

Quadtrees

Bernhard Mallinger

e0707663

TU Wien

1. März 2011

Betreut durch Univ.-Ass. Dipl.-Ing., BSc Christian Schauer

Inhaltsverzeichnis

I. Theorie	4
1. Einführung	4
1.1. Funktionsprinzip	5
2. Punktbasierte Quadrees	5
2.1. Point Quadtree	5
2.1.1. Suche	6
2.1.2. Einfügen	8
2.1.3. Löschen	9
2.2. Pseudo Quadtree	10
2.3. k -d Tree	12
3. Bereichsbasierte Quadrees	14
3.1. MX Quadtree	14
3.2. PR Quadtree	16
II. Praktikum	18
4. Problemstellung	19
4.1. Unknown Horizons	19
4.2. Technische Details	19
4.3. Problembeschreibung	20
4.4. Ursprünglicher Lösungsansatz	22
5. Implementierung	23
5.1. Wahl des Quadreetyp	23
6. Analyse	25

7. Conclusio	25
Abbildungsverzeichnis	26
Listings	26
Literatur	27

Teil I.

Theorie

1. Einführung

Quadtrees stellen eine Datenstruktur dar, welche von Binärbäumen abgeleitet sind. Sie erweitern dessen Prinzip der rekursiven, hierarchischen Aufteilung eines Raumes auf mehrere Dimensionen. Im Allgemeinen beschränkt man sich hier zur Vereinfachung auf zweidimensionale Daten, wobei die Verallgemeinerung auf k -dimensionale Daten trivial ist.¹

Anstatt wie bei Binärbäumen den Raum in jedem Schritt in zwei Unterbäume zu teilen, werden bei Quadrees vier Kinder verwendet, um alle möglichen Richtungen in zwei Dimensionen bezüglich des aktuellen Knoten abzudecken. Der Name „Quadtree“ leitet sich hiervon ab (lat. „quadri-“: „vier-“). Analog dazu bezeichnet man 3-dimensionale Bäume als „Octrees“. Es ist weiters einfach zu sehen, dass k -dimensionale Bäume 2^k Kinder pro Knoten aufweisen.²

Räumliche Strukturen wie Quadrees sind generell dann sinnvoll, wenn die Entfernung zweier Punkte für die Anwendung relevant ist. Beispiele hierfür sind unter anderem Repräsentation von Bildern (angrenzende Bereiche besitzen oftmals ähnliche Farben), Kollisionserkennung, wo das Interesse auf sich überschneidenden Strukturen liegt, oder Bereichsabfragen in Datenbanken.

Beim Zugriff auf die so in einem Quadtree gespeicherten Daten unterscheidet man zwischen verschiedenen Anfragetypen:³

Point Query. Mit dieser Anfrage versucht man herauszufinden, welche Daten sich bei einem bestimmten Punkt befinden. Antworten auf diese Anfrage können entweder *NULL* bzw. ein Datenobjekt sein.

Range Query. Hier interessiert man sich für die Knoten, welche innerhalb von Bereichen,

¹Vgl. [FB74], Seite 9

²Vgl. [BF79], Seite 16

³Basierend auf [Knu98], Seite 559 und der Erweiterung durch [Ben75a]

die hinsichtlich mehreren Dimensionen ausgedehnt sein können, liegen. Zurückgegeben wird eine möglicherweise leere Liste von Datenobjekten.

Neighborhood Query. Dieser Anfragetyp fokussiert sich auf Nähebeziehungen. Konkret werden die Knoten gesucht, welche die geringste Entfernung zu einem bestimmten Knoten aufweisen oder sich innerhalb eines gewissen Radius davon befinden. Der Rückgabewert kann hier eine Liste oder ein einzelnes Datenobjekt sein.

1.1. Funktionsprinzip

Wie eingangs erwähnt, wird bei einem Quadtree der Raum in jedem Schritt in zwei Dimensionen geteilt. In diesem Kontext bezeichnet man die Abschnitte der Unterteilung als „Quadranten“. Dies kann, wie Abbildung 1 auf Seite 6 demonstriert, direkt an den Koordinaten der Knoten stattfinden, je nach Quadtreotyp werden hier jedoch verschiedene Strategien eingesetzt.

