Zellenseiten steigt der Berechnungsaufwand. Im Ggs. zu den Projektionsverfahren ist auch eine Behandlung nicht-konvexer Zellen vorstellbar.

**Integrationsmethoden** Grundsätzlich bieten die verschiedenen Verfahrensklassen die gleichen Möglichkeiten bezüglich der Intensitätsintegration. Die Projektionsverfahren erlauben allerdings zusätzliche Approximationsverfahren, indem sie z.B. die Farbbeiträge über die Zellprojektion interpolieren oder einen *Splatting* Ansatz verfolgen (vgl. Abschnitt 2.1.1.2, S. 12 ff.).

Annahmen über die Verteilung der Datenwerte innerhalb einer Zelle haben direkte Auswirkungen auf die Effizienz der Integrationsmethoden (vgl. Abschnitt *Spezialfälle in geschlossener Form*, S. 9 ff.). Wenn numerische Simulationsergebnisse visualisiert werden sollen, ist es sinnvoll, für die Datenakquisition innerhalb der Zelle genau die Interpolationsverfahren zu verwenden, die auch in der Simulation eingesetzt wurden (vgl. [TSM94]). Werden Interpolationsverfahren höherer Ordnung eingesetzt, ist eine Lösung des Intensitätsintegrals in geschlossener Form nicht mehr möglich, und die angesprochenen auf einfachen Datenverteilungen basierenden Approximationsverfahren

verfälschen die Ergebnisse besonders stark. In einem solchen Fall ist es wahrscheinlich besser, zur Beschleunigung der Bildberechnung andere als die integrationsorientierten Approximationsverfahren anzuwenden (z.B. Subsampling bei Raycasting, Ausdünnung des Datenvolumens bei Projektionsverfahren).

Immer wenn global für einen ganzen Strahl Entscheidungen getroffen werden sollen, sind die *Raycasting* oder *Scanline* Ansätze besser geeignet als Projektionsverfahren, da die Strahlen am Stück traversiert werden. Wenn z.B. adaptiv über den Strahl hinweg zwischen verschiedenen Approximationsmethoden umgeschaltet werden soll (da man sich in den hinteren Regionen größere Fehler erlauben darf), hat man es leichter, wenn die Strahlen zusammenhängend abgearbeitet werden. Mit etwas mehr Verwaltungsaufwand läßt sich ein solches Verfahren allerdings auch bei Projektionsverfahren realisieren.

**Bildqualität, Artefakte** Das Auftreten von Artefakten im berechneten Bild ist in großem Maße von den eingesetzten Approximationsverfahren abhängig und nicht so sehr von der Art der Verfahrensklassen. In Abschnitt 7 werden z.B. typische Artefakte der Gouraud-Shading Approximation bei Projektionsverfahren näher beleuchtet.

Es existieren aber auch Artefakte, die typisch für die einzelnen Verfahrensgruppen sind. So kann es z.B. beim Raycasting passieren, daß man manche Volumenzellen gar nicht zur Bildberechnung heranzieht, da sie nicht explizit von einem Strahl getroffen wurden. Dies läßt sich durch das Aussenden mehrerer Strahlen pro Pixel (*supersampling*) korrigieren. Bei Projektionsverfahren muß man dafür Sorge tragen, daß die Ränder der Volumenzellen nicht mehrfach für verschiedene Zellen in die Projektion einbezogen werden.

Bei gemessenen Datenverteilungen, die in der Form irregulärer *sample points* vorliegen, steht man vor der schwierigen Entscheidung für eine „realistische" Interpolation zwischen diesen Punkten. Die Wahl eines geeigneten *Rekonstruktionsfilters* kann helfen, Artefakte zu vermeiden.

**Geschwindigkeit** Projektionsverfahren haben grundsätzlich einen Geschwindigkeitsvorteil gegenüber den Raycasting- und Scanline-Verfahren. Sie nutzen automatisch die Kohärenz innerhalb Volumenzellen. Man kann durch erhöhten Verwaltungsaufwand (Zwischenspeichern von Ebenengleichungen, Interpolationskoeffizienten, etc.) einen Teil dieser Kohärenz auch bei den anderen Verfahren teilweise nutzbar machen, einiges aber ist nicht übertragbar, wie z.B. die Möglichkeit einer schnellen Interpolation von Koeffizienten über die Zellprojektion während der Scan-Konvertierung.

Für sehr große Datensätze (viele Volumenzellen) büßen die Projektionsverfahren ihren Geschwindigkeitsvorteil ein, da die projizierten Zellen auf dem Bildschirm dann nur noch ein paar Pixel einnehmen. Bei der Wahl vergrößerter Bildausschnitte (*zooming*) und bei sehr großen Ergebnisbildern wird das Verhältnis dann wieder zugunsten der Projektionsverfahren verschoben.

Die Scanline-Verfahren, die in den 70er Jahren gerade mit der Intention entworfen worden waren, Kohärenz der Element-Anordnung zwischen einzelnen Bildzeilen zu nutzen, stellen sich für Volume Rendering Anwendungen als langsamste Gruppe heraus. Der Aufwand, *alle* Schnittpunkte eines Pixelstrahls mit den Volumenelementen zu bestimmen ist eben sehr viel höher, als nur das oberste Element zu bestimmen.

Für die Berechnung der Schnittpunkte und die Traversierung des Strahls sind bei diesen Verfahren zwei verschiedene Iterationen nötig, was Zeit kostet. Scanline Verfahren nutzen bei der Schnittpunktberechnung auch keinerlei Nachbarschaftsinformation, wie z.B. die *cell-to-cell* Raycaster. Das macht sie sehr universell einsetzbar, aber eben auch langsamer.

Es gibt ganz verschiedene Arten der Geschwindigkeitsbeschleunigung (z.B. Kohärenznutzung, genäherte Integration, adaptive Wahl des *Sample*-Abstandes, Volumenausdünnung bzw. *subsampling*, Parallelisierung) Die Projektionsverfahren sind am vielfältigsten, was den Einsatz verschiedener Approximationsverfahren angeht. Das spiegelt auch die tabellarische Übersicht wieder: Bei den Projektionsansätzen liegt eine große Bandbreite von Verfahren verschiedener Geschwindigkeit vor, während die Raycasting Ansätze generell im mittleren bis hinteren Geschwindigkeitsbereich angesiedelt sind.

Die Geschwindigkeitssteigerung durch Parallelisierung der Verfahren ist im Grunde ein Kapitel für sich und kann hier nur am Rande erwähnt werden. Raycasting- und Scanlineverfahren sind grundsätzlich erst einmal intuitiver zu parallelisieren als Projektionsverfahren, indem man die Aufteilung der Arbeit für die Prozessoren über den Bildraum vornimmt. Jeder Prozessor bekommt ein z.B. rechteckiges Teilgebiet des Ergebnisbildes zur Bearbeitung zugewiesen. Die Scanlineverfahren scheinen sehr günstig mit der Anzahl der Prozessoren zu skalieren ([Cha93]).

Wenn man bei den Projektionsverfahren eine Parallelisierung über den Objektraum vornimmt, muß die Reihenfolge der Teilaufgaben koordiniert werden (BTF oder FTB-Ordnung). Es ist allerdings auch möglich, die Projektionsverfahren ebenfalls über dem Bildraum zu parallelisieren. [Luc92] schlägt ein zweiphasiges Vorgehen vor, bei dem zunächst über den Objektraum und dann über den Bildraum parallelisiert wird.

**Beleuchtung** Ein einfaches Emissions-Absorptionsmodell zur Modellierung der Lichtverhältnisse im Datenvolumen wird von allen Verfahrensklassen gleichermaßen gut unterstützt. Die Unterschiede liegen eher in Randbereichen:

Mit Raycasting Verfahren ist man am flexibelsten, wenn man kompliziertere Lichttransfer-Modelle verwenden will. Angenommen man sieht äußere Lichtquellen vor und will Einfachstreuung mit *self shadowing* modellieren. Dazu muß man berücksichtigen, daß das Licht auf dem Weg zu dem jeweiligen Volumenpunkt, an dem die zum Auge reflektierte Intensität bestimmt werden soll, abgeschwächt wird. Somit ist es notwendig, von diesem Volumenpunkt aus einen Strahl zu jeder Lichtquelle zu senden und die Dichte entlang dieses Strahles aufzuintegrieren. Bei den Raycastingverfahren läßt sich das im Gegensatz zu Scanline- und Projektionsverfahren mit leichter Abänderung der bereits für die Volumentraversierung eingesetzten Strahlverfolgungsmethoden durchführen.

Raycasting und Scanlineverfahren lassen sich grundsätzlich einfacher mit *Raytracing* polygonal definierter Szenerien kombinieren als Projektionsverfahren. Für diese existieren allerdings eigene Verfahren zur Berücksichtigung polygonal definierter Oberflächen (z.B. [MHC90]).

Auch für das *Mapping* der Datenwerte auf Farb- und Transparenzwerte existieren Möglichkeiten, die sich besser mit dem Raycasting- oder Scanlineansatz verwirklichen lassen: Sabella ([Sab88]) schlägt beispielsweise eine Einfärbung des Pixels nach dem

Maximaldatenwert entlang eines Pixelstrahls vor, einer Größe, die bei den Strahlverfolgungsverfahren leicht zugänglich ist, bei Projektionsverfahren hingegen nicht ohne größeren Aufwand bestimmt werden kann (vgl. Fußnote 8, S. 19).

Allgemein ist es ein sehr schwieriger und zeitintensiver Vorgang, eine günstige *Mapping*-Einstellung zu finden. Van Gelder und Wilhelms ([VGW93]) schlagen zum schnelleren Austesten von Transferfunktionen für ihr Projektionsverfahren eine sogenannte *band option* vor. Der Benutzer gibt ein Datenintervall vor und Zellen mit Werten außerhalb dieses Intervalls werden einfach ausmaskiert und bei der Bildberechnung nicht berücksichtigt. Der Benutzer bekommt die Auswirkungen seiner Transferfunktion auf dem ausgewählten Datenbereich nun sehr viel schneller präsentiert. Solche und ähnliche (temporären) Datenreduktionen lassen sich sehr viel einfacher mit Projektionsverfahren als mit Raycasting- oder Scanlineverfahren realisieren.

**Flexibilität** Der letztgenannte Punkt — einfache Datenreduktion — fällt eigentlich schon in die Rubrik *Flexibilität*. Beim *Clipping* uninteressanter Gebiete ist der Projektionsansatz am flexibelsten. Auch was das Abwägen zwischen Geschwindigkeit und Genauigkeit angeht, bilden Projektionsverfahren auf irregulären Gittern eindeutig die flexibelste Gruppe (vgl. Abschnitte *Integrationsmethoden* und *Geschwindigkeit*).

Bei der Wahl der behandelbaren Gitter haben, wie bereits beschrieben, die Scanline-Algorithmen die Nase vorn. Raycasting-Methoden sind am flexibelsten, was die einsetzbaren Beleuchtungsstrategien angeht.

Generell geht der Trend dahin, Visualisierungsverfahren *adaptiver* zu gestalten. Sie sollen schneller als bisher auf benutzerdefinierte Änderungen (z.B. Blickrichtung, Transferfunktion) reagieren können und z.B. eine parametergesteuerte Kontrolle von Geschwindigkeit und Bildqualität ermöglichen.

Ein erster Schritt in die Richtung schnellerer Reaktion auf Benutzereingriffe sind mehrphasige Algorithmen, in denen die Berechnungsschritte für oft zu ändernde Aktionen (z.B. *Mapping* der Datenwerte) in einzelnen Phasen isoliert werden, so daß die Ergebnisse der vorher abgelaufenen Phasen nicht noch einmal neu berechnet werden müssen ([Luc92],[Cha93]).

**Datenstrukturen, Speicheranforderungen** Grundsätzlich kann man an vielen Stellen des Volume Rendering Vorgangs Rechenaufwand einsparen, indem man Teilergebnisse zwischenspeichert oder später benötigte Größen in einem *Preprocessing*-Schritt bestimmt und abspeichert. Die Speicherplatzanforderungen sind im Fall irregulärer Gitter allgemein noch relativ bescheiden (im Vergleich mit z.B. medizinischen Datensätzen aus der Computer-Tomographie). Dies hat seine Ursache darin, daß die Simulationsrechnungen, die auf irregulären Gittern arbeiten, bisher aus Gründen des Rechenaufwandes keine größeren Datensätze verarbeiten können und daß außerdem durch die unregelmäßige Struktur der Gitter schon mit weniger Zellen eine sehr genaue Modellierung vorgenommen werden kann.

Wenn Raycasting- oder Scanlineverfahren beschleunigt werden sollen, indem z.B. die Ebenengleichungen oder Interpolationskoeffizienten pro Zelle abgespeichert werden, dann ist ein deutlich größerer Speicheraufwand als bei den Projektionsverfahren notwendig.

**Vorsortierung der Zellen** An dieser Stelle soll noch einmal auf die bei Projektionsverfahren nötige Vorsortierung der Volumenzellen eingegangen werden. Es ist ein Nachteil, daß für jede Änderung der Blickrichtung oder der Perspektive eine Neusortierung vorgenommen werden muß. Andererseits existieren zumindest für zusammenhängende Tetraedergitter sehr schnelle Sortierverfahren ([Wil92b]), so daß dieser Aufwand im Vergleich zu den Bildberechnungszeiten kaum eine Rolle spielt.

Die Notwendigkeit der Vorsortierung kann durch die Wahl trivialer Beleuchtungsmodelle (vgl. Fußnote 9, S. 22) oder auch durch den Einsatz eines sogenannten *Alpha-Buffers* vermieden werden. Bei letzterem Ansatz werden alle Integrationsbeiträge der Volumenzellen nach ihrer z-Koordinate in einem globalen Buffer zwischengespeichert, anstatt daß ein direktes *Compositing* durchgeführt wird. Dadurch handelt man sich allerdings einen Geschwindigkeitsverlust durch eine zusätzliche Traversierung dieses Buffers für jeden Pixel, ähnlich wie bei den Scanline Verfahren, ein.

Schließlich soll nicht unerwähnt bleiben, daß der Vorgang der Bildberechnung bei den Projektionsverfahren bei einer *Back-To-Front* Sortierung einen erstaunlich guten Einblick in die Struktur der Datenverteilung ermöglicht. Dadurch daß sich das Bild von hinten nach vorne aufbaut, bekommt man die Strukturen im Innern der Szenerie gut vor Augen geführt. Eine solche „Animation“ kombiniert die Vorteile des Volume Renderings und einer Schnittflächenanimation.

Vom Aufwand her ist eine *Front-To-Back* Sortierung (bei Raycastingverfahren entsprechend eine *Front-To-Back* Traversierung des Strahls) günstiger, weil man die Integration abbrechen kann, wenn die akkumulierte Transparenz nahezu Undurchsichtigkeit erreicht hat.

## 2.3 Forschungsbedarf

In diesem Kapitel wurde ein ausführlicher Überblick über den derzeitigen Stand der Forschung im Bereich des *Volume Rendering* auf irregulären Gittern gegeben. Vieles bleibt noch zu verbessern, damit sich *Volume Rendering* im Bereich der wissenschaftlichen Visualisierung endgültig als Standardverfahren etabliert. An dieser Stelle sollen abschließend ein paar Bereiche aufgezählt werden, in denen Verbesserungen und Erweiterungen wünschenswert erscheinen.

- Die Visualisierung von höherdimensionalen Daten ist bisher erst in Ansätzen von Volume Rendering Methoden unterstützt worden. Dargestellt wurde ein Verfahren zur Visualisierung eines Strömungsfeldes, das die Ausbreitung von an beliebiger Stelle in dem Feld ausgestoßenem Rauch über die Zeit simulierte.

  Die Stärke von *Volume Rendering* liegt darin, auch in Standbildern einen aussagekräftigen Überblick eines ganzen Datenvolumens geben zu können. Um Vektorfelder in diesem Sinne direkt darzustellen, müssen geeignete Methoden gefunden werden, die Richtungsinformation an jedem Volumenpunkt abzubilden, ohne die Transparenz einzuschränken und ohne das Bild unübersichtlich werden zu lassen.

- Die Einbindung von Texturen (bzw. *Hypertextures*) in die verwendeten Beleuchtungsmodelle könnte bei der Darstellung dieser gerichteten Information eine Rolle

spielen. Texturen könnten auch zur Visualisierung eines lokalen Fehlers eingesetzt werden, wenn solche Information mit den Simulationsergebnissen mitgeliefert wird.

- Eine große Bandbreite von interaktiven und ungenauen bis hin zu rechenintensiven und sehr genauen Darstellungsverfahren ist sehr sinnvoll, um einen unbekannten Datensatz erst einmal überblicksartig erkunden zu können und dann genauere Bilder von interessanten Regionen zu berechnen. Interessant wäre es, ein *kontinuierliches* Abwägen zwischen diesen beiden Extremen zu erlauben und die Kontrolle über benutzerbestimmte Parameter (z.B. maximal zulässiger Fehler) zu gestalten.

- Hierzu wären gerade für den Fall irregulärer Gitter neue Methoden zur Erstellung von *reduced resolution meshes* wünschenswert, so daß eine kontrollierte Vergröberung von Gitterbereichen stattfinden könnte.

- Wie bereits im vorangegangenen Abschnitt unter dem Punkt *Flexibilität* erwähnt, müssen Methoden gefunden werden, die schneller auf bestimmte Benutzeraktionen (z.B. Änderung der Transferfunktion, neuer Blickwinkel) reagieren. Dies könnte durch Algorithmen geschehen, die eine komplette Neuberechnung in solchen Fällen vermeiden und statt dessen eine Art *Postoptimierung* der bereits berechneten Ergebnisse vornehmen.

- Volume Rendering gibt einen qualitativen Gesamteindruck von einer Szenerie. Es wäre sinnvoll, dieses Verfahren mit eher quantitativ ausgerichteten Methoden zu kombinieren (z.B. interaktive quantitative Wertebestimmung mit 3-D-Cursor). Dazu gehört auch, den Einfluß verschiedener Transferfunktionen deutlicher zu machen, z.B. durch Integrierung aussagekräftiger Legenden ins Resultatbild.

# Kapitel 3

# Anforderungsdefinition

Im Rahmen dieser Arbeit ist ein umfangreiches Programmsystem (VoRANG) zum Volume Rendering auf nichtstrukturierten Gittern entstanden.

In diesem Kapitel werden kurz die Anforderungen an das System VoRANG beschrieben. Es wird auf die Art der Eingangsdaten eingegangen, die Anforderungen in bezug auf Funktionalität, Geschwindigkeit und Steuerung werden aufgeführt und es wird verdeutlicht, wo die Prioritäten liegen.

Diese Anforderungsdefinition resultiert in Entwurfsentscheidungen für das Programmsystem, die im folgenden Kapitel diskutiert werden.

## 3.1 Charakterisierung der Eingangsdaten

Im Bereich *Scientific Visualization* stellt sich in zunehmendem Maße die Forderung nach einer adäquaten Darstellung von 3-D Skalar- und Vektorfeldern, die auf irregulären Gittern definiert sind. Am Konrad-Zuse-Zentrum Berlin werden z.B. adaptive numerische Algorithmen entwickelt, die Simulationsrechnungen auf *unstrukturierten Gittern* ausführen. Die Zellenform der Datengitter ist zur Zeit auf *Tetraeder* festgelegt, es ist jedoch vorstellbar, daß die Simulationsrechnungen auch auf andere Zelltypen ausgedehnt werden.

Das entstehende Programmsystem VoRANG soll u.a. zur Visualisierung von Ergebnissen solcher Simulationen eingesetzt werden.

Ein typisches Datengitter besteht aus etwa 5.000-100.000 Tetraedern. Die Daten sind wahlweise auf den Knotenpunkten, den Kanten, den Flächen oder den Zellen selbst definiert. Die Interpolation der Datenwerte über die Tetraeder kann sowohl linear als auch mit Methoden höherer Ordnung erfolgen.

Die Art der darzustellenden Daten ist eng an den jeweiligen Anwendungsfall gebunden. Ein typischer Anwendungsfall für eine Simulationsrechnung ist die Hyperthermieplanung[1]. Auf einem Tetraedergitter, das so gut wie möglich die betroffene Körperregion eines bestimmten Patienten approximiert und an den Grenzflächen zwischen verschiedenen Gewebetypen wesentlich feiner aufgelöst ist als inmitten von Gewebe-

[1]Hyperthermie ist eine Krebstherapie, in der Tumoren durch lokale Überhitzung von betroffenem Körpergewebe bekämpft werden. Meist wird diese Therapie in Kombination mit Chemo- oder Strahlentherapie eingesetzt.

kompartimenten, wird eine Hyperthermiebehandlung simuliert. Hohe Temperaturen im malignen Gewebe werden durch Einstrahlung und geeignete Überlagerung von Radiowellen erreicht. Im Laufe der Simulation soll die Ausrichtung der einstrahlenden Antennen optimiert werden. Zu diesem Zwecke müssen sowohl die elektromagnetischen Felder, als auch die absorbierte Leistung und die resultierende Temperatur berechnet und visualisiert werden.

Die Darstellung des skalaren Temperaturfeldes wäre ein typischer Anwendungfall für das Volume Rendering Verfahren.

