UNIVERSITÄT LEIPZIG

Institut für Informatik

Vorlesung
Geo-Informationssysteme II

Dr. Dieter Sosna

Sommersemester 2003

Hinweise, Anmerkungen und Verbesserungsvorschläge bitte an:

- Dr. Dieter Sosna*
- Georg Apitz**

- ein Teil des Scriptes geht auf Vorlagen von Heiko Stamer zurück

*<dieter@informatik.uni-leipzig.de>, http://www.informatik.uni-leipzig.de/~sosna

**<georg@apitz.de>, http://www.ge-org.com

UNIVERSITÄT LEIPZIG

Institut für Informatik

Die mittabstandstreue azimutale Abbildung der Erde ist winkel- und längentreu bezüglich des Hauptpunktes, hier die Stadt Leipzig.

Vorwort des Lesenden:

Die Idee zum vorliegenden Scriptum entstand während meiner Vorlesung „Geographische Informationssysteme II“, die ich im Wintersemester 2003/04 an der Universität Leipzig für Studenten der Fachrichtung Informatik gehalten habe. Weitere Interessenten kamen aus dem Bereich der Geographie. An dem Skript haben Studenten mitgearbeitet, denen ich auf diese Weise danken mˆchte:
Kap. x : Herr/Frau y

Dieter Sosna

Inhaltsverzeichnis

8 Quadtree-Strukturen, Buckets, Linearer Quadtree

8.1 *Kapiteleinführung*

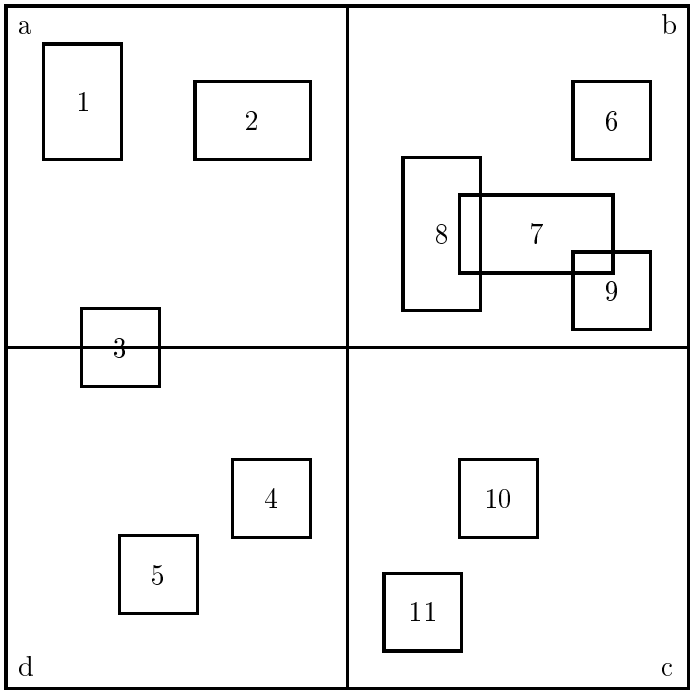
Die Qaudtree-Datenstruktur wird wegen ihrer Einfachheit im Allgemeinen für die Beschleunigung des Zugriffs auf die Objekte der 2D-Ebene benutzt. Es ist auch die Verallgemeinerung auf die 2^D-Bäume möglich. Im 3D-Raum wird die Octree-Datenstruktur für die Beschleunigung des Zugriffs eingesetzt.

Im folgenden skizzieren wir eine Variante für die Indexierung einer Sammlung von Rechtecken(mbb's genannt), die auf der Festplatte gespeichert sind. Diese Rechtecke sind neben Punkten für uns von Interesse, weil sie als minimale unbeschriebene Rechtecke(mbb's) von 2-dimensionalen geometrischen Objekten einen schnellen Vortest ermöglichen, ob z.B ein Punkt in einem Objekt liegt bzw. liegen kann.