Die einzelnen Knoten eines Baumes beinhalten Verweise auf vier Kindstrukturen, welche diejenigen Daten beinhalten, die sich vollständig in der entsprechenden Richtung des Knotens befinden. Die Koordinaten der zu speichernden Daten bestimmen demnach die Gestalt der Struktur. Die eigentlichen Daten werden je nach Quadtreotyp in allen Knoten oder nur den Blättern gespeichert.

Dieses Prinzip der rekursiven, hierarchischen Dekomposition erlaubt das effiziente Suchen in logarithmischer Zeit, sofern eine adäquate Balancierung sichergestellt ist.⁴

2. Punktbasierte Quadrees

2.1. Point Quadtree

Dieser Quadtreotyp wurde erstmals 1974 von Finkel und Bentley in [FB74] eingeführt und stellt die Basis vieler weiterer Entwicklungen dar.⁵ Point Quadrees kann man als direkte

⁴Siehe Kapitel 2.1.1 auf Seite 6

⁵Vgl. [BKOS00], Seite 318

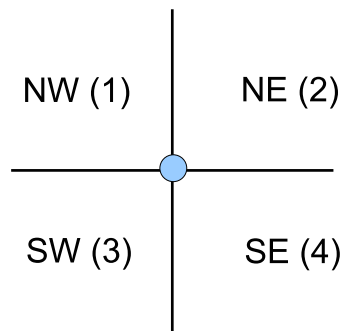


Abbildung 1: Alle Kinder eines Knotens repräsentieren Teilbäume, welche sich vollständig in NW-, NE-, SW- oder SE-Richtung des aktuellen Knoten befinden. Je nach Konvention können die Quadranten auch mittels ihrer Nummerierung referenziert werden.

Verallgemeinerung von binären Suchbäumen auf mehrdimensionale Räume auffassen.⁶ Ähnlich zu diesen erfolgt die Aufteilung direkt an den Koordinaten der Knoten, sowie an beiden Dimensionen gleichzeitig.

Abbildung 2 auf Seite 7 zeigt eine Instanz eines Point Quadrees in zwei üblichen Darstellungsvarianten: Baum- und Graphenform. In der Ersteren wird nur die Struktur des Baumes betrachtet, die Lage der Datenknoten ist ausschließlich relativ zueinander sichtbar. Die Zweitere konzentriert sich mehr auf die absolute Position der Punkte in einem Koordinatensystem und zeigt zusätzlich dazu die rekursive Dekompositionsstruktur ein. Die Bezeichnung der Knoten in der Grafik gibt hier die Reihenfolge des Einfügens an. Bereits hier wird deutlich, dass die Form des Baumes durch diese Reihenfolge determiniert wird.⁷

2.1.1. Suche

Der Algorithmus zur Suche von Koordinaten (Point Query) ergibt sich unmittelbar aus dem rekursiven Aufbauprinzip des Quadrees.

Die Suche beginnt an der Wurzel. In jedem Schritt werden je ein Vergleich der x - und y -Koordinate des aktuellen Knotens mit dem gesuchten Knoten durchgeführt, wodurch