## 3.2 Allgemeine Anforderungen

### Funktionalität

Die Farbe und Transparenz des zu visualisierenden Datenvolumens soll frei wählbar sein. Das Programmsystem soll auch dazu verwendet werden, mit verschiedenen Beleuchtungsstrategien zu experimentieren, daher soll das System in diesem Punkt leicht erweiterbar sein.

Die Datenverteilung soll aus beliebigen Blickrichtungen dargestellt werden können. Die Möglichkeit, scriptgesteuert Animationen berechnen zu lassen, ist wünschenswert. Es soll wahlweise ein perspektivischer oder orthogonaler Blick auf das Volumen möglich sein. Der besseren Orientierung halber soll das zugrunde liegende Gitter wahlweise mit dargestellt werden können.

Es sollen Bilder in beliebiger Größe berechnet werden können.

Das gesamte System soll so Hardware-unnabhängig wie möglich gehalten werden. Zumindest soll es auf den SUN und Silicon Graphics Workstations an der TU Berlin und am Konrad Zuse Zentrum lauffähig sein.

### Geschwindigkeit

Wie beschrieben, soll das Programm hauptsächlich zur Visualisierung von Simulationsergebnissen eingesetzt werden. Das schließt auch zeitkritische Simulationen ein. Gleichzeitig ist es hingegen besonders wichtig, eine wissenschaftlich exakte Visualisierung der Daten zu ermöglichen. Daher soll — in gewissen Grenzen — ein benutzerbestimmtes Abwägen zwischen Geschwindigkeit und Bildgenauigkeit möglich sein.

### Steuerung

Die Eingabe der Datengitter und der Beleuchtungsparameter (*mapping functions*) soll über Dateien mit entsprechend zu entwickelnden Dateiformaten erfolgen. Eine Script-Steuerung des gesamten Bildberechnungsvorganges ist wünschenswert, da über eine solche Schnittstelle eine einfache Kopplung mit anderen Programmen (z.B. für die numerischen Simulationen) vorgenommen werden kann.

## 3.3 Prioritäten

Bezüglich der Gittertypen liegt trotz der möglichen Erweiterung auf unstrukturierte Gitter beliebiger Zellenform die Priorität eindeutig auf *Tetraedergittern*. Auch wenn die zur Zeit in den Simulationsrechnungen verwendeten Gitter eine durchschnittliche Größe von nur etwa 20.000-30.000 Zellen haben, sollte das Volume Rendering Programm auf eine Größenordnung von etwa 150.000 Zellen ausgelegt sein, da die Größe der Datensätze mit steigender Rechenleistung (schnellerer Hardware) stetig zunimmt.

Der eindeutige Schwerpunkt in der Wahl zwischen Geschwindigkeit und Bildqualität ist auf eine wissenschaftlich exakte Bildberechnung zu legen. Das Programm sollte in der Lage sein, die Genauigkeit der Ergebnisse aus den Simulationsrechnungen direkt für die Visualisierung zu übernehmen.

*Volume Rendering* Bilder eignen sich sehr gut zur Veröffentlichung von Visualisierungsergebnissen, da auf wenig Raum ein hoher Informationsgehalt vermittelt werden kann. Daher ist es wichtig, Bilder in hoher Auflösung (Druck- bzw. Diaauflösung) berechnen zu können.

Was die Programmstruktur angeht, so ist eine leichte Erweiterbarkeit sehr wichtig, da das System auch als *Testbed* für verschiedene neu zu entwickelnde Darstellungsverfahren eingesetzt werden soll. Daher ist einer sauberen Programmierschnittstelle absoluter Vorrang gegenüber komfortabler Benutzerführung einzuräumen.

# Kapitel 4

# Entwurfsentscheidungen für das System VoRANG

Im folgenden werden auf die im vorangehenden Kapitel beschriebenen Anforderungen hin grundsätzliche Entwurfsentscheidungen für das Programmsystem VoRANG getroffen. Bei der Wahl des geeigneten Verfahrens zum *Volume Rendering auf nichtstrukturierten Gittern* wird auf die in Kapitel zwei gewonnenen Einsichten über Vor- und Nachteile verschiedener Ansätze zurückgegriffen und ein eigener Ansatz kreiert. Der Funktionsumfang des Systems wird beschrieben, wobei auch auf die Art der Programmsteuerung eingegangen wird. Es wird ein Eingabeformat für die Datengitter diskutiert und eine allgemeine Repräsentierung von Datenverteilungen vorgeschlagen. Die Strategie zur Bestimmung der Abbildungsfunktionen von Daten- auf Farb- und Transparenzwerte wird festgelegt und ein Dateiformat hierfür entwickelt. Schließlich werden verschiedene Approximationsverfahren für die Bildberechnung vorgeschlagen, die nebeneinander implementiert werden sollen, um ein Abwägen zwischen Geschwindigkeit und Bildqualität zu erlauben. Das Kapitel endet mit einer Spezifikation der durchzuführenden Tests.

## 4.1 Wahl des Volume Rendering Verfahrens

Der vorgegebene Gittertyp der *unstrukturierten Tetraedergitter* stellt keine außergewöhnlichen Anforderungen an das Volume Rendering Verfahren. Sowohl Raycasting- als auch Scanline- als auch Projektionsverfahren lassen sich grundsätzlich gut auf diese einfachsten unstrukturierten Gitter abstimmen. Ein Projektionsverfahren kann speziellen Nutzen aus dem Einsatz solcher Gitter ziehen, indem eine schnelle Tetraederprojektion nach dem Ansatz von [ST90] vorgenommen wird. Außerdem existiert ein schneller und effektiver Algorithmus für die Sortierung solcher Zellen ([Wil92b]). Zukünftige Erweiterungen auf andere Zellentypen können mit allen Verfahrensarten behandelt werden. Wenn bei den Projektionsverfahren die Tetraederstruktur aber erst einmal speziell zur Geschwindigkeitssteigerung genutzt wurde, ist es etwas schwieriger, als bei den Scanline- und Raycastingverfahren, eine solche Umstellung vorzunehmen.

Für die Implementierung eines Raycasting-Verfahrens spricht die etwas größere Flexibilität einer solchen Methode bei der Wahl von komplizierteren Beleuchtungsmodellen

(vgl. Abschnitt *Beleuchtung*, S. 40).

Die Anforderungsdefinition (Kapitel 3) legt ein Verfahren nahe, das flexibel zwischen verschiedenen Geschwindigkeitsstufen umschalten kann.

Scanlinemethoden scheiden aufgrund ihrer geringen Geschwindigkeit von vornherein als mögliches Verfahren aus. Die Vorteile dieser Verfahrensart — Behandlung beliebiger Gitter und gute Parallelisierungsmöglichkeiten — sind für die hier vorliegenden Anforderungen nicht von hoher Relevanz.

Es sollen Tetraedergitter in der Größenordnung von 100.000 Zellen bearbeitet werden. Ein durchschnittlich schnelles Raycasting Verfahren benötigt für solche Datensätze bei großen Ergebnisbildern (z.B. 1024x1024 Pixel) Rechenzeit von der Größenordnung mehrerer Stunden. Bei Anwendung von verschiedenen Beschleunigungsverfahren und Beschränkung der Ergebnisbildgröße kann diese Zeit zur Erzeugung von Testbildern eventuell auf 10 bis 20 Minuten gesenkt werden. Dieser Zeitaufwand ist, wenn man die Schwierigkeiten berücksichtigt, geeignete Parameter für die Bildberechnung zu finden, kaum tragbar für ein System, das den Anspruch hat, u.a. auch als Testumgebung für verschiedene neue Darstellungsmethoden (z.B. Beleuchtungsverfahren) zu dienen.

Bei Projektionsverfahren ist die Bandbreite zwischen schnellen Approximationen und genauer Bildberechnung wesentlich höher als bei den Raycasting Verfahren (vgl. Abschnitt *Geschwindigkeit*, S. 39). Noch dazu sind die Geschwindigkeitssteigerungen bei diesen Verfahren durch immer stärkere Approximationen in *einem* einzigen Bereich erzielbar, nämlich bei der Integration der Intensitäts- und Dichtebeiträge für die einzelnen Pixel der Zellenprojektion. Dies gewährleistet einen direkten Vergleich der einzelnen Approximationsstufen miteinander und mithin die Anordnung in einem einheitlichen Schema (innerhalb *eines* Programmoduls).

Das Verhältnis zwischen der Größe der Datensätze und der Bildgröße spricht im vorliegenden Anwendungsfall auch für die Projektionsverfahren. Je größer die von der Projektion *einer* Zelle bedeckte Fläche ist, desto mehr Nutzen kann der Algorithmus aus der Kohärenz innerhalb der Volumenzellen ziehen. Bei einer typischen Datensatzgröße von 100.000 Zellen, einer Bildgröße von 1024x1024 Pixeln und einer Bildausfüllung von z.B. 80% projiziert eine Volumenzelle durchschnittlich auf $\left(\frac{\sqrt{0.8}\cdot 1024}{\sqrt[3]{100.000}}\right)^2 \approx 20^2$ Pixel. Selbst bei einer Datensatzgröße von 150.000 Zellen und einer Bildgröße von nur 256x256 Pixeln würde die Projektion einer Zelle noch ungefähr 20 Pixel ausmachen.

Unter Einbeziehung all dieser Informationen wird die Entscheidung für die Implementierung eines *Projektionsverfahrens* getroffen. Es wird ein abgestuftes System von verschiedenen Approximationsverfahren für die Berechnung der Farb- und Dichtebeiträge der einzelnen Zellen entwickelt (s. Abschnitt 6.3, S. 64). Dabei wird auch die Tatsache genutzt, daß die zu visualisierenden Daten auf Tetraedergittern definiert sind. Es wird besonderer Wert auf Allgemeinheit und einfache Erweiterbarkeit speziell in folgenden Bereichen gelegt:

- *Behandelbare Datenverteilungen.* Die Dimension der Daten und die Art der Interpolation über den Zellen ist variabel. Die Daten können auf verschiedenen geometrischen Elementen (Punkten, Kanten, Flächen oder Zellen) definiert sein.

- *Beleuchtungsmethoden.* Als grundlegendes Modell für die Wahl der Farb- und Dichtekoeffizienten wird ein Emissions-Absorptions-Ansatz gewählt. Äußere Licht-

quellen sollen berücksichtigt werden können. Die Abbildungsfunktionen von Datenauf Farb- und Dichtewerte können vom Ort im Volumen und vom dort anliegenden Gradienten abhängig sein.

- *Bildberechnungsmethoden.* Das System soll in einfacher Weise um neue Approximationsverfahren für die Bildberechnung erweitert werden können.

## 4.2 Design und Funktionsumfang

Eine klar definierte Programmierschnittstelle und einfache Erweiterungsmöglichkeiten für das System, wie sie in der Anforderungsdefinition gefordert wurden, lassen sich am einfachsten mit einem objektorientierten Systementwurf realisieren. Das System VoRANG wird in diesem Sinne als Klassenbibliothek in C++ implementiert. Es wird keine bereits bestehende Grafikbibliothek verwendet, sondern eine eigene *Rendering Pipeline* implementiert. Das garantiert größtmögliche Flexibilität für die implementierten Algorithmen, da an jeder Stelle der *Pipeline* direkt eingegriffen werden kann. Weiterhin garantiert dieser Ansatz die geforderte Hardware-Unabhängigkeit. Dafür müssen eventuell Abstriche in der Geschwindigkeit gemacht werden.

Die Anforderungsdefinition legt weiterhin nahe, die Bedienung des *Volume Rendering* Programms über eine Skriptsprache vorzunehmen. Mittels einer solchen Schnittstelle können einfach und effizient animierte Bildsequenzen von Datenverteilungen erstellt werden und es kann eine Kopplung der *Volume Rendering* Umgebung mit anderen Programmen vorgenommen werden, indem diese einfach eine Skriptdatei erzeugen und dem System VoRANG zur Abarbeitung übergeben.

Eine mächtige universell einsetzbare und erweiterbare Skriptsprache ist *Tcl*. *Tcl* wurde Ende der 80er Jahre von John Ousterhout als wiederverwendbare Kommandosprache entwickelt und hat in den letzten Jahren zusammen mit dem X11 Toolkit *Tk* weite Verbreitung gefunden. *Tcl* stellt mächtige Sprachkonstrukte wie Steuervariablen und Schleifen zur Verfügung. Für eine ausführliche Einführung in *Tcl* sowie eine komplette Befehlsübersicht siehe [Ous94]. Die *Tcl* Sprache wird frei vertrieben und steht auf den für VoRANG relevanten Hardware-Plattformen zur Verfügung.

Für das Projekt VoRANG wird eine *Tcl*-basierte Skriptsprache entworfen. Das heißt, alle Funktionalität des Programms wird von Kommandos gesteuert, die als Erweiterung des *Tcl*-Kernels frei mit den *Tcl*-eigenen Sprachkonstrukten kombinierbar sind. Gleichzeitig existiert natürlich auch eine Programmierschnittstelle, d.h. jedes VoRANG-Kommando der Skriptsprache findet seine Entsprechung in einer Methode der äußersten *Volume Renderer*-Klasse im objektorientierten Systementwurf.

Die Funktionalität des Programmsystems VoRANG läßt sich in sechs verschiedene Gruppen einteilen: Bilddefinition, Datendefinition, Beleuchtung, Kameraeinstellungen, Transformationen und Berechnungsmethoden. Eine komplette Übersicht über alle Kommandos der implementierten Skriptsprache wird in Anhang A ab S. 85 gegeben. Im folgenden werden für die angesprochenen Gruppen die wichtigsten Entwurfsentscheidungen aufgeführt:

### Bilddefinition

Die Bildberechnung erfolgt vollkommen unabhängig von den Darstellungsmöglichkeiten der jeweiligen Hardware. Ein Bild wird in einem selbstdefinierten *Framebuffer* berechnet und nach Fertigstellung als Datei in beliebigem Bildformat (s.u.) abgespeichert. Bei der Bildberechnung wird *Gleitkommapräzision* zugrunde gelegt, d.h. jeder Bildpunkt wird als Quadrupel von `float`-Werten (r,g,b,alpha) repräsentiert. Durch die erhöhte Genauigkeit gegenüber einem Framebuffer mit einer Tiefe von nur 24 Bit können Artefakte beim *Compositing* vermieden werden.

Es können ca. 30 der gebräuchlichsten Bildformate erzeugt werden (s. Anhang A.2, S. 90). Dies wird unter Einsatz einer frei vertriebenen Bibliothek, der *San Diego Image Library* realisiert.

Zur einfachen Erstellung von Animationssequenzen wird ein Konzept der durchgehenden Numerierung berechneter Bilder angeboten.

Es können Bilder beliebiger Auflösung berechnet werden. Die Bildgröße ist dabei nur durch die Speicherkapazität der jeweiligen Hardware begrenzt.

Dem berechneten Bild können beliebige Hintergrundbilder unterlegt werden. Ein regelmäßiges Gittermuster als Hintergrund kann z.B. die im Bild auftretenden Transparenzen sehr viel deutlicher machen. Beim Einlesen dieser Hintergrundilder werden wiederum die verschiedenen bereits angesprochenen Bildformate unterstützt.

### Datendefinition

In der gegenwärtigen Programmversion werden ausschließlich Datenverteilungen auf Tetraedergittern verarbeitet. Das Datenformat und seine Implikationen werden im Abschnitt 6.1 ab S.60 behandelt.

Es können beliebig viele Datenverteilungen für ein Gitter gleichzeitig im Speicher gehalten werden und auch gleichzeitig in einer Bildberechnung berücksichtigt werden, wenn entsprechende *Mapping*-Verfahren definiert wurden, die aus der Kombination mehrerer Datenverteilungen Dichte- und Farbwerte ableiten.

### Beleuchtung

Das Beleuchtungskonzept beruht auf einem verallgemeinerten *Emissions-Absorptions-Modell*, wie es zu Anfang von Kapitel 2 eingeführt wurde.

Vor der Bilderzeugung muß der Benutzer Dateien zum *Color-* und *Alpha-Mapping* spezifizieren oder kann auf die ganz einfachen Defaultmethoden zurückgreifen, die in Abschnitt A.1, S. 87 beschrieben werden.

Die Strategie zur Abbildung der Datenwerte auf Farben und Transparenzen wird in Abschnitt 6.2 genauer erläutert. Dort werden auch die Dateiformate vorgestellt, über die die Beleuchtungsberechnungen gesteuert werden.

Es ist Funktionalität dafür vorgesehen, beliebig positionierte (parallelstrahlige) Lichtquellen zu definieren. Bisher existieren aber noch keine Abbildungsfunktionen, die Gebrauch von externen Lichtquellen machen.

### Kameraeinstellungen und Transformationen

Zur Bestimmung eines geeigneten Bildausschnittes kann der Benutzer an dreierlei Stellen Einstellungen vornehmen:

- Er kann die Kameraprojektion neu definieren. Sowohl orthogonale als auch perspektivische Projektion wird unterstützt.
- Er kann die Kamera auf eine neue Position bringen und in beliebige Richtung schwenken lassen.
- Er kann durch Angabe von Skalierungen, Translationen und Rotationen die Ausrichtung des *Datenvolumens* iterativ verändern. Dabei werden die Rotationen auf das jeweilige lokale Koordinatensystem des Datensatzes bezogen. Mit dieser Methode sind Animationen ohne Definition von Variablen über eine einfache Schleife realisierbar.

Die drei Einstellungen sind in dem Sinne unabhängig voneinander, daß die resultierenden Matrixoperationen in jeweils eigenen Matrizen festgehalten werden, die erst zu Beginn der Bildberechnung geeignet zu einer resultierenden Matrix zusammengefaßt werden.

### Berechnungsmethoden

Es wurde bereits dargestellt, daß ein Projektionsansatz die verschiedenen Anforderungen an das System VoRANG am geeignetsten erfüllen kann. Eine wesentliche Anforderung ist die Realisierung von verschiedenen Bildberechnungsmethoden, die durch geeignete Approximationsverfahren ein abgestuftes Abwägen zwischen Qualität und Geschwindigkeit der Bildberechnung erlauben.

Als geeigneter Teilalgorithmus für die Ansiedlung solcher Näherungsverfahren bietet sich im Rahmen der Projektionsverfahren die Integration der Intensitäts- und Dichtebeiträge für die einzelnen Pixel der Zellenprojektion an. Es werden vier verschiedene Approximationsstufen für diesen Teilalgorithmus realisiert. Diese werden in Abschnitt 6.3, S. 64 genau beschrieben. Es wird eine einheitliche Programmierschnittstelle definiert, so daß das Programm auf einfache Art und Weise um neue Näherungsverfahren ergänzt werden kann.

Die einzelnen Methoden werden über Kommandos der Skriptsprache angewählt, wobei ihnen eventuelle Parameter übergeben werden können.

Der implementierte Projektionsalgorithmus arbeitet im Prinzip sowohl mit einer *Back-To-Front* wie auch mit einer *Front-To-Back* Sortierung der Volumenzellen. In der aktuellen Programmversion ist allerdings nur der *Back-To-Front* Ansatz komplett implementiert worden.

## 4.3 Spezifikation der Testläufe

Durch geeignete Testläufe soll die Funktion und Effektivität der im Programm VoRANG implementierten Methoden gewährleistet werden. Folgende Tests sind durchzuführen:

- Unter Einsatz eines geeigneten Testdatensatzes ist ein Vergleich der Ergebnisse durchzuführen, die mit Hilfe der verschiedenen Approximationsverfahren erzielt werden können. Sichtbare Artefakte sollen belegt und gedeutet werden.

- Die Geschwindigkeit der verschiedenen Approximationsverfahren soll an einem geeigneten Beispiel dokumentiert werden.

- Es sind Visualisierungen mit verschiedenen *mapping functions* bei genauester Berechnungsmethode durchzuführen. Auftretende Artefakte sollen erklärt werden.

- Es ist eine Visualisierung für einen Anwendungsfall aus der Praxis zu erstellen (z.B. Temperaturverteilung über dem Körper eines Hyperthermiepatienten)

- Es soll eine Visualisierung vorgenommen werden, die die Flexibilität des Programmsystems in bezug auf verschiedene höherwertige Interpolationsverfahren oder höherdimensionale Datensätze oder komplexere Beleuchtungsstrategien verdeutlicht.

# Kapitel 5

# Systementwurf

Ein Softwareentwicklungsprozeß setzt sich aus drei Stufen zusammen:

- **Analyse**: Die funktionale Untersuchung der Problemstellung,
- **Design**: Die Erstellung einer geeigneten Gesamtstruktur für das System,
- **Implementierung**: Programmierung und Testen des Programmcodes.

Wichtig ist, daß dieser Ablauf nicht starr sondern iterativ vor sich geht und durch dynamische Konzepte wie *Prototyping*, *Experimentieren* und *Design-Analysen* ergänzt und durch *Dokumentation* und *Management* kontrolliert wird.

Ziel dieses Abschnitts ist die Erstellung eines objektorientierten Softwareentwurfes für das Projekt *VoRANG*. D.h. hier werden die ersten beiden Punkte, *Analyse* und *Design* adressiert.