Am Anfang betrachten wir ein rechteckiges Gebiet in einer Ebene zusammen mit den Rechtecken, die in diesem Gebiet liegen, als Grundgebiet. Diese Rechtecke(mbb's) werden durch die Koordinaten von diagonal liegenden Punkten beschrieben und bekommen außsuerdem Objektidentität(oid) zugewiesen. Damit besteht der Datensatz aus einem Paar(mbb,oid), d.h. aus dem Rechteck zusammen mit seiner Identität. Die Datensätze sollen in Buckets, die jeweils einer Plattenseite entsprechen, gespeichert werden.

Unseres Grundgebiet(Ebene, die auf ein Rechteck beschränkt ist) ist rekursiv in 4 Quadranten zerlegt und bleibt in diesem Anfangszustand, bis die Anzahl der Rechtecke, die einen Quadrant überlappen, kleiner oder gleich der Seitenkapazität ist. Die Quadranten werden als NordWest(NW), NordOst(NO), SüdWest(SW) und SüdOst(SO) bezeichnet. Der Index ist als ein geviertelter Baum(d.h. jeder innere Knoten hat je vier Kinder) dargestellt. Jedem Blatt ist eine Festplattenseite(Bucket) zugewiesen, die die Index-Einträge speichert. Wie im Falle mit ***Gridfile*** (siehe Kapitel 8 "Festgitter und Gridfile") tritt ein Rechteck in genau so vielen Blattquadranten auf, wie viele es überschneidet.

Das unten dargestellte Bild (Bild 8.1) zeigt als graphisches Beispiel die Partitionierung von der willkürlich von uns vorgegebenen Sammlung und ihr assozierter Baum. Dabei sind die Blätter mit den ihnen entsprechenden Rechtecken gekennzeichnet und die Seitengröfle wird in unserem Beispiel gleich 4 angenommen.



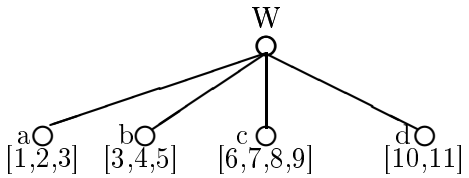


Bild 11.1

8.2 *Dynamisches Einfügen von Rechtecken*

Die Funktionsweise:

Ein Rechteck wird in jeden Blattquadrant eingefügt, den es überschneidet(vegleiche das Bild 8.1 mit dem Bild 8.2). Beim Einfügen eines Rechteckes werden alle Pfade verfolgt , die zu den vom Rechteck überschnittenen Blattquadranten führen (d.h., es wird in dem Baum nach allen vom Rechteck überschnittenen Blattquadranten gesucht).

Demzufolge ist das Einfügen nichts Anderes als die Suche mit dem einzigen Unterschied, dass bei der Suche kein einziger Eintrag hinzugefügt wird, sondern es wird festgestellt, ob ein Eintrag in dem Baum vorhanden ist oder nicht. Dabei wird Seite(Bucket) (sagen wir p) gelesen, die den Blättern zugewiesen wurde.

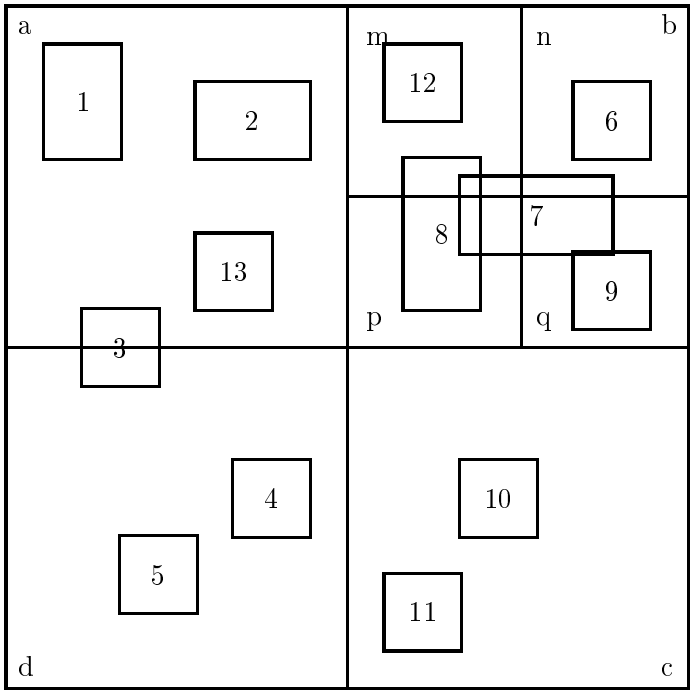
Beim Einfügen sind 2 Varianten möglich:

1. Entweder p ist nicht voll, dann wird der neue Eintrag hinzugefügt ohne den Quadrant in Subquadranten zu spalten und ohne die neue Buckets zu allokieren .
2. Oder p ist voll, dann wird
   - der Quadrant in vier Subquadranten gespalten,
   - der neue Eintrag zu den entstandenen Subquadraten hinzugefügt, die das Rechteck überschneidet,
   - drei neue Festplattenseiten allokiert, da die vorhandene Seite dann nicht mehr ausreicht.

   Im Endeffekt habe wir also 4 Buckets, zwischen denen jetzt die alte und neue Einträge verteilt sind.

Im folgenden führen wir ein Beispiel mit der verbalen Beschreibung und graphischen Darstellung (unter Zuilfenahme vom Bild 8.2) an:

1. Das Einfügen vom Rechteck 13 führt zu keiner Aufteilung des Quadranten. Es wird einfach zum Blattquadranten a hinzugefügt.
2. Das Einfügen vom Rechteck 12 führt zum Aufteilen des Quadrants b in vier Subquadranten m,n,p,q. Dieses Rechteck wird zu denjenigen neu gebildeten Blätterquadranten hinzugefügt, die er jeweils überschneidet. In unserem Falle wird es nur zum Subquadrant m hinzugefügt. Der Quadrant b wird in unserem Fall nur deswegen in Subquadranten aufgeteilt, weil die von uns am Anfang vereinbarte Seitekapazität von vier (Anzahl von Rechtecken, die im Quadranten enthalten sein müssen,also höchstens vier, um Quadrant nicht in Subquadranten aufzuteilen) überschritten wurde.


a
m
n
b
1
2
12
6
8
7
13
9
p
q
3
4
10
5
11
d
c

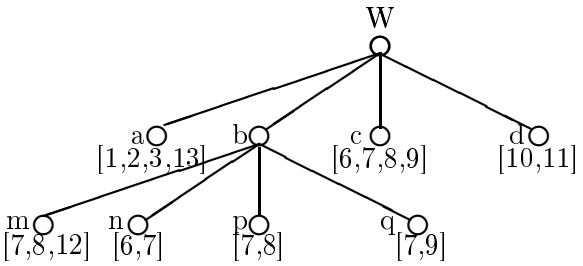

W
a
[1,2,3,13]
b
c
[6,7,8,9]
d
[10,11]
m
[7,8,12]
n
[6,7]
p
[7,8]
q
[7,9]

Bild 11.2

8.3 *Vor- und Nachteile*

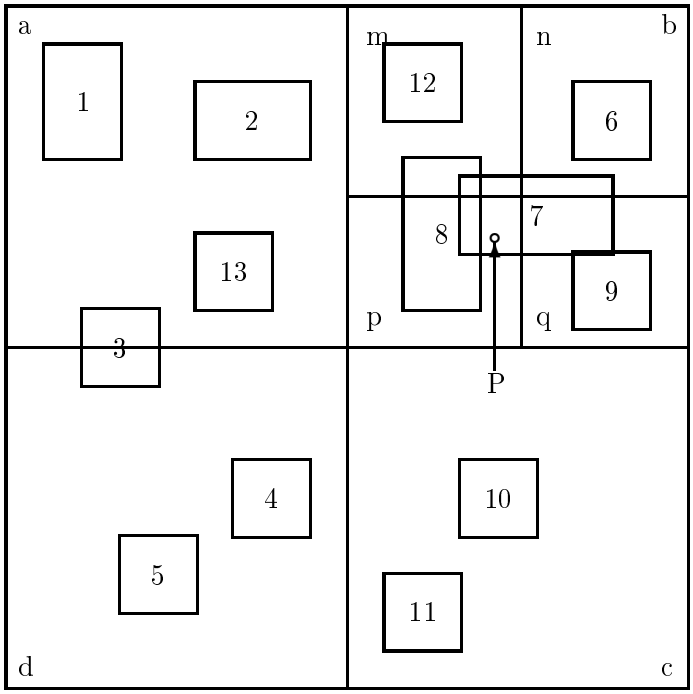
Diese Variante vom Quadtree entspricht nicht einigen Anforderungen von SAM(Spatial Access Methods), und zwar:

- Die Haupursache für den Mismatch ist Fan-Out(die maximale Anzahl von Sohnen bei jedem Quadrant), das gleich 4 ist, und das zum Belegen von nur einem kleinen Teil der Festplattenseite führt. Deshalb lässt sich der Quadtree nicht so einfach auf die Festplattenseite abbilden.
- Die Baumstrukturen mit dem groflen Fan-Out(solche wie B- oder R-Baum, die im Kaptitel 13 diskutiert werden) lassen sich effizienter auf die Festplattenseite abbilden und sind auch sehr gut für die Zugriffsmethoden auf den sekundären Speicher geeignet.
- Die Zeit, die für die Quadtree-Anfrage benötigt wird, ist vergleichbar mit der Baumtiefe, die ihrerseits ziemlich grofl sein kann. In dem Worst-Case ist jeder Baumknoten in einer getrennten Seite. Die Anzahl von Baumknotten in einer getrennten Seite und von Ein-/Ausgaben ist gleich der Baumtiefe.
- Die einzige Möglichkeit für die effiziente Abbildung der Baumknoten auf die Festpalltenseite besteht nur dann, wenn die Sammlung statisch ist. Im dynamischen Falle verschlechtert sich die Perfomance rasant.
- Ausserdem leidet der Quadtree genauso wie ***Gridfile***(siehe Kapitel 8) unter der Duplizierung von Objekten in verschiedenen Blättern.

8.4 *Motivation*

An dieser Stelle möchte ich noch einen kleinen Rückblick auf die ***Punktanfrage*** machen (siehe Kapitel 10 "Quadtree für Punkte"), ganz kurz die ***Fensteranfrage*** und die Raumfüllungskurve (in unserem Falle wird nur die Z-ordering-Kurve benötigt) beschreiben , da im letzten Teil dieses Kapitels und zwar bei dem linearen Quadtree dieses Wissen zum weiteren Verstehen benötigt wird.

Punktanfrage lässt sich sehr leicht mit dem Qaudtree realisieren. Es wird also ein Pfad von der Wurzel bis zu dem Blatt des Baumes verfolgt. Dabei wird auf jeder Ebene aus den vier möglichen Quadranten derjenige gewählt, der das Punktargument enthält. Das Blatt wird genauso gescannt und gelesen wie beim ***Grid*** (siehe Kapitel 8 "Festgitter und Gridfile"). Auf dem Bild 11.3 ist die Punktanfrage zusammen mit dem von der Wurzel bis zum Blatt des Baumes verfolgten Pfad und dem Ergebnis gleich 7 dargestellt. Für die ***Fensteranfrage*** müssen alle Blattquadranten gefunden werden, die sich das Fensterargument schneiden.



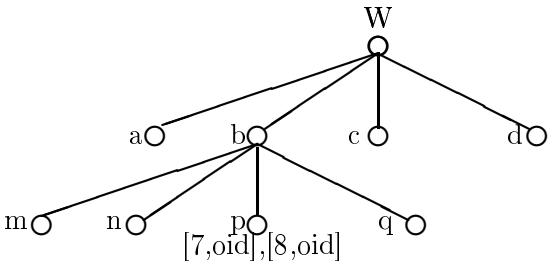


Bild 11.3

Eine ***Raumfüllungskurve*** legt eine totale Ordnung auf den Zellen von einem 2D-Gitter fest. Diese Ordnung ist sehr nützlich, da dadurch die Nähe der Zellen erhaltenbleibt. Das bedeutet, dass zwei Zellen, die sich im Raum in der Nähe befinden, fast genauso nah in der totalen Ordnung befinden werden. Selbst wenn es nicht immer der Fall ist, führen einige Kurven zu einer vernüftigen Annäherung diser "Näheeigenschaft". Die Ordnung soll unabhängig von der Lösung des Gitters stabil bleiben.

