Speicher- und Zeitbedarf von Methoden der Isoflächen-Extraktion

Jürgen Toelke, Dietmar Saupe

Institut für Informatik, Universität Leipzig
Augustusplatz 10-11, D-04109 Leipzig
toelke,saupe@informatik.uni-leipzig.de

Abstract

Isoflächen-Extraktion dient der Erzeugung von Hyperflächen gleicher Werte einer gegebenen Funktion $f : \mathbb{R}^n \to \mathbb{R}$. In der Praxis für Volumendaten ($n = 3$) ist die Funktion f in der Regel auf einen endlichen Definitionsbereich eingeschränkt und gegeben durch diskrete skalare Werte auf den Knoten eines regelmäßigen Gitters. Es existieren hierfür bereits Methoden: auf der einen Seite können die Marching-Cubes-Methode oder kleinere adaptive Bäume eingesetzt werden, die das Problem mit wenig Speicherbedarf lösen. Auf der anderen Seite besteht die Möglichkeit, mehr zusätzlichen Speicher in ein annähernd output-sensitives Verfahren wie die k-d-Tree- oder Intervallbaummethode zu investieren. Es besteht aber noch die Frage, wie ein Rechner mit dazwischen liegender Speichergröße optimal zugunsten der Geschwindigkeit der Isoflächen-Extraktion ausgenutzt werden kann. In diesem Artikel wird beschrieben, wie dieses Problem mit der Conditioned-Tree-Methode zu lösen ist. Der Conditioned Tree ist eine kombinierte Datenstruktur aus einer hierarchischen Raumunterteilung und Intervallbäumen, die geeigneten Raumsegmenten zugeordnet sind. Wir arbeiten hier mit den zentral unterteilten Conditioned Trees. Die Anzahl dieser Bäume hängt exponentiell von der Größe des Volumen-Datensatzes ab, es können also bei der Suche nach einem optimalen Baum nicht alle Kandidaten untersucht werden. Es ist aber möglich, bei geeigneter Wahl des Optimierungskriteriums dessen Lokalitätseigenschaft auszunutzen, die besagt, daß jeder Teilbaum eines optimalen Baums bezüglich des von ihm abgedeckten Teilproblems ebenfalls optimal ist. Wir betrachten hier zwei Eigenschaften, die ohne Zweifel diese Lokalitätseigenschaft haben: den Speicherbedarf $M(\mathcal{T})$ des Baumes $\mathcal{T}$ und den Erwartungswert der Suchzeit $T(\mathcal{T})$, wenn für die angeforderten Isowerte eine Häufigkeitsverteilung angenommen wird. Durch Einstellung eines Lagrange-Faktors λ läßt sich eine Kostenfunktion $C(\mathcal{T}) = M(\mathcal{T}) + \lambda \cdot T(\mathcal{T})$ zur Gewichtung von Speicher- und Zeitbedarf einsetzen, deren rekursive Minimierung stets den optimalen in der Klasse enthaltenen Baum bezüglich gegebenen Anforderungen an den Tradeoff Speicherplatz/Zeitbedarf ergibt.

1 Einführung

Wir gehen von einem dreidimensionalen Sample-Array (a_{xyz}) und einem Isowert γ aus und fassen in jeder Zelle des zugrundeliegenden Gitters die acht angrenzenden Gitterknotenpunkte (Voxel) zusammen. Jeder Zelle ist ein Intervall $[min, max)$ zugeordnet, das vom Minimum und dem Maximum der acht zugeordneten Datenwerte begrenzt ist. Für eine Visualisierung der zugehörigen Isofläche wird eine Liste aller Zellen benötigt, für die γ im zugehörigen Intervall $[min, max)$ liegt. Alle anderen Zellen werden als leere Zellen angesehen, die die Isofläche nicht schneiden.

Die direkte Methode der Isoflächen-Extraktion ist die Brute-Force-Methode, alle Zellen des Datensatzes zu durchsuchen. Sie wurde unter anderem von Lorensen/Cline in [7] beschrieben und benötigt außer für die Volumendaten keinen Speicher, aber eine stets lineare Rechenzeit (worst case). Schon durch den Einsatz von wenig zusätzlichen Speicher in einen Raumzerlegungsbaum, z. B. einen Octree wie in [12] von Wilhelms/van Gelder läßt sich der Vorgang beschleunigen. Bei der Extraktion der nicht-leeren Zellen werden die Teilbäume ignoriert, die zu Raumsegmenten gehören, welche keinen Teil der Isofläche enthalten. Wesentlich bessere Suchzeitergebnisse als die zerlegungsbaum-orientierten Methoden liefern die intervall-orientierten Methoden, die die gegebenen Zellen nach ihren Werten min, max geordnet in eine große Datenstruktur einordnen. Dazu gehören die k-d-Tree-Methode (beschrieben von Livnat/Shen/Johnson in [6]) und die Intervallbaum-Methode, die im Artikel [5] von Cignoni, Marino, Montani und Puppo beschrieben wird. Ebenfalls in diese Kategorie gehört die Idee von Shen, Hansen, Livnat und Johnson in [8], die Intervalle in eine Min-Max-Ebene, den sogenannten Span Space einzusortieren. Chiang und Silva präsentieren in [3] eine Variante der Intervallbaum-Methode, die den Baum aber nicht im Hauptspeicher, sondern in einer externen Datei ablegt, und deshalb nicht direkt mit den hier betrachteten Methoden vergleichbar ist. Zu ergänzen ist diese Liste noch durch die Methode von Bajaj, Pascucci und Schikore in [1], sich ausgehend von kleinen Listen repräsentativer Intervalle (Seed Sets) durch die gesuchte Isofläche zu arbeiten. Shen und Johnson stellen mit [9] schliesslich eine inkrementelle Methode der Isoflächen-Extraktion vor.

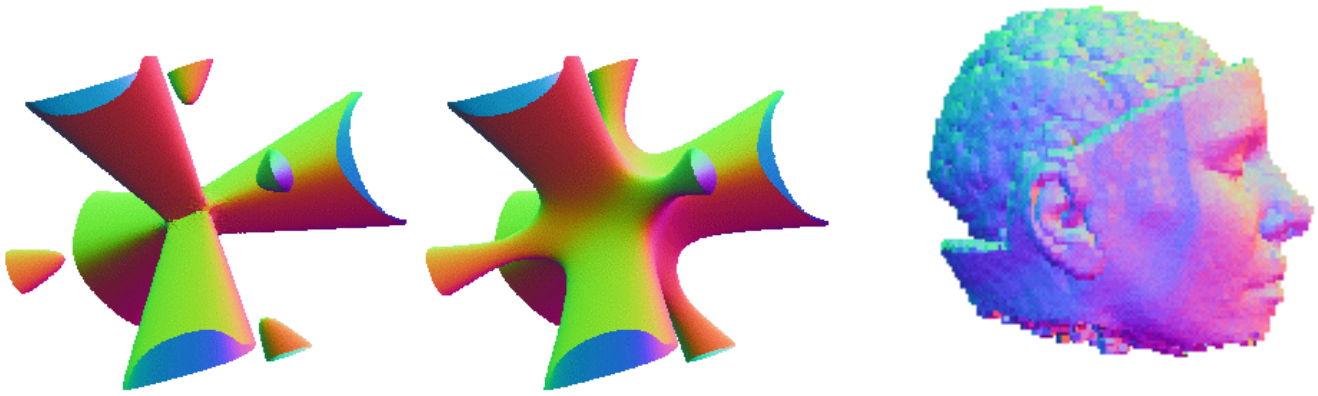

Abbildung 1: Drei Beispiel-Isoflächen: die Kummer-Oberfläche zu $f(x,y,z) = (x^2+y^2+z^2-2)^2+5(1-z-x\sqrt{2})(1-z+x\sqrt{2})(1+z-x\sqrt{2})(1+z+x\sqrt{2})$ mit zwei verschiedenen Isowerten und ein $128 \times 128 \times 84$ MR-Scan eines menschlichen Kopfes vom Chapel Hill Volume Rendering Test Dataset, aufgenommen von Siemens Medical Systems, downloaded aus [2]/brain.