Nachdem im vorangehenden Kapitel durch die Entwurfsentscheidungen deutlich gemacht wurde, mit welchen Methoden ein *Volume Rendering* Bild von einem irregulär definierten Datensatz zu erstellen ist, soll nun die Programmstruktur festgelegt werden, die dieses Vorgehen in Software umsetzt.

Der Entwurf soll die Vorteile objektorientierter Programmierung nutzen. Insbesondere wird auf strukturierten, verständlichen Aufbau, Wiederverwendbarkeit von Komponenten und auf einfache Erweiterbarkeit Wert gelegt.

Um zu einer in diesem Sinne funktionalen Klassenhierarchie zu gelangen, wird das System zunächst mittels *Datenfluß-* und *Funktionsablaufdiagrammen* skizziert (Analyse). Allgemeine und konkrete Ziele, die beim Entwurf Beachtung finden sollen, wurden bereits in Kapitel 3 in einer Anforderungsliste fixiert. Hier wird nun begründet eine Klassenhierarchie für das Programm erstellt und in *Vererbungs-* und *Abhängigkeitsdiagrammen* dokumentiert (Design).

## 5.1 Analyse

Die Problemstellung läßt sich folgendermaßen umreißen: Es wird eine grafische Szenerie aufgebaut, die aus den zu visualisierenden Daten, ihren (bezüglich des bildgebenden Modells definierten) Eigenschaften und aus datenunabhängigen Umgebungseigenschaften besteht. Diese Szenerie wird benutzergesteuert zu einem Bild verarbeitet.

Insgesamt ergeben sich also die konzeptionellen Aufgaben *Datenweltaufbau*, *Bildberechnung* und *Steuerung*.

### 5.1.1 Datenfluß

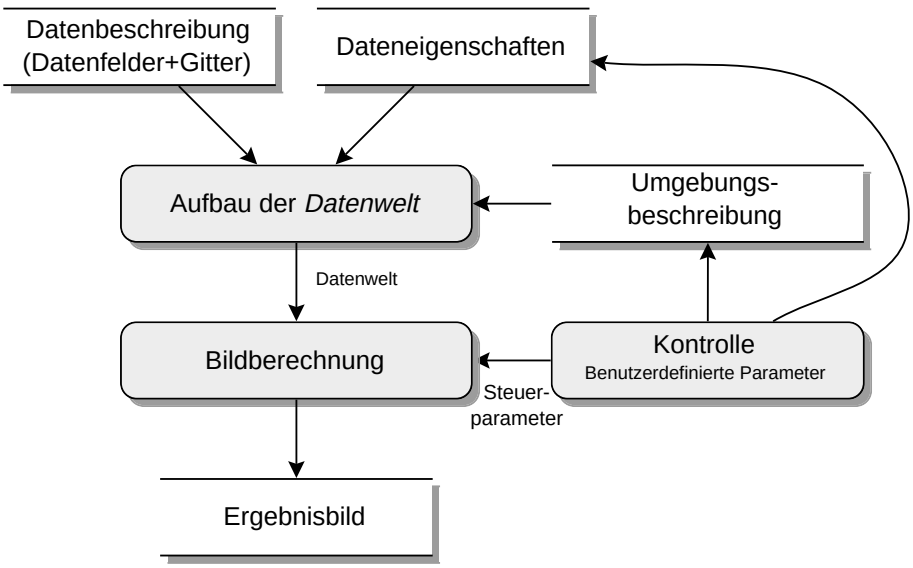



Abbildung 5.1: *Level 0 - Datenflußdiagramm*

Das Diagramm in Abb. 5.1 beschreibt auf der gröbsten Stufe die Datenströme zu und zwischen den elementaren konzeptionellen Aufgaben. Die abzubildende Welt wird aus den vorliegenden Daten (Datenfelder und Gitter), deren Eigenschaften (*Shading*-Modell, Transferfunktionen für Farbgebung und Transparenz) und der Umgebungsbeschreibung (z.B. Blickrichtung, Position von Lichtern, Größe des Ergebnisbildes) erstellt.

Es besteht eine funktionale Aufteilung der Szenenbeschreibung in Daten, Materialeigenschaften und Umgebungseigenschaften. Viele der Material- und Umgebungsparameter lassen sich vom Benutzer kontrollieren.

Daten- und Kontrollstrom sind voneinander getrennt. Die Bildberechnung wird auch direkt durch Steuerparameter beeinflußt (z.B. Angabe eines maximalen *Sampling* Abstandes ).

### 5.1.2 Funktionsablauf

Neben dem grundlegenden schematischen Datenfluß spielt der geordnete Ablauf der einzelnen Aktivitäten eine entscheidende Rolle. Um diesen aufzuzeigen, wird hier ein Funktionsablaufdiagramm (Abb. 5.2) für den *Volume Rendering* Vorgang erstellt. Es sei angemerkt, daß dieses Diagramm im Vergleich zum *Level 0 - Datenflußdiagramm* auf einer etwas niedrigeren Abstraktionsebene angesiedelt ist, d.h. konkretere Einzelheiten berücksichtigt.

Die Pfeile in diesem Diagramm spiegeln den zeitlichen Ablauf der Aktionen wieder.

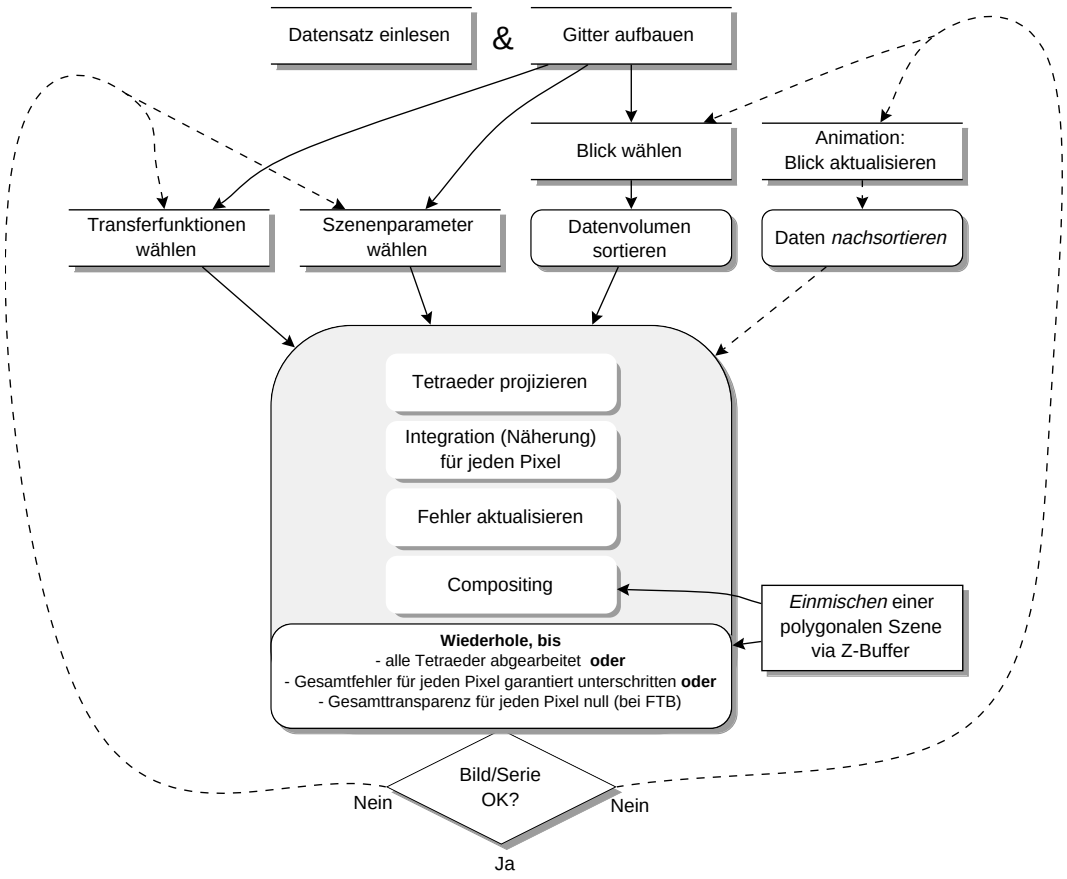



Abbildung 5.2: *Funktionsablaufdiagramm*

Die Schleife in der Mitte des Diagramms repräsentiert den funktionalen Komplex der *Bildberechnung*, während sich der *Aufbau der Datenwelt* und die *Kontrolle der Systemparameter* zeitlich verschränkt im oberen Diagrammteil widerspiegeln.

Die Wahl der Blickrichtung mußte von den übrigen Szenenparametern getrennt werden, da die Sortierung des Datenvolumens nur von diesem einen Parameter abhängt.

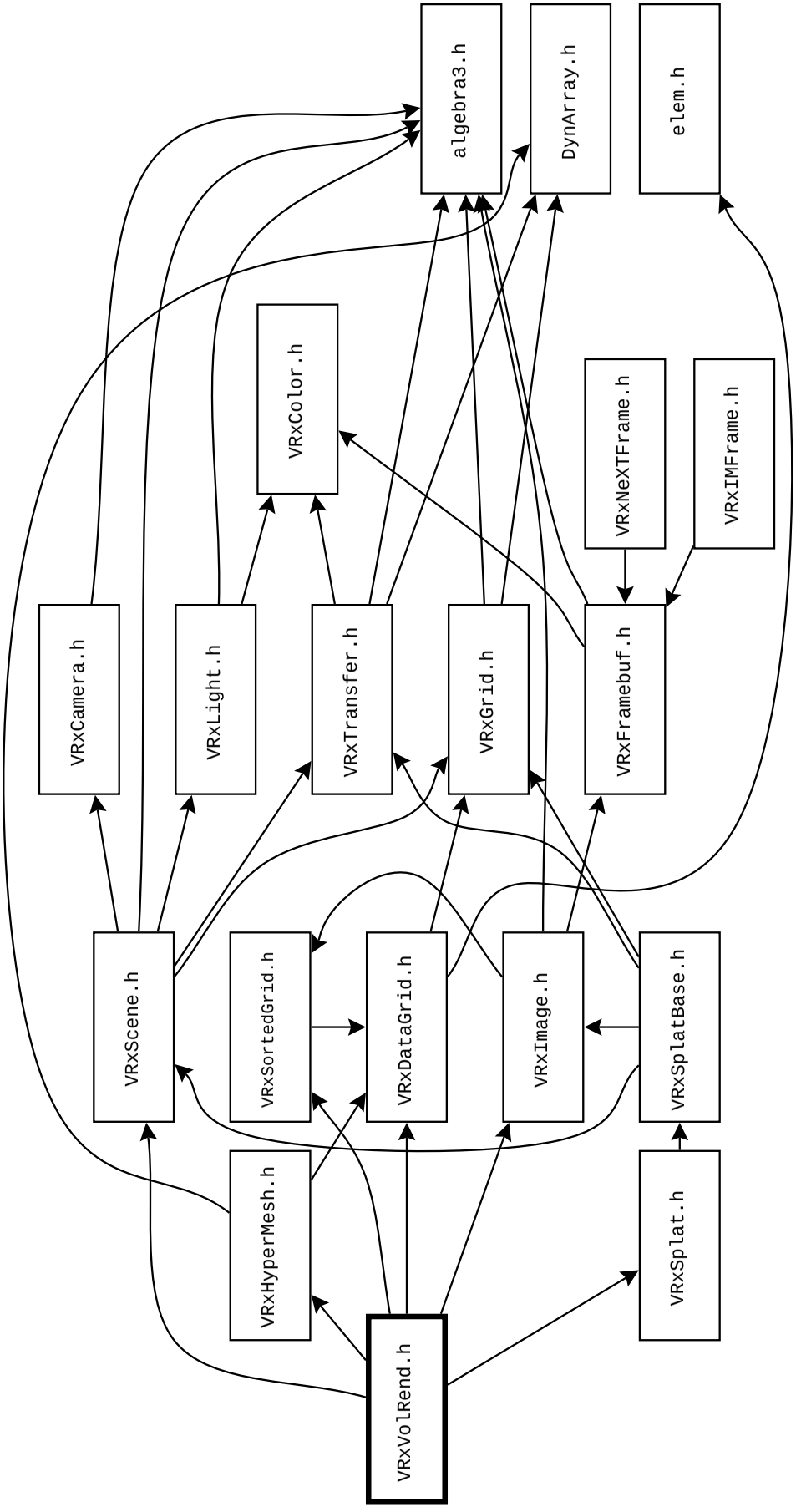
## 5.2 Design

Im Designschritt wird ein strukturierter Klassenentwurf erarbeitet, d.h. die Problemlösung wird in möglichst in sich abgeschlossene Komponenten unterteilt, die im Idealfall als Spezialfälle allgemeiner, wiederverwendbarer Konzepte formuliert werden können.

### 5.2.1 Übersicht über die Programmodule und ihre Klassen

```
VoRANG.cc  -------------  Aufruf des Volume Renderers

VRxVolRend.{h,cc} ------  Die aeusserste Volume Rendering Klasse,
                          Tcl-Schnittstelle

Dateiformat Datengitter: Lexicalizer und Parser
-----------------------------------------------
HyperMeshLex.y   -------  flex (lex) - Grammatik
HyperMesh.y  -----------  Bison (yacc) - Grammatik
VRxHyperMesh.{h,cc} ----  Datentypen fuer das Dateiformat HyperMesh

Gitter und Daten
----------------
VRxGrid.{h,cc}  --------  Die Gitter-Datenstrukturen (Geometrie)
VRxDataGrid.{h,cc} -----  Gitterdaten und Feldverteilungen

Sortierung der Volumenzellen
----------------------------
VRxSortedGrid.{h,cc}  --  Die (abstrakte) Basisklasse zum Sortieren
VRxConcaveSort.{h,cc} --  Sortieren von Gittern mit Konkavitaeten
VRxMPVOSort.{h,cc}  ----  Der Standard  MPVO Algorithmus

Weitere Klassen im Bereich Datenwelt
------------------------------------
VRxScene.{h,cc} --------  Verwaltung der Objektszenerie (+Matrizen !)
VRxCamera.{h,cc} -------  Einstellungen fuer die Kamera und Projektionen
VRxTransfer.{h,cc} -----  Mapping-Strategien (Color & Alpha)
VRxLight.{h,cc} --------  Externe Lichtquellen,
                          bisher noch nicht implementiert!

Bildberechnung
--------------
VRxSplatBase.{h,cc} ----  Basisklasse fuer Projektionsmethoden
VRxSplat.{h,cc} --------  Verschiedene Approximationsverfahren
polygon.{h,cc}  --------  U.a. Scan Converting von Polygonen

Bildverwaltung
--------------
VRxImage.{h,cc}  -------  Compositing und Bildverwaltung
VRxFramebuf.{h,cc}  ----  Basisklassen: Framebuffer
VRxIMPicture.{h,cc} ----  SUN/SGI Realisierung (mit SDSC IMtools)
VRxNeXTPicture.{h,cc} --  NeXT Realisierung

Allgemeine Datenstrukturen
--------------------------
VRxColor.{h,cc}  -------  Repraesentierung von Farben
algebra3.{h,cc}  -------  Klassen fuer 3D-Vektoralgebra
utils.{h,cc}  ----------  Diverse Hilfs-Funktionen

Template-Klassen
----------------
DynArray.h   -----------  Dynamisches Array
DynUniqArray.h  --------  Erweiterungen des Dynamischen Arrays
HashTable.h  -----------  Hash Table
```

Hierarchie der Header Dateien im Projekt *VoRANG*



### 5.2.2 Vererbung

Eine wesentliche Überlegung zur Erstellung der Klassenhierarchie ist, daß das Konzept der Vererbung nur dort eingesetzt wird, wo es tatsächlich Anwendung findet, beispielsweise bei der Ableitung von Spezialfällen aus generischen, wiederverwendbaren Klassen oder für die Bereitstellung mehrerer Implementationsalternativen für ein globales Konzept.

Es wird *nicht* die Erstellung *eines* gemeinsamen Ableitungsbaumes für das gesamte Programm verfolgt, da eine solche Baumstruktur für Programme erfahrungsgemäß zu eine schlechten Wiederverwendbarkeit der einzelnen Komponenten führt. Stattdessen wird auf eine sogenannte *Wald*struktur hingearbeitet.

Im Projekt VoRANG wird Vererbung in drei verschiedenen Ausprägungen eingesetzt: Zum Zwecke von *Alternativimplementierungen*, *Parallelimplementierungen* und *Spezialisierungs-Implementierungen*.

Beispiele für *Alternativimplementierungen* sind die Ableitung dreier verschiedenen Sortierungsklassen von `VRxSortedGrid` und die Ableitung zweier Bildspeicherungsklassen (für verschiedene Hardware-Plattformen) von `VRxPicture` (Datei **VRxFramebuf.cc**). In beiden Fällen wird pro Programmlauf, bzw. sogar pro *Compilierung*, nur eine der jeweils zwei Möglichkeiten realisiert.

Beispiele für *Parallelimplementierungen* sind die Ableitung diverser Realisierungen des Konzeptes „Datenfeld“ von der Klasse `VRxDataField` und auch die Ableitung der verschiedenen Projektions-Approximationen von der Klasse `VRxSplat`. Von diesen parallelen Möglichkeiten der Realisierung werden während eines Programmlaufes durchaus mehrere benutzt. Es wird aktiver Gebrauch vom Konzept des *dynamischen Bindens* gemacht.

Schließlich gibt es mehrere Beispiele für das Konzept der *Spezialisierungs-Vererbung*. Die Ableitung der Tetraederklasse von der Polyederklasse in **VRxGrid.{h,cc}** oder die Ableitung der Klasse `VRxNode` von `vec3` aus demselben Modul sind nur zwei davon.

In den zur Illustration auf der folgenden Seite beispielhaft herausgegriffenen Vererbungsdiagrammen repräsentieren Striche zwischen den Klassensymbolen Vererbungen. Die weiter oben stehenden Klassen übernehmen dabei die Rolle der Eltern.

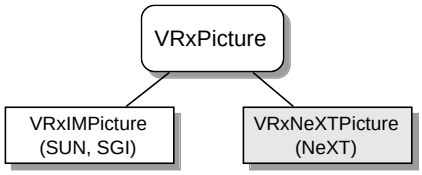



Abbildung 5.3: *Vererbungsrelation: Alternativimplementierungen*

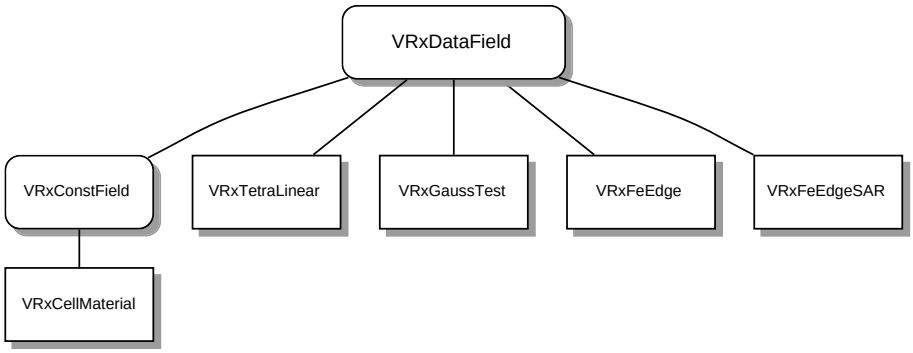



Abbildung 5.4: *Vererbungsrelation: Parallelimplementierungen*

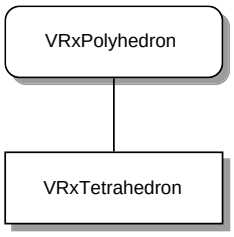



Abbildung 5.5: *Vererbungsrelation: Spezialisierung*

# Kapitel 6

# Ausgewählte Fragen der Implementierung

In diesem Kapitel wird näher auf ein paar wesentliche Punkte der Implementierung eingegangen. Dabei wird auch die Erweiterbarkeit des Programmsystems diskutiert. Themenschwerpunkte sind die Definition von Feldverteilungen, Mapping von Daten- auf Farb- und Dichtewerte, und Näherungsmethoden für die Bildberechnung.

Der Benutzer des Programms VoRANG hat diverse Freiheitsgrade in der Festlegung seiner Bildparameter. Er muß

- eine geeignete Interpolationsfunktion für die wahrscheinlich auf Samplepunkten definierten Daten wählen.
- geeignete Abbildungsfunktionen von den Datenwerten auf Farb- und Dichtewerte definieren, und zwar in der großen Bandbreite von einfachen Abbildungstabellen bis hin zur Implementierung ausgeklügelter prozeduraler Shadinggleichungen.
- eine geeignete Approximationsmethode, eine passende Integrationstechnik und eventuell einen günstigen Abtastabstand für die Bildberechnung wählen.

Diese drei Themengebiete werden in ganz verschiedenen Modulen innerhalb des Programmpaketes behandelt. Im folgenden werden diese Gebiete näher betrachtet.

## 6.1 Gitter- und Datenbeschreibung

Wie bereits erwähnt, werden in der gegenwärtigen Programmversion von VoRANG ausschließlich Datenverteilungen auf Tetraedergittern verarbeitet, diese können jedoch in sehr vielfältiger Weise definiert werden.

