Published: 2. Oldenburger 3D-Tage, 2003, In: Luhmann (Hrsg.): Optische 3D-Messtechnik - Photogrammetrie - Laser-Scanning, Wichmann-Verlag, 2003, 117-124.

Hash-Bäume und andere mehrdimensionale Punktstrukturen zur Speicherung von Laserscanner-Daten

Thomas BRINKHOFF

Zusammenfassung

Beim Laserscanning fallen umfangreiche Datenmengen an, deren Verwaltung und Speicherung besonderer Datenstrukturen bedarf. In diesem Beitrag wird die Fragestellung betrachtet, ob mehrdimensionale Punktstrukturen, die als Indexstrukturen für Geodatenbanken entwickelt worden sind, für die persistente Speicherung von Laserscanner-Daten eingesetzt werden können. Es werden insbesondere sogenannte Hash-Bäume und R-Bäume vorgestellt und anhand konkreter Laserscanner-Daten näher untersucht.

1 Einleitung

Der Einsatz von *Laserscanning* zur Erfassung von räumlichen Daten gewinnt in den letzten Jahren ständig an Bedeutung (LUHMANN 2002). Dabei fallen mehrdimensionale Punktwolken an, die zu Auswertungszwecken weiterverarbeitet werden müssen (NIEMEIER & KERN 2001). Aus den Messwerten ergeben sich ummittelbar die kartesischen Koordinatenwerte (x,y,z) und – bei einigen Laserscannern – zusätzlich ein Intensitätswert für die empfangene Signalstärke. Somit ist das Ergebnis einer Messung zunächst eine Menge von drei- oder vierdimensionalen Punkten.

Aufgrund des *großen Datenumfangs* – mehrere Millionen Datenpunkte bei steigender Tendenz – ist es nicht zweckmäßig, die Punkte als herkömmliche Grafikobjekte in eine CAD-Zeichnung zu übernehmen (SCHWERMANN & EFFKEMANN 2002). Generell führt das Vorhalten der gesamten Punktwolke im Hauptspeicher zu großen anfänglichen Ladezeiten und ist schlecht skalierbar, da ab einer gewissen Schwelle Teile des Hauptspeichers auf den (langsamen) Hintergrundspeicher ausgelagert werden müssen.

Eine Alternative stellt der Einsatz von *persistenten Datenstrukturen* dar, die die Daten auf dem Hintergrundspeicher halten und so organisieren, dass nur Teile der Daten bei Bedarf eingelesen werden. Für den Einsatz in Datenbanksystemen wurde eine Reihe solcher Datenstrukturen entwickelt, die dort *Indexstrukturen* genannt werden. Herkömmliche Indexstrukturen für alphanumerische Daten können nicht (ohne weiteres) für Geodaten verwendet werden. Daher wurden spezifische Indexstrukturen für Geodatenbanken und Geoinformationssysteme entwickelt (BRINKHOFF 2003). Eine Kategorie von Indexstrukturen stellen *Punktstrukturen* dar, die in der Lage sind, mehrdimensionale Punktdaten dynamisch auf dem Hintergrundspeicher zu organisieren und räumliche Anfragen zu bearbeiten.

In diesem Beitrag soll die Fragestellung untersucht werden, ob solche Punktstrukturen für die Speicherung von Laserscanner-Daten eingesetzt werden können. Dazu werden im zwei-

ten Abschnitt Indexstrukturen aus dem Bereich der Geodatenbanken vorgestellt. Ein besonderer Augenmerk gilt dabei Hash-Bäumen und R-Bäumen. In Abschnitt 3 wird der Einsatz solcher Indexstrukturen für die Speicherung von Laserscanner-Daten näher betrachtet. Der Beitrag schließt mit einer Zusammenfassung und einem Ausblick auf zukünftige Arbeiten.