	Speicherbedarf Volumendaten (MB)	Speicherbedarf Suchbaum (MB)	Speicherbedarf gesamt (MB)	Suchzeit pro Extraktion (ms)
Brute Force	81.78	0	81.78	7571
Octree	81.78	36.39	118.17	236.1
KD-Tree	81.78	324.3	460.1	57.0
Intervallbaum	81.78	809.8	891.6	5.4

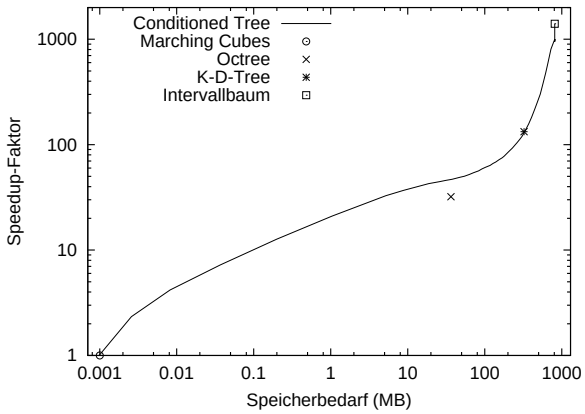

Abbildung 2: Hier sind zu den bekanntesten Methoden der Isoflächen-Extraktion jeweils der Speicherplatz- und der durchschnittliche Zeitbedarf für eine Isoflächen-Extraktion am Beispiel des Kummer-Datensatzes der Größe $350 \times 350 \times 350$ aufgeführt. Eine Auswertung des Speicher- und Zeitbedarfs der Brute-Force-Methode (Marching Cubes nach Lorensen, Cline: [7]) mit der Octree-Methode (Wilhelms, Gelder: [12]), der KD-Tree-Methode (Livnat, Shen, Johnson: [6]) und der Intervallbaum-Methode (Cignoni, Marino, ...: [5]) ergibt, daß bei steigendem Speicherplatz das selbe Extraktionsproblem in kürzerer Zeit gelöst werden kann. Wenn n die Länge des Datensatzes und k die Länge der Ausgabe bezeichnet, dann entstehen bei diesen Methoden folgende asymptotische Rechenzeiten: Brute-Force-Methode $O(n)$, Octree-Methode $O(k + k \cdot log(n))$, k-d-Tree $O(k + \sqrt{n})$ und Intervallbaum $O(k + log(n))$.

Abbildung 2 zeigt die charakteristische Leistung der Verfahren am Beispiel eines Volumendatensatzes der Größe $350 \times 350 \times 350$. Die Beschleunigung der Zellextraktion durch einen Intervallbaum beträgt ca. 1400, benötigt aber etwa 810 MB zusätzlichen Speicher. Für einen Rechner mit nur 200 MB müßte das Verfahren z. B. auf die wesentlich langsamere Octree-Methode zurückfallen. In diesem Paper füllen wir die Lücke zwischen den bekannten state-of-the-art Verfahren in optimaler Weise durch einen hybriden Algorithmus auf, das sogenannte Conditioned-Tree-Verfahren, dessen Ergebnisse durch die durchgezogene Linie des Diagramms in Abbildung 2 dargestellt ist.

Der hybride Ansatz besteht aus einem adaptiven Raumzerlegungsbaum, an dessen Blättern ein Flag anzeigt, ob der entsprechende Subblock mit der Brute-Force-Methode oder unter Verwendung eines zusätzlichen Intervallbaums zu untersuchen ist.

Im nächsten Abschnitt dieses Textes führen wir alle Methoden auf, aus denen sich die Conditioned-Tree-Methode zusammensetzt. In Abschnitt 3.1 beschreiben wir die Struktur des Conditioned Trees, und zeige dann in Abschnitt 3.2, wie ein optimaler Conditioned Tree zu einem gegebenen Parameter λ konstruiert werden kann. In Abschnitt 4 werden die

ermittelten Ergebnisse sowie einige Bilder der durchgeführten Experimente gezeigt, und schliesslich die Schlußfolgerungen in Abschnitt 5.

Die Visualisierungen der Isoflächen wurden mit einem interaktiven, mit OpenGL geschriebenen Programm erzeugt, das zugunsten der Geschwindigkeit anstelle der extrahierten Zellen einfache Quadrate (GL_Points in der jeweils passenden Größe) darstellt. Die Farbe oder der Helligkeitswert der Quadrate ergibt sich durch eine diskrete Approximation der Ebenennormalen. Bei den speichersparenden Methoden wie dem Marching-Cubes- und Octree-Verfahren wird der Hauptanteil der Rechenzeit des Visualisierungsprogramms für die Isozellen-Extraktion benötigt. Diese wird für das k-d-Tree- und Intervallbaumverfahren beschleunigt, so daß die Geschwindigkeit nun hauptsächlich vom Rendering der Isofläche abhängt. In Abbbildung 2 ist die Geschwindigkeit der wichtigsten Suchverfahren dargestellt.

2 Die untersuchten Methoden

2.1 Die Brute-Force-Methode

Die Brute-Force-Methode ist die einfachstmögliche Methode, Isoflächen zu extrahieren und hat den Vorteil, daß sie keinen zusätzlichen Speicherplatz benötigt. Das Prinzip der Brute-Force-Methode ist es, den Isowert mit dem Minimum und dem Maximum aller Zellen zu vergleichen, es wird also für einen Isowert γ der folgende Suchalgorithmus durchgeführt:

- Für alle Zellen C des Datensatzes:
  - Bestimme das Minimum $min(C)$ und das Maximum $max(C)$ der Datenwerte der acht angrenzenden Voxel
  - Wenn $\gamma \in [min(C), max(C))$, dann gib die Zelle aus

Dieser Algorithmus hat für $n \times n \times n$ große Datensätze $O(n^3)$ Rechenzeit, während der Output (bei nicht-fraktalen Flächen) nur $O(n^2)$ Größe hat. In diesem Paper geht es darum, die Suchzeit durch den Einsatz einer zusätzlichen Datenstruktur mit vorgegebenem Speicherbedarf optimal zu verkürzen.

2.2 Der adaptive Zerlegungsbaum

Der adaptive Zerlegungsbaum bietet eine Methode der Isoflächen-Extraktion, die durch den Einsatz von nur wenig Speicher eine erhebliche Beschleunigung gegenüber der Brute-Force-Methode ermöglicht. Seine Struktur verkörpert eine hierarchische Raumzerlegung des gegebenen Volumendatensatzes mit allen darin vorkommenden Zellen. Ausgehend von der Wurzel der Baums, der den ganzen Datensatz repräsentiert, wird der von jedem inneren Knoten dargestellte Teilblock in Unterblöcke zerlegt, die in die Nachfolger des Knotens eingetragen sind. Dabei enthält jeder Knoten das Minimum und das Maximum der in seinem Teilblock enthaltenen Volumendaten, sowie die erforderlichen Informationen zur Position des Teilblocks. Der adaptive Baum beschleunigt die Suche nach einem gegebenen Isowert, weil nach der relativ schnellen Untersuchung des Baums auf Intervalle, in denen γ enthalten ist, nur noch die Teilblöcke der Zerlegung untersucht werden müssen, die zu diesen Intervallen gehören.

Als Grundgerüst des Conditioned Trees werden hauptsächlich binäre adaptive Bäume benutzt, bei denen jeder Block in genau zwei Unterblöcke zerlegt wird. Es gibt hierzu die Alternative, Octrees zu verwenden, in denen jeder Teilblock würfelförmig ist und sich in acht Sub-Würfel aufsplittet. Diese lassen sich jedoch wiederum durch binäre Bäume simulieren, weil sich jede Zerlegung in acht Teile durch drei übereinandergelagerte Zerlegungen in jeweils zwei Teile realisieren lässt. Für die Wahl der Baumtiefe gibt es viele verschiedene Möglichkeiten, einige Beispiele sind hier aufgelistet:

- Der Baum endet bei einer vorher festgelegten Tiefe
- Ein Baumknoten wird zum Blatt erklärt, wenn die Anzahl der Zellen seines Volumendaten-Blocks, ein vorgegebenes Limit unterschreitet. Wilhelms und van Gelder verwenden in [12] einen Octree, dessen Blätter jeweils acht Zellen umfassen.
- Ein Knoten ist ein Blatt, wenn das Intervall $[min, max)$, das als Abbruchkriterium für die Suche dient, eine festgelegte Länge unterschreitet
- Es gibt auch Optimierungsverfahren, um den Baum zu bestimmen: Tree growing, tree pruning, die Reduktion eines großen Baumes mit dem BFOS-Algorithmus ([4]) oder das in diesem Text beschriebene Conditioned-Tree-Verfahren ohne Intervallbäume.

Eine oft angewandte Art der Blockunterteilung ist es, einen einem Baumknoten zugewiesenen Block durch eine achsenparallele Ebene in die beiden den Nachfolgerknoten zugewiesenen Teilblöcke zerlegt wird. Das hat den Vorteil, daß für die Zerlegung nur wenig Informationen gebraucht werden: die Ausrichtung der achsenparallelen Ebene, für die es nur die

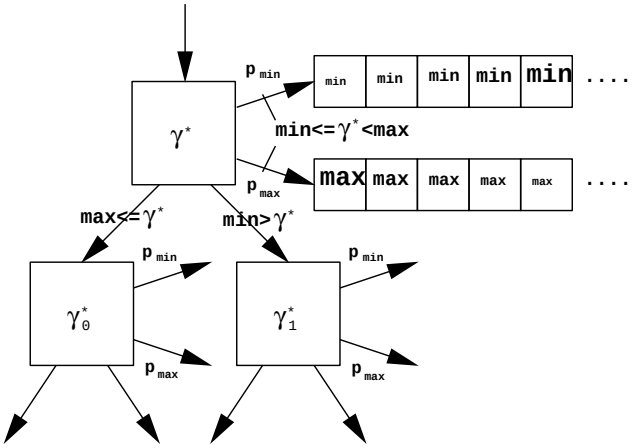

Abbildung 3: Die Struktur des Intervallbaums

drei Möglichkeiten XY, XZ und YZ gibt, und ihre genaue Position. Es kann zum Beispiel die Ausrichtung XZ und die Position 63 sein, dann wäre dadurch die zur x- und z-Achse parallele Ebene mit der Gleichung $y = 63$ definiert. Diese Spezifizierungen können auch prozedural erzeugt werden, z. B. könnte ein Block stets in der Mitte seiner längsten Seite geteilt werden. Dadurch ist die explizite Abspeicherung der Splitebenen nicht notwendig, sie kann aber trotzdem für die Einsparung von Rechenzeit hilfreich sein.