### 6.1.1 Dateiformat für Gitter und Daten

Im Rahmen der wissenschaftlichen Simulationsrechnungen, in deren Verlauf das Programm VoRANG zur Visualisierung von Feldverteilungen eingesetzt werden soll, muß

ein reibungsloser Datenaustausch zwischen den verschiedenen Simulationsphasen *Gittergenerierung*, *Numerische Lösungsverfahren* und *Visualisierung* gewährleistet werden.

In der derzeitigen Situation werden die einzelnen Programme zur Simulation und Visualisierung nicht direkt gekoppelt. Stattdessen findet die Visualisierung nur als *Postprocessing* auf den in Dateien abgelegten Simulationsergebnissen statt. Der Datenaustausch hierzu ging bisher über sehr einfache ASCII-Formate vonstatten, die, wenn sie den wachsenden Anforderungen nicht mehr genügten, durch ein neues spezielleres Format ergänzt wurden. Dies führte zu einer Vielzahl verschiedener spezieller Dateiformate.

Um hier Abhilfe zu schaffen, wurde ein gemeinsames Dateiformat für die genannten Teilschritte der Computersimulationen definiert, das allgemein und flexibel genug ist, um die von Numerik und Visualisierung benötigten Informationen gleichermaßen zu berücksichtigen, und das dabei auch für absehbare zukünftige Erweiterungen offen ist.

Dieses gemeinsame Dateiformat für Tetraedergitter genügt folgenden Anforderungen:

- Daten können wahlweise ASCII- oder binärkodiert gespeichert werden.
- Man hat die Möglichkeit, entweder nur das Gitter oder nur Daten oder beides zusammen in einer Datei zu speichern.
- Man kann in einer Datei beliebig viele Datensätze für *ein* Gitter abspeichern.
- Neben Gitterpunkten und Tetraedern können auch Kanten und Dreiecke abgespeichert werden.
- Die „Kodierung“ der Tetraeder erfolgt wahlweise über die zugehörigen Punkte, Kanten oder Dreiecke. Entsprechend erfolgt die Kodierung der Flächen wahlweise über die zugehörigen Punkte oder Kanten.
- Daten können sowohl auf Punkten, Kanten, Dreiecken und/oder Tetraedern definiert werden.
- Für die Angabe von Koordinaten, Indizes und Simulationsergebnissen sind unterschiedliche Datentypen erlaubt. Z.B. können Gleitkommazahlen als `double` oder `float` definiert werden, was z.B. eine platzsparende Speicherung im Binärformat ermöglicht.
- Für alle Dreiecke kann zusätzlich ein Verweis auf den darüber- bzw. darunterliegenden Tetraeder angegeben werden.
- Tetraeder sind rechtshändig orientiert. Punkt-, Kanten- oder Flächenindizes müssen entsprechend sortiert abspeichert werden.
- Das Format bietet Beschreibungsmechanismen für beliebige Datenfeldverteilungen, unter anderem auch zur Festlegung der Interpolationsmethode.

Das Dateiformat mit dem Namen *HyperMesh* wird in Anhang B ab S. 92 detailliert beschrieben.

### 6.1.2 Datenverteilungen

Das entscheidende Element in einer Datengitter-Datei sind die Feldverteilungen, die mit dem Schlüsselwort `field` eingeleitet werden. Das Programm VoRANG visualisiert ausschließlich solche Feldverteilungen. Welche im Datenfile aufgeführte(n) Feldverteilungen visualisiert werden soll(en), wird in den *Mapping*-Dateien festgelegt.

Was eine Feldverteilung auszeichnet, ist die Interpolationsmethode, durch die – irgendwo auf dem Gitter definierte – Rohdaten in eine kontinuierliche Funktion über alle Tetraeder umgesetzt werden. Verwirklicht wird dieses Konzept der Feldverteilung durch die in **VRxDataGrid.{h,cc}** angesiedelte Basisklasse `VRxDataField` und ihre abgeleiteten Klassen. Eine Instanz von `VRxDataField` bezieht sich auf ein oder mehrere Rohdatenpakete (Klasse `VRxData`), also z.B. auf ein paket von auf Knoten definierten Daten*samples*. Sie stellt die spezifischen Methoden `initInterpolation` und `interpolateField` bereit.

### 6.1.3 Erweiterung des Programms um neue Interpolationsmethoden

Hier ein paar Hinweise zum Erweitern des Programms VoRANG um eine neue Interpolationsmethode. Die Vorgehensweise ist stichpunktartig die folgende:

- Ableiten der Klasse `VRxDataField`. Der *Constructor* bekommt alle verwendeten Datenpakete (`VRxData`) und sonstige Informationen über die neue Datenverteilung übergeben, wie es auch bei den bestehenden Unterklassen der Fall ist.
- Bereitstellen der beiden Methoden `initInterpolation` und `interpolateField` in der neuen Klasse.
- Einbauen der neuen Datenfeldverteilung in das Datengitterformat *HyperMesh*: Anpassen der Dateien **HyperMeshLex.y** (Lexicalizer) und **HyperMesh.y** (Parser). Aus dem Parser heraus wird auch der Constructor der neuen Feldverteilung aufgerufen.
- Einbauen der Formaterweiterung in die Methoden zum Abspeichern des aktuellen Datengitters: `writeHyperMesh` und `writePartialGrid` in Klasse `VRxDataGrid`.

## 6.2 Color- und Alpha-Mapping

Die Abbildung von Datenwerten auf Emissions- und Absorptionswerte beeinflußt entscheidend das resultierende Bild des Datenvolumens. Für das Projekt *VoRANG* soll die Abbildungsstrategie so frei wie möglich wählbar sein. Es soll ein *Testbed* für verschiedene Strategien entstehen, in dessen Rahmen auch mehrere Methoden bei der Erstellung *eines* Bildes kombinierbar sein sollen.

Ergebnis des *Color Mapping* Vorganges ist, daß für einen bestimmten Ort im Volumen sowohl ein Farbwert bestimmt wird, der die Emission an diesem Volumenpunkt repräsentiert, als auch ein Wert `alpha`, der die Absorption an diesem Volumenpunkt angibt. Im Projekt *VoRANG* wird dieses Ergebnis zweistufig durch Color- und

Alpha-Mapping-Dateien und durch verschiedene `Color-` und `Alpha-Shading`[1] Methoden erreicht, die als abgeleitete Klassen (Parallelimplementierungen) unter Basisklassen `VRxColorShading` und `VRxAlphaShading` zur Verfügung gestellt werden.

Die Mapping Dateien beinhalten dabei die Parameter (oft Abbildungstabellen) für die einzelnen Shading-Methoden.

In den *Mapping*-Dateien, deren Format in Anhang C ab S. 100 ausführlich beschrieben wird, kann konkret folgendes festgelegt werden:

- Die Shadingmethode(n) (durch eindeutige Bezeichner).
- Die Abbildung von Datenwerten in Koeffizienten/Farbwerte/Alphawerte (durch tabellarische Auflistung).
- andere Parameter für diese Shadingmethode(n)

Im Falle des einfachen Emissions-/Absorptionsmodelles übernehmen die *Shading* Methoden beispielsweise bloß die Verwaltung einer LookUp-Tabelle und brauchen gar keine abschließende Berechnung mehr auszuführen. Im Falle einer Anwendung des Phong'schen Beleuchtungsmodells hingegen wird beim *Color Shading* zwar ebenfalls eine LookUp-Tabelle verwaltet, aber die aus dieser Color-LookUp-Tabelle erhaltenen Werte sind dann Koeffizienten für den ambienten, den diffusen, und den spiegelnden Anteil des Lichtes am angegebenen Volumenpunkt. Aus diesen Koeffizienten muß unter Berücksichtigung der Lichtquellen und des Gradienten am aktuellen Volumenpunkt dann der endgültige „Emissionswert" bestimmt werden.

Es ist durchaus nicht unabdingbar, daß in den Mapping-Dateien eine tabellarische Abbildung von Datenwerten in was für Koeffizienten auch immer notiert wird. Es ist beispielsweise eine `Alpha-Shading` Methode `ISOVALUE` denkbar, die als Parameter nur die Angabe mehrerer Datenwerte fordert, an denen Isoflächen erscheinen sollen, samt einer Angabe, wie stark diese Isofläche betont werden soll. Die Art der angegebenen Parameter in den *Mapping*-Dateien ist also einzig und allein von der gewählten *Shading* Methode abhängig.

Verschiedene Shading-Strategien können zur Erstellung *eines* Bildes kombiniert werden. Für bestimmte Datenwerte soll Shading-Methode `X` und für andere Werte Methode `Y` verwendet werden. Die für einen solchen Fall verwendete Syntax ist ebenfalls in Anhang C beschrieben.

### 6.2.1 Die Implementierung des Mapping Vorganges

Sämtliche Aktionen, die direkt mit dem Mapping Vorgang zu tun haben, sind in den Dateien **VRxTransfer.{h,cc}** realisiert. Die Klassen, die den Bildberechnungsklassen eine Schnittstelle zum Mapping anbieten sind `VRxColorMapping` und `VRxAlphaMapping`.

[1] Im Rahmen dieses Projektes wird folgende sprachliche Konvention verwendet: Color- und Alpha-*LookUp* bedeutet das rein tabellarische Abbilden eines Datenwertes aus der zu visualisierenden Feldverteilung in Farb- bzw. Dichtewerte. Eine prozedurale Abbildung von Koeffizienten oder Parametern unter Zuhilfenahme weiterer Größen – wie z.B. des aktuellen Ortes im Datengitter, des Ortsgradienten, oder der aktuellen Volumenzelle – in Farb- oder Dichtewerte wird in Ermangelung eines besseren Begriffs im folgenden Color- bzw. Alpha-*Shading* genannt. Der gesamte Vorgang, der durchaus beide Stufen hintereinandergeschaltet umfassen kann wird mit Color- bzw. Alpha-*Mapping* bezeichnet.

Dies sind jedoch nicht die Klassen, die man ableiten muß, um eigene Mapping Strategien zu implementieren. Diese zwei Klassen sind nur dafür zuständig, die Mapping Dateien einzulesen, herauszufinden, welche *Shading* Methoden verwendet werden sollen, und Instanzen der entsprechenden Shading-Klassen zu erzeugen, indem sie die zur jeweiligen Shading Klasse gehörenden Parameterblöcke aus den Mapping Dateien heraussuchen und an den Constructor dieser Shading Klasse weiterreichen. Außerdem muß für den Fall, daß verschiedene Shading Methoden in Kombination eingesetzt werden sollen, in dieser „vorgeschalteten" Mapping Stufe entschieden werden, welche Shading Methode für welchen Datenwert angewendet wird.

Die Shading Klassen, die sämtlich von `VRxColorShading` und `VRxAlphaShading` abgeleitet sind, repräsentieren genau die verschiedenen Mapping Methoden, die z.B. im Fall des Color Mappings durch die Schlüsselwörter

`CONSTANT, IDENTITY, SIMPLE_EMITTER, ABSORPTION_WEIGHTED` und `CONST_CELL`

in den Color Mapping Dateien spezifiziert werden.

### 6.2.2 Erweiterung des Programms um neue Mapping Methoden

Hier ein paar Hinweise zum Erweitern des Programms VoRANG um eine neue Color Mapping Methode. Vollkommen analog kann auch eine neue Alpha Mapping Methode definiert werden. Es müssen ausschließlich Änderungen in **VRxTransfer.{h,cc}** vorgenommen werden. Die Vorgehensweise ist stichpunktartig die folgende:

- Ableiten der Klasse `VRxColorShading`. Der *Constructor* kann ganz frei gestaltet werden, sollte aber einen Eingabestrom `istream*` mit in der Parameterliste haben, denn nach Konvention ist er für das Einlesen seines persönlichen Parameterblocks selbst zuständig. Die Klasse `VRxColorMapping` hat dafür zu sorgen, daß der Streampointer auf die richtige Stelle zeigt, wenn er dem neuen Constructor übergeben wird.
- Die Methode `color` der neuen Klasse ist für das Mapping eines einzelnen Datenwerts zuständig und bekommt dafür den Datenwert selbst, die Nummer des aktuellen Tetraeders, die absolute Gitterposition und den (schon früher bestimmten) `alpha`-Wert an dieser Stelle übergeben. Aus all dieser Information muß nun ein Farbwert gebastelt werden.
- Wie bereits angesprochen, muß die Klasse `VRxColorMapping` noch leicht erweitert werden. Diese Klasse enthält schließlich den Parser für das Mapping Format. In der Methode `readMappingFile` muß die neue Methode eingeklinkt werden.

## 6.3 Approximationen für die Bildberechnung

In diesem Abschnitt werden die gegenwärtig implementierten Approximationsmethoden beschrieben, die die Intensitäts- und Dichtebeiträge der Zelle für alle Pixel der Zellprojektion bestimmen.

In diesem Bereich hat der Benutzer einige Entscheidungen zu treffen: Er muß eine Approximationsmethode auswählen (s. 6.3.1) und für den Fall, daß er `exact` oder `interpolateCoeff` gewählt hat, auch noch eine Integrationsmethode (s. 6.3.2). Für die Methode `exact` muß er zusätzlich noch einen maximalen Abtastabstand vorgeben. Alle diese Entscheidungen haben sowohl Einfluß auf die Qualität des Ergebnisbildes als auch auf die Dauer der Bildberechnung.

Die für all diese Funktionalität zuständigen Klassen in der Hierarchie sind `VRxSplat` und deren Unterklassen (`VRxTestSplat, VRxExactSplat, VRxCoeffInterpSplat` und `VRxColorInterpSplat`). Die Basisklasse `VRxSplat` ist in den Dateien **VRxSplatBase.{h,cc}** angesiedelt, während die Unterklassen in **VRxSplat.{h,cc}** zu finden sind. Es sei hier erwähnt, daß der Begriff *Splat* hierbei als zusammenfassender Oberbegriff für die Aktionen *Projektion*, *Integration* und *Compositing* verwendet wird und nicht etwa als der Bezeichner für nur eine spezielle grobe Approximationsart.

Die Programmierschnittstelle zur Definition derartiger „Splatting"-Approximationsmethoden ist recht einfach: Die Approximationsverfahren sind in einer Methode `projectTetra` der Splatting-Klassen angesiedelt, die als Parameter das aktuelle Tetraeder übergeben bekommt. Diese Methode projiziert sämtliche vier Knoten des Tetraeders in die Bildebene und teilt diese Projektion geeignet in Dreiecke auf (vgl. Abschnitt 2.2.2.2, S.30). Dann müssen die Farb- und Dichtewerte bestimmt werden, die die Zelle zu jedem Pixel der Projektion beiträgt — dies übernimmt das jeweilige Approximationsverfahren — und diese Beiträge werden Pixel für Pixel einer *Compositing*-Methode übergeben.

Die Methode `projectTetra` ist als virtuelle Funktion definiert. Für jedes neue Approximationsverfahren wird durch Ableitung von der Basisklasse `VRxSplat` eine neue Version von ihr bereitgestellt. Die Methoden zur Projektion, Aufspaltung der Projektion in Dreiecke und *scan conversion* der Dreiecke sind in der Basisklasse selbst realisiert und können von allen realisierten Approximationsverfahren gleichermaßen verwendet werden. Die einzelnen Verfahren unterscheiden sich nur darin,

a) welche Werte sie während der *scan conversion* über die Zellprojektion interpolieren lassen und

b) wie sie aus diesen Werten die Intensitäts- und Dichteintegrale für jeden Pixel bestimmen.

Die verschiedenen Integrationsgleichungen (s. 6.3.2) sind als statische Funktionen definiert und können somit auch von allen Approximationsverfahren verwendet werden. Sie werden aber bisher nur von `VRxExactSplat` und `VRxCoeffInterpSplat` benötigt.

### 6.3.1 Approximationsmethoden

Die vier implementierten Verfahren gehen wie folgt vor:

#### „Exakte" Methode

Die genaueste der vier Methoden bekommt einen Parameter $\Delta s$ übergeben, der den maximal erlaubten Abstand zwischen zwei *sample points* angibt. Es wird dann dementsprechend an jedem einzelnen Pixel das Intervall zwischen vorderer Zellenwand und

hinterer Zellenwand in kleinere Teilintervalle von höchstens der Länge $\Delta s$ unterteilt. An allen Stützpunkten dieser Teilintervalle wird mit den für die aktuelle Datenverteilungen gültigen Interpolationsmethoden ein Datenwert bestimmt, dieser wird mit Hilfe der *mapping functions* auf Emissions- und Absorptionskoeffizienten abgebildet und die Teilintegrale für die Dichte und die Intensitäten werden nach einer vom Benutzer spezifizierten Integrationsmethode (6.3.2) bestimmt. Jeder auf diese Weise bestimmte Beitrag der Teilintegrale wird direkt an die *Compositing* Methode (in der Klasse `VRxImage`) weitergereicht und von dieser ins Ergebnisbild eingebracht.

Wenn als Parameter $\Delta s$ ein Wert übergeben wird, der für einen beliebigen Pixel größer ist als das gesamte Integral zwischen vorderer und hinterer Zellenwand, wird der Parameter vernachlässigt und das gesamte Integral (*Zellenintegral*) direkt nach der spezifizierten Integrationsnäherung ausgewertet.

#### Methode der „interpolierten Koeffizienten“

Bei dieser Methode wird das *Zellenintegral* für jeden Pixel ohne Berücksichtigung weiterer Stützstellen wiederum nach der jeweils spezifizierten Integrationsnäherung (6.3.2) bestimmt. Allerdings werden die Emissions- und Absorptionskoeffizienten nicht mehr für jeden Pixel an Vorder- und Hinterwand der Zelle bestimmt, indem die *mapping functions* auf die dort anliegenden Datenwerte angewandt werden. Stattdessen werden die Koeffizienten nun nur noch an den Knoten der Projektion bestimmt und dazwischen linear interpoliert. Die Dichte und Intensität wird jedoch immer noch nach der angegebenen Methode für jeden Pixel integriert.

#### Farbinterpolations-Methode

In dieser gröbsten der implementierten Approximationen wird nun auch noch auf das Auswerten der Integrale an jedem Pixel verzichtet und statt dessen werden nach der *Gouraud Shading* Interpolationsmethode aus [ST90] direkt Farb- und Transparenzwerte über die Zellprojektion interpoliert, wie das in Abschnitt 2.1.1.2, S. 12 beschrieben wird. Die dort angeführte Änderung der *Compositing* Gleichung wird durch die Verwendung einer zweiten Compositing-Methode aus `VRxImage` realisiert.

#### Test-Methode

Diese Methode ist im Grunde überhaupt keine Realisierung einer Volume Rendering Technik. Sie dient nur der Darstellung der Gitter*geometrie*. Es werden alle Volumenzellen mit semitransparenten Zellwänden dargestellt. Die Farbe ist einheitlich rot. Der Grad der Undurchsichtigkeit wird durch einen benutzergesteuerten Parameter zwischen durchsichtig (0.0) und undurchsichtig (1.0) festgelegt. Clipping findet bei dieser Methode nicht statt. Diese Methode erfüllt vor allem den Zweck als Testmethode zur korrekten Ausrichtung des Datenvolumens.

### 6.3.2 Integrationsmethoden

Von ebenso entscheidender Bedeutung wie die Wahl der geeigneten Approximationsmethode ist die Auswahl der günstigsten Integrationsmethode. Bei den drei angebotenen

Möglichkeiten ist das keine Frage der Ausführungsgeschwindigkeit – die Berechnungen sind von ihrem Aufwand her durchaus ähnlich. Hingegen ist keine Integrationsmethode für alle Anwendungsfälle günstiger als die andere. Es existieren für jede einzelne Methode „Problemfälle". Gute Beispiele dafür sind die Anwendung der Trapezregel-Integration bei großflächig undurchsichtigen Gebieten oder die Anwendung der konstanten E/A-Integration auf Gebiete mit konkreter Emission aber einer Absorption von null. Für kohärente hochgeradig undurchsichtige Gebiete liefert die *gewichtete konstante E/A-Integration* bessere Ergebnisse als die normale *konstante E/A-Integration*, hingegen führt erstere insgesamt wesentlich häufiger zu Artefakten als letztere.

Eine interessante Alternative könnte die in Abschnitt 6.3.3 hergeleitete exakte Integration für den Fall linear variierender (und voneinander unabhängiger) Emission und Absorption sein, die aber bisher noch nicht implementiert wurde.

In allen drei nachfolgenden Fällen wird die *Intervalltransparenz* $\theta_k$ nach dem gleichen Schema, nämlich nach Trapezregel (Gleichung 2.14) aufintegriert:

$$\theta_k = e^{-\frac{1}{2}(\chi(s_{k-1})+\chi(s_k))\,\Delta s_k), \text{ mit } \Delta s_k=(s_k-s_{k-1})}$$

Das ist eine exakte Berechnung für den Fall, daß der Absorptionskoeffizient zwischen den beiden Stützstellen linear variiert (also z.B. lineare Interpolation der Originaldaten über die Zelle und gleichzeitig lineare Alpha-Mappingfunktion). Für alle anderen Fälle ist dies bereits eine Approximation.