An dieser Stelle möchte ich eine der bekanntesten Raumfüllungskurven darstellen - die Z-Oder-Kurve (auch Z-Ordering-Kurve genannt). Zur Beschreibung der Z-Oder-Kurve benötigen wir ein Gitter, das darunter liegen soll. Dieses Gitter ist ein Array von N×N Zellen, wobei $N = 2^d$ ist. Es kann als ein kompletter Quadtree mit der Tiefe gleich d angesehen werden.

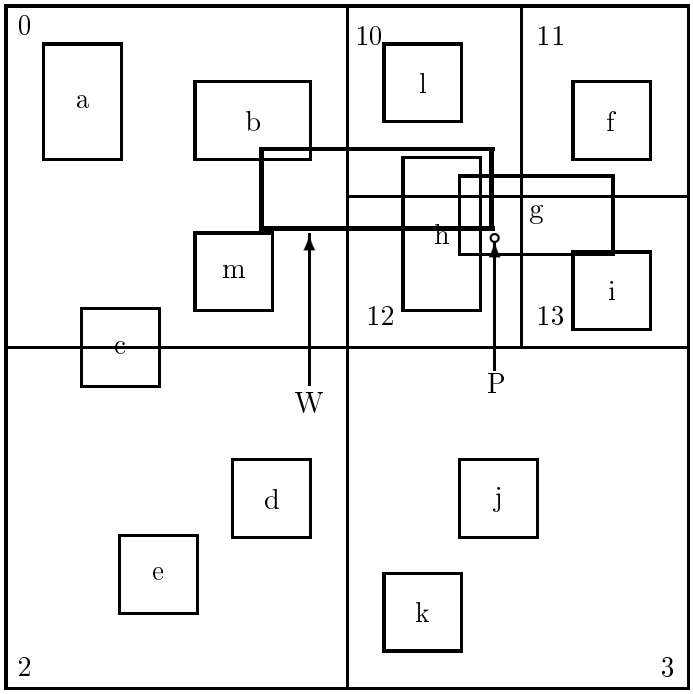
Z-Order:

Ein Label ist assoziert mit jedem Knoten des kompletten Quadtree und wird dabei als ein String über dem Alphabet (0, 1, 2, 3) gebildet. Das Label für die Wurzel ist ein leeres String. Die Söhner NW (NO, SW, SO) von dem internen Knoten k werden als k.0 (k.1, k.2, k.3) notiert, wobei "." als die String-Konkatenation bezeichnet wird.

Wenn die Zellen mit der Stringlänge d gelabelt sind, dann können wir sie gemäfl ihrer Labels in der lexikographischen Ordnung sortieren. Nehmen wir als Beispiel die Tiefe d=3 und sortieren die Labels in der aufsteigende Reiehenfolge. Nach der Sortierung befindet sich die Zelle 212 vor der Zelle 300 und nach der Zelle 21. Diese z-artige Ordnung (siehe unten das Bild 8.4) hat zum Name Z-Order beigetragen.

Beim Quadtree-Labeling gibt es Einiges zu beachten:

- Die Ordnung von Blättern entspricht dem Scannen von Blättern von links nach rechts.
- Die Labels haben verschiedene Gröflen. Also wenn ein Blatt B eine Gitterzelle Z enthält, dann ist das Label b von B ein Präfix von dem Label z von Z und wir haben $b < z$, wo das Zeichen "<" die aufsteigende lexikographische Ordnung auf den Strings bedeutet. Zum Beispiel $3 < 31 < 312$.
- Die Gröfle vom Label ist gleich der Tiefe vom Blatt in diesem Baum, dem dieses Label zugeordnet wurde.
- Das Label von einem Blatt kann als der Pfad von der Wurzel zu diesem Blatt angesehen werden.