2 Indexstrukturen für Geodatenbanken

2.1 Indexierung in Datenbanksystemen

Ziel eines Datenbanksystems ist es, große Mengen von Datensätzen in einer dynamischen Umgebung persistent zu speichern. Dabei organisiert das Datenbankmanagementsystem die Daten in *Datenbankblöcken*. Der Zugriff auf den Hintergrundspeicher erfolgt blockweise, d.h. bei einem Zugriff auf einen Datensatz wird mindestens ein vollständiger Block übertragen, der unter Umständen mehrere Datensätze enthält.

Ein *Index* ist ein dynamisches Inhaltsverzeichnis, das die schnelle Suche nach den Datenbankblöcken mit Datensätzen unterstützt, die eine Anfragebedingung erfüllen. Datenstrukturen, die zur Nutzung als Datenbankindex entwickelt worden sind, nennt man *Indexstrukturen*. In heutigen Datenbanksystemen werden typischerweise B-Bäume oder Hash-Verfahren (bzw. deren Varianten) als Indexstrukturen eingesetzt. Ein *B-Baum* ist ein dynamischer, balancierter Baum, bei dem ein Knoten einem Datenbankblock entspricht. B-Bäume speichern die Daten sortiert nach ausgewählten Schlüsselattributen (**Abb. 1**, links). Zur Anfragebearbeitung wird ein B-Baum bei der Wurzel beginnend zielgerichtet durchlaufen. *Hash-Verfahren* berechnen den Speicherort (d.h. die Blockadresse) eines Datensatzes mit Hilfe einer *Hash-Funktion*, die auf ausgewählte Schlüsselattribute angewendet wird (**Abb. 1**, rechts). Hash-Verfahren erlauben eine schnelle Suche gemäß diesen Schlüsselattributen, weisen aber Effizienzprobleme auf, falls auf ungleich im Datenraum verteilten Daten Bereichsanfragen zu bearbeiten sind.

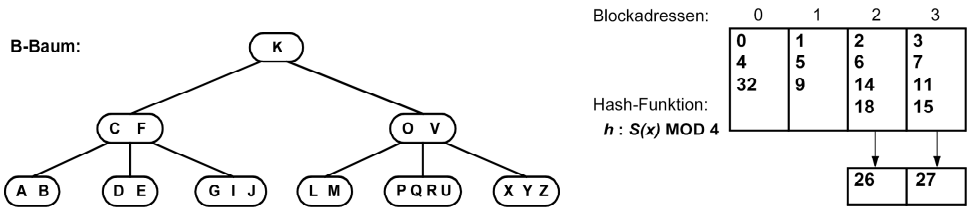

Abb. 1: Beispiel für einen B-Baum und für ein Hash-Verfahren.

2.2 Indexierung von Geo- und Punktdaten

Da B-Bäume erfordern, dass die Daten nach einer linearen Ordnung sortiert werden können, und da Hash-Verfahren Probleme bei Ungleichverteilungen bzw. Bereichsanfragen haben, können herkömmliche Indexstrukturen nicht (ohne weiteres) für Geodaten verwen-

det werden. Somit wurden spezifische *Rechteckstrukturen* zur Speicherung von zwei- oder mehrdimensionalen ausgedehnten Objekten in Geodatenbanken und Geoinformationssystemen entwickelt (BRINKHOFF 2003). *Punktstrukturen* sind Indexstrukturen für zwei- oder mehrdimensionale Punktdaten.

Das *Gridfile* (auch Gitterdatei genannt) (NIEVERGELT ET AL. 1984) ist eine solche mehrdimensionale Punktstruktur. Dabei handelt es sich um ein Hash-Verfahren, bei dem die Hash-Funktion durch ein gitterförmiges Verzeichnis *(Grid Directory)* ersetzt wird. Das Directory speichert in seinen Zellen die Adressen von Blöcken, die die eigentlichen Daten enthalten (**Abb. 2**). Allerdings weist das Gridfile Schwächen bei ungleich verteilten oder bei korrelierten Daten auf.