### Konstante Emissions/Absorptions-Integration

Bei der Bestimmung der kumulativen Intervalltransparenz wurde bereits implizit ein Mittelwert der an den beiden Stützstellen $s_k$ und $s_{k-1}$ anliegenden Absorptionskoeffizienten ermittelt:

$$\chi_\emptyset = 0.5 \cdot (\chi(s_{k-1}) + \chi(s_k))$$

Analog werden die Emissionswerte an den beiden Stützstellen gemittelt:

$$\eta_\emptyset = 0.5 \cdot (\eta_{s_k} + \eta_{s_{k-1}})$$

Es wird nun angenommen, daß die Werte $\eta_\emptyset$ und $\chi_\emptyset$ konstant über dem Intervall $[s_{k-1}, s_k]$ vorliegen. Nach Gleichung 2.12 wird dann die akkumulierte Intervallintensität bestimmt:

$$b_k = \frac{\eta_\emptyset\,(1-\theta_k)}{\chi_\emptyset}$$

Die beiden Werte $\theta_k$ und $b_k$ werden an die *Compositing*-Gleichung übergeben.

Es bleibt anzumerken, daß dieses Vorgehen eine exakte Lösung der Lichttransfergleichung (2.5) über dem angegebenen Intervall für den Fall darstellt, daß Emission und Absorption über diesem Intervall proportional zueinander verlaufen. Anderenfalls ist dies eine Approximation, die allerdings desto genauer wird, je kleiner die Intervalle werden.

Für den Fall jedoch, wo Emission vorhanden ist, jedoch keine Absorption, tritt folgende Schwierigkeit auf: $\chi_\emptyset$ ist gleich null. Damit nicht durch null geteilt wird, wird $\chi_\emptyset$ bei der berechnung von $b_k$ in diesem Fall auf einen sehr kleinen Wert (0.0000001) gesetzt, was in einer akkumulierten Intervallintensität von null resultiert, denn der Zähler von Gleichung 2.12 ergibt sich exakt zu null. So kommt es dadurch, daß der unbestimmte Ausdruck $\frac{0}{0}$ in diesem Fall zu null ausgewertet wurde, zu der Situation, daß eine Emission von null berechnet wird, obwohl durchaus eine Emission spezifiziert wurde.

#### Gewichtete Konstante E/A-Integration

Das Vorgehen ist bei dieser Methode das gleiche wie bei der vorhergehenden, mit dem Unterschied, daß als „konstanter Emissionskoeffizient" eine andere Mittelung der Koeffizienten an den Intervallgrenzen verwendet wird:

$$\eta_\emptyset = \eta_{s_{k-1}} \cdot \theta_k + \eta_{s_k} \cdot (1 - \theta_k)$$

Die Idee dahinter ist, daß z.B. im Fall eines vollkommen undurchsichtigen Gebietes, wo nur der allervorderste Emissionswert eine Rolle spielen kann, es eine ungünstige Wahl ist, den Emissionswert gleichberechtigt zwischen vorderer und hinterer Intervallgrenze zu mitteln.

Was allerdings sehr negativ bei dieser Methode zu Buche schlägt, ist, daß sie den Vorteil des korrekten Ergebnisses bei proportionaler Emission und Absorption aufgibt, weil $\chi_\emptyset$ und $\eta_\emptyset$ nun sozusagen nicht mehr an derselben Stelle abgegriffen werden und wahrscheinlich auch keine Stelle in dem angegebenen Integral existiert, an der genau dieses Verhältnis von $\chi$ und $\eta$ vorherrscht.

#### Integration nach Trapezregel

In diesem Fall wird die akkumulierte Intervallintensität $b_k$ nach Trapezregel (Gleichung 2.15) approximiert:

$$b_k = \frac{1}{2}(\eta(s_{k-1})\,\theta_k + \eta(s_k))\,\Delta s_k$$

Bei hoher Undurchsichtigkeit des betreffenden Gebietes und großen Intervallen (z.B. über den gesamten Zelldurchmesser) bekommt man ein Problem an den Zellrändern, wo $\Delta s_k$ gegen null geht. Die Emission geht dann dort gegen Null, obwohl die Opazität immer noch recht groß ist. Und das führt zu deutlich dunklen Zellrändern.

### 6.3.3 Mathematisch exakte Lösung für lineare Emission und Absorption

Im folgenden wird eine exakte Lösung für die Bestimmung von $\theta_k$ und $b_k$ für den allgemeinen Fall hergeleitet, daß Emission und Absorption über dem Intervall $[s_{k-1}, s_k]$ linear verlaufen.

Wenn $\rho(\boldsymbol{x})$ im Zelleninneren linear interpoliert wird, wird das Skalarfeld dort durch folgende Parameterfunktion angegeben:

$$\rho(x,y,z) = c_1 + c_2\,x + c_3\,y + c_4\,z \tag{6.1}$$

Da das Skalarfeld an den 4 Eckpunkten des Tetraeders bekannt ist, kann man in einem linearen Gleichungssystem mit vier Gleichungen und vier Variablen nach den Parametern $c_1, c_2, c_3, c_4$ auflösen.

Will man die Dichtefunktion $\rho$ in Abhängigkeit des Strahlparameters $s$ ausdrücken, so setzt man die Parameterform des Strahles durch die Zelle,

$$\boldsymbol{x}(s) = \begin{pmatrix} x_0 \\ y_0 \\ z_0 \end{pmatrix} + s \begin{pmatrix} x_r \\ y_r \\ z_r \end{pmatrix}$$

mit $\binom{x_0}{y_0}_{z_0}$ als Strahleneintrittspunkt und $\binom{x_r}{y_r}_{z_r}$ als Richtungsvektor in obige Gleichung ein und erhält

$$\rho(s) = (c_1 + c_2\,x_0 + c_3\,y_0 + c_4\,z_0) + s \cdot (c_2\,x_r + c_3\,y_r + c_4\,z_r), \tag{6.2}$$

oder in Kurzschreibweise: $\rho(s) = k + l \cdot s$.
Nimmt man nun die linearen Transferfunktionen $d(\rho)$ und $i_\nu(\rho)$ als durch

$$i_\nu(\rho) = p_\nu + q_\nu \cdot \rho \quad \text{und} \quad d(\rho) = v + w \cdot \rho$$

gegeben an, so ergeben sich für die Emissions- und Absorptionsfunktionen $\eta$ und $\chi$ in Abhängigkeit vom Strahlparameter $s$ folgende Gleichungen:

$$\eta_\nu(s) = p_\nu + q_\nu\,k + q_\nu\,l\,s \quad \text{und} \quad \chi(s) = v + w\,k + w\,l\,s$$

Für den Fall linearer Absorption gilt Gleichung 2.14 zur exakten Bestimmung von $\theta_k$. Unter Verwendung von $\theta_k$ läßt sich dann $b_k$ wie folgt bestimmen:

$$\begin{aligned} b_k &= \int_{s_{k-1}}^{s_k} p_\nu + q_\nu\,k + q_\nu\,l\,s \cdot e^{-\int_s^{s_k} v+w\,k+w\,l\,s'\,ds'}\,ds \\ &= \frac{q_\nu}{w}\,(1-\theta_k) + \frac{\sqrt{\frac{\pi}{2}}(w\,p_\nu - v\,q_\nu)\,(\mathrm{Erfi}(\frac{v+w\,k+w\,l\,s_k}{\sqrt{2}\sqrt{w}\sqrt{l}}) - \mathrm{Erfi}(\frac{v+w\,k+w\,l\,s_{k-1}}{\sqrt{2}\sqrt{w}\sqrt{l}}))}{e^{\frac{(v+w\,k+w\,l\,s_k)^2}{2\,w\,l}}\,w^{\frac{3}{2}}\,\sqrt{l}} \end{aligned} \tag{6.3}$$

Hierbei ist $\mathrm{Erfi}(x)$ eine Integralfunktion der Gaussverteilung, die auch imaginäre Argumente zuläßt. $\mathrm{Erfi}(x)$ ist folgendermaßen definiert:

$$\mathrm{Erfi}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{u^2} du \quad \text{und} \quad \mathrm{Erfi}(ib) = i\frac{2}{\sqrt{\pi}} \int_0^b e^{-u^2} du$$

für reelle Zahlen $x$ und $b$.

Wenn $l = 0$, ergibt sich der Fall konstanter Emission und Absorption (vgl. 2.12):

$$b_k = \frac{\left(1 - e^{(v+wk)\cdot(s_{k-1}-s_k)}\right) \cdot (p_\nu + q_\nu k)}{v + wk}$$

Wenn $w = 0$, ergibt sich:

$$b_k = \frac{-lq_\nu + p_\nu v + q_\nu kv + q_\nu lvs_k + e^{(s_{k-1}-s_k)\cdot v} \cdot (q_\nu l - p_\nu v - q_\nu kv - lq_\nu vs_{k-1})}{v^2}$$

und für den Fall, daß $w = 0 \wedge v = 0$:

$$b_k = 0.5 \cdot (s_k - s_{k-1}) \cdot (2p_\nu + 2q_\nu k + q_\nu ls_k + q_\nu ls_{k-1})$$

### 6.3.4 Erweiterung des Programms um neue Approximationen

Hier ein paar Hinweise zum Erweitern des Programms VoRANG um eine neue Approximationsmethode. Die Vorgehensweise ist stichpunktartig die folgende:

- Ableiten einer neuen Unterklasse von `VRxSplat`. Diese sollte in den Dateien **VRxSplat.{h,cc}** angesiedelt werden.
- Bereitstellen
  einer statischen Funktion `myDrawPoint(int x, int y, PolygonVertex *pv)`, die vom Scan-Konvertierer für jeden getroffenen Pixel aufgerufen wird und die dafür verantwortlich ist, daß die richtigen Werte zum *Compositing* gelangen.
- Implementierung der Methode `projTetra(VRxPolyhedron& tetra)` in der neuen Klasse. Das sollte in Analogie zu den bestehenden `projTetra`-Methoden geschehen, d.h.: Korrektes Ausfüllen der Knotentabelle `cellNodes`, Nutzung der Methode `project`, Vorbereiten der bei der Scan-Konvertierung zu interpolierenden Werte, Aufruf der Scan-Konvertierung (`polygon_scan`) unter Angabe der eigenen `myDrawPoint`-Funktion.
- Verankern der neuen Approximation in der Skriptsprache: Erweitern der Methode `methodCmd` in der Klasse `VRxVolRend` in der Datei **VRxVolRend.cc**.

# Kapitel 7

# Testläufe und Resultate

In diesem Kapitel werden verschiedene Aspekte der in VoRANG implementierten Bildberechnungsmethoden getestet und die Ergebnisse ausgewertet. Die Art der Tests wird dabei von den Vorgaben aus Abschnitt 4.3, S. 51, maßgeblich beeinflußt. Zunächst wird der Testdatensatz *Farbquader* für zwei verschiedene konstante Absorptionskoeffizienten unter Einsatz der unterschiedlichen Approximationsverfahren visualisiert. Es werden die Artefakte erläutert, die insbesondere bei der Anwendung der *Farbinterpolations*-Näherung entstehen. Es wird ein Geschwindigkeitsvergleich für die verschiedenen Näherungsverfahren gegeben.

Der nächste Test beschäftigt sich mit dem Auftreten hoher Helligkeitskontraste bei ungünstiger Wahl der Transferfunktionen für Emissions- und Absorptionskoeffizienten.

Für eine Temperaturverteilung aus dem Anwendungsfall der bereits in Abschnitt 3.1 vorgestellten *Hyperthermieplanung* wird unter Einsatz zweier verschiedener *Color Mappings* eine Visualisierung vorgenommen.

Schließlich wird ein Beispiel für eine Visualisierung gegeben, bei der eine exakte Berechnung der Farb- und Dichteverteilungen unabdingbar ist. Dieses Beispiel arbeitet mit höherdimensionalen Eingabedaten und verwendet eine nichtlineare *Mapping*-Vorschrift.

## 7.1 Ergebnisse der Approximationsverfahren

Der Testdatensatz *Farbquader* besteht aus 12 Tetraedern. Die Feldverteilung, die über diesem Gitter definiert ist, ist vektoriell: An jedem der 12 Knoten des Datensatzes ist eine bestimmte Farbe definiert. Innerhalb der Tetraeder wird eine lineare Interpolation dieser Farbvektoren angenommen. Der Absorptionskoeffizient wird für das gesamte Volumen als konstant angenommen (Abbildungsmethode `VRxConstAlphaShading`), als Vektor der Emissionskoeffizienten wird direkt der Wert der Feldverteilung übernommen (Abbildungsmethode `VRxIdentColorShading`).

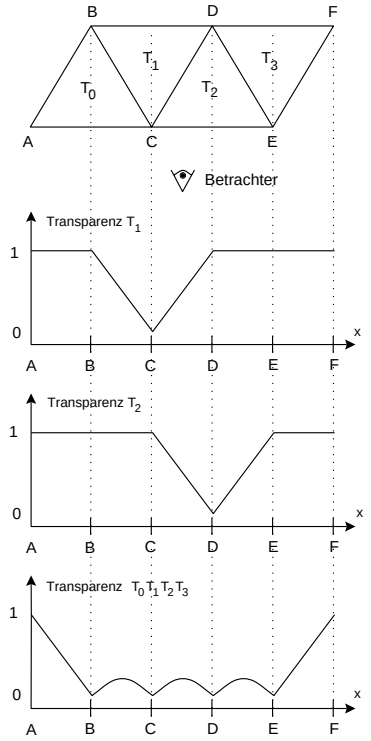
Abbildung 6.1 zeigt die Ergebnisse für drei verschiedene Näherungsverfahren, wobei der konstant gewählte Absorptionskoeffizient, der direkt die „Undurchsichtigkeit“ des Volumens beeinflußt, ungefähr in der Mitte zwischen völliger Transparenz und Undurchsichtigkeit angesiedelt ist. Von oben nach unten sind die Ergebnisse der *exakten Methode*, der *Methode interpolierter Koeffizienten* und der *Farbinterpolationsmethode*

zu sehen. Für die exakte Methode wurde der *maximale Sampleabstand* derart festgelegt, daß die längsten auftretenden Intervalle zwischen Vorder- und Rückwand der transformierten Zelle in ca. 15 Teilintervalle aufgeteilt wurden.

Bei einem Vergleich der ersten beiden Bilder erkennt man, daß die Transparenz über den beiden Volumina identisch ist, was mit den Erwartungen übereinstimmt, da die zur Integralbestimmung angewandte Approximation (*„konstante Absorption“*) in diesem Fall ja tatsächlich exakte Ergebnisse liefert, da die Absorption wirklich konstant definiert wurde. Beim Vergleich der Farbgebung zeigt sich, daß das Approximationsverfahren für die Integrale, nämlich `weightedConstEA`, durch die gewichtete Mittelwertbildung zur Bestimmung einer „pseudo-konstanten“ Emission die vordere Farbe ein klein wenig zu stark betont. Denn im Vergleich zur (exakteren) Farbgebung im oberen Bild wird der grüne Bereich in der vorderen oberen Ecke zu weit ausgedehnt.

Bei einer Betrachtung des untersten Ergebnisbildes fällt auf, daß der gleiche Absorptionskoeffizient wie bei den genaueren Approximationen hier eine sehr viel transparentere Darstellung bewirkt. Abgesehen davon ist die Transparenz über dem Volumen bei diesem eher niedrigen Absorptionskoeffizienten doch recht realistisch verteilt (relativ konstanter Bereich im mittleren Gebiet, abfallend zum linken und rechten Rand hin), obwohl sie ja linear statt exponentiell interpoliert wurde. Auch die Farbgebung stimmt grundsätzlich mit der aus Bild 2 überein.

Für einen höheren konstanten Absorptionskoeffizienten ändert sich die Situation nun allerdings schlagartig (vgl. Abbildung 6.2) Während beim Vergleich der oberen beiden Bilder die Beobachtung bezüglich der falsch gewichteten Intensitätsapproximation noch einmal bestätigt wird, aber ansonsten eine sehr hohe Übereinstimmung festgestellt werden kann, treten nun für die Farbinterpolationsmethode deutliche Artefakte zu Tage. Zunächst sei angemerkt, daß die im untersten Bild dargestellte Visualisierung die größte Opazität wiederspiegelt, die man mit dieser Methode überhaupt erreichen kann, was schon mal ein großes Manko ist. Während die Farbgebung durch die lineare Interpolation über die Zellprojektionen noch relativ gut approximiert wird, ist die Transparenz im mittleren Bereich des Bildes nun bei weitem nicht mehr so homogen, wie sie eigentlich sein sollte. Stattdessen werden plötzlich Kanten, und dadurch vor allem auch Schnittpunkte von Kanten überbetont. Im folgenden soll kurz dargestellt werden, wie es dazu kommt.




Nebenstehende Abbildung verdeutlicht schematisch, was passiert, wenn bei konstantem Absorptionskoeffizienten der integrierte Transparenzwert linear über die Zellenprojektion interpoliert wird, statt auf korrekte Weise für jeden Pixel mit einem exponentiellen Ausdruck ermittelt zu werden. Die Szene (in 2D) besteht aus vier Dreiecken zu denen der Betrachter in gleicher Ebene positioniert ist (vgl. Abbildung). Die durch die Approximationsmethode bestimmte resultierende Transparenz für Dreieck $T_1$ ist im zweiten Bild von oben dargestellt. Entsprechend für $T_2$ im dritten Bild von oben. Wenn man die Transparenzen der Dreiecke nun einander überlagert, wie dies durch den *Compositing* Schritt passiert, ergibt sich der Tranzparenzverlauf der untersten Abbildung. Man erkennt deutlich die Überbetonung der Kanten durch Gebiete niedriger Transparenz im Vergleich zum korrekten Referenzmodell, das eine konstante Transparenz zwischen B und E aufweisen müßte. In [SBM94] findet sich eine ausführliche Diskussion dieses Phänomens.

## 7.2 Abhängige Emission und Absorption

In Abbildung 6.3 ist über dem bereits beschriebenen quaderförmigen Datengitter eine Skalarwertverteilung definiert, die von links nach rechts stetig ansteigt. Diese wurde visualisiert, wobei der *Absorptionskoeffizient* proportional zum Skalarwert definiertwurde.

Im oberen der beiden Bilder wurde der Emissionskoeffizient über dem Volumen als ein Rot konstanter Intensität gewählt. Man erkennt, daß je undurchsichtiger das Volumen nach rechts hin wird, die resultierende Intensität drastisch abnimmt, da durch die höhere Absorption im rechten Bildteil weniger Intensität bis zum Auge durchdringen kann.

Da dies (wahrscheinlich) ein eher unerwünschter Effekt ist, bietet es sich an, Absorption und Emission zu koppeln, indem der Emissionskoeffizient proportional zum Absorptionskoeffizienten wächst. Das entsprechende Bild ist in der unteren Hälfte von Abbildung 6.3 zu sehen.

## 7.3 Temperaturverteilung

In Abbildung 6.4 ist die Visualisierung einer simulierten Temperaturverteilung über der Beckenregion eines Patienten während einer Hyperthermiebehandlung zu sehen. Die Absorptionskoeffizienten nehmen linear mit der Temperatur zu. Es wurden zwei verschiedene *Color Mappings* eingesetzt, wobei nach dem Vorschlag aus dem letzten

Testfall die Intensität der Emission proportional mit der Absorption zunimmt. Im oberen Bild verdeutlicht ein Farbverlauf von blau über grün und gelb hin zu rot zunehmende Temperatur. Die höchste Temperatur wird im Gebiet der Prostata erreicht, wo in diesem Fall der Tumor angesiedelt ist. Man erkennt aber auch weitere lokale Temperaturmaxima im Bereich der Leistengegend, die nach Möglichkeit minimiert werden sollten.

Das untere Bild zeigt dieselbe Temperaturverteilung bei einer anderen Farbgebung (von rot bei niedrigen Temperaturen hin zu gelb bei hohen). Die scheinbare Unstetigkeit in der Temperaturverteilung am Rande des Tumors ist ein Artefakt der durch die (ungünstige) Wahl einer stückweise linearen und an dieser Stelle ungewöhnlich stark ansteigenden Transferfunktion bei gleichzeitiger Anwendung der Approximationsmethode `interpolateCoeff` zustande kommt.

## 7.4 Nichtlineare Datenverteilung

Zum Abschluß der Testreihe soll nun ein Beispiel angeführt werden, bei dem nur die exakte Berechnungsmethode das gewünschte Ergebnisbild korrekt ermitteln kann. Über *einer* Volumenzelle wird als Vektorfeld an jedem Punkt das Quadrupel der sogenannten *baryzentrischen Koordinaten* $(b_0, b_1, b_2, b_3)$ definiert. In Abbildung 6.5 ist nun die Skalarverteilung $s = b_0 \cdot b_1 \cdot abs(b_1 - b_0)$[1] abgebildet, wobei die Absorptions- und Emissionskoeffizienten proportional zu dieser Skalarverteilung gewählt wurden.

Dieses Beispiel zeigt, daß mit der exakten Bildberechnungsmethode auch nichtlineare Datenverteilungen innerhalb einer Volumenzelle adäquat visualisiert werden können.