Unabhängig von der Art der Zerlegung, die einem adaptiven Baum zugrundeliegt, lässt sich folgender rekursiver Algorithmus für die Suche verwenden. Im folgenden ist die einparametrige Prozedur $AdaptiveSearch(k, \gamma)$ definiert, deren Parameter ein Baumknoten k und ein Isowert γ sind. Eine Isofläche auf dem ganzen Datensatz wird durch $AdaptiveSearch(w, \gamma)$ bestimmt, wobei w die Wurzel des Baums ist.

- Wenn nach dem Intervalltest im Knoten k $\gamma \in [min(k), max(k))$ gilt, dann
  - Wenn k ein Blatt ist, dann untersuche alle Zellen, die im k zugeordneten Block enthalten sind, ob sie in der Isofläche enthalten sind.
  - Wenn k ein innerer Knoten ist, dann rufe für alle Nachfolger k' von k $AdaptiveSearch(k', \gamma)$ auf

2.3 Der Intervallbaum

Der Intervallbaum benötigt viel zusätzlichen Speicher, weil darin jede Zelle des Volumen-Datensatzes mit ihrem Intervall gesondert gespeichert werden muss. Er stellt ebenfalls eine hierarchische Zerlegung dar, zerlegt aber nicht den Definitionsbereich, sondern den Wertebereich der Volumendaten.

Jeder Knoten des Intervallbaums enthält ausser den Zeigern auf die beiden Nachfolgerknoten (die auch Null sein können) einen Unterteilungswert γ^* und je einen Zeiger auf eine Min- und Maxliste (siehe Abb. 3). In den Intervallbaum werden alle Zellen mit ihren Intervallen $[min, max)$ in je eine der Min- und Maxlisten eingetragen:

- Wir starten in der Wurzel des Intervallbaums und haben die vollständige Liste aller Intervalle $[min, max)$
- Diese werden nach dem Unterteilungswert γ^* in drei Gruppen sortiert, abhängig davon, ob $\gamma^* \in [min, max)$, $max \leq \gamma^*$ oder $min > \gamma^*$ gilt. Die geeignete Wahl von γ^* ist eine Optimierungsfrage, die in der noch ausstehenden Dissertation näher erläutert wird. Ein einfacher Ansatz ist es, γ^* genau in der Mitte des von den Intervallen benutzten Bereichs zu wählen. Cignoni verwendet in [5] die Strategie, alle min- und max-Grenzen der Intervalle in eine monoton steigende Liste einzuordnen, in der der Wert γ^* jeweils so gewählt wird, daß er diese Liste in der Mitte teilt.
- Alle Intervalle mit $max \leq \gamma^*$ werden an den linken Nachfolgerknoten weitergegeben, und alle Intervalle mit $min > \gamma^*$ an den rechten Nachfolgerknoten. Dort werden sie in gleicher Weise verarbeitet und durch ein anderes γ^* zerlegt.

  Nachfolgerknoten, die kein Intervall erhalten, brauchen natürlich nicht mehr betrachtet zu werden.
- Alle Intervalle mit $\gamma^* \in [min, max)$ werden sowohl in die Minliste, als auch in die Maxliste des Knotens eingefügt.
- Die Minliste ist nach aufsteigendem min-Wert, die Maxliste nach absteigendem max-Wert der darin enthaltenen Intervalle zu sortieren.

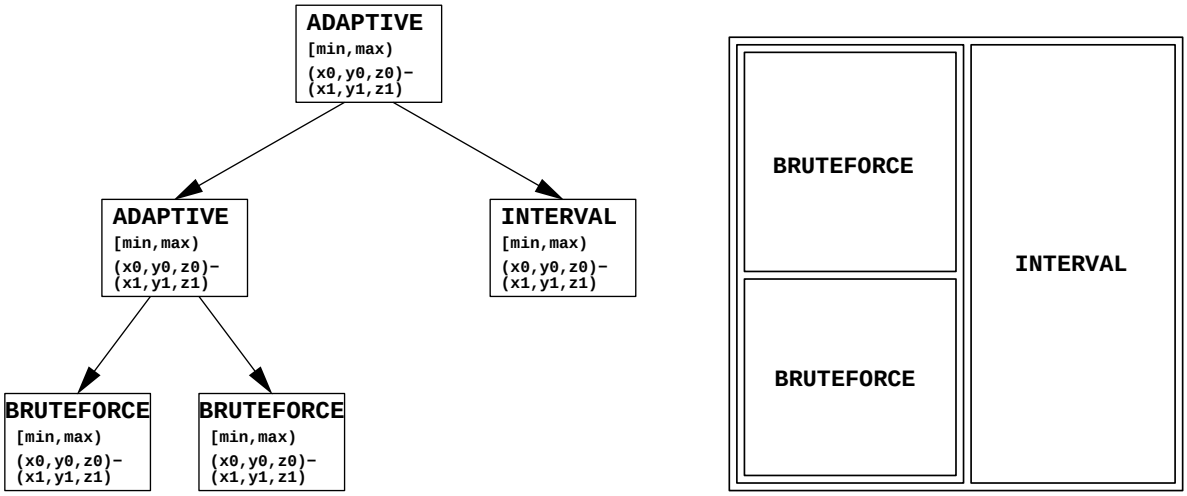

Abbildung 4: Die hybride Datenstruktur für die Conditioned-Tree-Methode. Daneben ist die zugehörige Partition der Volumendaten dargestellt.

Die Suche nach Zellen mit Intervallen $[min, max)$, in denen ein vorgegebener Isowert γ enthalten ist, kann nun folgendermaßen durchgeführt werden:

- Wir starten in der Wurzel des Baums.
- Es braucht nur ein Pfad des Baums betrachtet zu werden. Dafür wird in jedem Knoten, in dem ein Vergleich $\gamma < \gamma^*$ ergibt, der linke Nachfolgerknoten, und für $\gamma > \gamma^*$ der rechte Nachfolgerknoten besucht. Wenn $\gamma = \gamma^*$ gilt oder der Pfad an einem Blatt des Baums ankommt, ist der letzte Knoten des Suchpfads erreicht.

  Es ist durch die Vorsortierung der Intervalle sichergestellt, daß alle in der Ergebnisfläche enthaltenen Zellen mit ihren Intervallen in jeweils einer der Min-/Maxlisten vorkommen, auf die die Knoten des Suchpfads verweisen.
- Während der Suchpfad durchlaufen wird, wird in jedem Knoten die geeignete Liste untersucht. Diese ist für $\gamma < \gamma^*$ die Minliste und für $\gamma > \gamma^*$ die Maxliste. Für $\gamma = \gamma^*$ kann eine der beiden Listen komplett ausgegeben werden, weil γ in allen darin eingetragenen Intervallen liegt.
- Wir betrachten nun den Fall $\gamma < \gamma^*$. Es ist bekannt, daß γ^* in allen Intervallen $[min, max)$ der Minliste liegt, also gilt $\gamma < \gamma^* < max$. Zu untersuchen sind die Intervalle nur noch auf die Ungleichung $min \leq \gamma$. Da die Liste der Intervalle nach aufsteigendem min vorsortiert ist, gilt diese Ungleichung für die ersten i Intervalle für ein geeignetes i. Diese werden also durchlaufen und ausgegeben.
- Für $\gamma > \gamma^*$ erfolgt die Suche analog in der Maxliste.

Die Rechenzeit der Intervallbaum-Methode für n^3 große Datensätze (und nicht fraktale Flächen) ist $O(n^2)$ und annähernd output-sensitiv, d. h., die meiste Zeit wird nicht für die Suche selbst, sondern für die Ausgabe der Liste aller Isozellen benötigt. Die Methode wurde von von Cignoni, Marino, Montani und Puppo in [5] beschrieben.

3 Der Conditioned Tree

3.1 Die Struktur des Conditioned Trees

Das hier vorgestellte Conditioned-Tree-Verfahren ist eine Methode der Isoflächen-Extraktion, bei der die Datenstruktur von einem Parameter λ abhängt, der angibt, wie wichtig die Geschwindigkeit des Suchprozesses im Vergleich zum Speicherbedarf ist. Dieser kann dann zu jedem vorgegebenen Speicher- oder Zeitwert eine optimale Lösung liefern. Formell ausgedrückt ist für ein gegebenes λ die folgende Lagrange-Kostenfunktion zu minimieren:

$$C(\mathcal{T}) := M(\mathcal{T}) + \lambda \cdot T(\mathcal{T})$$

Dabei ist $\mathcal{T}$ die zu optimierende Baumstruktur, $M(\mathcal{T})$ die Größe des von ihr belegten Speichers und $T(\mathcal{T})$ der Erwartungswert der Rechenzeit, die für eine Isoflächenextraktion in dieser Struktur gebraucht wird.