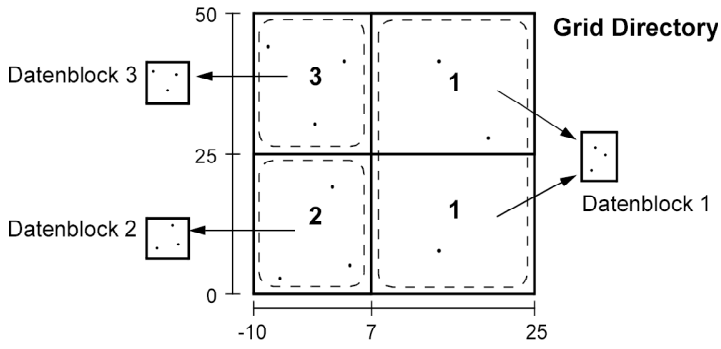

Abb. 2: Beispiel für ein Gridfile.

Die Partitionierung des Datenraums durch Gridfiles hat folgende Eigenschaften:

- die Regionen, die durch einen Datenbankblock repräsentiert werden *(Blockregionen)*, sind rechteckig,
- der Datenraum wird vollständig durch Blockregionen überdeckt und
- die Blockregionen sind disjunkt.

Für leistungsfähige Punktstrukturen muss nach SEEGER (1989) mindestens eine dieser Eigenschaften aufgegeben werden.

2.3 Hash-Bäume

Hash-Bäume sind mehrdimensionale Punktstrukturen, die ein Hash-Verfahren mit einer Baumstruktur verbinden. Ein typisches Beispiel für einen Hash-Baum ist das *BANG-File* (Balanced and Nested Grid File) von FREESTON (1987). Das BANG-File ist ein hierarchischer Baum, dessen Knoten Blockregionen enthalten, die sich an einer Gitterstruktur orientieren. Die Blockregionen sind im Gegensatz zum normalen Gridfile nicht rechteckig. Stattdessen werden von einem Rechteck, das eine Blockregion umgibt, alle eingelagerten Rechtecke, die kleinere Blockregionen repräsentieren, abgezogen. Dadurch hat eine Blockregion ggf. eine unregelmäßige Form und kann in mehrere nichtzusammenhängende Gebiete zerfallen. Das nachfolgende Beispiel (**Abb. 3**) zeigt für eine Punktmenge, die in etwa auf einer Sinuskurve liegt, die Partitionierung des Datenraums durch ein BANG-File. Dabei

sind jeweils alle Blockregionen einer Ebene des Directorybaums zusammengefasst dargestellt.

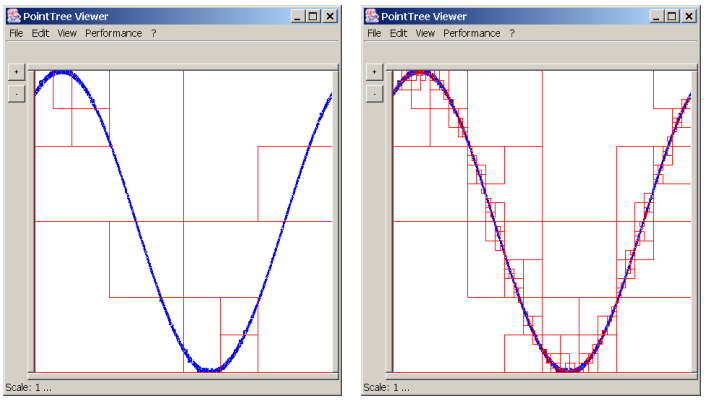

Abb. 3: Beispiel für die Partitionierung des Datenraums durch ein BANG-File.

Ein anderes Beispiel ist der *Buddy Tree* (SEEGER & KRIEGEL 1990). Der Buddy Tree ist ein hierarchischer Baum, dessen Knoten rechteckige Blockregionen enthalten, die sich an einer Gitterstruktur orientieren. Allerdings müssen die Blockregionen im Gegensatz zum Gridfile und dem BANG-File nicht alle Teile des Datenraums überdecken (**Abb. 4**).