⁶Vgl. [Sam90], Seite 48

⁷Vgl. Kapitel 2.1.2 auf Seite 8

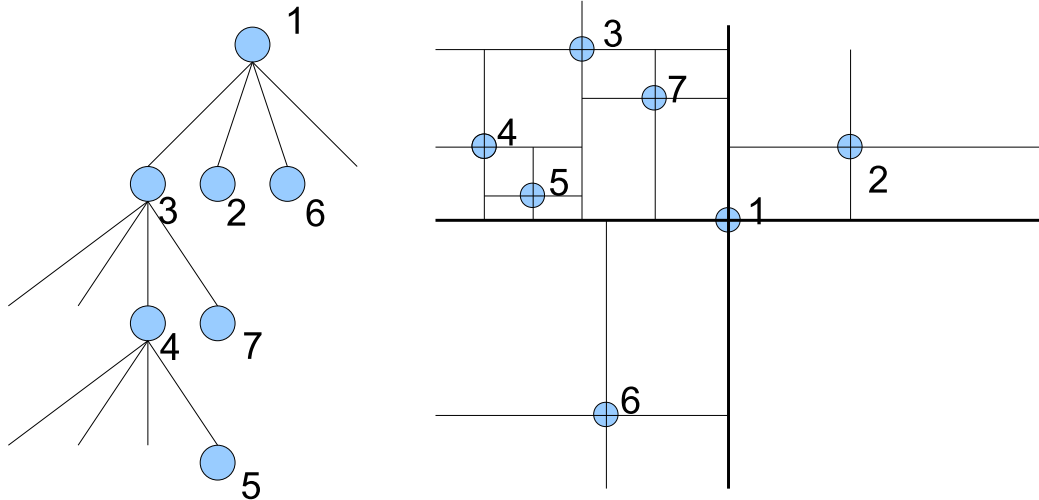


Abbildung 2: Gegenüberstellung: Ein Point Quadtree in Baum- bzw. Graphendarstellung

der Quadrant bestimmt wird, in welchen sich der gesuchte Knoten befindet. Ist dieser Quadrant leer, d.h. ist kein Kindknoten bezüglich dieser Richtung am aktuellen Knoten vorhanden, wird die Suche erfolglos abgebrochen; andernfalls wird sie bei dem Kindknoten, welcher diesen Quadranten abdeckt, rekursiv fortgesetzt, bis die Koordinaten des gesuchten Knoten mit jenen des aktuellen Knoten übereinstimmen.

Durch die rekursive Aufteilung kann hier im Average Case mit einer Gesamtpfadlänge⁸ von $N \log_4 N$ gerechnet werden, was in empirischen Untersuchungen mit Einfügungen in zufälliger Reihenfolge bestätigt wurde.⁹ Dies führt in der Folge bei der Suche nach einzelnen Punkten zu einer Laufzeit von $O(\log_4 N)$. Allerdings hängt diese Größe von der Balancierung des Baumes ab, wofür es beim Point Quadtree keine Garantien gibt,¹⁰ was im Falle einer Entartung (Worst Case) zu einer linearen Laufzeit ($O(n)$) führt.¹¹

Die Bereichssuche gestaltet sich ähnlich der Punktabfrage. Der essentielle Unterschied besteht darin, dass hier die Suche möglicherweise in mehreren Quadranten eines Knotens fortgesetzt werden muss. Die zu durchsuchenden Quadranten werden bestimmt, indem überprüft wird, ob der Bereich dieses Quadranten mit dem gesuchten Bereich überlappt. Weiters

⁸engl. TPL: Total path length

⁹Vgl. [FB74], Seite 4

¹⁰Vgl. Kapitel 2.1.2 auf Seite 8

¹¹Vgl. [Sam90], Seite 52

wird für jeden Knoten, welcher besucht wird, festgestellt, ob dieser sich in dem spezifizierten Bereich befindet; dieser wird gegebenenfalls zurückgegeben.¹² In dieser Formulierung wird ebenso sofort deutlich, dass die gesuchte Region nicht unbedingt ein Rechteck bilden muss.

2.1.2. Einfügen

Das Verfahren zum Einfügen neuer Daten baut direkt auf die Suche auf. In dem Quadranten, in welchem eine normale Suche abbrechen würde, wird das Datum eingefügt, indem an der entsprechenden Stelle, die sich offensichtlich Weise in diesem Quadranten befinden muss, der Datenknoten platziert wird. Konkret geschieht dies durch Setzen des diesem Quadranten zugeordneten Zeigers im Elternknoten. Der asymptotische Aufwand dieser Operation ist natürlich identisch mit jenem der Suche.