Bei der Bestimmung des Erwartungswerts wird auf der Menge der möglichen Isowerte eine Verteilung angenommen, abhängig davon, wie häufig diese vom Benutzer angefordert werden. Der Einfachkeit halber werden wir im folgenden annehmen, daß die Gleichverteilung auf dem Wertebereich der Volumendaten vorliegt. Wenn eine andere Häufigkeitsverteilung bekannt ist, lässt sich die Methode verallgemeinern, indem eine der folgenden beiden Umformungen vorgenommen wird:

- Die Volumendaten werden so skaliert, daß wir eine Gleichverteilung erhalten. Das ist mathematisch bei allen stetigen Verteilungen möglich, deren Verteilungsfunktion $\gamma \mapsto P(-\infty, \gamma)$ bekannt ist; dafür wird jeder Eintrag a_{xyz} des Datensatzes durch $P(-\infty, a_{xyz})$ ersetzt.
- Die nachfolgenden Formeln zur Berechnung der erwarteten Suchzeit im Fall der Gleichverteilung lassen sich auch auf den allgemeinen Fall erweitern, wenn $P(-\infty, \cdot)$ bekannt ist. Dafür wird an jeder Stelle, an der eine Wahrscheinlichkeit $P[min, max)$ benötigt wird, anstelle von $\rho(max - min)$ die sich aus dieser Funktion ergebende Wahrscheinlichkeit eingesetzt.

Die Struktur des Conditioned Trees ist folgendermassen aufgebaut: Das Kernstück ist ein binärer adaptiver Partitionsbaum, wie er in diesem Artikel vorher schon beschrieben wurde. Die Knoten dieses Baums repräsentieren Teilblöcke der gegebenen Volumendaten und enthalten einen Flag, der aussagt, ob der Baum hier weiter verzweigt, und — falls ein Blatt vorliegt — wie der entsprechende Block weiterverarbeitet wird. Dieser Flag kann die drei Werte $BRUTEFORCE$ (0), $INTERVAL$ (1) und $ADAPTIVE$ (2) annehmen, die folgende Bedeutungen haben:

- $BRUTEFORCE$ bedeutet, dass der Knoten ein Blatt ist und dessen Teilblock gegebenenfalls mit der Brute-Force-Methode durchsucht wird.
- $INTERVAL$ bedeutet, dass der Knoten ebenfalls ein Blatt ist, von dem aber ein Zeiger auf die Wurzel eines Intervallbaums verweist. Dieser enthält alle Zellen des Teilblocks mit ihren Intervallen und kann gegebenenfalls schnell durchsucht werden.
- $ADAPTIVE$ bedeutet, dass der Knoten ein innerer Knoten ist und bearbeitet wird, indem man wie im normalen adaptiven Baum einen Intervalltest durchführt und weiter verzweigt.

Die speichersparendste Version des Conditioned Trees enthält gleich in der Wurzel den Flag $BRUTEFORCE$ und besteht nur aus dieser Wurzel; sie führt nach einer Untersuchung der Baumwurzel zum klassischen Marching-Cube-Verfahren. Die Version, die dagegen die schnellste Extraktion ermöglicht, hat in der Wurzel den Flag $INTERVAL$ und einen Zeiger auf einen Intervallbaum mit allen Volumendaten; in diesem Fall wird die ganze Datenstruktur mit dem Intervallbaum-Verfahren durchsucht. Zwischen diesen beiden Extremen "platzsparend-langsam" und "speicherintensiv-schnell" lassen sich alle möglichen Conditioned Trees in einem Speicher-Zeit-Diagramm einordnen.

Die in der Datenstruktur des adaptiven Baums vorkommenden Knoten sind folgendermassen aufgebaut:

- Ein Knoten des adaptiven Baums, der das Grundgerüst bildet, enthält die Einträge:

  $x_0, y_0, z_0, x_1, y_1, z_1$
  $flag$
  min, max
  p_1, p_2

  - $x_0, ..., z_1$ sind die Grenzen des repräsentierten Teilblocks.
  - $flag$ ist der Flag, der zwischen $BRUTEFORCE$ (0), $INTERVAL$ (1) und $ADAPTIVE$ (2) entscheidet.
  - min und max sind der minimale und der maximale Wert der Volumendaten innerhalb des Blocks. Diese werden für den Intervalltest $\gamma \in [min, max)$ verwendet, der entscheidet, ob unterhalb des Knotens weiter untersucht werden muss.
  - p_1 und p_2 sind Zeiger auf andere Datenstrukturen. Im Fall $flag = BRUTEFORCE$ werden beide ignoriert, für $flag = INTERVAL$ zeigt p_1 auf die Wurzel des Intervallbaums und für $flag = ADAPTIVE$ zeigen p_1 und p_2 auf die Nachfolger des Knotens im adaptiven Baum. Es besteht ein weiterer Lösungsansatz, abhängig vom Wert von $flag$ die Datenstruktur des Baumknotens bereits vor p_1 bzw. p_2 abbrechen zu lassen. Die Frage, ob der so zurückgewonnene Speicher die für die komplexere Fallunterscheidung benötigte Rechenzeit aufwiegt, ist noch nicht experimentiell geklärt worden.

- Ein Knoten des Intervallbaums enthält:

  $flag, \gamma^*, count$
  p_{left}, p_{right}
  p_{min}, p_{max}

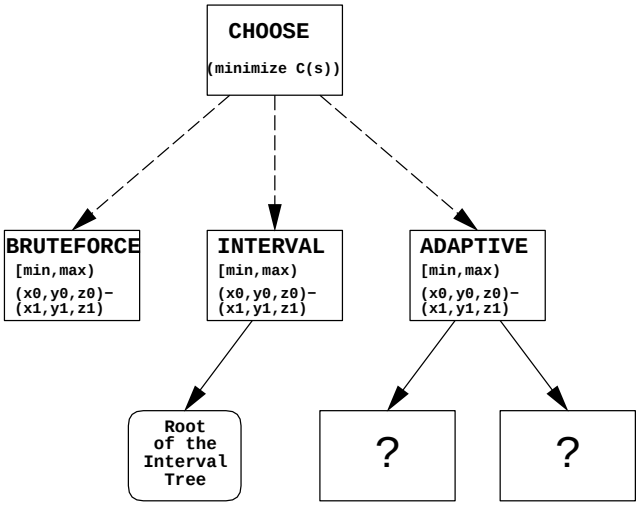

Abbildung 5: Der Conditioned Tree wird optimiert, indem in jedem Knoten des adaptiven Baums die beste der drei Alternativen gewählt wird

  - Durch den Wert von $flag$ wird entschieden, ob der Knoten einen linken Nachfolger (1), einen rechten Nachfolger (2), keinen Nachfolger (0) oder beide Nachfolger (3) hat.
  - γ^* ist der Grenz-Isowert, der in allen Intervallen der Listen p_{min} und p_{max} enthalten sowie für die Aufspaltung in einen linken und einen rechten Teil zuständig ist.
  - *count* ist die Anzahl der Elemente, die in den Listen p_{min} und p_{max} eingetragen sind.
  - p_{left} und p_{right} sind Zeiger auf die Nachfolger des Knotens.
  - p_{min} und p_{max} sind Zeiger auf das jeweils erste Element der p_{min}- und p_{max}-Liste, die die selben Intervalle nach aufsteigendem min und nach absteigendem max enthalten.

- Die Listen zu p_{min} und p_{max} von jedem Intervallbaumknoten sind Arrays von *count* Cellinfo-Strukturen, die folgende Informationen enthalten:

  x_0, y_0, z_0
  min im Fall p_{min}
  max im Fall p_{max}

  - x_0, y_0, z_0 sind die Koordinaten des obersten Eckpunkts der Zelle
  - min und max sind der minimale und der maximale Datenwert der acht Eckpunkte; diese definieren das Intervall für den Intervallbaum. Abhängig vom Vorkommen in der Min- oder Maxliste wird nur eine der beiden Intervallgrenzen für die Suche benötigt.

Im folgenden Abschnitt wird beschrieben, wie der bezüglich einer vorgegebenen Kostenfunktion optimale Conditioned Tree rekursiv bestimmt wird.

3.2 Die Optimierung der Kostenfunktion $C(\mathcal{T})$ für einen vorgegebenen Lagrange-Wert λ

Um die zugrundeliegende Kostenfunktion zu minimieren, benutzen wir rekursive Formeln für den Speicherbedarf $M(s)$ und den geschätzten Erwartungswert der Rechenzeit für eine Extraktion $T(s)$ für jeden Knoten s des Baumes $\mathcal{T}$. Dabei sind die auf Baumknoten definierten Funktionen $M(s)$ und $T(s)$ Erweiterungen von den nur für komplette Bäume bestimmten Funktionen $M(\mathcal{T})$ und $T(\mathcal{T})$. $M(s)$ ist definiert als der Speicherbedarf des Unterbaums mit der Wurzel s. $T(s)$ ist der Erwartungswert der Suchzeit während einer Extraktion, die unterhalb des Knotens s verbracht wird.