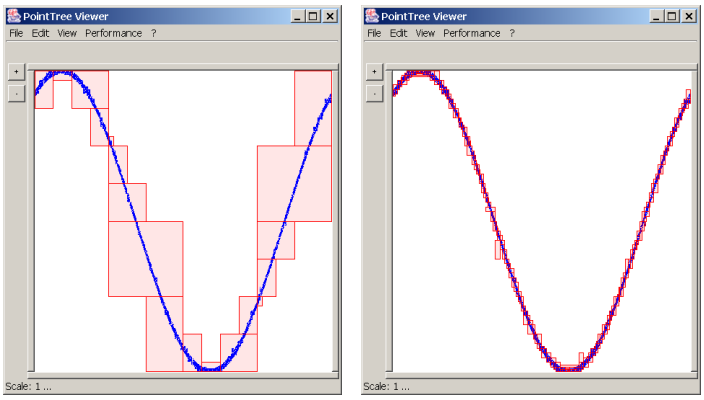

Abb.4: Beispiel für die Partitionierung des Datenraums durch einen Buddy Tree.

2.4 R-Bäume

Der *R-Baum* (GUTTMAN 1984) ist eine Indexstruktur, die sowohl als mehrdimensionale Punkt- als auch als Rechtecksstruktur verwendet werden kann. Der R-Baum hat ähnliche Eigenschaften wie der B-Baum, ohne dass er eine spezifische Sortierung der Daten erfordert. Die Daten in den Knoten des R-Baums werden rekursiv durch minimal umgebende

zwei- oder mehrdimensionale achsenparallele Rechtecke beschrieben. Diese Blockregionen dürfen sich überlappen und brauchen nicht alle Teile des Datenraums überdecken. Die verschiedenen Varianten des R-Baums unterscheiden sich insbesondere in ihrer *Überlaufstrategie*. Diese definiert bei einem Blocküberlauf aufgrund einer Einfügeoperation die Kriterien, nach denen die Datensätze oder Blockregionen des betroffenen Blocks auf zwei Blöcke aufgeteilt werden. Das nachfolgende Beispiel zeigt links die Partitionierung des Datenraums auf der obersten Ebene des R-Baums und rechts einen Detailausschnitt, der die Überlappung zwischen den Blockregionen deutlich macht (**Abb. 5**).

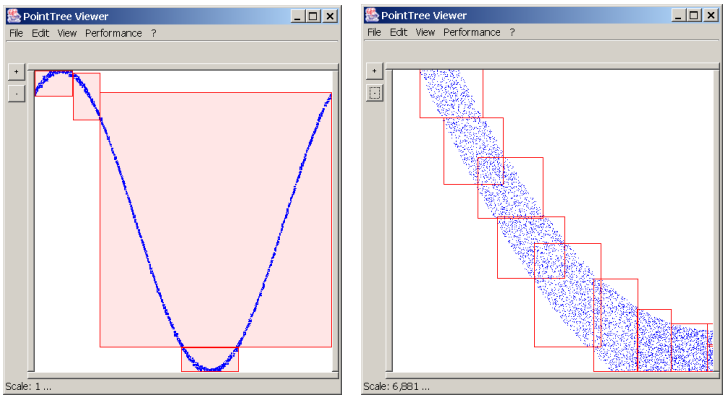

Abb. 5: Beispiel für die Partitionierung des Datenraums durch einen R-Baum.