Der neu eingefügte Knoten teilt den Raum wiederum an seinen x - bzw. y -Koordinaten und erweitert so die Struktur des Baumes. Es ist schnell ersichtlich, dass hier, genau wie bei einem Binärbaum, Entartungen auftreten können, da sich die Struktur dynamisch mit jedem eingefügten Element herausbildet.

Ein besonders Problem stellen hier etwa geordnete Daten da. Werden diese in der gegebenen Reihenfolge eingefügt, degeneriert der Baum zu einer linearen Liste, wodurch ein linearer Such- und somit gleichzeitig auch Einfügearbeit entsteht, wie Abbildung 3 auf Seite 9 demonstriert.

Sind andererseits beim Aufbau des Baumes bereits alle Daten verfügbar, kann durch folgende Strategie eine optimale Balancierung erreicht werden: Der Median der sortierten Liste von Daten wird als Wurzel verwendet, die restlichen Elemente werden in 4 Gruppen eingeteilt, welche den Quadranten dieser Wurzel zugeordnet werden. Auf jede dieser Unterteilungen wird der Prozess rekursiv angewendet. Dadurch kann erreicht werden, dass für jeden Knoten kein Kind mehr als die Hälfte aller Subknoten von diesem enthält, was empirisch zu 15% geringeren Gesamtpfadlängen führt.¹³

¹²Vgl. [FB74], Seite 5f

¹³Vgl. [FB74], Seite 8f

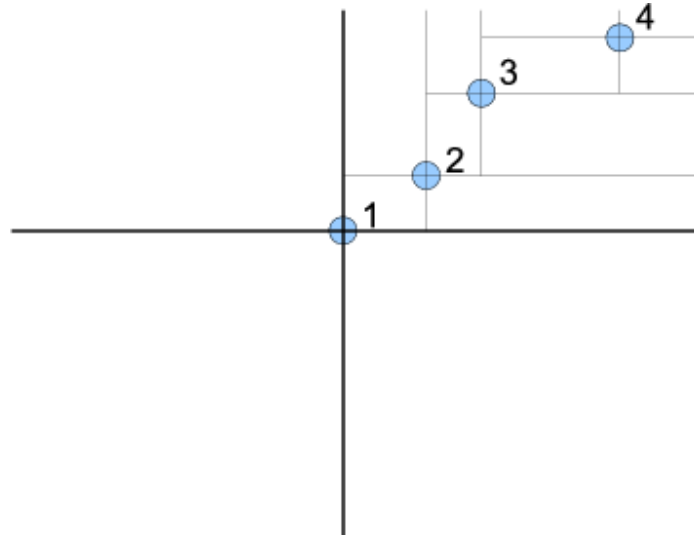


Abbildung 3: Ein Entarteter Point Quadtree

2.1.3. Löschen

Das Entfernen eines Knotens aus einem Quadtree ist im Allgemeinen eine komplexe Operation, da die Knoten alleine die Struktur des Baumes bilden, und diese auch nach dem Löschen weiterhin gewisse Eigenschaften innehaben muss. Daher ist es notwendig, eventuelle Kinder eines entfernten Knoten gemäß dem Baumaufbau neu zu platzieren.¹⁴ Bei Binärbäumen stellt dies kein allzu großes Problem dar, da in einem eindimensionalen Raum immer ein geeigneter Ersatz in Form des nächsten Knotens für einen zu entfernenden Knoten zur Verfügung steht.¹⁵ Im Fall von Quadrees muss eine Substitution hingegen Bedingungen hinsichtlich zweier Dimensionen genügen, wodurch es ein Näheverhältnis wie im eindimensionalen Fall nicht gibt.¹⁶ Hier ist ein Ersatz nur dann möglich, wenn ein Blattknoten existiert, für den sich im Bereich zwischen den Achsen, welche durch die Koordinaten des Blattknoten bzw. deren des zu entfernenden Knoten verlaufen, kein anderer Knoten befindet.¹⁷