Das Prinzip der Optimierung ist es nun, für jeden Knoten die beste der drei Möglichkeiten $BRUTEFORCE$, $INTERVAL$ und $ADAPTIVE$, also diejenige mit dem kleinsten Wert für $C(s) := M(s) + \lambda \cdot T(s)$ zu finden. Für die Alternative $ADAPTIVE$ wird dabei die rekursive Voraussetzung gemacht, dass die beiden Unterbäume bereits optimiert worden sind. Da die Kostenfunktion C sich additiv auf der Struktur des Baumes verhält, führt das Prinzip, in jedem Knoten das lokale Optimum zu wählen, stets zum globalen Optimum unter allen möglichen Bäumen.

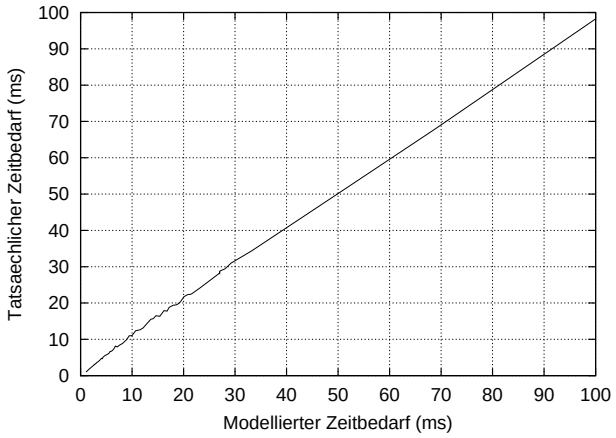

Abbildung 6: Das Diagramm zeigt die Abhängigkeit der gemessenen Zeit von der geschätzten Zeit einer Testreihe von Isoflächen-Extraktionen dar, bei denen der Parameter λ variiert wird. Hier sieht man, daß das gewählte Modell mit sämtlichen Parametern eine gute Wahl für die Schätzung der erwarteten Rechenzeit ist.

Der Einfachkeit halber betrachten wir für die Zerlegung von Teilblöcken jeweils nur eine Möglichkeit, obwohl sich ein grösserer Block auf viele Arten durch eine achsenparallele Ebene in zwei Teile trennen läßt. Von diesen betrachten wir nur die Möglichkeit, bei der der die Teilungsebene senkrecht zur längsten Kantenrichtung des Blocks steht und diesen — abgesehen von einer Rundung auf ganze Zahlen — genau in der Mitte teilt. Die unter Berücksichtigung dieser Einschränkung möglichen Conditioned Trees nennen wir zentral unterteilte Conditioned Trees.

Die Berechnung von $M(s)$ ist (fast) trivial; die Bestimmung von $T(s)$ wird im folgenden beschrieben. Eine experimentelle Bestimmung des Zeitbedarfs T durch partielle Isoflächen-Extraktion und direkte Messung wäre für die Baumoptimierung nicht sinnvoll. Diese müsste nämlich für jeden möglichen Teilbaum durchgeführt werden, was viel zu hohe Rechenzeitkosten hätte. Es gibt aber die schnellere Möglichkeit, jedem Rechenschritt, der bei der Extraktion durchgeführt wird, eine Konstante zuzuordnen, und die sich so ergebende Gesamtrechenzeit rekursiv über den Aufbau des Baumes zu berechnen. Die Konstanten dafür, $c_{CELL}, c_{QUAD}...$, lassen sich durch lineare Approximation und Minimierung des quadratischen Fehlers aus den Ergebnissen bereits ausgewerteter Conditioned Trees bestimmen. Je mehr verschiedenartige Bäume dafür verwendet werden, desto genauer wird natürlich das Ergebnis. Die lineare Approximation hat — umgerechnet in Nanosekunden Prozessorzeit — folgende Näherungswerte ergeben:

$$c_{INTINIT} = 216 \quad c_{INTCELL} = 4.38 \quad c_{CELL} = 24.2 \quad c_{ADAP} = 92.2$$

Die genaue Bedeutung dieser Konstanten wird sich im Laufe der nachfolgenden Zeitkostenanalyse ergeben. Die Linearität des Graphen im TIME-TIME-Diagramm auf Abbildung 6 zeigt, daß mit dieser Wahl der Konstanten eine hinreichend große Genauigkeit erreicht wird.

Wir nehmen nun bei der rekursiven Optimierung für jeden betrachteten Knoten an, daß der adaptive Baum oder ein Teilbaum bereits vollständig konstruiert ist. Für jeden Knoten p dieses Baums sei $T(p)$ der berechnete Erwartungswert des Zeitbedarfs für die Isowertsuche in allen Strukturen unterhalb des Knotens. Wenn r die Wurzel des Conditioned Trees ist, dann ist also $T(r)$ der Erwartungswert des Zeitbedarfs für eine Isoflächen-Extraktion abzüglich der Untersuchung der Wurzel r selbst.

Warum die Wurzel des jeweils untersuchten Teilbaums nicht mitgezählt wird, wird sich aus der rekursiven Berechnungsformel von T ergeben. Jedenfalls macht dieser Abzug nichts aus, weil der Zeitbedarf für die Untersuchung der Wurzel stets konstant ist.

Wie bereits erwähnt wurde, nehmen wir nun an, dass der Isowert γ einer Gleichverteilung unterliegt, also gilt für alle Intervallgrenzen a, b mit $\gamma_{min} \leq a < b \leq \gamma_{max}$:

$$P(\gamma \in [a,b)) = \rho(b-a) \quad \text{mit} \quad \rho := \frac{1}{\gamma_{max} - \gamma_{min}}$$

Es ergeben sich abhängig vom Typ des Knotens p für die Berechnung von $T(p)$ drei Fälle:

- **Fall 1: p ist ein $BRUTEFORCE$-Knoten**

Zur Bestimmung der erwarteten Rechenzeit müssen wir die Anzahl der im entsprechenden Block enthaltenen Zellen mit der Wahrscheinlichkeit multiplizieren, dass der Block wirklich untersucht wird:

$$T(p) = (x_1(p) - x_0(p))(y_1(p) - y_0(p))(z_1(p) - z_0(p)) \cdot \rho(max(p) - min(p)) \cdot c_{CELL}$$

Dabei ist c_{CELL} die geschätzte Zeit, die für die Untersuchung einer Zelle benötigt wird.

Eine erweiterte und etwas schnellere Version der Brute-Force-Methode verwendet jeweils die Informationen der letzten vier Voxel einer Zelle für die nächste Zelle, die ebenfalls von diesen Voxeln begrenzt wird. Wenn dieses Modell benutzt wird, dann lautet die Zeitformel:

$$T(p) = \rho(max(p) - min(p)) \cdot (x_1(p) - x_0(p))(y_1(p) - y_0(p)) \cdot [(z_1(p) - z_0(p)) \cdot c_{QUAD} + c_{INIT}]$$

Dabei ist c_{INIT} die Zeit, die benötigt wird, um eine Reihe von Zellen für die Suche zu initialisieren, z. B. für den Test der ersten Vierergruppe von Voxeln, die mit γ verglichen werden. c_{QUAD} ist die Zeit, die dann noch für jede zu untersuchende Zelle verbleibt.

- **Fall 2: p ist ein $INTERVAL$-Knoten**

  Im wesentlichen ist der Intervallbaum I "output sensitive", d. h., die Rechenzeit ist annähernd proportional zur Länge der Ausgabe (vgl. [5]) Dieser Umstand kann für eine erste Näherungsformel für $T(p)$ genutzt werden, indem alle im Baum enthaltenen Zellen betrachtet werden und der Erwartungswert für deren Ausgabe bestimmt wird:

$$T(p) \approx \sum_{c \text{ Zelle in } I} \rho(max(c) - min(c)) \cdot c_{CELLOUT}$$

  Exakter wird die erwartete Rechenzeit berechnet, indem die Zeiten addiert werden, die aus der Suche entlang des von γ abhängigen Baumpfads und der Ausgabe der entspechenden Stücke der Min- oder Maxlisten jedes Knotens resultieren.

  Jeder innere Knoten p_I des Intervallbaums wird mit einem Definitionsbereich $[min(p_I), max(p_I))$ assoziiert, so daß p_I genau dann untersucht wird, wenn γ in diesem Bereich liegt. Für die Wurzel des Intervallbaums $root_I$ erhalten wir die Grenzen dieses Definitionsbereichs durch einfache Übernahme vom direkt darüberliegenden adaptiven Baumknoten p:

$$min(root_I) = min(p), \quad max(root_I) = max(p)$$

  Für die Nachfolger eines bereits bestimmten Intervallbaumknotens erhalten wir die Definitionsbereiche, indem der schon bekannte Bereich durch den Grenz-Isowert γ^* zerlegt wird:

$$min(left(p_I)) = min(p_I), \quad max(left(p_I)) = \gamma^*(p_I)$$

$$min(right(p_I)) = \gamma^*(p_I), \quad max(right(p_I)) = max(p_I)$$

  Um nun daraus $T(p)$ zu bestimmen, müssen wir die Erwartungswerte aller Zeiten aufaddieren, die für die Untersuchung der Intervallbaumknoten und der zugehörigen Listen benötigt werden:

$$T(p) = \sum_{p_I \text{ Knoten in } I} \rho(max(p_I) - min(p_I)) \cdot (c_{INTINIT} + c_{INTCELL}) + \sum_{c \text{ Zelle in } I} \rho(max(c) - min(c)) \cdot c_{INTCELL}$$

  $c_{INTCELL}$ ist dabei die Zeit, die für den Test einer Zelle aus der Min- oder Maxliste benötigt wird. Das muss für jede extrahierte Zelle plus eine pro untersuchtem Baumknoten getan werden, bei der der Test negativ ausgeht und die Listenbearbeitung terminiert.