2.5 Ein erstes Fazit

Die vorgestellten Punktstrukturen sind dynamische Datenstrukturen, die das Einfügen und Löschen von Punktdaten erlauben. Sie unterstützen – als Indexstrukturen für Datenbanken – die persistente Speicherung von Daten auf dem Hintergrundspeicher. Alle genannten Punktstrukturen sind für zwei-, drei- oder höherdimensionale Punktdaten geeignet und erlauben der Verarbeitung von *räumlichen Anfragen*. Solche Anfragen sind

- räumliche Selektionsanfragen, wie *Punktanfragen* und *Rechtecksanfragen*,
- die Bestimmung der k nächsten Nachbarn (*Nearest Neighbor Queries*, siehe z.B. (HJALTASON & SAMET 1999)) und
- der geometrische Verbund (*Spatial Join*, siehe z.B. (BRINKHOFF ET AL. 1993)).

Die Punktstrukturen sind *lokal ordnungserhaltend*, das heißt, räumlich eng benachbarte Punkte werden mit hoher Wahrscheinlichkeit in dem gleichen Block gespeichert. Dies ist für die Verarbeitung von räumlichen Anfragen essentiell, da der Zugriff auf einen auf dem Hintergrundspeicher gespeicherten Block im Vergleich zu anderen Rechneroperationen sehr langsam ist. *Globale Ordnungserhaltung* fordert, dass Blöcke, die durch räumlich nahe Blockregionen beschrieben sind, mit hoher Wahrscheinlichkeit physisch nah auf dem Hintergrundspeicher abgelegt werden. Dies stellen die genannten Punktstrukturen nicht sicher. Anstelle von speziellen Verfahren (siehe z.B. (HUTFLESZ ET AL. 1988), (BRINKHOFF 2001))

kann aber auch eine nachträgliche vollständige Reorganisation der Punktstruktur zu einer globalen Ordnungserhaltung führen (siehe Abschnitt 3.2).

3 Einsatz für Laserscanner-Daten

3.1 Vorbereitung

Um die in den Abschnitten 2.3 und 2.4 vorgestellten Punktstrukturen auf ihre Einsatzfähigkeit für Laserscanner-Daten untersuchen zu können, mussten eine Reihe von vorbereitenden Tätigkeiten durchgeführt werden:

- Zwei rund 15 Jahre alte Implementierungen des BANG-Files und des Buddy Trees in der Programmiersprache Modula-2, die auf Sun-3-Workstations mit Motorola-Prozessoren ausgerichtet waren (und damit nicht auf SPARC-Workstations ablauffähig sind), wurden auf eine Entwicklungsplattform portiert, die aktuelle Intel-Prozessoren unterstützt. Auf dieser Basis wurde eine einheitliche API entworfen und implementiert.
- Die Modula-2-Implementierungen wurden zusammen mit einer aktuellen Java-Implementierung für R-Bäume unter einer einheitlichen Java-Benutzeroberfläche integriert (**Abb. 3, 4, 5**). Als R-Baum-Varianten wurde neben den traditionellen Verfahren von GUTTMAN (1984) und dem leistungsfähigeren *R*-Baum* von BECKMANN ET AL. (1990) auch eine von BECKMANN & SEEGER (2003) revidierte Version des R*-Baums *(RR*-Baum)* implementiert.

Die Datenstrukturen wurden mit einem Datensatz getestet, der von der Aufnahme einer Gebäudefassade stammt und rund 3,66 Mio Datenpunkte umfasst (**Abb. 6**). Auf dieser Basis konnten für die Hash-Bäume einige notwendige Parameter abgeleitet werden. Hier ist insbesondere die Stellenzahl für die Beschreibung von Blockregionen zu nennen.