0
10
11
a
b
l
f
g
h
m
i
12
13
c
W
P
d
j
e
k
2
3

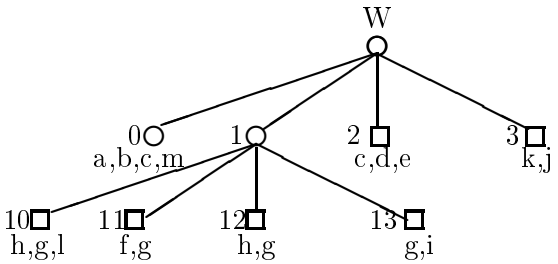

W
0
a,b,c,m
1
2
c,d,e
3
k,j
10
h,g,l
11
f,g
12
h,g
13
g,i

Bild 11.4

9 *Linearer Quadtree*

9.1 *Linearer Quadtree mit dem B***-Baum*

Wenn ein Eintrag [mbb,oid] einem Quadtree-Blatt mit dem Label b zugewiesen und in einer Seite mit Adresse p gespeichert wurde, dann wird in einem B*-Baum die Sammlung von Paaren (b,p) indexiert, die aufs Label b verweisen. (sihe Das Bild 11.5)

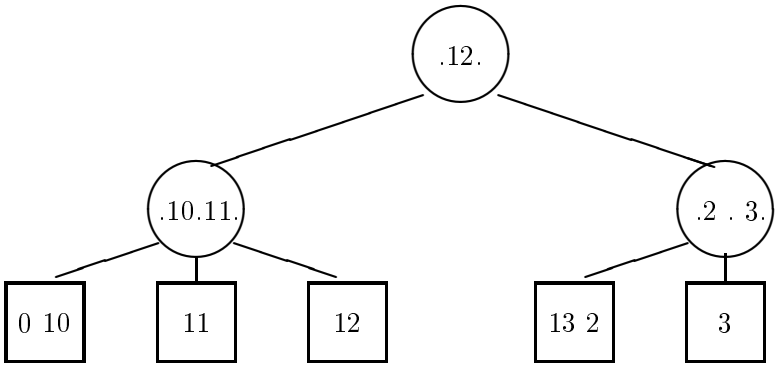


Bild 11.5 (2,3)B*-Baum

Solche Struktur führt zu einer guten Packung von Quadtree-Labels in den Bläettern vom B*-Baum. Diese Packung ist dynamisch. Sie ist vorhanden sowohl während des Einfügens als auch während des Löschens von Objekten in die Kollektion (z. B. sind die Einträge, die den Blättern mit den Lables 0 und 10 entsprechen, in dem selben Knoten). Solches Schema hat das gleiche Redundanzproblem wie die früher präsentierte Variante des Quadtree. D.h. Knoten, die verschiedene Qaudtree-Quadranten überlappen, sind in den Seiten duplieziert, die diesen Blättern zugewiesen sind (z. B. g ist 4 Mal in der gleichen Seiten gespeichert). Im folgenden schauen wir uns lineare Strukturen an, die Duplizierung von Knoten vermeiden.

9.2 *Punktanfrage mit dem linearen Quadtree*

Um die Identifikatoren von Objekten zu bekommen, deren Knoten(mbb) einen Punkt P enthält, benötigt man folgende 3 Schritte:

1. *Punktlabel berechnen.*

   Berechne das Label b (ein String der Länge d) von der Gitterzelle, die den Punkt P enthält. Das wird mit Hilfe der Methode *Punktlabel* des unten angegebenen Algorithmus bewerkstelligt (siehe LQ-PunktAnfrage).

2. *Retrieval vom Quadtree-Blatt in dem B*-Baum.*

   Sei B das Label vom Quadtree-Blatt, das P enthält. Wenn B=b, dann befindet sich das Qaudtree-Blatt in der Tiefe d. Wenn aber B ein Präfix von b ist, dann ist seine Länge kleiner als d und der Blattquadrant enthält die Gitterzelle, die P enthält. Genaugenommen entspricht dieser Quadrant dem Eintrag in dem B*- Baum, dessen Schlüssel der gröflte Wert ist, der kleiner als oder gleich b ist. Die Suche nach solchen Einträgen ist keine Basisfunktion von B*- Bäumen. Diese Funktion wird in unserem Algorithmus MaxInf(b) genannt (siehe LQ-PunktAnfrage).