[1] Diese Verteilung ist eine relevante Größe beim Einsatz von Interpolationsmethoden höherer Ordnung in *Finite Elemente Methoden* (FEM)

# Kapitel 8

# Zusammenfassung

Die vorliegende Arbeit setzt sich intensiv mit dem Thema *Volume Rendering auf irregulären Gittern* auseinander.

Es wird ein ausführlicher Überblick über den aktuellen Stand der Forschung gegeben. Dabei wird auf die Wahl geeigneter mathematischer Modelle für die Darstellung semitransparenter Dichtewolken eingegangen, eine Taxonomie von auf irregulären Gittern arbeitenden *Volume Rendering Verfahren* wird vorgenommen und eine Übersicht von bisher implementierten Verfahren wird präsentiert.

Nach Analyse der Anforderungen, die an ein neu zu implementierendes Programmsystem VoRANG zum *Volume Rendering auf nichtstrukturierten Gittern* gestellt werden, wird ein eigener Ansatz zum Volume Rendering auf Tetraedergittern entwickelt, dessen zentrale Punkte ein kontrolliertes Abwägen von Bildqualität und Geschwindigkeit sowie Flexibilität und einfache Erweiterbarkeit in den Bereichen

- Repräsentierung beliebiger Datenverteilungen (Interpolation höherer Ordnung, höherdimensionale Feldverteilungen),
- Realisierung allgemeiner Abbildungsmodelle von Daten- auf Farb- und Dichtewerte
- Implementierung neuer Approximationsverfahren zur Bildberechnung

sind.

Es wird ein Einblick in die objektorientierte Struktur des Programmsystems gegeben und abschließend werden Testergebnisse präsentiert.

# Literaturverzeichnis


[Cha90] Judy Challinger. Object-Oriented Rendering of Volumetric and Geometric Primitives. Master's thesis, University of California, Santa Cruz, CA, 1990.

[Cha93] Judith Ann Challinger. *Scalable Parallel Direct Volume Rendering for Nonrectilinear Computational Grids*. PhD thesis, University of California, Santa Cruz, Santa Cruz, CA, 1993.

[DCH88] R.A. Drebin, L. Carpenter, and P. Hanrahan. Volume rendering. *Computer Graphics, Proceedings of SIGGRAPH '88*, 22(4):65–74, 1988.

[Frü94] Thomas Frühauf. Raycasting of Nonregularly Structured Volume Data. In *Eurographics '94, Computer Graphics Forum*, volume 3(13), pages C293–C303, Blackwell Publishers, Oxford, GB, September 1994.

[FvDFH90] J.D. Foley, A. van Dam, S.K. Feiner, and J.F. Hughes. *Computer Graphics, Principles and Practice*. Addison-Wesley, Reading, MA, second edition, 1990.

[Gal91] Richard S. Gallagher. Span Filtering: An Optimization Scheme For Volume Visualization of Large Finite Element Models. In *Proceedings Visualization '91*, pages 68–75, San Diego, CA, October 1991.

[Gar90] Michael P. Garrity. Raytracing Irregular Volume Data. In *Computer Graphics 24/5, San Diego Workshop on Volume Visualization , Nov. 1990*, pages 35–47, ACM SIGGRAPH, New York, NY, 1990.

[Gie92] Christopher Giertsen. Volume Visualization of Sparse Irregular Meshes. *IEEE Computer Graphics & Applications*, 12:40–48, March 1992.

[GN89] Richard S. Gallagher and Joop C. Nagtegaal. An Efficient 3-D Visualization Technique for Finite Element Models and Other Coarse Volumes. *Computer Graphics, Proceedings of SIGGRAPH '89*, 23(3):185–194, July 1989.

[HHS94] H.C. Hege, T. Höllerer, and D. Stalling. Volume Rendering, Mathematical Models and Algorithmic Aspects. Technical Report TR 93-7, Konrad-Zuse-Zentrum für Informationstechnik Berlin, Berlin, Germany, May 1994.

[Kaj86] J.T. Kajiya. The Rendering Equation. *Computer Graphics, Proceedings of SIGGRAPH '86*, 20(4):143–150, August 1986.

[KH84] J.T. Kajiya and B.P. Von Herzen. Ray Tracing Volume Densities. *Computer Graphics, Proceedings of SIGGRAPH '84*, 18(3):165–173, July 1984.

[Koy92] Koji Koyamada. Fast Traverse of Irregular Volumes. In Tosiyasu L. Kuni, editor, *Visual Computing, Proceedings of the 10th International Conference on Computer Graphics*, pages 295–312. Springer, Tokyo, 1992.

[Lev88] Marc Levoy. Display of surfaces from volume data. *IEEE Computer Graphics and Applications*, 8(3):29–37, May 1988.

[Lev90] Marc Levoy. Efficient ray tracing of volume data. *ACM Transactions on Graphics*, 9(3):245–261, July 1990.

[Luc92] Bruce Lucas. A Scientific Visualization Renderer. In *Proceedings* Visualization *'92, Boston, MA, Oct. 92*, pages 227–233, IEEE Computer Society Press, Los Alamitos, CA, October 1992.

[MBC93] Nelson Max, Barry Becker, and Roger Crawfis. Flow Volumes for Interactive Vector Field Visualization. In G. M. Nielson and Dan Bergeron, editors, *Proceedings Visualization '93*, pages 19–24, San Jose, CA, 1993.

[MHC90] Nelson Max, Pat Hanrahan, and Roger Crawfis. Area and Volume Coherence for Efficient Visualization of 3D Scalar Functions. In *Computer Graphics 24/5, San Diego Workshop on Volume Visualization , Nov. 1990*, pages 27–33, ACM SIGGRAPH, New York, NY, 1990.

[Nee90] Henry Neeman. A Decomposition Algorithm for Visualizing Irregular Grids. In *Computer Graphics 24/5, San Diego Workshop on Volume Visualization , Nov. 1990*, pages 49–56, ACM SIGGRAPH, New York, NY, 1990.

[Neu93] Ulrich Neumann. *Volume Reconstruction and Parallel Rendering Algorithms: A Comparative Analysis*. PhD thesis, University of North Carolina at Chapel Hill, Chapel Hill, NC, 1993.

[Ous94] John K. Ousterhout. *Tcl and the Tk Toolkit*. Addison-Wesley, Reading, MA, may 1994.

[RW92] Shankar Ramamoorthy and Jane Wilhelms. An Analysis of Approaches to Ray-Tracing Curvilinear Grids. Technical Report UCSC-CRL-92-07, University of California, Santa Cruz, CA, 1992.

[Sab88] P. Sabella. A rendering algorithm for visualizing 3d scalar fields. *Computer Graphics, Proceedings of SIGGRAPH '88*, 22(4):51–58, 1988.

[Sak90] Georgios Sakas. Fast Rendering of Arbitrary Distributed Volume Densities. In *Proceedings Eurographics '90*, pages 519–530, Elsevier Science Publishers B.V., 1990.

[SBM94] Clifford Stein, Barry Becker, and Nelson Max. Sorting and Hardware Assisted Rendering for Volume Visualization . In *1994 Symposium on Volume Visualization, Washington, D.C.*, pages 83–90, ACM SIGGRAPH, New York, NY, October 1994.

[Spe90] Don Speray. Volume Probes: Interactive Data Exploration on Arbitrary Grids. In *Computer Graphics 24/5, San Diego Workshop on Volume Visualization , Nov. 1990*, pages 5–12, ACM SIGGRAPH, New York, NY, 1990.

[ST90] Peter Shirley and Allan Tuchman. A Polygonal Approximation to Direct Scalar Volume Rendering. In *Computer Graphics 24/5, San Diego Workshop on Volume Visualization , Nov. 1990*, pages 63–70, ACM SIGGRAPH, New York, NY, 1990.

[TSM94] Behnam Tabatabai, Emanuela Sessarego, and Harald Mayer. Volume Rendering on Non-regular Grids. In *Eurographics '94, Computer Graphics Forum*, volume 3(13), pages C247–C258, Blackwell Publishers, Oxford, GB, September 1994.

[UK88] C. Upson and M. Keeler. V-BUFFER: Visible Volume Rendering. *Computer Graphics, Proceedings of SIGGRAPH '88*, 22(4):59–64, August 1988.

[Use91] Sam Uselton. Volume Rendering for Computational Fluid Dynamics: Initial Results. Technical Report RNR-91-026, NAS-NASA Ames Research Center, Moffet Field, CA, 1991.

[VGW93] Allen Van Gelder and Jane Wilhelms. Rapid Exploration of Curvilinear Grids Using Direct Volume Rendering. In G. M. Nielson and Dan Bergeron, editors, *Proceedings Visualization '93*, pages 70–77, San Jose, CA, 1993.

[WCAR90] Jane Wilhelms, Judy Challinger, Naim Alper, and Shankar Ramamoorthy. Direct Volume Rendering of Curvilinear Volumes. In *Computer Graphics 24/5, San Diego Workshop on Volume Visualization , Nov. 1990*, pages 41–46, ACM SIGGRAPH, New York, NY, 1990.

[Wes90] Lee Westover. Footprint Evaluation for Volume Rendering. *Computer Graphics*, 24(4):367–376, 1990.

[Wil92a] Jane Wilhelms. Pursuing Interactive Visualization of Irregular Grids. In *Workshop Visualisierung - Rolle von Interaktivität und Echtzeit*, pages 1–9, GMD Sankt Augustin, Schloß Birlinghoven, June 1992. Preprints.

[Wil92b] Peter Williams. Visibility Ordering Meshed Polyhedra. *ACM Transactions on Graphics*, 11(2):103–126, April 1992.

[Wil92c] Peter L. Williams. Interactive Splatting of Nonrectilinear Volumes. In *Proceedings* Visualization *'92, Boston, MA, Oct. 92*, pages 37–44, IEEE Computer Society Press, Los Alamitos, CA, 1992.

[Wil92d] Peter Lawrence Williams. *Interactive Direct Volume Rendering of Curvilinear and Unstructured Data.* PhD thesis, University of Illinois at Urbana-Champaign, Urbana, Illinois, 1992.

[WM92] Peter Williams and Nelson Max. A Volume Density Optical Model. In Kaufman and Lorensen, editors, *1992 Workshop on Volume Visualization*, pages 61–68. ACM SIGGRAPH, NY, 1992.

[WVG91] Jane Wilhelms and Allen Van Gelder. A Coherent Projection Approach for Direct Volume Rendering. *Computer Graphics, Proceedings of SIGGRAPH '91*, 25(4):275–284, July 1991.

# Anhang A

# Scriptsprache zu VoRANG

Die Scriptsprache zu VoRANG baut auf Tcl (Version 7.3) auf und umfaßt daher alle Sprachkonstrukte dieser allgemeinen Scriptsprache, wie zum Beispiel Steuervariablen und Schleifen (`while`, `for` und `foreach`).

Um folgende Kommandos wurde der Tcl-Kernel erweitert:

## A.1 Kommandos

### Bildangaben

`imagesize <xpixels> <ypixels>`
(Größe des Ergebnisbildes)
*Default: 512x512*

`output <basename>.<format>`
(Der Grundname des Ergebnisbildes. Dieser Name wird noch durch die aktuelle Bildnummer ergänzt. Beispiel: "`output test.rgb`" resultiert in "`test.0001.rgb`".
Die Angabe `<format>` legt das Bildformat des Ergebnisbildes fest (`tiff, rgb, rla ...`) Die verwendbaren Dateiformate werden von der *San Diego Image Library* unterstützt. Sie sind auf Seite 90 im einzelnen aufgeführt)
*Defaultwert: "*`vorang.tiff`*"*

`set_nr <image_number>`
(Setzt die aktuelle Nummer die bei der nächsten Bildausgabe in den Dateinamen integriert wird)
*Default-Anfangswert:* `0`*, dann Inkrementierung bei jeder neuen Bildberechnung.*

`background {<filename>, clear}`
(Ein strukturierter Hintergrund kann die dreidimensionale Orientierung stark erleichtern. Bei Angabe des Hintergrundbildes sollen in einer späteren Programmversion auch `rgbz`-Bilder (ohne transparente Bildteile) erlaubt sein. Das *Volume Rendering* wird dann dieser Szenerie übergelagert. Bei Angabe des Parameters `clear` anstelle eines

Dateinamens, wird das Hintergrundbild auf uniform weiß geändert)
*Defaulthintergrund: uniform weiß*

## Angaben zu den Daten

`volume <filename>`
(In der Volumendatei sind Gitter und Daten oder nur das Gitter repräsentiert. Zusammen mit den Daten wird auch die Interpolationsmethode festgelegt)
*Fehlt die Angabe eines Datenvolumens, oder werden in den Color- und Alphamapping Dateien Daten referenziert, die nicht definiert wurden, wird ein angestoßener Bildberechnungs-Vorgang mit einer Fehlermeldung abgebrochen. Das Einlesen eines neuen Datensatzes macht automatisch alle bisherigen Angaben zur Mapping-Strategie und alle Datentransformations- und Projektionsangaben hinfällig. Statt der alten Angaben (falls vorhanden) werden dann die jeweiligen Defaultwerte angenommen.*

`data [+] <filename>`
(In der hier angegebenen Datei stehen Datenverteilungen OHNE Gitter. Es muß bereits ein zu diesen Daten passendes Gitter bereitstehen. Die Daten in dieser Datei ersetzen oder ergänzen die evtl. bereits vorhandenen Daten, je nach angegebenem `'+'`.)
*Wenn die bisherigen Feldverteilungen* ersetzt *werden (kein `'+'` im Kommando), sind automatisch alle bisherigen Mapping-Angaben hinfällig und werden durch die jeweiligen Default-Methoden ersetzt.*

`save_volume <filename> [<binary>, <ascii>] [small]`
(Ohne Parameter speichert `save_volume` das aktuelle Datenvolumen im *ausführlichen Format*, welches von VoRANG am schnellsten eingelesen werden kann (Gitterpunkte, Flächen über Punktangabe, Tetraeder über Flächenangabe). Man kann angeben, ob man die Datei im ASCII- oder Binärformat abspeichern will. Spezifiziert man `small`, wird das Datenvolumen im *kompakten Format* gespeichert (nur Knoten und Tetraeder und evtl. zusätzliche datentragende Elemente). Bei großen Datensätzen kann es, auch unter Geschwindigkeitsaspekten, sinnvoll sein, kompakt abzuspeichern, da die IO-Zeiten (reines Einlesen ohne Parsing) dann geringer sind.)
Dieses Kommando kann auch zum Wandeln verschiedener Formate ineinander verwendet werden.
*Default:* `<ascii>`*, ausführliches Format.*

`save_subset <filename> <field_nr> [<binary>, <ascii>]`
(Speichert einen Teil des aktuellen Datengitters, der durch die Maskierungs-Feldverteilung `field_nr` definiert wird. Die Feldverteilung `<field_nr>` muß vom Typ `Constant` (Konstanter Wert pro Zelle) sein. Jede Zelle, die in dieser Feldverteilung einen von null verschiedenen Wert zugeteilt hat, wird beim Speichern berücksichtigt. Bei diesem Kommando wird automatisch das *kompakte Format* (s.o.) verwendet.)
*Default für optionalen Parameter:* `<ascii>`

## Angaben zu Beleuchtung und Darstellung

`addlight <red> <green> <blue> <rx> <ry> <rz>`
(Noch nicht implementiert! Hinzufügen einer unendlich entfernten Punktlichtquelle in Richtung (rx,ry,rz) vom Koordinatenursprung.
Lichtquellen werden nur dann berücksichtigt, wenn die verwendete Methode zum *Shading* (Befehl `colormap`) sie miteinbezieht. Zur Zeit ist noch keine solche Methode implementiert)
*Fehlt diese Angabe, wird eine Standardlichtquelle an der Kameraposition angenommen. Diese wird automatisch entfernt, wenn der Benutzer eigene Lichtquellen definiert.*

`removelight <n>`
(Noch nicht implementiert! Das als `<n>`'tes eingetragene Licht wird wieder entfernt)
*Eine Standardlichtquelle an der Kameraposition (s.o.) ist nicht numeriert und braucht auch nicht entfernt zu werden.*

`colormap <filename>`
(Über diese Datei wird die Abbildung von Datenwerten in Farbwerte gesteuert. Dabei kann auf mehrere *Shading Methoden* zurückgegriffen werden.
Vgl. Dateiformat zum *Color Mapping* (Anhang C, S. 100)
*Bei fehlender Angabe einer Datei wird intern eine Default Methode bereitgestellt. Eine solche Methode muß unabhängig von Typ und Dimension der zu visualisierenden Datenverteilung sein und ist daher sehr rudimentär: Konstante Emission (rot = (1 0 0)) über dem gesamte Volumen*

`alphamap <filename>`
(In der angegebenen Datei wird das Mapping von Datenwerten in Absorptionswerte spezifiziert, wofür verschiedene Methoden bereitstehen. Vgl. Dateiformat zum *Alpha Mapping* (Anhang C, S. 100)
*Bei fehlender Angabe einer Datei wird intern eine Default Methode bereitgestellt. Eine solche Methode muß unabhängig von Typ und Dimension der zu visualisierenden Datenverteilung sein und ist daher sehr rudimentär: Konstante Absorption (Wert 2) über dem gesamte Volumen*

`linedraw {on, off, <parameter>}`
(Durch dieses Kommando wird spezifiziert, ob die Linien des Gitters, auf dem die zu visualisierende Datenverteilung definiert ist, mit dargestellt werden sollen. Durch den Parameter `on` wird angegeben, daß *alle* Gitterlinien mit ausgegeben werden sollen, durch `off` wird veranlaßt, das Gitter nicht mit darzustellen. Mit der dritten Option ist eine kontrollierte Ausgabe von Linien möglich: In Abhängigkeit einer Feldverteilung des Typs *constant cell* werden nur für bestimmte Zellen die Gitterlinien mit ausgegeben.
`<parameter>` besteht aus einer Zahl und einer Zahlenliste
"`<field_nr> {<value1>, <value2>...}`".
`<field_nr>` gibt die Nummer einer Feldverteilung an, die vom Typ *constant cell* sein muß. Die Zahlen in der Liste repräsentieren die Werte, bei deren Auftreten in der

Feldverteilung die Gitterlinien für die entsprechende Zelle gezeichnet werden[1])
*Defaultwert:* `on`

## Angaben zur Kameraprojektion

`persp_camera <angle> <aspect> <near> <far>`
(Es wird eine Kamera mit perspektivischer Projektion definiert, unter Angabe der Parameter Blickwinkel, Seitenverhältnis und Abstand zu vorderen und hinteren Clipping-Ebenen. Da die Parameter `<near>` und `<far>` Abstände sind, müssen sie positiv sein.)
ACHTUNG: Für den Fall daß der Betrachterstandpunkt vor oder innerhalb des Datenvolumen liegt, kann die Sortierung der Datenzellen nicht ordnungsgemäß erfolgen und der Bildberechnungsvorgang muß abgebrochen werden.

`ortho_camera <left> <right> <bottom> <top> <front> <back>`
(Es wird eine Kamera mit orthogonaler Projektion definiert unter Angabe des Sichtvolumens relativ zum Kamerastandpunkt)

Wenn keine Angaben zum Kameratyp gemacht werden, wird defaultmäßig eine orthogonale Kamera gewählt, die als Sichtvolumen (Clipping !) die *Bounding Box* des aktuellen Datengitters annimmt. ACHTUNG: Eventuelle Gittertransformationen beeinflussen die Default-Clipping-Einstellungen NICHT. Dies kann verwirrende Ergebnisse verursachen.

