7 Raumzugriffsstrukturen

Ausgedehnte Objekte

In Nichtstandard-Datenbankssytemen, wie z.B. in Geographischen Datenbanksystemen, werden nicht nur Punktobjekte, sondern auch *ausgedehnte Datenobjekte* gespeichert:

- *Linien*
- *Linienzüge*
- *Flächen.*

Eine für Geographische Datenbanken besonders wichtige Objektklasse stellen *Flächen mit Löchern* dar. Sie werden durch *einfache Polygone mit Löchern (EPL)* repräsentiert.

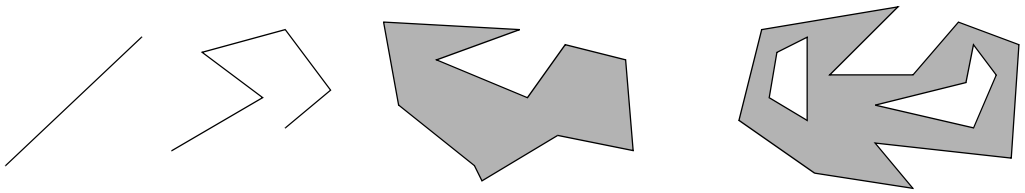
7.1 Grundlagen

- ***Einfaches Polygon***

  Polygon, bei dem sich kein Paar nicht-aufeinanderfolgender Kanten schneidet.
- ***Einfaches Polygon mit Löchern***

  Einfaches Polygon, aus dem disjunkte, einfache Polygone herausgeschnitten sind.

Im weiteren wird insbesondere auf die Verwaltung und Speicherung von einfachen Polygonen mit Löchern eingegangen.



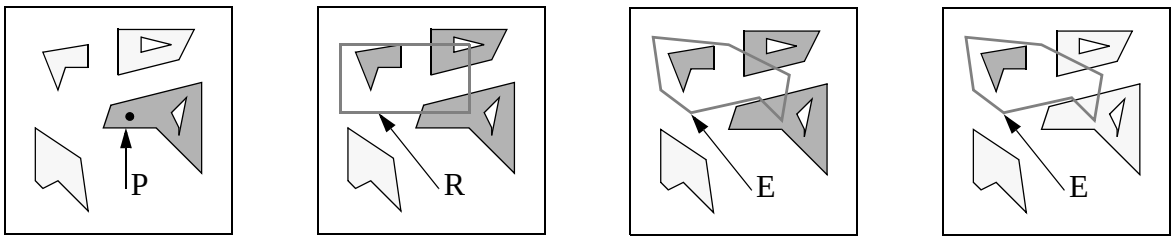
Linie Linienzug Fläche (einfaches Polygon) Fläche mit Löchern (EPL)

Anfragen

Eine Indexstruktur soll Anfragen effizient unterstützen. Für die Objektklasse der EPL gibt es eine Reihe von wichtigen *Basisanfragen*:

- *Point Query:* Gegeben ein Punkt P; finde alle EPL, die P enthalten.
- *Window Query:* Gegeben ein Rechteck R; finde alle EPL, die R schneiden.
- *Region Query:* Gegeben ein EPL E; finde alle EPL, die E schneiden.
- *Enclosure Query:* Gegeben ein EPL E; finde alle EPL, die in E enthalten sind.
- *Containment Query*: Gegeben ein EPL E; finde alle EPL, die E vollständig enthalten.

Auf diesen Basisanfragen können dann komplexere Anfragen in einem Geoinformationssystem aufsetzen.



Point Query Window Query Region Query Enclosure Query

Approximationen

Einfache Polygone mit Löchern (die ja eine beliebige Komplexität besitzen können) lassen sich nicht direkt mittels Indexstrukturen abgespeichern (z. B. weil man sonst keine minimale Anzahl von Einträgen pro Seite garantieren könnte).

⇒ Eine *Approximation* der EPL über eine einfachere Objektklasse wird notwendig.

Eine solche Approximation sollte *konservativ* sein, d.h. das approximierte Objekt sollte vollständig in der Approximation enthalten sein. Eine Reihe von Approximationen wurde vorgeschlagen, die in Abschnitt 7.4 behandelt werden. Im folgenden betrachten wir die einfachste dieser Approximationen, nämlich das *achsenparallele minimal umgebende Rechteck* (MUR).

Zweistufige Anfragebearbeitung [BHKS 93]

Die Anfragebearbeitung von approximiert organisierten Objekten muß zweistufig ablaufen:

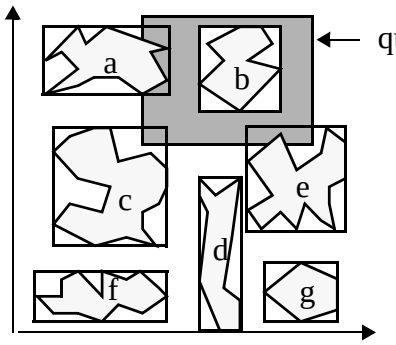
1. Filterschritt

Zunächst werden die Objekte über die Indexstruktur eingelesen, die gemäß der Approximation die Anfrage erfüllen. Man erhält somit eine *Kandidatenmenge*, die Obermenge zur eigentlichen Lösungsmenge ist.