3. *Zugriff aufs Blatt und Scannen vom Blatt*

   Wenn der Eintrag im B*- Baum gefunden wurde, dann muss auf die Seite mit der Adresse p zugegriffen, alle Paare [mbb,oid] in dieser Seite gescannt und anschließend geprüft werden, welcher der Knoten(mbb) den Punkt P enthält. Das Ergebnis ist die Liste von Objektidentifikatoren.

Das oben beschrieben Algorithmus ist auf dem oben stehenden Bild 8.4 dargestellt.

- *Schritt 1:*

  Aus den Koordinaten vom Punkt P findet man, dass die Gitterzelle, die P enthält, das Label 120 hat.

- *Schritt 2:*

  Mit Hilfe von der Funktion MaxInf stellt man schnell fest, dass das maximale Label, das kleiner als 120, das Label 12 ist.

- *Schritt 3:*

  In diesem Schritt bleibt es nur noch auf die Seite zuzugreifen , zu scannen und die Knoten aufs Enthalten vom Punkt P zu prüfen.

LQ-PunktAnfrageAlogrithmus: Quelle: Buch

```
LQ-PunktAnfrage(P:punkt):set(oid)
    begin
    Ergebnis=0
//Schritt 1: berechne das Label vom Punkt
     b = PunktLabel(P)
//Schritt 2: der Eintrag [B,p] wurde beim Traversieren vom B*- Baum mit //dem Schlüssel
b bekommen
     [L, p]=MaxInf(b)
//Schritt3: bekomme die seite und gebe die Objekten zurück
    Seite = LeseSeite(p)
    for each e in Seite do
     if (e.mbb contains P) then Ergebnis+=e.oid
    end for
    reutrn Ergebnis
    end
```

9.3 *FensterAnfrage mit dem linearen Quadtree*

Der Gedanke ist das Interval I von Labels zu berechnen, das alle Qaudranten abdeckt, die das Fenster W überschneidet. Dann stellt man eine Bereichsanfrage auf dem B*- Baum in dem Interval I.

Der Berechnungsvorgang wird in 3 Schritten abgewickelt, die im folgenden beschrieben werden:

1. Berechne das Label b vom NW-Fenstervertex. Das geschieht genauso wie in dem Schritt 1 vom PunktAnfrageAlgorithmus. Dann berechne MaxInf(b) als im Schritt 2 von diesem Algorithmus. Das ergibt dann [B,p], wobei das Label B die unterste Grenze vom Interval ist. Als Nächstes berechnen wir das Label b' vom SO-Fenstervertex mit MaxInf. Das führt zu [B',p'], wobei B' ist die oberste Grenze.
2. Stelle eine Bereichsanfrage im Interval [B,B'] auf dem B* -Baum, welche dann alle Einträge [b,p] zurückgibt, deren Label b in diesem Interval liegt.
3. Für jeden B*- Baumeintrag e=[b,p] berechne den Quadranten, der als e.b markiert ist, mit der Funktion Quadrant(e.b). Wenn der Quadrant(e,b) das Fenster überschneidet, dann greife auf die Quadrantenseite e.p zu und teste jeden von Quadtree-Einträgen auf die ‹berlappung mit W.

Sehen wir jetzt das Bild 8.4 auf dem das Anfragefenster mit dem Argument W abgebildet ist.

1. *Schritt 1:*

   Das Label vom NW-Fenstervertex von W ist 012 und das Label von SO-Fenstervertex ist 121. Dann suchen wir im B* - Baum Bidl 11.5 mit der Funktion MaxInf das gröflte Quadtreelabel, das kleiner als 012 und 121 ist. Wir bekommen MaxInf(012)=01 und MaxInf(121)=12.

2. *Schritt 2:*

   In diesem Schritt schreiben wir eine Bereichsanfrage auf dem B*- Baum (sihe Bild 8.4) in dem Interval [01,12]

3. *Schritt 3:*

   Für jeden Eintrag, der während des Scannens gefunden wurde, greifen wir auf die Seite nur dann zu, wenn W den Quadranten(e.b) überlappt.