Ursprünglich wurde vorgeschlagen, alle Kinder des zu entfernenden Knoten neu einzufügen, was einen sehr aufwändigen Prozess darstellen kann.¹⁸ Sollte kein Knoten das oben

¹⁴Vgl. [FB74], Seite 8

¹⁵Vgl. [Sam80], Seite 2

¹⁶Vgl. [Sam80], Seite 2

¹⁷Vgl. [Sam80], Seite 3

¹⁸Vgl. [FB74], Seite 8

genannte Kriterium erfüllen, kann nur durch günstige Wahl des Ersatzknotens die Anzahl der neu einzufügenden Knoten minimiert werden, wie etwa in [Sam80] gezeigt wird.

Eine weitere Optimierung kann dadurch erreicht werden, dass gelöschte Knoten nicht aus dem Baum entfernt, sondern nur als gelöscht markiert werden, um so die Struktur zu erhalten. Die markierten Knoten dürfen bei zukünftigen Suchen nicht mehr zurückgegeben werden. Nach jeder Entfernung erhöht sich hierdurch der Anteil der „toten“ Knoten, was sich negativ auf die Performance aller Operationen auswirkt, und periodische Restrukturierungsmaßnahmen des Baumes erfordert. Durch dieses Verfahren kann folglich der Aufwand einzelner Löschoptionen abgefedert werden, auf Kosten von regelmäßigen Wartungskosten.

2.2. Pseudo Quadtree

Der Pseudo Quadtree ist eine Abwandlung des Point Quadrees, welche Löschen in logarithmischer Zeit sowie einfachere Rebalancierung bei Deformation ermöglicht. Diese Form des Quadrees wurde 1982 von Overmars und van Leeuwen in [OL82] entwickelt.

Dem Pseudo Quadtree liegt die Analyse zu Grunde, dass die Probleme beim Entfernen von Knoten und bei der Balancierung bei Point Quadrees daher stammen, dass die Datenpunkte selbst die Struktur des Baumes bilden, und so nur äußerst wenig Flexibilität vorhanden ist.¹⁹ Aus diesem Grund wurde unterscheidet sich der Pseudo Quadtree davon insofern, als dass die Dekomposition des Raum an beliebigen Punkten stattfinden kann.

Konkret funktioniert das Einfügen beim Pseudo Quadtree wie folgt: Der Quadrant eines Datenpunktes wird gesucht. Ist dieser leer, wird der Knoten hier eingefügt, ohne den Raum an dessen Koordinaten weiter zu unterteilen. Befindet sich hier hingegen bereits ein anderer Knoten, muss eine neue Untergliederung eingeführt werden. Hier können beliebige Strategien verwendet werden, um einen Teilungspunkt zu bestimmen, wodurch ein der Anwendung angepasster, gut balancierter Baum entstehen kann. Als einzige Postcondition dieser Strategie muss garantiert werden, dass die Knoten in verschiedene Quadranten eingeteilt werden.²⁰

Abbildung 4 auf Seite 11 zeigt einen Pseudo Quadtree, welcher dieselben Daten wie der

¹⁹Vgl. [OL82], Seite 6

²⁰Vgl. [OL82], Seite 6

Point Quadtree in Abbildung 2 auf Seite 7 beinhaltet. In diesem Fall wurde der Mittelpunkt von zwei Knoten als Punkt zur Teilung verwendet.