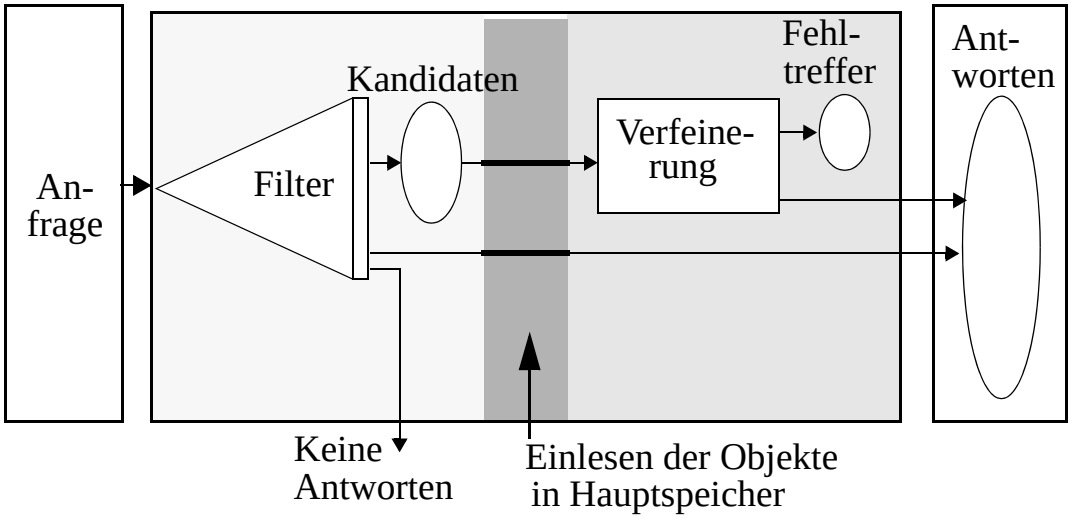
2. Verfeinerungsschritt

Im zweiten Schritt wird die exakte Darstellung der Kandidaten herangezogen und untersucht, ob das EPL tatsächlich die Anfrage erfüllt.

Beispiel: Window-Query



→ a und b sind sicher *Antworten*
→ f, d und g sind sicher keine Antworten
→ c und e sind *Kandidaten*
→ c ist ein *Fehltreffer* (*false hit*, d. h. ein Kandidat, der keine Antwort ist)



7.2 Von Punkt- zu Rechteckzugriffsstrukturen

Mit den bisher vorgestellten Indexstrukturen *(Punktzugriffsstrukturen)* können nicht ohne weiteres achsenparallele Rechtecke abgespeichert werden, sie können jedoch als Basis für *Raumzugriffsstrukturen* benutzt werden. Es gibt drei Techniken, um von Punkt- zu Rechteckzugriffsstrukturen zu kommen (siehe [SK 88]):

- Punkttransformation
- Clipping
- Überlappende Regionen.

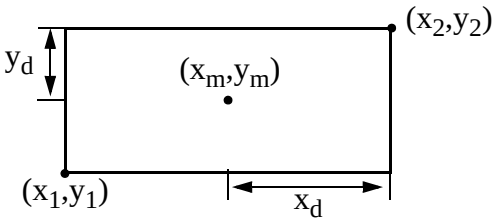
Punkttransformation

Bei der *Punkttransformation* werden die minimal umgebenden n-dimensionalen Rechtecke in 2n-dimensionale Punkte überführt und in einer 2n-dimensionalen Punktzugriffsstruktur abgespeichert. Es lassen sich zwei Transformationen unterscheiden:

- *Mittentransformation*

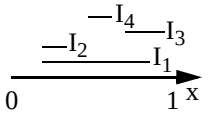
  Das Rechteck wird durch den Mittelpunkt (x_m,y_m) und die jeweilige halbe Ausdehnung x_d und y_d beschrieben.
- *Eckentransformation*

  Das Rechteck wird durch diagonal gegenüberliegende Eckpunkte (x_1,y_1) und (x_2,y_2) repräsentiert.

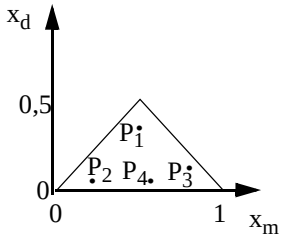


Transformation des Datenraumes und der Anfragebereiche (für 1-dimensionale Intervalle):

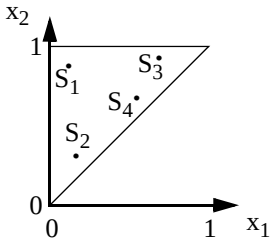
Transformation von Intervallen I_j



in 2-dim. Punkte:

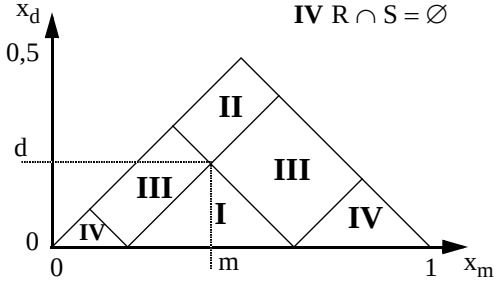


$P_j = (x_m,x_d)$
(Mittentransformation)



$S_j = (x_1,x_2)$
(Eckentransformation)

I $R \subseteq S$
II $R \supseteq S$
III $R \cap S \neq \varnothing$
IV $R \cap S = \varnothing$



Anfrageräume zur Suche von Intervallen R bzgl. eines Intervalls S = (m,d) (Mittentransformation)

Eigenschaften

- Bei der Eckentransformation liegen die meisten Daten auf einer Diagonalen durch den Datenraum.

  ⇒starke Korrelation der Daten
- Bei der Mittentransformation verlaufen die Anfragegrenzen nicht mehr orthogonal zur Datenraumpartitionierung.

  ⇒komplexere Implementierung der Anfrage

  ⇒mehr Datenseiten werden geschnitten
- Die geometrischen Verhältnisse gehen bei der Punkttransformation durch die Einbeziehung der Rechtecksausdehnung verloren.

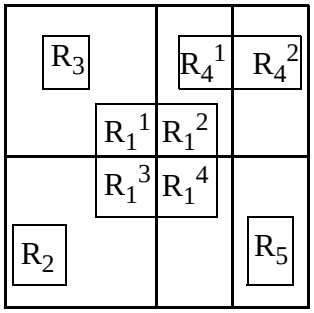
  ⇒Raumbezogene Anfragen verlieren dadurch deutlich an Effizienz.
- Die Punkttransformation ist auf jede Punktzugriffsstruktur anwendbar. So müssen z.B. die Einfüge- oder Lösch-Operationen nicht geändert werden.

Anwendung

Der Leistungsvergleich in Abschnitt 6.1 zeigt, daß der *Buddy-Baum* für die Punkttransformation gut geeignet ist (siehe Verteilung “Diagonal”). Der *LSD-Baum* wurde speziell als Indexstruktur für mehrdimensionale Punkte entwickelt, die durch Punkttransformation aus Rechtecken entstanden sind.