  $c_{INTINIT}$ ist die Zeit, die für die Untersuchung eines Intervallbaum-Knotens gebraucht wird, also für den Vergleich des Isowerts γ mit dem eingetragenen Grenz-Isowert γ^*, die Initialisierung der Suche in der entsprechenden Min- oder Maxliste und den Übergang zum linken bzw. rechten Nachfolger.

  Anstatt nach dem jeweils letzten auszugebenden Element einer Min- oder Maxliste des Intervallbaums linear von vorne nach hinten zu suchen, kann dieses auch in nur logarithmischer Zeit durch binäre Suche ermittelt werden. Die Routine zur Ausgabe der Isozellen wird dadurch beschleunigt, weil deren Schleifenabfrage ein einfacher Vergleich des Zählers mit dem so gefundenen Wert ist, anstatt wie bisher ein Vergleich des Isowerts mit dem Min- oder Max-Eintrag im Array, was zu einer signifikanten Reduzierung der Konstanten $c_{INTCELL}$ führt. Diese Optimierungsmöglichkeit ist im Programm implementiert und wird auch benutzt, der verschwindend kleine logarithmische Zeitanteil wird jedoch im Modell ignoriert.

- **Fall 3: p ist ein $ADAPTIVE$-Knoten**

  Dies ist der Fall, der die Berechnung von $T(p)$ auf eine rekursive Formel führt. Alles unter dem Knoten p zu untersuchen, bedeutet, die Nachfolger von p und alles unterhalb dieser Nachfolger zu untersuchen, also gilt:

$$T(p) = \rho(max(p) - min(p)) \cdot c_{ADAP} + T(left(p)) + T(right(p))$$

  Dabei ist c_{ADAP} ein konstanter Wert, der die Zeit für die Untersuchung der beiden Nachfolger des betrachteten adaptiven Baumknotens angibt.

Wie an diesen Formeln zu sehen ist, enthält jede den gleichen Faktor $\rho = \frac{1}{\gamma_{max} - \gamma_{min}}$. Dieser kann also im Optimierungsalgorithmus weggelassen werden, ohne dass sich dadurch an der Anordnung der verglichenen Zeitwerte etwas ändert.

Diese Art der Rechenzeitoptimierung hat einen großen Vorteil: obwohl die Anzahl der zentral unterteilten Conditioned Trees im Vergleich zur Größe des Datensatzes exponentiell steigt, brauchen bei weitem nicht alle Alternativen zur Bestimmung des besten unter ihnen betrachtet zu werden. Wenn S die Anzahl der Zellen im Datensatz bezeichnet, dann ist nachweisbar, daß der Zeitaufwand für die Optimierung sich durch $O(S \cdot log^2(S))$ abschätzen läßt. Es werden nämlich für jeden Knoten des adaptiven Baumanteils bei der Optimierung die folgenden Schritte durchgeführt. S' sei dabei die Anzahl der Zellen des betrachteten Teilblocks.

- Berechnung der Rechenzeit für die Brute-Force-Suche im Block: $O(1)$
- Addition der Rechenzeiten der beiden adaptiven Bäume mit der Zeit für die Untersuchung des vereinenden Wurzelknotens: $O(1)$
- Analyse des Zeitaufwandes für die Suche im Intervallbaum. Das für die Liste der Zellen erforderliche Grundgerüst des Intervallbaums wird gebildet und ausgewertet. Bei geeigneter Vorsortierung ist das in $O(S' log(S'))$ Rechenzeit möglich.
- Die rekursive Analyse des Intervallbaumgerüsts kostet selbst im schlimmsten Fall, daß S' einelementige Min- und Maxlisten auftreten, $O(S')$ Rechenzeit.
- Der Vergleich der drei Kostenwerte kostet $O(1)$; überflüssig gewordene adaptive Teilbäume und Intervallbäume können in $O(S')$ Rechenzeit entfernt werden.

Zusammengenommen wird für jede Ebene des Conditioned Trees $O(S \cdot log(S))$ Rechenzeit gebraucht, also ergibt sich für den ganzen Baum $O(S \cdot log^2(S))$.

4 Ergebnisse und Bilder

Die Conditioned-Tree-Methode wird hier auf verschiedene Typen von Datensätzen angewandt. Sämtliche hier dargestellten Ergebnisse sind auf einem SGI Origin 200 mit 270-MHZ-Prozessor IP27 (eigentlich vier, diese wurden aber nicht gleichzeitig auf das Problem angesetzt) und 1536 MB Speicher gemessen worden. Für diesen Rechner wird ein Programm benutzt, das zu einem gegebenen Datensatz mit dem Lagrange-Parameter λ den ganzen Bereich durchläuft, den Speicher- und Zeitbedarf misst und in einer Datei abspeichert. Dabei wird eine steigende Folge von Parameterwerten $\lambda_1, ..., \lambda_n$ angestrebt, die den Bereich hinreichend dicht abdeckt, so dass für den Speicherbedarf zweier aufeinanderfolgender Werte $\frac{M(\lambda_{i+1})}{M(\lambda_i)} < 1 + \delta$ für ein vom Benutzer vorgegebenes $\delta > 0$ erfüllt ist. Die dafür verwendeten optimalen Conditioned Trees und ihre Intervallbäume werden ebenfalls für eine spätere Verwendung abgespeichert und archiviert. Über den Zeitbedarf dieses Preprocessing-Schritts läßt sich keine verbindliche Aussage machen, weil er nicht nur von der Größe des Datensatzes abhängt, sondern auch von den Anforderungen an den Speicherbedarf der Conditioned Trees, wie gut er einen vom System vorgebebenen Wert ausschöpfen muß. Ferner ist es möglich, eine bereits bestehende Baumliste mit einem großen Wert für δ weiter zu verfeinern. Die Rechenzeiten des Programms reichen von einigen Minuten für die Erzeugung kleiner, aber schon gut verwendbarer Baumlisten (z. B. für $\delta = 1$) bis zu mehreren Stunden für die Erzeugung einer Liste von optimalen Bäumen, bei denen jeder Speicherbedarf bis auf wenige Prozent Abweichung erreicht werden kann (z. B. $\delta = 0.05$). Die Messung der durchschnittlichen Zeit pro Flächenextraktion erfolgt durch eine Simulation der Gleichverteilung auf dem Bereich der möglichen Isowerte, d. h., indem eine Folge von äquidistanten Isowerten verwendet wird, die den ganzen Wertebereich abdecken. Der Durchschnitt der so gemessenen Suchzeiten ist ein gutes Maß für den tatsächlichen Erwartungswert der Suchzeit.

Abbildung 7 zeigt die Speicher-Zeit-Graphen der Kummer-Oberfläche, in den Größen 100x100x100, 200x200x200, 250x250x250 und 400x400x400. Abbildung 8 macht das gleiche mit einer anderen algebraischen Funktion, die der Heart-Oberfläche zugrundeliegt, und die Abbildungen 9 und 10 zeigen Isoflächen-Bilder und Graphen von zwei gemessenen

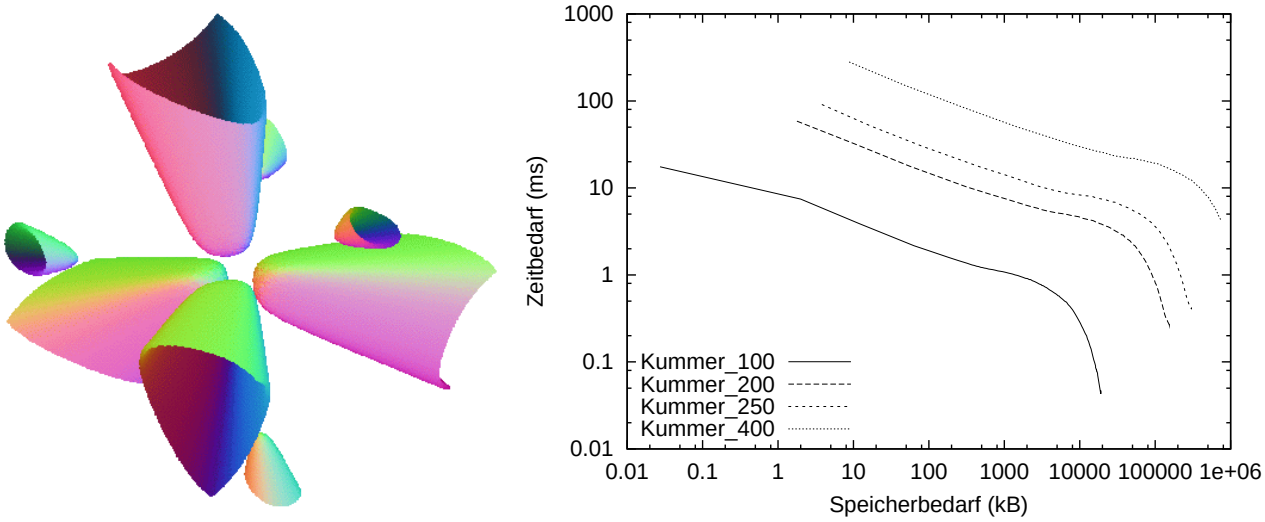

Abbildung 7: Die Conditioned-Tree-Kurven der Kummer-Oberfläche im Speicher-Zeit-Diagramm; die Funktion wurde bereits zu einem Bild in der Einführung definiert.