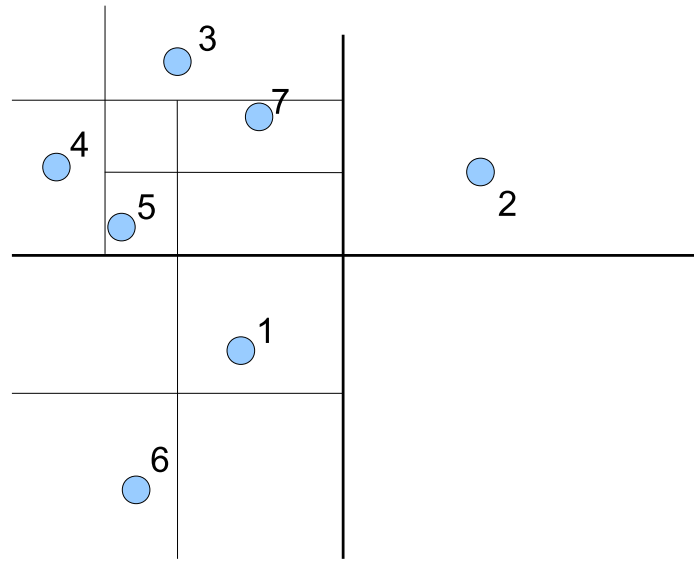


Abbildung 4: Pseudo Quadtree

Sollte trotz einer angepassten Strategie ein Ungleichgewicht entstehen, kann der Subbaum, dessen Wurzel der höchste Knoten ist, bei welchem eine unbalancierte Verteilung seiner Kindknoten festgestellt wird, neu strukturiert werden, indem eine neue Aufteilung der vorhandenen Knoten berechnet wird.²¹ Es kann gezeigt werden, dass im allgemeinen Fall eine Teilung möglich ist, in der sich in jedem der 4 Quadranten höchstens ein Drittel der Knoten befinden.²² Mit dieser Methode kann weiters garantiert werden, dass ein Pseudo Quadtree aufgebaut werden kann, dessen Höhe garantiert $\lceil \log_{d+1} n \rceil$ nicht überschreitet, wobei n die Anzahl der Daten darstellt.²³

Durch diese Aufteilung des Raumes an beliebigen Punkten befinden sich Daten schließlich nur noch an den Blättern. Dies wiederum ist der optimale Fall bezüglich dem Entfernen von Knoten – ein Knoten ohne Kinder kann jederzeit ohne zusätzlichen Aufwand gelöscht werden. Da das Entfernen eines Knotens immer auch das Suchen desselben erfordert, entsteht somit ein logarithmischer Aufwand.

²¹Vgl. [OL82], Seite 4f

²²Vgl. [OL82], Seite 7

²³Vgl. [OL82], Seite 8

2.3. k -d Tree

Diese Variante wurde von Bentley kurz nach dem Point Quadtree 1974 in [Ben75b] entwickelt. Der Name k -d deutet an, dass dieser Baum k -dimensionale Schlüssel verwendet.

Der k -d Tree kann wie der Point Quadtree als Verallgemeinerung des Binärbaums gesehen werden, hier wird hingegen ein etwas anderer Ansatz verfolgt: Anstatt den Raum gleichzeitig in allen Dimensionen zu partitionieren und somit 2^k Kinder pro Knoten zu erfordern, wird in jedem Schritt nur eine Dimension berücksichtigt und so eine binäre Unterteilung von eigentlich mehrdimensionalen Schlüsseln ermöglicht. Diese Mehrdimensionalität wird bei k -d Trees weiters dadurch ausgedrückt, dass die Dimensionen, bezüglich derer an einer gewissen Tiefe geteilt wird, beliebig alternieren. Im zweidimensionalen Fall könnte hier etwa bei geraden Tiefenebenen an der x -Achse bzw bei ungeraden an der y -Achse geteilt werden, wie etwa Abbildung 5 auf Seite 13 zeigt. Ursprünglich wurde genau dieses Abwechslungsmuster vorgeschlagen,²⁴ wobei dieses auch dynamisch beliebig gewählt und so an die Daten angepasst werden kann²⁵. Verwendet man hier einen dynamischen Ansatz, muss jedoch bei jedem Knoten der Diskriminator mitgespeichert werden.

Dieser Typ bietet gegenüber einem Point Quadtree Vorteile bezüglich des Platzbedarfs: anstatt 2^k sind hier nur 2 Kinder pro Knoten erforderlich. Dies löst das Problem von vielen unnötigen Leereinträgen, welche bei einem Point Quadtree bei einer großen Dimensionsanzahl auftreten können.