Clipping

Die Idee ist, daß ein Rechteck in jede Datenregion eingefügt wird, die es schneidet.



5 Rechtecke
9 Datensätze

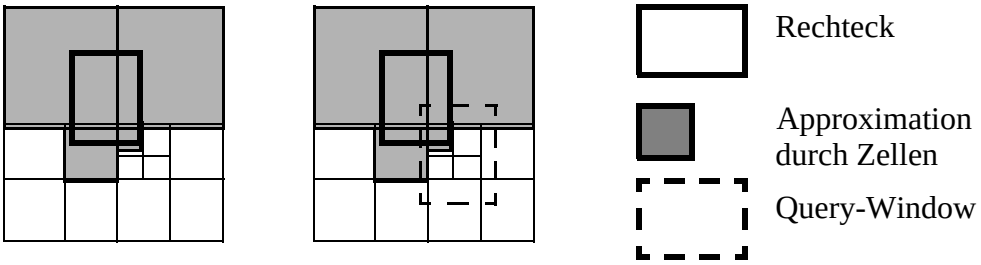
Eigenschaften:

- Die Zahl der Datensätze in der Indexstruktur steigt stärker als die Zahl der gespeicherten Rechtecke. Dieses gilt insbesondere, wenn die Rechtecke im Verhältnis zum Datenraum groß sind oder die Partitionierung schon sehr fein geworden ist.
- Die Indexstruktur sollte Überlaufseiten zulassen, da ein Gebiet, in dem sich mehr Rechtecke überlappen, als in eine Datenseite passen, nicht weiter partitioniert werden kann.
- Einfügen und Löschen werden erheblich aufwendiger.
- Bereichsanfragen degenerieren in der Leistung aufgrund der hohen Zahl von mehrfach eingelesenen identischen Rechtecken.

Anwendung

Mit Hilfe des Clipping lassen sich Rechtecke folgendermassen in einem *PR-Quadtree* abspeichern. Es werden die Zellen des PR-Quadtrees berechnet, die ein gegebenes Rechteck minimal umgeben.

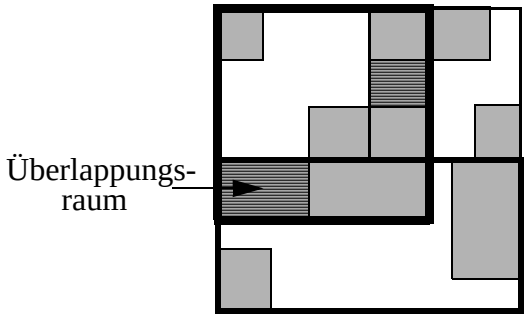
Alle diese Zellen erhalten einen Eintrag für das Rechteck.



In obigem Beispiel wird das gespeicherte Rechteck für die gegebene Window-Query viermal gefunden.

Überlappende Regionen

Die Kernidee der Technik der überlappenden Regionen ist, daß die Partitionierung des Datenraumes nicht mehr disjunkt ist. Somit können sich Seitenregionen überlappen und ein Clipping der Rechtecke wird überflüssig.



Hauptproblem: Überlappung der Directoryregionen.

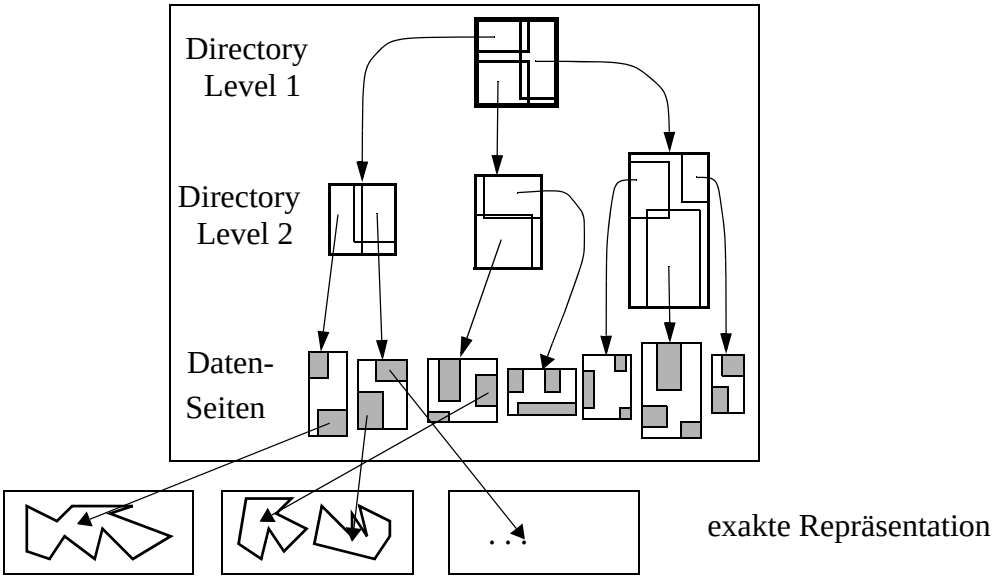
- Fällt eine Anfrage in einen Überlappungsraum, müssen mehrere Pfade im Suchbaum untersucht werden.

  $\Rightarrow$ Die Überlappung sollte möglichst klein gehalten werden.

Anwendung

Die Technik überlappender Regionen wird beim *R-Baum* und dessen Varianten verwendet.