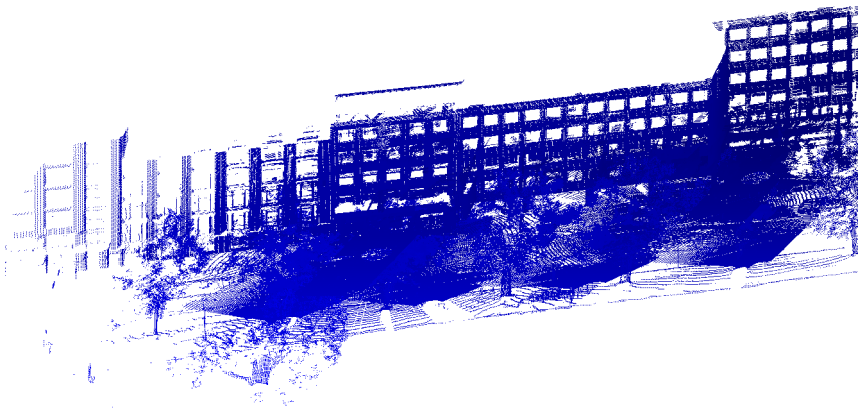

Abb. 6: Darstellung des Testdatensatzes.

3.2 Aktueller Entwicklungsstand

Zum *Aufbau der Punktstrukturen* können die Punktdaten aus XYZ-Dateien eingelesen und in die Indexstruktur eingefügt werden. Dabei entsteht relativ unabhängig von der verwendeten Struktur ein Speichermehrbedarf von rund 68%, der für die Beschreibung der Blockregionen im Directory und insbesondere durch nicht vollständig gefüllte Datenblöcke entsteht. Letzteres ist der Preis dafür, dass es sich um dynamische Datenstrukturen handelt, also nachträglich Daten eingefügt und gelöscht werden können, ohne dass sich das Leistungsverhalten der Indexstruktur dadurch entscheidend verschlechtert.

Der *Durchsatz beim Einfügen* der Testdaten betrug auf einem Pentium IV-PC (2 GHz, 256 MB Hauptspeicher) für das BANG-File, den Buddy Tree bzw. den RR*-Baum rund 7100, 8100 bzw. 5500 Datenpunkte/Sekunde. Negativ aus dem Rahmen fiel der R*-Baum, der nur ca. 2500 Punkte pro Sekunde schaffte.

Ein exzellentes Leistungsverhalten zeigen alle Indexstrukturen bei der *Extraktion von Datenpunkten*, die in einem kleineren Anfragequader liegen. Auf die Daten kann ohne einen vorherigen Ladevorgang sofort zugegriffen werden. Die Anfragen werden zielgerichtet zu den Blöcken hingeführt, in denen die gesuchten Daten gespeichert sind. Damit reichen wenige Zugriffe auf den Hindergrundspeicher für die Bearbeitung solcher Anfragen aus.

Für viele Anwendungszwecke ist eine *Überblicksdarstellung* der Gesamtdaten erforderlich. Diese kann man dadurch unterstützen, dass man Datenpunkte, die in relativ großen Datenregionen liegen, einzeln darstellt, während man bei kleineren Datenregionen nur die Region darstellt, so dass man den betroffenen Datenblock nicht einlesen braucht. Die Anzahl der Blockzugriffe auf den Hintergrundspeicher kann damit deutlich reduziert werden. Die tatsächliche Laufzeit für eine solche Vorgehensweise ist allerdings solange nicht befriedigend, wie die Daten nur lokal ordnungserhaltend gespeichert sind, da dann (fast) jeder Blockzugriff zu einer Aktivität auf dem Hintergrundspeicher führt. Daher wurde die Funktionalität der Indexstrukturen um eine *globale Reorganisation* erweitert, die in der Reihenfolge eines Tiefendurchlaufs die Blöcke neu auf dem Hintergrundspeicher anordnet. Dadurch kann eine globale räumliche Ordnung erzielt werden. Der Zeitbedarf für eine Überblicksdarstellung verringert sich dadurch in etwa um den Faktor 10.

4 Zusammenfassung

In diesem Beitrag wurde Einsatz von Indexstrukturen, die für die Organisation von multidimensionalen Daten in Datenbanksystemen entwickelt wurden, für die persistente Verwaltung von Laserscanner-Daten erörtert. Als potenziell geeignete Verfahren wurden Hash-Bäume (wie das BANG-File und der Buddy Tree) und R-Bäume untersucht. Mit Hilfe einer Implementierung dieser Punktstrukturen konnten erste Erfahrungen gesammelt und hier vorgestellt werden.