LQ-FensterAnfrageAlogrithmus: Quelle: Buch

```
LQ-FensterAnfrage(W:rectangle):set(oid)
begin
    Ergebnis=0
//Schritt 1: Aus den Fenstervertexen W.nw und W.se berechnet man
//           das Interval [B,B']. Dazu wird die 2-malige Suche im B*- Baum
//           benötigt.
    b =PunktLabel(W.nw);[B,p]=MaxInf(b);
    b'=PunktLabel(W.se);[B',p']=MaxInf(b');
//Schritt 2: Man berechnet die Menge Q von B*- Baumeinträgen [b,p]
             in b ∈ [B,B']
    Q=BereichAnfrage([B,B'])
//Schritt 3: Man greift auf jeden Eintrag in der Menge Q zu, dessen Quadrant
//           W überlappt.
       for each q in Q do
          if (Quadrant(q.b) overlaps W) then
             Setie = LeseSeite(q.p)
// Es wird die Quadtree-Seite gescannt
             for each e in Seite do
               if (e.mbb overlaps W) then Ergebnis+=(e.oid)
             end for
          end if
       end for
//Es werden Ergebnisse sortiert und Duplikate eliminiert
    Sort(Ergebnis);RemoveDuplikate(Ergebnis);
     return Ergebnis
end
```

Literatur:

Literatur

[1] *ABC Kartenkunde*
Brockhaus Leipzig, 1983

[2] Bauer, Manfred: *Vermessung und Ortung mit Satelliten*,
Wichmann-Verlag, Karlsruhe, 1997, ISBN 3-87907-309-0

[3] Bill, Fritsch: *Grundlagen der Geoinformationssysteme, Bd. 1*
Wichman-Verlag, Heidelberg, 1994,

[4] Breusing, Dr. A: *Das Verebnen der Kugeloberfläche für Kartennetzentwürfe*
Verlag von H. Wagner und E. Debes, Leipzig, 1892

[5] Dana, P.H.: `http://www.utexas.edu/depts/grg/gcraft`,
University of Texas, Austin

[6] Hake, Grünreich: *Kartographie*
Walter de Gruyter, 1994

[7] Kuntz, Eugen: *Kartennetzentwurfslehre*,
2. Auflage, Wichmann-Verlag, Karlsruhe, 1990, ISBN 3-87907-186-1

[8] Schröder, Eberhard: *Kartenentwürfe der Erde*,
B.G. Teubner, Leipzig, 1988, ISBN 3-8171-1016-2
auch als Taschenbuch Band 61, Verlag H. Deutsch, 1999

Mathematische Literatur

FW39 Finsterwalder, R.: *Photogrammetrie*,
De Gruyter, Berlin, 1939, (UB:ZW Math. Geo_2-75)

FL94 Fischer, W.; Lieb, I.: *Funktionentheorie*,
Vieweg, 1994, ISBN 3-528-67247-1

Kl83 Klotzek, B.: *Einführung in die Differentialgeometrie*,
Band I und II, Deutscher Verlag der Wissenschaften, Berlin, 1983

La77 Langwitz, D.: *Differentialgeometrie*,
B.G. Teubner, Stuttgart, 1977, ISBN 3-519-12215-4

Lo97 Lorenz, F.: *Funktionentheorie*,
Spektrum Akad. Verlag, 1997, ISBN 3-8274-0197-6

Pr57 Privalov, I.I.: *Einführung in die Funktionentheorie I*,
B.G. Teubner, Leipzig, 1957
(in Lehrbuchsammlung der UB Leipzig mehrfach vorhanden)

Sch90 Schöne: *Differentialgeometrie*,
Reihe MINÖL, Band 6, B.G. Teubner, Leipzig, 1990, ISBN 3-322-00409-0

Karten und Datenquellen

- Landesvermessungsamt Sachsen
  Verzeichnis der Karten und digitalen Daten
  erscheint jährlich (?)
  Anschrift: Olbrichtplatz 3, 01072 Dresden, PF 100244

Software

- Wessel, Pål; Smith, Walter H.F.: *GMT - The Generic Mapping Tools*
  `ftp://gmt.soest.hawaii.edu/pub/gmt`