Beispiel:



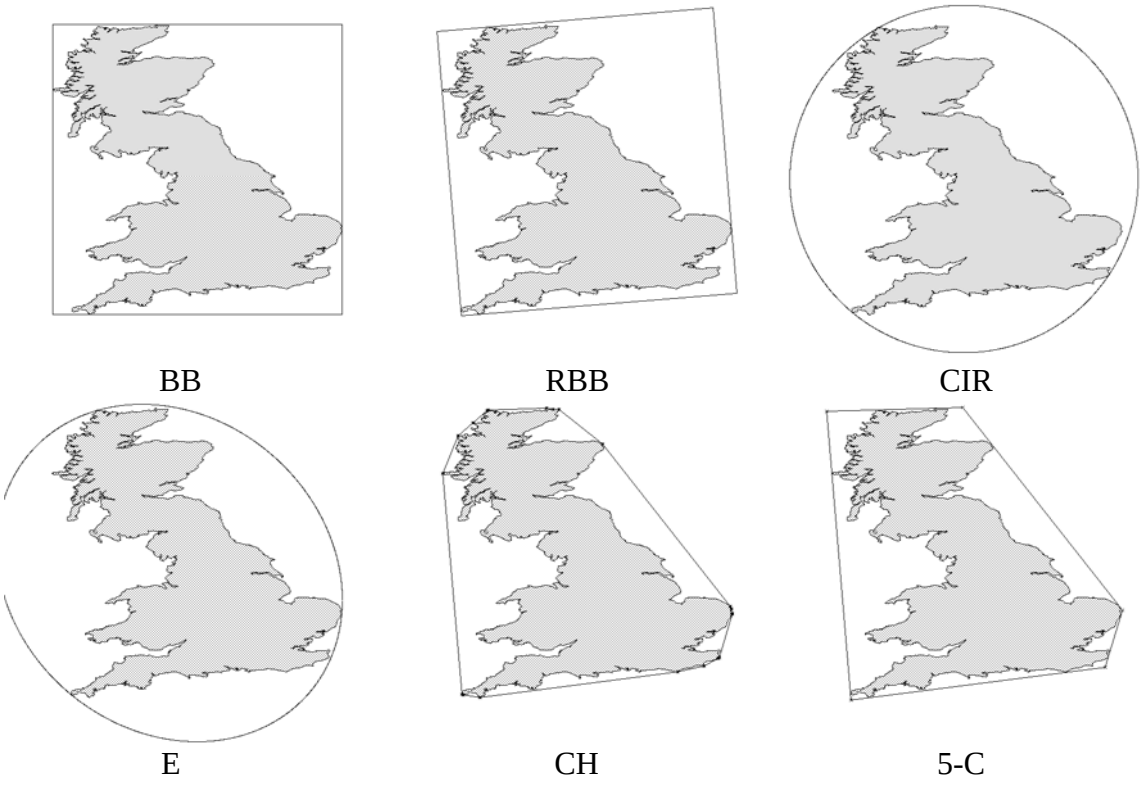
7.3 Approximation

Idee

- möglichst viele Objekte, die keine Antworten sind, ohne Zugriff auf ihre exakte Repräsentation ausscheiden
- den aufwendigen Verfeinerungsschritt nur für möglichst wenige Objekte durchführen

Verschiedene Approximationen

- *achsenparalleles, minimal umgebendes Rechteck (BB).*
- *minimal umgebender Kreis (CIR)* [Oos 90].
- *konvexe Hülle (CH)* [Gün 89].
- *gedrehtes, minimal umgebendes Rechteck (RBB)*
- *minimal umgebende Ellipse (E)*
- *minimal umgebende konvexe Vier- und Fünfecke (4-C bzw. 5-C)*



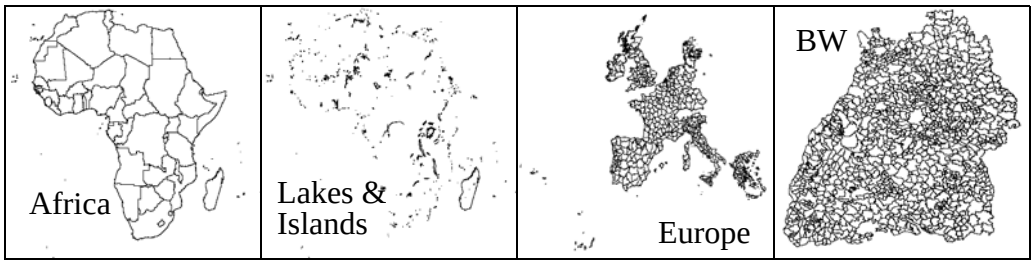
Approximationsgüte

$$Q_{\text{Appr}} = \frac{A(Appr[O])}{A(O)} \times 100\%$$

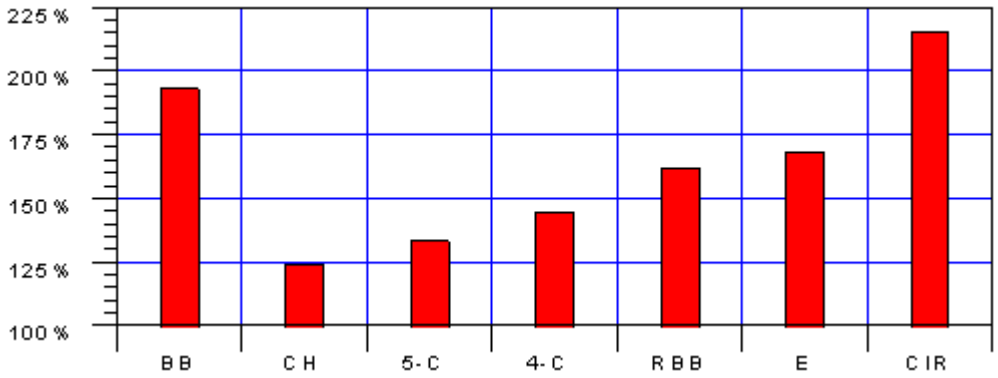
- A: Fläche, O: Objekt, Appr[O]: Approximation von O
- Q_{Appr} = 100% wenn Approximation optimal, d. h. Appr[O] = O
- je kleiner Q_{Appr}, umso besser die Approximation

Experimentelle Untersuchung der Approximationsgüte [BKS 93]

Verwendete Daten



Durchschnittsergebnisse über alle Landkarten



Interpretation

- CH hat im Unterschied zu 5-C beliebig viele Parameter, trotzdem approximiert 5-C fast gleich gut.
- RBB: 1 Parameter mehr als BB ⇒ 31% Verbesserung.
- 5-C: 6 Parameter mehr als BB⇒ 60% Verbesserung.