Zwei wichtige Aufgaben stehen in der näheren Zukunft an: 1. Die Definition typischer Anfrageprofile auf Laserscanner-Daten, so dass das Leistungsverhalten der einzelnen Indexstrukturen detailliert untersucht und miteinander verglichen werden kann. 2. Die Be-

trachtung, ob die lokale Ordnungserhaltung bzw. die räumlichen Hierarchien, die man durch den Aufbau von Indexstrukturen erzielt, zur Unterstützung weiterer Analyseschritte genutzt werden können. Hier ist insbesondere die Extraktion und Approximation von Flächen und Kanten (z.B. in Anlehnung an die Vorgehensweise von NIEMEIER & KERN (2001)) zu betrachten.

5 Literatur

Beckmann N., H.-P. Kriegel, R. Schneider & B. Seeger (1990): *The R*-tree: An Efficient and Robust Access Method for Points and Rectangles*. Proceedings ACM SIGMOD International Conference on Management of Data, Atlantic City, NJ, S. 322-331.

Beckmann N. & B. Seeger (2003): *Ready for System Integration: A Revised R*-tree with Improved Insertion and Search Performance*. Technischer Bericht Universität Marburg.

Brinkhoff T. (2001): *Using a Cluster Manager in a Spatial Database System*. Proceedings 9th ACM International Symposium on Advances in Geographic Information Systems (ACM-GIS), Atlanta, GA, S. 136-141.

Brinkhoff T. (2003): *Geodatenbanksysteme.* Lehrbuch in Vorbereitung.

Brinkhoff T., H.-P. Kriegel & B. Seeger (1993): *Efficient Processing of Spatial Joins Using R-trees*. Proceedings ACM SIGMOD International Conference on Management of Data, Washington, DC, S. 237-246.

Freeston M. (1987): *The BANG file: A new kind of grid file*. Proceedings ACM SIGMOD International Conference on Management of Data, San Francisco, CA, S. 260-269.

Guttman A. (1984): *R-trees: A Dynamic Index Structure for Spatial Searching*. Proceedings ACM SIGMOD International Conference on Management of Data, Boston, S. 47-57.

Hjaltason G.R. & H. Samet (1999): *Distance Browsing in Spatial Databases*. ACM Transactions on Database Systems, Vol. 24, No. 2, S. 265-318.

Hutflesz A., H.-W. Six & P. Widmayer (1988): *Globally Order Preserving Multidimensional Linear Hashing*. Proceedings 4th International Conference on Data Engineering, Los Angeles, CA, S. 572-579.

Luhmann T. (Hrsg.) (2002): *Photogrammetrie und Laserscanning, Anwendung für As-Built-Dokumentation und Facility Management*. Herbert Wichmann Verlag.

Niemeier W. & F. Kern (2001): *Anwendungspotentiale von scannenden Messverfahren*. In: U. Weferling et al. (Hrsg.): Von Handaufmaß bis High Tech, Verlag Philipp von Zabern, S. 134-140.

Nievergelt J., H. Hinterberger & K.C. Sevcik (1984): *The Grid File: An Adaptable, Symmetric Multikey File Structure*. ACM Transactions on Database Systems, Vol. 9, No. 1, S. 38-71.

Schwermann R. & C. Effkemann (2002): *Kombiniertes Monoplotting in Laserscanner- und Bilddaten mit PHIDIAS*. In: (LUHMANN 2002), S. 57-70.

Seeger B. (1989): *Entwurf und Implementierung mehrdimensionaler Zugriffsstrukturen*. Dissertation, Universität Bremen.

Seeger B. & H.-P. Kriegel (1990): *The Buddy Tree: An Efficient and Robust Access Method for Spatial Databases*. Proceedings 16th International Conference on Very Large Data Bases, Brisbane, Australien, S. 590-601.