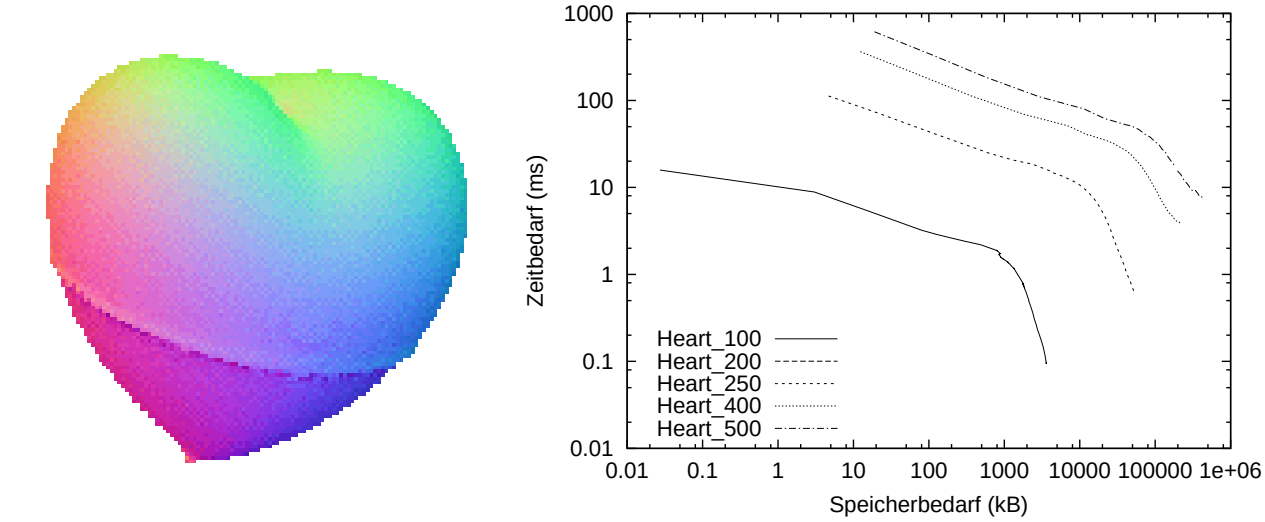

Abbildung 8: Ein Beispielbild und die Conditioned-Tree-Kurven der Heart-Oberfläche, Funktion $f(x,y,z) = (x^2+y^2+z^2-1)^3 - y^3(x^2+0.1z^2)$ im Bereich $[-2,2) \times [-2,2) \times [-2,2)$

Datensätzen, Bighead (256x256x225) und Engine (256x256x110). Diese Datensätze lassen sich im Gegensatz zu den mathematischen Funktionen nicht in verschiedenen Grössen verwenden, weil ihre Grösse bereits festgelegt ist.

Abbildung 9 zeigt zusätzlich den Memory-Time-Graphen, der den von den Volumendaten benötigten Speicherplatz mitzählt. An diesem Beispiel wird deutlich, wie gering der für den adaptiven Baum benötigten Speicherplatz im Vergleich zu den Volumendaten ist, der aber die Rechenzeit schon stark reduziert. Das Diagramm in Abbildung 10 zerlegt den Speicherbedarf graphisch in den Bedarf für adaptive Bäume und für Intervallbäume. Dadurch ist zu sehen, daß der adaptive Baum mit steigendem Parameterwert wächst, dann aber wieder auf einen Knoten zusammenschrumpft.

5 Schlußfolgerungen und weitere Arbeiten

Mit der Conditioned-Tree-Methode haben wir ein parameterabhängiges Verfahren entwickelt, das die Lücke zwischen den bekannten Verfahren der Isoflächen-Extraktion schließt. Für jede Vorgabe zwischen dem Speicherbedarf der Volumendaten selbst und dem Speicherbedarf, der für das Intervallbaum-Verfahren benötigt wird, stellt das Optimierungsschema eine Datenstruktur zur Verfügung, die diesen Speicherbereich optimal zugunsten der Geschwindigkeit ausschöpft.

Die Forschungen zum Conditioned Tree sind weitestgehend abgeschlossen; eine detaillierte Beschreibung der Ergebnisse

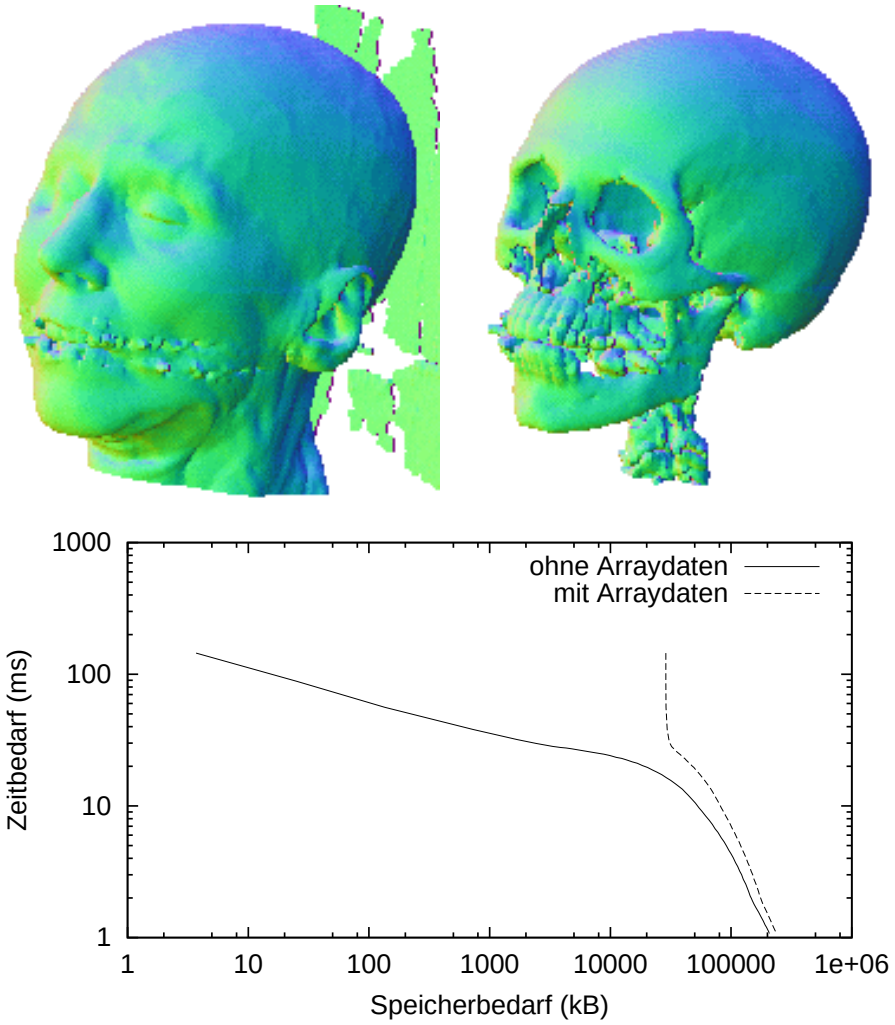

Abbildung 9: Ein menschlicher Kopf und sein Schädel, ein $256 \times 256 \times 225$ CT-Scan von [2]/bighead. Ursprünglich waren diese Daten vom Chapel Hill Volume Rendering Test Dataset und stammen aus dem North Carolina Memorial Hospital. Die zugehörige Memory-Time-Kurve wird zum Vergleich mit und ohne den Speicherbedarf für die Volumendaten gezeigt. Am oberen, annähernd senkrecht verlaufenden Teil der zweiten Kurve sieht man, daß der adaptive Baum im Vergleich zu den Volumendaten verschwindend klein ist.