Vorteile genauerer Approximationen

- Zugriff auf die exakte Repräsentation für weniger Objekte (weniger Seitenzugriffe)
- Verfeinerungsschritt für weniger Objekte (weniger CPU-Zeit)
- z. B. Point-Queries: die Verbesserung der Verarbeitungszeit ist proportional zur Verbesserung der Approximationsgüte

Nachteile genauerer Approximationen

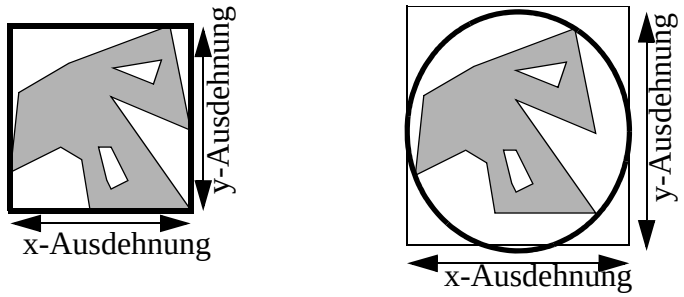
- höherer Speicherplatzbedarf auf Datenseiten, z. B. bei Annahme von 4 Bytes pro Parameter:

	CH	5-C	4-C	RBB	E	BB	CIR
Q_{Appr}	124	133	144	162	168	193	215
Speicherplatzbedarf in bytes	32 - 640	40	32	20	20	16	12

⇒ mehr Datenseiten nötig

- größere Ausdehnung der Approximationen

  z. B.



Ausdehnung$_{Appr}$ = x-Ausdehnung x y-Ausdehnung, Ausdehnung$_{BB}$=Ausdehnung$_{CH}$=100%

	5-C	4-C	RBB	E	CIR
Ausdehnung $_{Appr}$ (%)	121	144	151	122	142

⇒ größere Datenseiten-Regionen

- höherer Aufwand für die Berechnung der Approximationen und für Tests auf den Approximationen

Frage:

Wird der Overhead im Filterschritt durch größere Gewinne im Verfeinerungsschritt gerechtfertigt? Siehe die folgende Untersuchung der Performance von Point-Queries mit den betrachteten Approximationen.

Untersuchung von Point-Queries mit Approximationen

- Annahme: die gegebene Approximation ersetzt die BB-Approximation.
- Annahme: der Verfeinerungsschritt ist pro Objekt c-mal so aufwendig wie der Filterschritt (c > 2 für alle Testdaten).
- Leistungsmaß ist die Antwortzeit.

	Reduktion # Fehltreffer	**Overhead im Filterschritt**		**Break-Even Punkt für c =**		**Speicher-platz-bedarf** [bytes]
		100% erfolgreiche Query	60% erfolgreiche Query	100% erfolgreiche Query	60% erfolgreiche Query	
CH	36 %	70 %	62 %	1,94	1,72	32-640
5-C	31 %	44 %	39 %	1,42	1,26	40
4-C	25 %	43 %	38 %	1,72	1,52	32
RBB	16 %	30 %	28 %	1,88	1,75	20
E	13 %	18 %	15 %	1,38	1,15	20
CIR	- 11 %	5 %	6 %	/	/	12

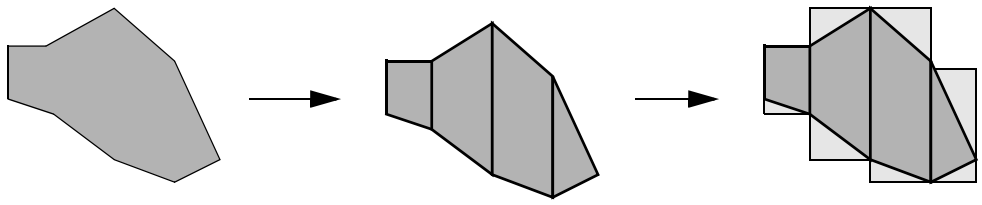
Zusammenfassende Bewertung

- Der Kreis ist nicht geeignet als Approximation.
- Die Ellipse besitzt den niedrigsten Break-Even-Punkt.
- Das 5-Eck liefert den besten Trade-Off zwischen Leistungsverbesserung, Speicherplatzbedarf und Break-Even-Punkt.
- Alle Leistungsverbesserungen wachsen mit c, d. h. mit der Komplexität der Objekte.

7.4 Dekomposition

Idee

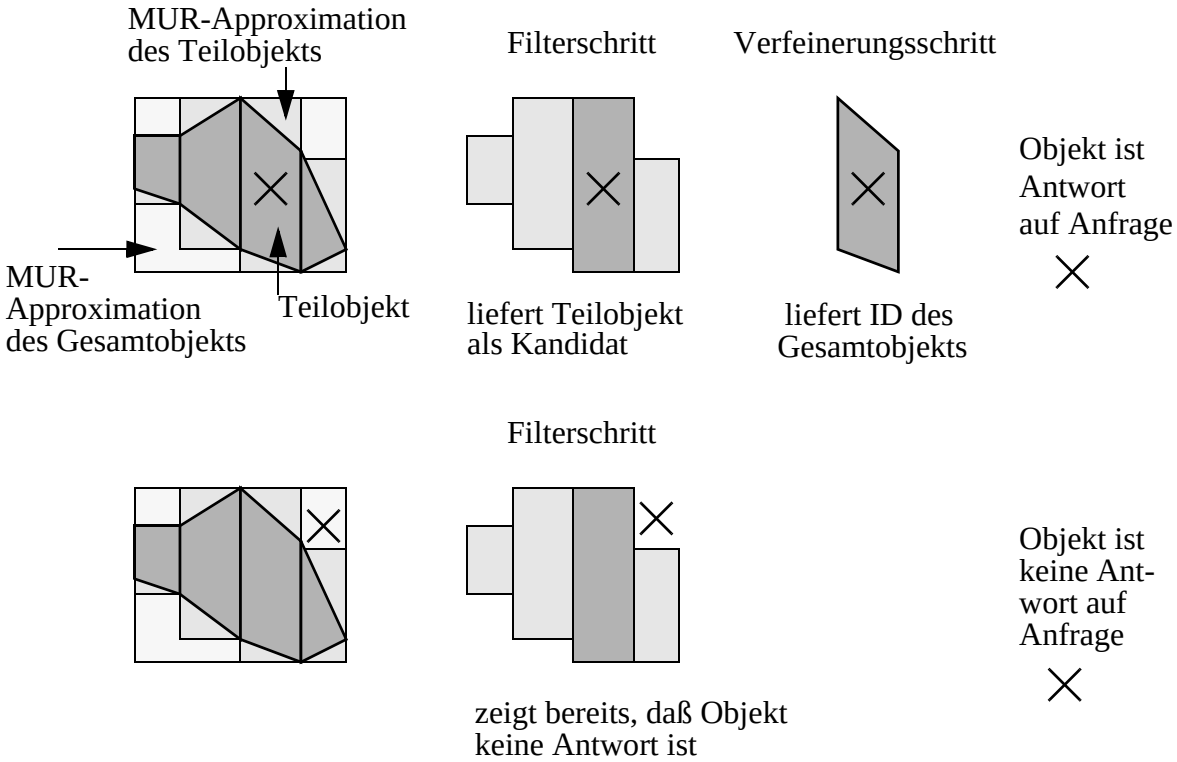
- Ziel: Tests auf der exakten Repräsentation der Objekte (im Verfeinerungsschritt) vereinfachen.
- Beobachtung: für das Ergebnis eines Tests ist i. A. nur ein kleiner Teil des Objekts relevant.
- Vorgehen: Dekomposition der EPL in einfache Teilobjekte und Approximation der Teilobjekte durch minimal umgebende Rechtecke.