## Angabe von Transformationen

`vol_scale <x> <y> <z>`
`vol_translate <x> <y> <z>`
`vol_rotate <x> <y> <z> <degree>`
(Skalierung, Translation und Rotation des Datenvolumens. Insbesondere wichtig in Schleifen für Animationssequenzen. Rotationen werden auf den Schwerpunkt der *Bounding Box* des Volumens an dessen aktueller Position bezogen)
*Die Ausmaße des Volumens sind in der Volumendatei durch die Gitterkoordinaten festgelegt. Vor Anwendung jeglicher Transformation wird das Volumen zu Beginn automatisch mit dem Schwerpunkt der* Bounding Box *in den Ursprung verlagert.*

`look_at <x> <y> <z> <refx> <refy> <refz> <ux> <uy> <uz>`
(Zusätzlich zu den geometrischen Transformationen des Datenvolumens kann auch der Ort und die Blickrichtung des Betrachters auf einen Referenzpunkt verändert werden. Mit den drei letzten Parametern wird die *up direction* der Kamera spezifiziert.)
*Default:* `0 0 0 0 0 -1 0 1 0`

[1]Diese Art der Steuerung hört sich kompliziert an, ist aber sehr funktional. Für einen Datensatz aus der Hyperthermisimulation beispielsweise kann so auf einfache Weise das Gitter für bestimmte Gewebetypen (Knochen, Muskel, Fett, etc.) ein- und ausgeblendet werden. Man kann auch beliebige *Maskierungs*verteilungen definieren, bei denen man einfach den konstanten Zellwert der „interessanten“ Zellen auf einen bestimmten Wert setzt und für diesen dann die Darstellung der Gitterlinien einschaltet

*Wenn Kameraposition und Referenzpunkt als identisch angegeben werden (was sie nicht sollten !), wird die Kameraposition beibehalten und als* Blickrichtung *die negative z-Achse angenommen (also der Referenzpunkt z.B. auf* `(Kameraposition + {0,0,-1})` *gesetzt. Wenn die* up direction *nicht orthogonal zur Blickrichtung ist, wird sie automatisch korrigiert.)*

## Berechnungsmethoden

`method {exact, interpolateCoeff, interpolateColor, test} [params]`
(Mit diesem Kommando wird eine Projektionsmethode gewählt, wobei man zwischen Genauigkeit und Geschwindigkeit abwägen kann.
Die Methode `exact` bestimmt die Intensitätsbeiträge für jeden Pixel akkurat nach dem Lichttransfermodell, wobei die Integrationsgenauigkeit über Angabe des Maximalabstandes zwischen zwei *sample points* gesteuert wird.
Die Methode `interpolateCoeff` nimmt die Integration immer über dem gesamten Integral zwischen Eintritts- und Austrittspunkt des Pixelstrahles vor und *interpoliert* außerdem die dafür benötigten Emissions- und Absorptionskoeffizienten auf den Aussenflächen der Volumenzelle statt sie über die *mapping functions* zu bestimmen.
Die Methode `interpolateColor` führt nicht mehr für jeden Punkt der Zellprojektion eine Integration durch sondern interpoliert die endgültigen Farbbeiträge über die Zellprojektion.
Die `test`-Methode hingegen bildet nur die Geometrie des Gitters (und nicht die Datenverteilung) ab, indem die Volumenzellen mit semitransparenten Zellwänden dargestellt werden. Auch Clipping findet bei dieser Methode nicht statt. Sie ist damit u.a. zum „Orten“ des Datenvolumens geeignet.
Jede Methode bekommt eigene Parameter übergeben:

```
exact :          <integration> <Schrittweite> <Hell.faktor>
interpolCoeff :  <integration> <Hell.faktor>
interpolColor :  <Hell.faktor>
test :           <Opazität>
```

Dabei gilt:

```
<integration>:   [trapez, constEA, weightedConstEA]
<Schrittweite>:  double, z.B. 0.1
<Hell.faktor>:   double, z.B. 5
```

Der Parameter `<integration>` wählt eine Integrationsmethode aus. Die Gesamtdichte $\theta_k$ wird in allen drei Fällen nach Trapezregel bestimmt, unterschiedlich werden die Intensitätsbeiträge $b_k$ angenähert (vgl. Darstellung der Approximationen in Abschnitt 6.3 ab S. 64).
Bei der Methode `exact` gibt `<Schrittweite>` den Maximalabstand (im Weltkoordinatensystem) zwischen zwei *sample points* an.
`<Hell.faktor>` ist ein Skalierungsfaktor für den Lichtintensitätsbeitrag jeder Zelle (zum Aussteuern der Helligkeit des Volumens).
`<Opazität>` gibt bei der Methode `test` den Grad der Undurchsichtigkeit aller Zellenwände an (0.0 - 1.0).

*Default für dieses Kommando:* `test 0.5`

`render_btf [{online, remote}]`
`render_ftb [{online, remote}]`
(Die Bildberechnung wird angestoßen und endet mit Abspeichern des fertigen Bildes. Beim (optionalen) Parameter `online` wird das Bild während der Berechnung online in einem Grafikfenster aufgebaut (nur auf NeXTStep-Plattform implementiert). Nach Fertigstellung des Bildes, wird einem eine Angabe über den größten Farbwert (R,G oder B) im entstandenen Bild ausgegeben. Dazu muß man wissen, daß im Framebuffer beliebig große `float` Werte erlaubt sind, für das Abspeichern des Resultatbildes aber alle Werte über 1.0 „geclippt“ werden. Um sicher zu gehen, daß man sich keine vermeidbaren Artefakte einfängt, sollte man das Bild mit dem Helligkeitsparameter des Kommandos `method` also so aussteuern, daß der ausgegebene Wert kleiner gleich 1.0 ist.
`render_ftb` ist bisher nicht vollständig implementiert.)
*Defaultwert für Parameter:* `remote`.

## A.2 Unterstützte Bildformate

Folgende Bildformate werden von der Bibliothek der *San Diego Image Tools* und damit auch vom Programm *VoRANG* unterstützt:

| | |
|---|---|
| bmp | Microsoft Windows bitmap image file |
| cur | Microsoft Windows cursor image file |
| eps | Adobe Encapsulated PostScript file |
| gif | Compuserve Graphics image file |
| hdf | Hierarchical Data File |
| ico | Microsoft Windows icon image file |
| icon | Sun Icon and Cursor file |
| iff | Sun TAAC Image File Format |
| mpnt | Apple Macintosh MacPaint file |
| pbm | PBM Portable Bit Map file |
| pcx | ZSoft IBM PC Paintbrush file |
| pgm | PBM Portable Gray Map file |
| pic | PIXAR picture file |
| pict | Apple Macintosh QuickDraw/PICT file |
| pix | Alias image file |
| ppm | PBM Portable Pixel Map file |
| pnm | PBM Portable aNy Map file |
| ps | Adobe PostScript file |
| ras | Sun Rasterfile |
| rgb | SGI RGB image file |
| rla | Wavefront raster image file |
| rle | Utah Run length encoded image file |
| synu | SDSC Synu image file |

| | |
|---|---|
| tga | Truevision Targa image file |
| tiff | Tagged image file |
| viff | Khoros Visualization image file |
| x | AVS X image file |
| xbm | X11 bitmap file |
| xwd | X Window System window dump image file |

# Anhang B

# Dateiformat: Gitter und Daten (HyperMesh)

## B.1 Übersicht

```
                                                          --+
# HyperMesh 3D ASCII|BINARY 0.6                             | Kopfzeile
                                                          --+
Materials {                                               --+
  { id=0, name="Tumor",   eps=1.2567, sigma= 986 },         | Materialien
  { id=1, name="Knochen", eps=89.345, sigma=7435 }          |
}                                                         --+
                                                          --+
nNodes = 4                                                  |
nEdges = 6                                                  | Anzahl der
nTriangles = 4                                              | Elemente
nTetrahedra = 1                                             |
nExtra = 2                                                  |
                                                          --+
Nodes   { float[3] Coordinates } = @1                     --+
Edges   { short[2] FromTo } = @2                            |
Triangles       { int[3] Nodes } = @3                       |
Triangles       { int[3] Edges } = @3a                      | Deklaration der
Triangles       { int[2] UpperLower } = @3b                 | Gittergeometrie
Tetrahedra      { int[4] Nodes } = @4                       |
Tetrahedra      { int[6] Edges } = @4a                      | (@txt : Marker)
Tetrahedra      { int[4] Triangles } = @4b                  |
                                                          --+
NodeData        { double } = @5                           --+
NodeData        { complex } = @6                            |
NodeData        { float[3] } = @7                           |
EdgeData        { float } = @8                              | Deklaration der
EdgeData        { int ClassId } = @9                        | (Roh-)Daten
TriangleData    { float } = @10                             |
TetrahedronData { float } = @11                             | (@txt : Marker)
TetrahedronData { byte Materials } = @12                    |
ExtraData       { float[5] } = @13                          |
                                                          --+
# Skalarfeld linear interpoliert                          --+
Field "myTemp" { double Temperature } = Linear(@5)          |
```

```
                                                        |
# Vektorfeld linear interpoliert                        |
Field { float[3] } = Linear(@7)                         |
                                                        | Datenfelder
# Vektorfeld mit Edge-Elementen                         |
Field { double[3] EField } = EdgeElem(@8)               | (Bezug auf
                                                        |  Rohdaten,
# Skalarfeld (SAR) -- ueber Vektorfeld (edges) berechnet|  Angabe der
Field "myPower" { double Power } = EdgeElemSAR(@8,@12)  |  Interpola-
                                                        |  tionsart)
# Interpolation hoeherer Ordnung                        |
Field { float } = HigherOrder(@8,@10,@11)               |
                                                        |
# Skalarfeld mit konstanten Werten pro Tetraeder        |
Field { int Materials} = Constant(@12)                  |
                                                        |
# Testfeld mit sich ueberlagernden Gaussverteilungen    |
Field { float } = GaussTest(@13)                        |
                                                      --+
# Data:
# Nodes: coordinates                                  --+
@1                                                      |
0.0 0.0 1.0                                             |
0.0 1.0 1.0                                             |
1.0 1.0 1.0                                             |
0.0 1.0 0.0                                             |
                                                        |
# edges: Nodes 1&2                                      |
@2                                                      | Auflistung der
3 2                                                     |
2 4                                                     |
1 3                                                     | (Roh-)Daten
3 4                                                     |
2 1                                                     |
1 4                                                     | fuer jeden
                                                        | aufgetretenen
# triangles: Nodes 1&2&3                                | Marker
@3                                                      |
3 2 4                                                   |
1 3 4                                                   |
2 1 4                                                   |
1 2 3                                                   |
                                                        |
@3a                                                     |
...                                                     |
                                                        |
# usw. fuer alle marker ...                            |
...                                                     |
...                                                     |
                                                        |
@13                                                     |
1 1 1 0.5 2.5                                           |
3 3 3 0.5 5.0                                           |
                                                      --+
```

## B.2 Formatbeschreibung

### Dateiheader

Das hier beschriebene Dateiformat enthält zunächst eine Art Vorspann, der in jedem Fall ASCII-kodiert ist, und aus dem alle Einzelheiten über das File ersichtlich sind. Dazu gehören die Größe des Gitters, ob das Gitter selbst oder nur die Daten kodiert sind, die verwendeten Datentypen usw. Die Datei wird zeilenweise interpretiert. Kommentarzeilen sind durch ein `#` in der ersten Spalte gekennzeichnet. Leerzeilen werden ignoriert. Die gesamte Datei wird durch einen Kommentar der folgenden Art eingeleitet:

`# HyperMesh` ⟨*dimension*⟩ ⟨*encoding*⟩ ⟨*version*⟩

Dabei kann *dimension* wahlweise `2D` oder `3D` und *encoding* `ASCII` oder `BINARY` sein. *version* bezeichnet die Versionsnummer, zur Zeit `0.7`.

### Materialblock

Mitunter sollen an bestimmte Datenwerte bestimmte Bedeutungen gebunden werden (z.B. Zuordnung Bytewert - Gewebetyp). Daher kann zu Beginn der Gitterdatei ein Aufzählungstyp definiert werden: `Materials`.

Deklariert werden verschiedene Materialien in der folgenden Blockanweisung:

`Materials {`
⟨*material_definition 1*⟩`,`
⟨*material_definition 2*⟩`,`
`...`
`}`

Eine Materialzeile ⟨*material_definition*⟩ hat den folgenden Aufbau:

`{ id=`⟨*byte_value*⟩`, name="`⟨*string*⟩`", [eps=`⟨*float*⟩`, sigma=`⟨*float*⟩ `...`
`]}`

Die Angabe einer Identifikationsnummer und eines Namens für ein Material ist obligatorisch. Optional können noch beliebig viele andere ⟨*name*⟩=⟨*value*⟩ Paare aufgeführt werden, in obigem Fall werden zwei Gewebetypkonstanten (`eps` und `sigma`) definiert.

Man verwendet diesen Enumerationstyp, indem man ein Feld von Identifikationsnummern definiert. Materialwerte kann man überall dort verwenden, wo man auch den Datentyp `byte` einsetzen kann. Man kennzeichnet ihren Einsatz durch die Angabe des *denominators* `Materials` hinter dem Datentyp:

```
byte Materials
```

## Angaben zur Gittergeometrie

### Anzahl der einzelnen Elemente

Zunächst wird die Anzahl der Knoten, Kanten, Dreiecke und Tetraeder im Gitter durch folgende Zeilen festgelegt:

`nNodes` = ⟨*Number of Nodes*⟩

`nEdges` = ⟨*Number of Edges*⟩

`nTriangles` = ⟨*Number of Triangles*⟩

`nTetrahedra` = ⟨*Number of Tetrahedra*⟩

`nExtra` = ⟨*Number of ExtraData Blocks*⟩

*Extra Data* sind solche Datenangaben, die nicht an ein geometrisches Element im Gitter gebunden sind. `nExtra` gibt an, wieviele solche Datenelemente neben dem Gitter aufgeführt werden sollen (Defaultwert: 1).

Es müssen nicht Angaben zu allen 4 Arten von Gitterelementen und *Extra Data* gemacht werden, sondern nur zu denjenigen, auf die später Bezug genommen wird. Wird keine Gittergeometrie mit abgespeichert, sondern nur Daten, so muß die Anzahl derjenigen Elemente angegeben werden, auf denen Daten definiert werden.

### Gittergeometrie

Nachdem festgelegt ist, wieviele Elemente verschiedenen Typs das Gitter enthält, kann durch die folgenden Anweisungen die räumliche Anordnung dieser Elemente beschrieben werden.

Wenn das File Gitterpunkte enthält, so wird darauf wie folgt hingewiesen:

`Nodes { ⟨type⟩ Coordinates } =` ⟨*marker*⟩

Hier bezeichnet *type* den Datentyp, durch den die Gitterpunkte gegeben sind. Folgende Basisdatentypen stehen bereit und haben die angegebenen Wertebereiche:

| | |
|---|---|
| byte | 0...255 |
| short | -32768...32767 |
| int,long | $-2^{31}...(2^{31}-1)$ |
| float | ⟨*single precision real*⟩ |
| double | ⟨*double precision real*⟩ |

Von allen diesen Basistypen dürfen auch Arrays beliebiger Größe spezifiziert werden (z.B. `short[5]`). In der Regel werden Gitterpunkte in 3D mit `float[3]` oder `double[3]` spezifiziert werden.
*marker* ist ein Verweis auf eine nachfolgende Stelle in der Datei, an der dann die Gitterpunkte wirklich abgelegt sind. Verweise haben die Form `@`⟨*text*⟩ und müssen eindeutig sein.

Entsprechende Zeilen wie für die Knotenpunkte gibt es auch für Kanten, Dreiecke und Tetraeder:

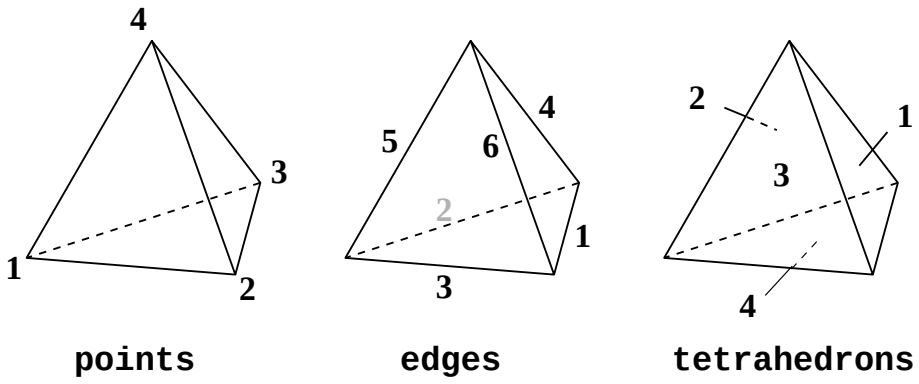



Abbildung B.1: Reihenfolge der Indizes für verschiedene Kodierungsarten.

`Edges` { ⟨*type*⟩ `FromTo`} = ⟨*marker*⟩

`Triangles` { ⟨*type*⟩ `[Nodes, Edges, UpperLower]`} = ⟨*marker*⟩

`Tetrahedra` { ⟨*type*⟩ `[Nodes, Edges, Triangles]`} = ⟨*marker*⟩

Für die Kanten muß *type* ein Feld eines ganzahligen Datentyps der Länge 2 sein, zum Beispiel `long[2]`. Die beiden Einträge werden als Indizes auf Gitterpunkte interpretiert. Für Dreiecke werden Felder der Länge 3 oder 2 und für Tetraeder solche der Länge 4 oder 6 verwendet.

Die Art und Weise, wie die Geometriedaten für Dreiecke und Tetraeder interpretiert werden, wird durch die Angabe der Kodierungsart festgelegt:

Hinter der Typangabe von `Triangles` steht z.B. eines der Schlüsselwörter `Nodes`, `Edges` oder `UpperLower`. Im Falle von `Nodes` und `Edges` muß ⟨*type*⟩ ein 3-dimensionales Feld von ganzzahligen Werten sein und der Marker verweist auf eine Auflistung von Zahlentripeln, die als Indizes auf Gitterpunkte bzw. als Indizes auf Kanten interpretiert werden. Im dritten Fall (`UpperLower`) werden in dem Datenblock auf den der Marker verweist, für jede Dreiecksfläche die Indizes der beiden angrenzenden Tetraeder aufgelistet (null bei Außenflächen). Der Datentyp hierzu muß entsprechend ein zweidimensionales Feld eines ganzzahligen Typs sein.

In Analogie zu den Dreiecksflächen wird hinter der Typangabe von `Tetrahedra` angegeben, ob die Tetraeder über Knoten-, Kanten- oder Flächenreferenzen definiert werden. Die Reihenfolge, in der die Indizes dann im Datenblock anzugeben sind, ist in Abb. B.1 dargestellt.

## Angaben zu den Gitterdaten

### Deklaration der (Roh-)daten

Falls die Volumendatei Datenwerte auf den Gitterpunkten enthält, so wird auf diese durch eine Zeile der folgenden Art hingewiesen:

`NodeData` { ⟨*type*⟩ [⟨*denom*⟩] } = ⟨*marker*⟩

Hier kann *type* ein beliebiger Datentyp oder ein Feld davon sein. Für ein Skalarfeld kann man z.B. einfach `double` angeben, für ein Vektorfeld beispielsweise `float[3]`.

Für Daten auf Kanten, Dreiecken und Tetraedern und für `ExtraData` gibt es analoge Zeilen:

`EdgeData` { ⟨*type*⟩ [⟨*denom*⟩] } = ⟨*marker*⟩

`TriangleData` { ⟨*type*⟩ [⟨*denom*⟩] } = ⟨*marker*⟩

`TetrahedronData` { ⟨*type*⟩ [⟨*denom*⟩] } = ⟨*marker*⟩

`ExtraData` { ⟨*type*⟩ [⟨*denom*⟩] } = ⟨*marker*⟩

Mittels der optionalen Angabe ⟨*denom*⟩ kann man die Rohdaten noch durch einen beliebigen Bezeichner klassifizieren. Die Verwendung solcher Bezeichner kann zwischen den verschiedenen Benutzern dieses Austauschformates verabredet werden. Z.B. verweist der Bezeichner `Materials` darauf, daß die Werte des aktuellen Datentyps als Materialien interpretiert werden sollen (s. Abschnitt zum Materialblock, S. 94).

Es dürfen beliebig viele Datenzeilen in einer Datei aufgeführt sein, jede mit einem Vorwärtsverweis auf die Daten selbst. Auch dürfen Datenzeilen gleicher Art wiederholt auftreten. Auf diese Weise kann ein Gitter mit beliebig vielen auf ihm definierten Datensätzen abgespeichert werden.

### Deklaration der Datenfelder

Die Bezeichner `NodeData`, `EdgeData`, `TriangleData` und `TetrahedronData` beschreiben Datensamples auf geometrischen Elementen ohne besonderen semantischen Bezug. Was jedoch durch das Programm VoRANG visualisiert wird, sind *Feldverteilungen*, d.h. Datenfelder, die an jedem infinitesimalen Ort im Volumengitter einen bestimmten Wert annehmen. Die oben genannten Rohdaten sind nur als Stützpunkte oder Beschreibungsparameter zur Definition solcher Feldverteilungen über dem Datengitter zu verstehen. Zur Deklaration von Feldverteilungen unter Angabe einer beliebigen Interpolationsmethode steht die Anweisung `Field` bereit:

`Field ["`⟨*string*⟩`"] { `⟨*type*⟩ `[`⟨*denom*⟩`] } = `⟨*Interpol*⟩`(`⟨*marker*⟩`[, `⟨*marker*⟩`, ...])`

⟨*string*⟩ ist ein optionaler Name für die jeweilige Feldverteilung. Die Typdeklaration in den geschweiften Klammern hat exakt die gleiche Syntax wie bei der Rohdatendeklaration und legt den Typ der Feldverteilung fest, also den Rückgabetyp der Interpolationsfunktion. Hinter dem Gleichheitszeichen wird nun diese Interpolationsfunktion spezifiziert. Dies geschieht über einen Bezeichner und eine Liste von Markern. Denn eine solche Interpolationsfunktion bezieht sich auf einen oder mehrere Rohdatenblöcke. Jeder Marker, der in einer `Field` Deklaration verwendet wird, muß bereits vorher in einer Rohdatendeklaration aufgetaucht sein. Und der Typ dieses Datenblocks muß genau zur jeweiligen Interpolationsmethode in der Felddeklaration passen. Der gleiche Marker, also auch die gleiche Rohdatendefinition, kann in beliebig vielen Felddeklarationen verwendet werden.

Folgende Interpolatoren können gewählt werden:

`Linear(`⟨*marker*⟩`)`:
Feldverteilung mit Sample-Werten auf den Knoten. Für alle anderen Gitterkoordinaten soll der Datenwert durch lineare Interpolation über das betroffene Tetraeder bestimmt werden (bel. Skalar- oder Vektordaten).