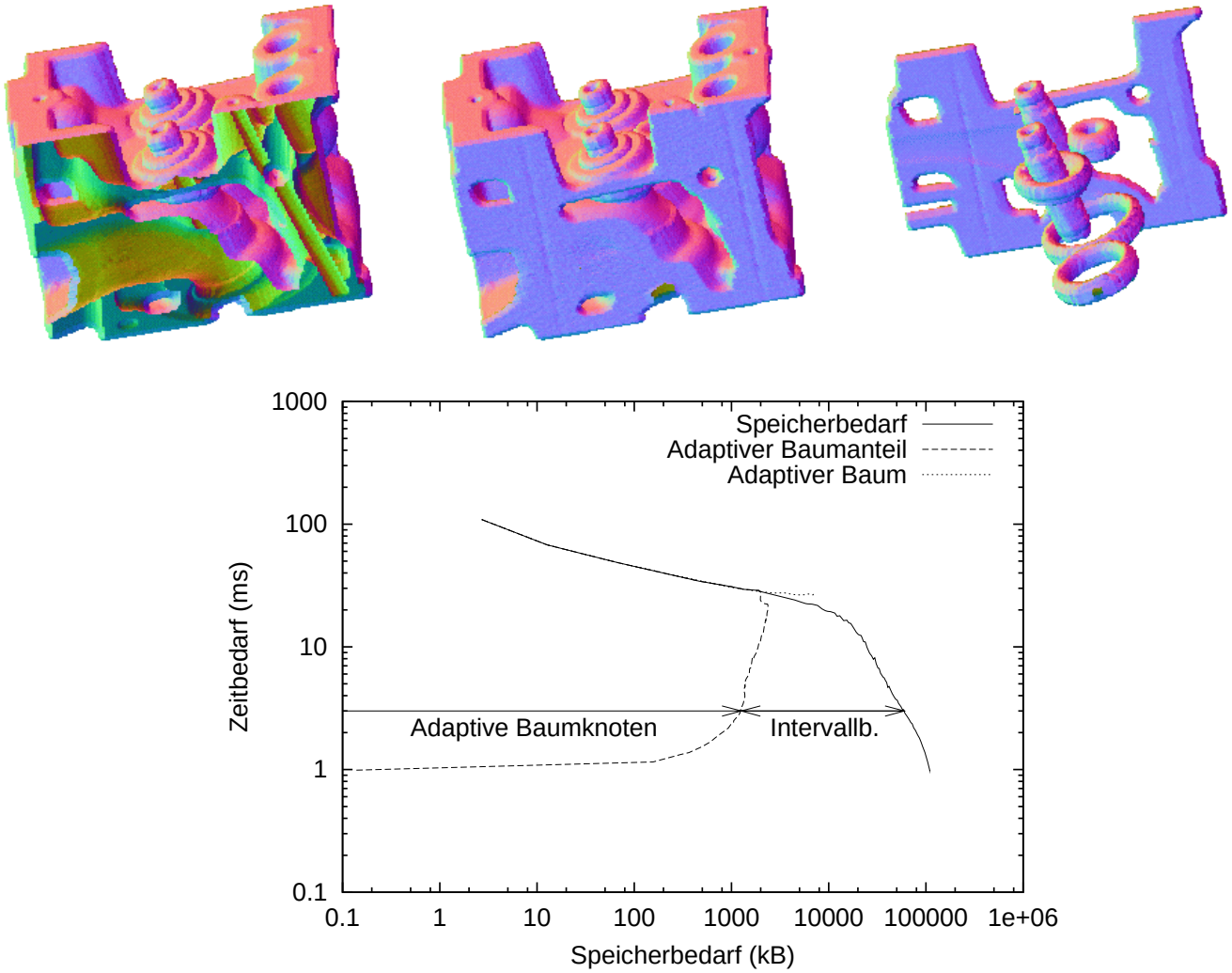

Abbildung 10: Das Innenleben einer Maschine, ein $256 \times 256 \times 110$ CT-Scan von [2]/engine. Das obere rechte Bild zeigt die härtesten Bauteile der Maschine. Unten ist das Memory-Time-Diagramm des Engine-Datensatzes dargestellt. Die rechte Kurve zeigt den ganzen zusätzlichen Speicherbedarf, die linke Kurve den Teil davon, der durch Knoten des adaptiven Baums belegt wird. Die Fläche dazwischen zeigt also den Speicherbedarf der Intervallbäume. Die oberste Kurve zeigt die Abhängigkeit der Rechenzeit vom Speicher bei einem normalen adaptiven Baum, ohne Zulassung von Intervallbäumen. Sie endet schon vor dem Erreichen des vollständigen Zerlegungsbaums auf natürliche Weise, weil der adaptive Baum ab einer bestimmten Tiefe keine Vorteile mehr für die Rechenzeit bringt.

wird bald in der Dissertation [10] folgen. Die folgenden ergänzenden Fragestellungen zu diesem Thema werden z. Zt. bearbeitet:

- Der Conditioned Tree benötigt sowohl in den Blattknoten des adaptiven Baums, als auch in vielen Knoten des Intervallbaums Speicherplatz für Zeiger, die aufgrund der Wahl des Flags nicht benutzt werden. Ist es sinnvoll, zugunsten der Speicherplatzersparnis neue Knotenstrukturen zu definieren, die diese Zeiger nicht mehr enthalten, oder würde sich dadurch eine komplexe Fallunterscheidung ergeben, deren Rechenzeitbedarf diese Einsparung wieder aufhebt ?

- In den Baumknoten sind auch $x_0, ..., z_1$, die Grenzkoordinaten des zugeordneten Teilblocks, enthalten. Die Dateien, in denen das Grundgerüst der Conditioned Trees abgespeichert wird, enthalten aber nur die Flags, die in jedem Knoten zwischen den Alternativen $BRUTEFORCE$, $INTERVAL$ und $ADAPTIVE$ entscheiden; daraus werden die für zentral unterteilte Conditioned Trees eindeutig bestimmten Blockgrenzen rekonstruiert. Ist es günstiger, Speicher zu sparen, indem $x_0, ..., z_1$ auch aus der Datenstuktur weggelassen werden, und dafür zusätzliche Rechenzeit für die Rekonstruktion der Grenzen während der Extraktion in Kauf zu nehmen ?

- Die Grundidee des Conditioned Trees ist es, durch Erweiterung der wählbaren Unterstrukturen auch die Optimierungsmöglichkeiten zu verbessern, wenn eine Formel für die Schätzfunktion ihrer mittleren Rechenzeit bekannt ist. So können insbesondere auch k-d-Trees als Unterbäume von Conditioned Trees bei der Optimierung betrachtet und eingebaut werden.

- Was in diesem Bericht aus Platzgründen nicht erwähnt ist, ist eine bereits realisierte Methode, die zu einer gegebenen Liste von Intervallen (wie sie sich aus den hier untersuchten Volumendaten-Zellen ergibt) einen bezüglich des Speicherbedarfs und der mittleren Suchzeit optimalen Intervallbaum bestimmt.

6 Literatur

[1] Chandrajit L. Bajaj, Valerio Pascucci, Daniel R. Schikore, *Fast isocontouring for improved interactivity*, ACM Siggraph/IEEE Symposium on Volume Visualization (1996), pp. 39–46.

[2] *Volume Datasets*, Chapel Hill Volume Rendering Test Dataset, Volume I, ftp://www-graphics.stanford.edu/pub/volpack/data .

[3] Yi-Jen Chiang, Claudio T. Silva, *I/O optimal isosurface extraction*, Proceedings IEEE Visualization 1997, pp. 293–300.

[4] Philip A. Chou, Tom Lookabaugh, Robert M. Gray, *Optimal pruning with applications to tree-structured source coding and modeling*, IEEE Transactions on Information Theory, vol.35, no.2 (1989), pp. 299–315.

[5] Paolo Cignoni, Paola Marino, Claudio Montani, Enrico Puppo, Roberto Scopigno, *Speeding up isosurface extraction using interval trees*, IEEE Transactions on Visualization and Computer Graphics, vol.3, no.2 (1997), pp. 158–170.

[6] Yarden Livnat, Hai-Wei Shen, Christopher R. Johnson, *A near optimal isosurface extraction algorithm using the span space*, IEEE Transactions on Visualization and Computer Graphics, vol.2, no.1 (1996), pp. 73–84.

[7] William E. Lorensen, Harvey E. Cline, *Marching Cubes: a high resolution 3D surface construction algorithm*, ACM Computer Graphics, vol. 21, no. 4 (1987), pp. 163–196.

[8] Han-Wei Shen, Charles D. Hansen, Yarden Livnat, Christopher R. Johnson, *Isosurfacing in span space with utmost efficiency (ISSUE)*, Proceedings IEEE Visualization 1996, pp. 287–294.

[9] Han-Wei Shen, Christopher R. Johnson, *Sweeping Simplices: A fast iso-surface extraction algorithm for unstructured grids*, Proceedings IEEE Visualization 1995, pp. 143–150.

[10] Jürgen Toelke, Optimierung des Platz-Zeit-Tradeoffs bei der Isoflächen-Extraktion von Volumendaten, Dissertation, Institut für Informatik, Univ. Leipzig, voraussichtlich Dez. 2000.

[11] Chris Weigle, David C. Banks, *Complex-valued contour meshing*, Proceedings IEEE Visualization 1996, pp. 173–180.

[12] Jane Wilhelms, Allen Van Gelder, *Octrees for faster isosurface generation*, ACM Transactions on Graphics, vol.11, no.3 (1992), pp. 201–227.