Gesamt-Objekt | Teilobjekte | Teilobjekte mit MUR (in Zugriffsstruktur)

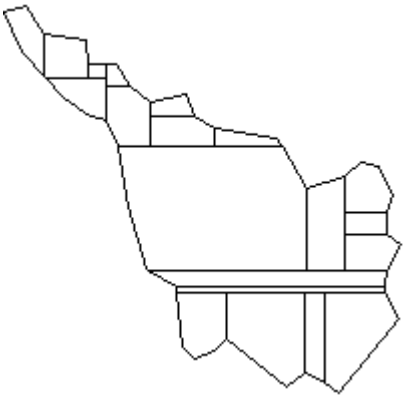
Anfragebearbeitung mit Dekomposition (Point Query)

- Der Filterschritt liefert mit Hilfe einer Raumzugriffsstruktur alle Teilobjekte, deren Approximation den Anfragepunkt enthalten.
- Über einen Algorithmus aus der Computational Geometry wird nun getestet, ob der Punkt tatsächlich im Kandidaten-Teilobjekt liegt.
- Wenn der Test dieses bestätigt, kann das zugehörige Gesamtobjekt in die Antwortmenge der Anfrage mit aufgenommen werden.



Verschiedene Dekompositionen

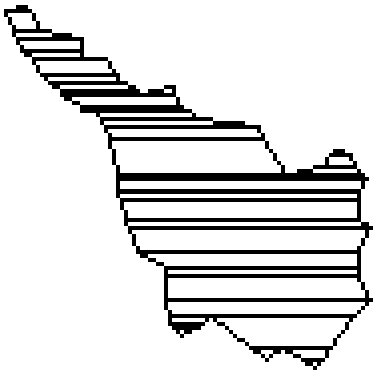
(n = Zahl der Eckpunkte des EPL)



konvexe Dekomposition

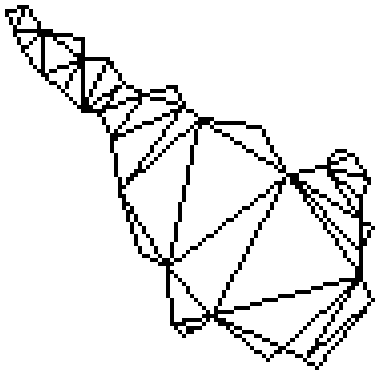
in konvexe Polygone

≈ *n*/2 Komponenten



Dekomposition in Trapeze

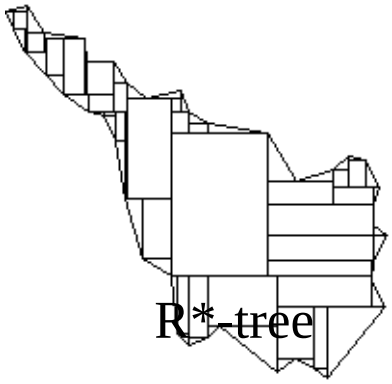
≈ *n* Komponenten



Triangulation

Dekomposition in Dreiecke

≈ *n* Komponenten



Heterogene Dekomposition

in Dreiecke und Rechtecke

≈ 1.8**n* Komponenten

Eigenschaften

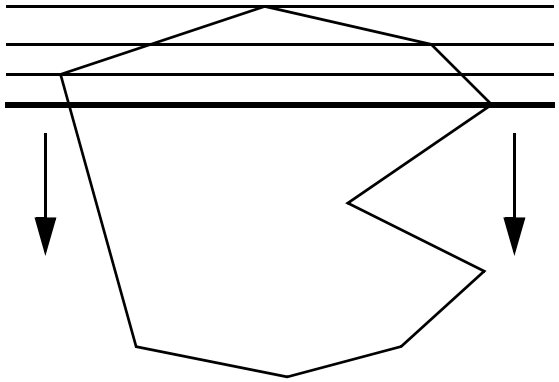
- lineare Anzahl einfacher Komponenten
- Die Typen der Komponenten sind so gewählt, daß sie gut durch MUR approximiert werden können.

Vorteile

- Die Teilobjekte lassen sich durch die MUR besser approximieren als das Gesamtobjekt.
  ⇒ Der Filterschritt arbeitet genauer, so daß sich die Kandidatenmenge verkleinert.
- Es entstehen einfachere Teilobjekte (weniger Kanten).
  ⇒ Der Verfeinerungsschritt kann effizienter durchgeführt werden.

Nachteile

- Rechen-Aufwand für die Dekomposition
  z. B. für die Zerlegung in Trapeze :



Sortiere die Eckpunkte nach Y-Koordinate.