`Constant(`⟨*marker*⟩`)`:
Datenfeld mit Sample-Werten auf den Tetraedern. Die Datenverteilung wird als für das jeweilige Tetraeder konstant angenommen (bel. Skalar- oder Vektordaten).

`GaussTest(`⟨*marker*⟩`)`:
Diese Interpolationsfunktion erwartet die Angabe von Rohdaten des Typs `ExtraData`. Und zwar werden in dem von ⟨*marker*⟩ referenzierten Datenblock mehrere (`nExtra`) sich überlagernde Gaussverteilungen definiert. Der Datentyp der Rohdaten muß `float[5]` sein, und zwar werden der Reihe nach x-, y- und z-Koordinate, der Verteilungsparameter $\rho$ und die Amplitude einer Gaussverteilung aufgelistet.
Auf diese Weise stellt diese Interpolationsfunktion eine analytisch definierte Testdatenverteilung zur Verfügung, die im Grunde unabhängig vom Datengitter ist.

`EdgeElem(`⟨*marker*⟩`)`:
Ergebnis einer FEM Rechnung mit Lösungen auf den Kanten. Skalare Werte auf den Kanten implizieren Interpolation für ein *Vektor*feld über dem gesamten Datengitter.

`EdgeElemSAR(`⟨*marker1*⟩`,`⟨*marker2*⟩`)`:
Ableitung einer Leistungs-Verteilung (SAR) aus dem Ergebnis einer FEM Rechnung mit Lösungen auf den Kanten. Skalare Werte auf den Kanten implizieren Interpolation für ein *Vektor*feld über dem gesamten Datengitter. Unter Zuhilfenahme der Materialeigenschaften der jeweiligen Volumenzellen (zweiter Marker !) wird hieraus ein Wert der spezifischen Absorptionsrate (SAR) ermittelt.

`HigherOrder(`⟨*marker1*⟩`,`⟨*marker2*⟩`,`⟨*marker3*⟩`)`:
Vorstellbare Interpolation höherer Ordnung, die sich sowohl auf Daten auf Knoten (⟨*marker1*⟩), Daten auf Kanten (⟨*marker2*⟩) als auch auf Daten auf Tetraedern (⟨*marker3*⟩) bezieht.

Die letzten drei Interpolationsfunktionen (`EdgeElem`, `EdgeElemSAR` und `HigherOrder`) sind im Projekt VoRANG zwar berücksichtigt, aber bisher nicht vollständig implementiert.

## Allgemeines

Falls die Datei zum Beispiel keine Gitterpunkte oder keine Daten auf Kanten enthält, so fehlen einfach die entsprechenden Zeilen, hier `Nodes` und `EdgeData`. Zusätzlich kann jedoch auch explizit darauf hingewiesen werden durch

```
Nodes = NULL
```

oder

```
EdgeData = NULL
```

Wenn die Gitterpunkte fehlen, so darf die Datei keine Kanten, Dreiecke oder Tetraeder mit Indizes auf Gitterpunkte enthalten. Es ist jedoch zum Beispiel denkbar, daß Kanten und Dreiecke fehlen, während Punkte und Tetraeder mit Indizes auf Punkte vorhanden sind.

## Datensektion

Mit den bisher vorgestellten Zeilen ist der Vorspann im File abgeschlossen. Es folgen jetzt noch die wirklichen Koordinaten, Indizes und Datenwerte. Diese werden mit den zuvor definierten Markern eingeleitet und folgen unmittelbar nach der Zeile, die den Marker enthält. Im Fall einer ASCII-Datei könnten zum Beispiel Gitterkoordinaten wie folgt abgespeichert werden:

```
@1
0.000 0.000 0.000
1.000 0.000 0.000
0.000 1.000 0.000
...
```

Im Fall einer ASCII-Datei enthält jede Zeile genau einen Eintrag. Für binärkodierte Daten wird das Format einer „Standardworkstation“ verwendet, also *big-endian* Byte-Ordering, sowie Gleitkommazahlen in IEEE-Darstellung. Es ist empfehlenswert, bei Binärdateien unmittelbar nach dem Vorspann eine Kommentarzeile einzuschieben, die einen Seitenvorschub *Ctrl-L* enthält. Auf diese Weise kann der Vorspann problemlos mit `more` gelesen werden.

# Anhang C

# Dateiformat: Color- und Alpha-Mapping

## C.1 Übersicht

```
                                                                --+
# VoRANG Colormap 1.0                                             | Kopfzeile
                                                                --+
DataField 0                                                     --+
CategoryField 1                                                   | Feldreferenzen
                                                                --+
                                                                --+
SIMPLE_EMITTER field:global from:0 to:6 @1                        |
SIMPLE_EMITTER field:global from:7 to:9 @1                        | Shading-Methoden
CONST_CELL field:1 from:6 to:7 @2                                 |
                                                                --+
@1                                                              --+
0 0 0       0                                                     |
0 0 0       5                                                     | Shading-
0.5 0.1 0   7                                                     | Parameterblock 1:
1 0.3 0     10                                                    |
10 7 0      12.0                                                  | RGB-Tabelle
15 12 0     12.2                                                  |
25 25 0     12.3537998                                            |
                                                                --+
@2                                                              --+
0 1 5       6                                                     | Parameterblock 2
                                                                --+
```

## C.2 Formatbeschreibung

Es wird je eine Datei für das Farb- und für das Alpha-Mapping bereitgestellt.

## Header

Die erste Zeile enthält einen Kommentar, der auf den Typ der Datei verweist:

`# VoRANG Colormap` ⟨*version*⟩ bzw.:

`# VoRANG Alphamap` ⟨*version*⟩

Die aktuelle Versionsnummer ist in beiden Fällen 1.0.

## Feldreferenzen

Zunächst muß festgelegt werden, welche Datenverteilung überhaupt visualisiert werden soll. Denn im Dateiformat für die Definition von Gitter und Datenverteilungen (HyperMesh) können, eingeleitet durch das Schlüsselwort `Field`, viele verschiedene Feldverteilungen gleichzeitig definiert werden (vgl. B.2, S. 97). Beispiel:

```
Field "myTemp" { double Temperature } = Linear(@5) # Feldverteilung 0
Field { float[3] } = Linear(@7)                    # Feldverteilung 1
Field { int Materials} = Constant(@12)             # Feldverteilung 2
```

Diese werden intern nach der Reihenfolge ihrer Auflistung in der Volumendatei von 0 bis $n$ durchnumeriert.
In den Mapping-Dateien wird nun mittels der Zeile

`DataField` ⟨*nr*⟩

die Feldverteilung ⟨*nr*⟩ global für die Visualisierung zugrunde gelegt.

Nun ist es durchaus möglich, mehrere Shading-Methoden bei der Berechnung eines Bildes kombiniert einzusetzen. Zu diesem Zwecke muß ein Mechanismus bereitstehen, mit dem kontrolliert zwischen verschiedenen Methoden umgeschaltet werden kann. Dieses Umschalten geschieht in den Mapping-Dateien über die ausgezeichnete Datenverteilung `CategoryField`. Die (optionale) Zeile

`CategoryField` ⟨*nr*⟩

bezeichnet diejenige Feldverteilung, anhand derer Werte die eine oder die andere Shading-Methode zum Einsatz kommt. Diese Feldverteilung muß skalar sein.

## Shading-Methoden

Die für die Bildberechnung eingesetzten Shading-Methoden werden durch Zeilen der folgenden Art festgelegt:

`<method> [field:{`⟨*nr*⟩`,global}] [from:`⟨*value*⟩ `to:`⟨*value*⟩`] `⟨*marker*⟩

Mit `<method>` wird eine der Shading Methoden angewählt, die im System *VoRANG* durch Subclassing von `VRxColorShading` und `VRxAlphaShading` bereitgestellt wurden. In der derzeitigen Version sind das für das *Color-Shading*:

`CONSTANT`, `IDENTITY`, `SIMPLE_EMITTER`, `ABSORPTION_WEIGHTED` und `CONST_CELL`

Und für das *Alpha-Shading*:

`CONSTANT`, `SIMPLE_ABSORBER`, `BARY_SHADING` und `CONST_CELL`

Für den Fall, daß die Feldverteilung `DataField` mit *einer* einheitlichen Methode visualisiert werden soll, wird nur eine Methodenzeile obiger Art in der Mapping Datei aufgeführt. Nur wenn mehrere Methoden in Kombination eingesetzt werden sollen, treten mehrere Methodenzeilen auf. Und auch nur dann machen die Deklaration für `CategoryField` und die Angaben `field:` und `from:...to:` Sinn:
Über die Angabe `field:` kann lokal für die aktuelle Methode die zu visualisierende Feldverteilung geändert werden. Ist dies erwünscht, sollte hier eine andere Nummer als für `DataField` angegeben werden. Soll hingegen die globale Feldverteilung verwendet werden, sollte man die Deklaration `field` entweder einfach weglassen oder `global` spezifizieren. Es ist nicht sinnvoll, die Nummer von `DataField` zu wiederholen, weil dies ein erneutes Auswerten der Interpolationsfunktion für diese Feldverteilung bei jedem Mapping-Vorgang zur Folge hätte. Die Angabe von `global` hat also Vorteile bezüglich des Rechenaufwandes, weil das Datum, das gemappt werden soll, nicht noch einmal neu bestimmt werden muß.

Es ist vorstellbar, daß in zukünftigen Versionen Shading Methoden implementiert werden, die ihre Abbildungsstrategie von *mehreren* Feldverteilungen abhängig machen (z.B. `Colorwash` Methoden). Dafür könnte die hinter `field:` angegebene Verteilung die globale Feldverteilung ergänzen statt sie zu ersetzen.
Über die Angaben `from:`⟨*value*⟩ und `to:`⟨*value*⟩ geschieht das kontrollierte Umschalten zwischen den einzelnen Shading Methoden. ⟨*value*⟩ gibt hier Werte der „Maskierungsfeldverteilung“ `CategoryField` an. Immer wenn ein Datenwert auf Farb- oder Alpha-Werte gemappt werden soll und wenn zwischen mehreren anwendbaren Shading-Methoden entschieden werden muß, wird in der Mapping-Methode diese Feldverteilung für den aktuellen Volumenpunkt ausgewertet. Anhand des resultierenden Wertes $v$ wird dann die anzuwendende Shading Methode bestimmt: Es wird die erste aufgelistete Methode angewendet, in deren Zeile ein `[from,to[` Intervall[1] angegeben ist, das $v$ beinhaltet.
⟨*marker*⟩ sind Verweise auf nachfolgende Sektionen in der Datei, an denen die Tabellen und/oder Parameter für die durch `<method>` spezifizierte Shading-Methode aufgeführt sind, und haben die Form `@`⟨*text*⟩. Sie müssen eindeutig einen Datenblock bezeichnen, dürfen aber beliebig oft in Deklarationen verwendet werden.

## Parameterblöcke für die einzelnen Methoden

Bei der Festlegung der Shading-Methoden fanden Vorwärtsverweise auf Parameterblöcke für die jeweiligen Methoden statt. Diese Blöcke werden im folgenden beschrieben. In ihnen werden Umwandlungstabellen und/oder spezielle Parameter notiert. Jede Methode ist dabei grundsätzlich völlig frei in der Art und Notation der Parameter, die

[1] also einschließlich Wert `from` und ausschließlich Wert `to`

sie benötigt. Die Auswertung des Parameterblocks übernimmt der *Constructor* der jeweiligen Shadingklasse, der bei der Implementierung neuer Shading-Strategien entsprechend entworfen werden muß. Oft werden als Parameterblöcke Tabellen verwendet, die einen Wert der zu visualisierenden Feldverteilung auf ein beliebig großes Zahlentupel abbilden. Deshalb ist die Verwaltung solcher Tabellen in einer eigenen Klasse `VRxDataTable` realisiert, auf die von mehreren Shading-Unterklassen aus zugegriffen werden kann.

Ein Parameterblock besteht grundsätzlich aus allen Datenwerten zwischen zwei Markern oder zwischen dem letzten Marker und dem Dateiende.

Vorbemerkung zum Wertebereich der Mapping-Resultate: Die Emissions- und Absorptionswerte die von den Mapping-Methoden zurückgeliefert werden, entsprechen den in Kapitel 2 eingeführten Koeffizientenverteilungen $\eta$ und $\chi$. Es sind also Werte zwischen null und unendlich zulässig. Sinn macht jedoch nur die Ausschöpfung eines Intervalls von 0 bis etwa 30.

### Parameter für die *Color Shading* Methoden

#### CONSTANT:

Als Parameterblock werden einfach drei Gleitkommazahlen angegeben, die die Rot-, Grün- und Blaukomponente der konstanten Emission definieren. Beispiel:

```
CONSTANT @1
@1
2  1.0  0.0
```

#### IDENTITY:

Diese Methode erwartet überhaupt keinen Parameterblock, da die Werte aus der Datenverteilung direkt als Emissionsfarbe am jeweiligen Volumenpunkt interpretiert werden. Die zu visualisierende Datenverteilung muß bei Anwendung dieser Mapping Methode vom Typ `float[3]` sein. Die Angabe eines Markers erübrigt sich bei Spezifizierung dieser Methode:

`IDENTITY [field:{`⟨*nr*⟩`,global}] [from:`⟨*value*⟩ `to:`⟨*value*⟩`]`

#### SIMPLE_EMITTER:

Es wird als Parameterblock eine Tabelle angegeben, die Stützpunkte für eine stückweise lineare Abbildung von Datenwerten auf RGB-Tripel enthält. Die ersten drei `float` Werte einer Zeile bestimmen das RGB-Tripel, dahinter folgt die Angabe des zugehörigen Datenwertes, der von Zeile zu Zeile streng monoton ansteigen muß. Die Farbwerte zu Datenwerten, die zwischen zwei Stützpunkten liegen, werden linear interpoliert. Ein Datenwert, der kleiner ist als der Datenwert der ersten Zeile wird konstant auf das RGB-Tripel der ersten Zeile abgebildet und ein Wert der größer ist als der Datenwert der letzten Zeile wird konstant auf das RGB-Tripel der letzten Zeile abgebildet. Formatbeispiel:

```
SIMPLE_EMITTER @1
@1
0    0    0       0
0.0  0.5  0.5     0.5
0.0  0.0  1.0     0.9
2    2    0       1.0
```

### ABSORPTION_WEIGHTED:

Es wird exakt dasselbe Format für den Parameterblock benutzt wie bei der Methode `SIMPLE_EMITTER`. Der Unterschied dieser Methode gegenüber `SIMPLE_EMITTER` liegt darin, daß der aus der Tabellen-Abbildung resultierende Wert noch mit dem Absorptionswert der aktuellen Volumenstelle multipliziert wird. Dies ermöglicht eine einfache Behandlung des Falles (annähernd) proportionaler Emission und Absorption: Man hält die Emission möglichst konstant (evtl. Änderung des Farbwertes aber gleiche Intensität (= größter Wert aus R,G und B)). Die Gewichtung mit der Absorption übernimmt diese Methode.

### CONST_CELL:

Im Fall dieser Methode muß die zu visualisierende Feldverteilung vom Typ `byte` sein und durch die Interpolationsart `Constant` definiert werden. Die im Parameterblock angegebene Tabelle muß dann für jeden Bytewert, der auftreten kann, ein RGB Zahlentripel definieren. Wie in den vorangehenden beiden Methoden steht in einer Zeile der Datenwert ganz hinten und das RGB-Tripel vorne. Beispiel:

```
CONST_CELL @1
@1
0 3 1.8       0
0 3 1.8       1
0 3 1.8       2
0 0 3         3
0.5 0.15 0.04 4
25 0 0        5
```

## Parameter für die *Alpha Shading* Methoden

### CONSTANT:

Als Parameterblock wird genau eine Gleitkommazahl angegeben, die den Wert der konstanten Absorption definiert. Beispiel:

```
CONSTANT @1
@1
2
```

**SIMPLE_ABSORBER:**

Diese Methode funktioniert analog zur Colormap-Methode `SIMPLE_EMITTER`. Nur wird statt eines RGB-Tripels zu Beginn jeder Tabellenzeile im Parameterblock ein einfacher Absorptionswert abgetragen. Beispiel:

```
SIMPLE_ABSORBER @1
@1
0    0
0.25 0.8
1.00 1.0
```

**BARY_SHADING:**

Diese Methode wurde als Beispiel für eine einfache benutzerdefinierte Visualisierung von höherdimensionalem Dateninput implementiert. Die Feldverteilung ist ein vierdimensionales Feld von `float` Werten und entspricht im gewählten Beispiel den baryzentrischen Koordinaten in einem Tetraeder. Es wurden fünf Methoden implementiert, um aus den Vektorelementen $v_1, v_2, v_3$ und $v_4$ einen Absorptionswert zu berechnen.

Im Parameterblock legt die erste Zahl genau die Auswahl einer dieser fünf Methode fest und die zweite Zahl ist ein Skalierungsfaktor *scale* für den resultierenden Absorptionswert $\alpha_{res}$.
Die 5 Berechnungsschemata sind im einzelnen:

1. $\alpha_{res} = scale \cdot v_1$
2. $\alpha_{res} = scale \cdot v_1 \cdot v_2$
3. $\alpha_{res} = scale \cdot v_1 \cdot v_2 \cdot abs(v_1 - v_2)$
4. $\alpha_{res} = scale \cdot v_1 \cdot v_2 \cdot v_3$
5. $\alpha_{res} = scale \cdot {v_1}^2 \cdot v_2 \cdot v_3$

Beispiel für das Format:

```
BARY_SHADING @1
@1
3 150
```

**CONST_CELL:**

Diese Methode funktioniert analog zur Colormap-Methode `CONST_CELL`. Nur wird statt eines RGB-Tripels zu Beginn jeder Tabellenzeile im Parameterblock ein einfacher Absorptionswert abgetragen. Beispiel:

```
CONST_CELL @1
@1
1.0   0
1.0   1
```

```
1.0   2
1.0   3
0.1   4
5     5
```

# Anhang D

# Das Projekt auf Diskette

Der gesamte Programmtext des Programmsystems VoRANG, sowie Testdatensätze und Beispieldateien zur *Scriptsteuerung* und zum *Color-* und *Alpha-Mapping* sind auf der am hinteren Umschlagdeckel befestigten MS-DOS Diskette der Arbeit beigefügt.

## D.1 Übersicht über die Ordner

```
VoRANG/  -------------------------- Projektverzeichnis

          DATA/    ---------------- Die HyperMesh Datens"atze

          FORMATE/ ---------------- Beschreibung der Dateiformate
                                    als Postscript-Datei

          IMAGES/  ---------------  Hintergrundbilder

          MAPPINGS/  -------------  Color- und Alpha- Mapping
                                    Dateien

          SCRIPTS/  --------------  Steuerungsskripte
                                    (f"ur Kommandozeile)

          SOURCE/   --------------  Der komplette source-code

                 FEM/     --------  Finite Elemente Code
                                    (z.B. Interpol.)

                 FlexBison/ ------  Der Parser und Lexicalizer
                                    (yacc & lex bzw. bison & flex)

                 include/ --------  Alle eigenen Header-Files


                 include_others/ -  Die fremden Header-Files

          TIMING/ ----------------  Kram, Notizen, Protokolle f"ur
                                    Testl"aufe und Zeitmessung

          VoRANGlibs/ ------------- Drei benutzte Fremd-Libraries
```

## D.2 Übersicht über die mitgelieferten Beispiele

```
farbraum.script   ------ Der transparente Farbquader. Anwendung
                         verschiedener Integrations- und Approximations-
                         verfahren.

farbraumOpaq.script ---- Der undurchsichtige Farbquader. Hier
                         sieht man krasse Artefakte bei
                         Trapezregel-Integration und grossen
                         Intervallen

teapot.script  --------- Teapot

nonLinTetra.script ----- Die nichtlineare Verteilung in EINEM
                         Tetraeder (Baryzentrische Koordinaten)

selfMade.script  ------- Ein Beispiel f"ur Artefakte bei
                         verschiedenen Integrationsverfahren

propEA.script  --------- Am Beispiel des roten Quaders wird
                         deutlich, dass es manchmal sinnvoll
                         ist, Emission und Absorption zu koppeln

schleife.script  ------- Beispiel f"ur eine Animationsschleife
                         (nonLinTetra, unsymmetrisches Rot.Zentrum)

HyperRed.script  ------- Temperaturverteilung mit Farbe von
                         Rot nach Gelb

HyperGreen.script ------ Temperaturverteilung mit Colormap von
                         Blau nach Gr"un nach Gelb nach Rot

HyperKombi.script  ----- Der Hyperthermie Datensatz.
                         Knochen als graublauer Block,
                         drumherum rote Temperaturverteilung.
                         (SIMPLE_EMITTER & CONST_CELL)

bin2Asc    ------------- Wandlung von Bin"arformat nach ASCII

gauss.script ----------- Darstellung von drei sich "uberlagernden
                         Gaussverteilungen. Beispiel f"ur eine
                         analytische Funktion, vorget"auscht durch
                         die Interpolationsmethode.
```