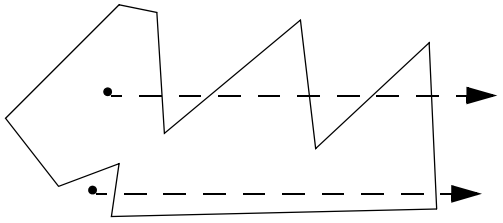
Für jedes Element der sortierten Liste definiere ein Trapez durch das aktuelle und das nächste Listenelement.

⇒ Laufzeit der Dekomposition in Trapeze ist O (n*log n), wobei n die Anzahl der Eckpunkte des EPL bezeichnet.

- Es sind mehrere Teilobjekte für ein EPL zu verwalten und zu speichern. Im obigen Fall der Dekomposition in Trapeze z. B. können es bis zu n-1 Teilobjekte sein.
  ⇒ Größerer Speicherbedarf.
  ⇒ Komplexere Handhabung von Objekten (Objekt-ID, Konsistenz).

Exkurs in die algorithmische Geometrie:

Beispiel: Test, ob Punkt in Polygon liegt

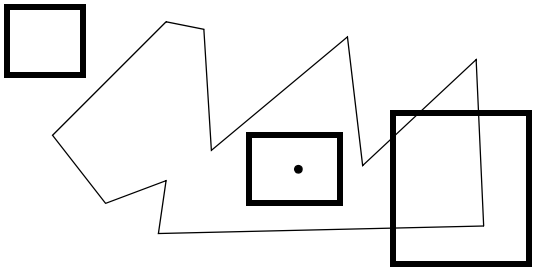


Zähle die Schnittpunkte.

Falls Anzahl
ungerade: Punkt liegt im Polygon
gerade: Punkt liegt nicht im Polygon.

Laufzeit O(n), n = Anzahl der Kanten

Beispiel: Test, ob Window ein Polygon schneidet



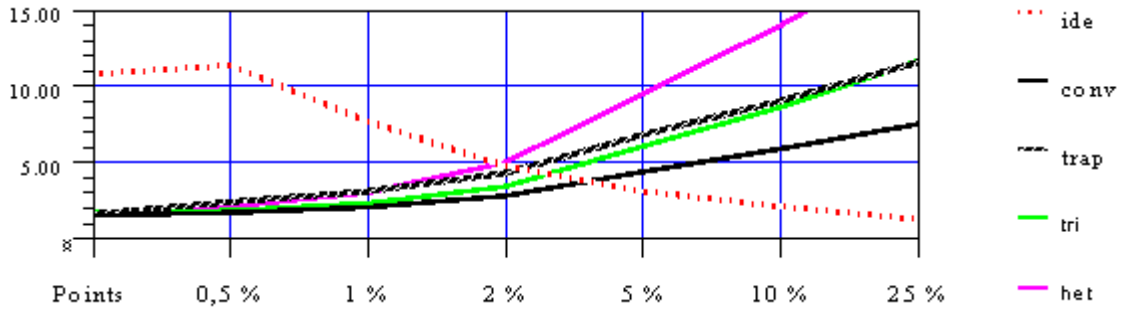
Schneidet eine der Kanten das Window?
Wenn ja: Antwort = ja
Wenn nein:

Nehme einen Punkt des Window.
Liegt er im Polygon?
Wenn ja: Antwort = ja.
Wenn nein: Antwort = nein.

Laufzeit O(n), n = Anzahl der Kanten

Experimentelle Untersuchung von Window-Queries [KHS 91]

(CPU-Zeit in msec pro Antwort)



Größe des Query-Window

Ergebnisse

Dekompositionen mit linearer Anzahl von Komponenten

- sind sehr gut für selektive Queries (kleine Antwortmenge)
- degenerieren für wenig selektive Queries (große Antwortmenge).

Vergleich der bisherigen Ansätze zur Dekomposition

- zwei Ansätze: Identität (d. h. keine Dekomposition der Objekte) und Zerlegung in einfache Komponenten (im folgenden Vergleich z. B. in Trapeze)
- Komplexität der Komponenten: Zahl ihrer Eckpunkte

	Anzahl der Komponenten	Komplexität der Komponenten	Leistung bei Queries :	
			selektive	wenig selektive
Identität	1	n	schlecht	gut
einfache Komponenten	n	4	gut	schlecht

⇒ Benötigt wird eine gute *Balance* zwischen Anzahl und Komplexität der Komponenten.

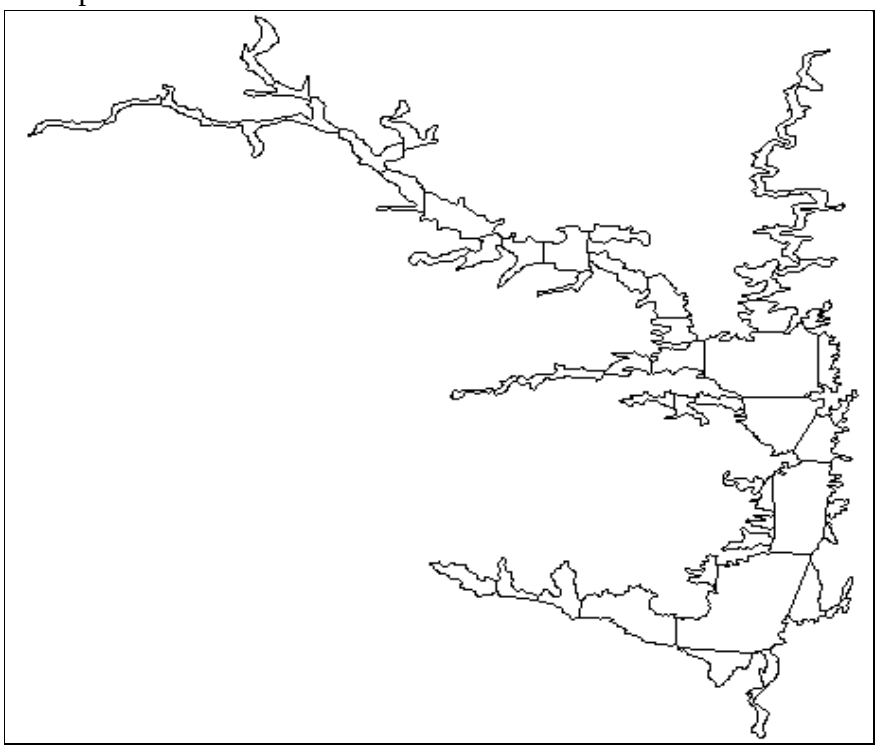
Balancierte Dekomposition [SK 93]

- Siehe obiger Vergleich: das Produkt aus Anzahl der Komponenten und ihrer Komplexität ist O(n).
- Wurzelkriterium für balancierte Dekomposition: die Komplexität der bei einer Dekomposition erhaltenen Komponenten soll im Intervall

$$[c\sqrt{n},\ 2c\sqrt{n} + 1]$$

liegen, wobei c eine Konstante ist (z. B. c = 1) und n die Anzahl der Eckpunkte des Gesamtobjekts bezeichnet.

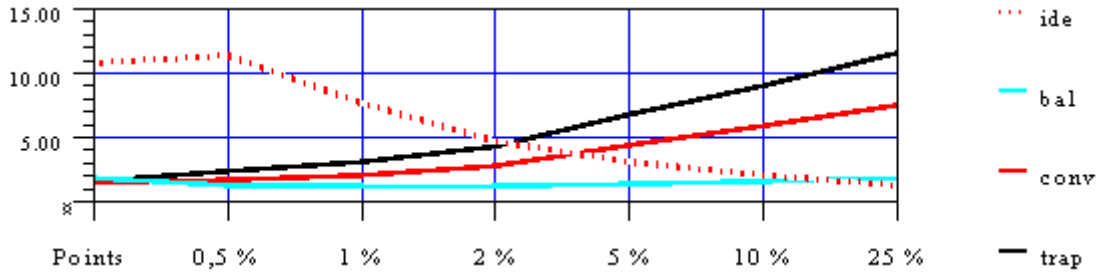
- Beispiel: Volta See



n = 5449

57 Komponenten

Experimentelle Untersuchung der balancierten Dekomposition



Durchschnittliche Antwortzeit für verschiedene Window-Größen **(**in msec pro Antwort)

==> Die balancierte Dekomposition

- degeneriert nicht für wenig selektive Window-Queries
- verbessert die Leistung im Vergleich zur Identität um einen Faktor von bis zu 10.

7.5 Literatur

Literatur (Raumzugriffsstrukturen)

[BKSS 90] Beckmann N., Kriegel H.-P., Schneider R., Seeger B.: *'The R*-tree: An Efficient and Robust Access Method for Points and Rectangles'*, Proc. ACM SIGMOD Int. Conf. on Management of Data, Atlantic City, N.J., 1990, pp. 322-331.

[Gün 89] Günther O.: *'The Design of the Cell Tree: An Object-Oriented Index Structure for Geometric Databases'*, Proc. IEEE 5th Int. Conf. on Data Engineering, Los Angeles, CA., 1989, pp. 598-605.

[Gut 84] Guttman A.: *'R-trees: A Dynamic Index Structure for Spatial Searching'*, Proc. ACM SIGMOD Int. Conf. on Management of Data, Boston, MA., 1984, pp. 47-57.

[Oos 90] Oosterom P. J. M.: *'Reactive Data Structures for Geographic Information Systems'*, PhD-thesis, Dept. of Computer Science at Leiden University, Netherland, 1990.

[PSTW 93] Pragel B. U., Six H.-W., Toben H., Widmayer P.: *'Towards an Analysis of Range Query Performance in Spatial Data Structures'*, Proc. ACM SIGMOD Principles of Database Systems, Washington, 1993, pp. 214-221.

[SK 88] Seeger B., Kriegel H.-P.: *'Techniques for Design and Implementation of Efficient Spatial Access Methods'*, Proc. 14th Int. Conf. on Very Large Databases, Los Angeles, CA., 1988, pp. 360-371.

Literatur (Anfragebearbeitung)

[BHKS 93] Brinkhoff T., Horn H., Kriegel H.-P., Schneider R.: *'A Storage and Access Architecture for Efficient Query Processing in Spatial Database Systems'*, Proc. 3rd Int. Symp. on Large Spatial Databases, Singapore, 1993, in: Lecture Notes in Computer Science, Vol. 692, Springer, 1993, pp. 357-376.

[BKS 93] Brinkhoff T., Kriegel H.-P., Schneider R.: *'Comparison of Approximations of Complex Objects used for Approximation-based Query Processing in Spatial Database Systems'*, Proc. 9th Int. Conf. on Data Engineering, Vienna, Austria, 1993, pp. 40-49.

[BKS 93] Brinkhoff T., Kriegel H.-P., Seeger B.: *'Efficient Processing of Spatial Joins Using R-trees'*, Proc. ACM SIGMOD Int. Conf. on Management of Data, Washington DC, 1993, pp. 237-246.

[BKSS 94] Brinkhoff T., Kriegel H.-P., Schneider R., Seeger B.: *'Multi-Step Processing of Spatial Joins'*, Proc. ACM SIGMOD Int. Conf. on Managment of Data, Minneapolis, MN, 1994, pp. 197-208.

[Kri 91] Kriegel H.-P., Heep P., Heep S., Schiwietz M., Schneider R.: *'An Access Method Based Query Processor for Spatial Database Systems'*, Proc. Int. Workshop on Database Management Systems for Geographical Applications, Capri, Italy, 1991, in: Geographic Database Management Systems, Springer, 1992, pp. 273-292.

[KHS 91] Kriegel H.-P., Horn H., Schiwietz M.: *'The Performance of Object Decomposition Techniques for Spatial Query Processing'*, Proc. 2nd Symp. on Large Spatial Databases, Zurich, Switzerland, 1991, in: Lecture Notes in Computer Science, Vol. 525, Springer, 1991, pp. 257-276.

[SK 93] Schiwietz M., Kriegel H.-P.: *'Query Processing of Spatial Objects: Complexity versus Redundancy'*, Proc. 3rd Symp. on Large Spatial Databases, Singapore, 1993.