Weiters müssen bei einem Point Quadtree bei einer Suche in jedem Schritt alle Dimensionen verglichen werden. Bei einer großen Anzahl von Dimensionen kann dies überflüssig sein, wenn sich etwa Datensätze bereits durch eine echte Teilmenge ihrer Schlüssel einteilen lassen. In diesem Fall reicht das Überprüfen dieser Teilmenge, was in einem Point Quadtree nicht realisierbar ist, sich mit dem k -d Tree jedoch effizient verwirklichen lässt.²⁶

Die Operationen Suche und Einfügen ergeben sich direkt aus dem Funktionsprinzip des k -d Trees, beide benötigen offensichtlich eine logarithmische Laufzeit im Average Case²⁷ (sofern

²⁴Vgl. [Ben75b], Seite 2

²⁵Vgl. [Sam90], Seite 66

²⁶Vgl. [Sam90], Seite 66

²⁷Vgl. [Ben75b], Seite 1

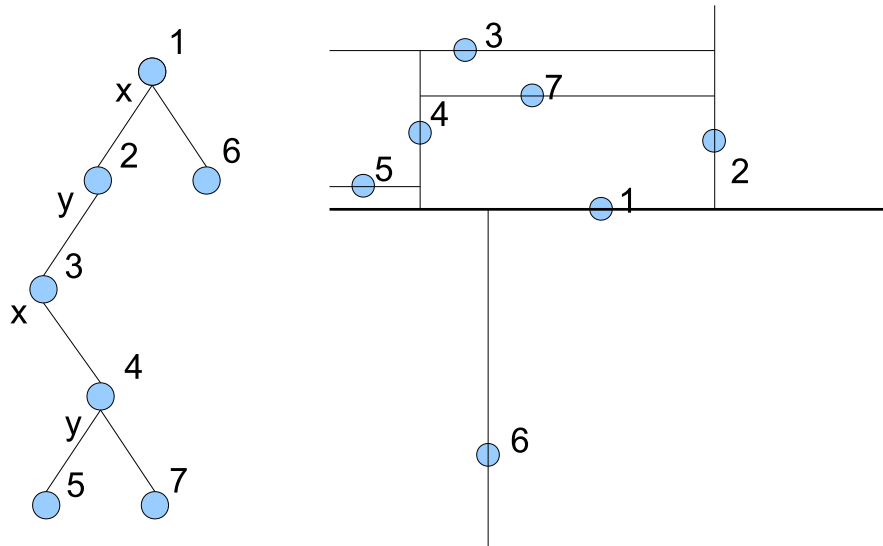


Abbildung 5: Ein k -d Tree: Die Beschriftung links der Unterteilungen gibt die Achse an, bezüglich welcher die Knoten aufgeteilt werden.

Bezüglich den vorhergegangenen Abbildung wurden hier die Koordinaten der Knoten leicht verändert, da in diesem Fall ein äußerst schlecht balancierter k -d Tree entstehen würde, was der Autor aus Demonstrationszwecken vermeiden möchte.

der Baum nicht entartet). Schwieriger ist hingegen das Löschen: Nach dem Entfernen eines Knotens darf die Struktur des k -d Trees nicht verletzt werden, weshalb ein geeigneter Ersatz gefunden werden muss. Für dieses Problem wurde eine Strategie entwickelt, welche auf der Lösung des analogen Problems bei Binärbäumen basiert und zugleich die Mehrdimensionalität im k -d Tree berücksichtigt. Der wesentliche Aspekt dieses Verfahrens ist, dass der Ersatzknoten nicht wie beim Binärbaum immer ein Blatt ist, sondern im Allgemeinen auch Kinder hat, wodurch der Ersatzknoten auch substituiert werden muss.²⁸ Anstatt im Worst Case (bei Löschung der Wurzel) n Neueinfügungen mit dem naiven Ansatz benötigt diese Methode nur $O(\log n)$ (bei einer Knotenanzahl von n).

Um diesem Problem gänzlich zu entgehen kann ähnlich dem Pseudo Quadtree²⁹ auch ein Pseudo k -d Tree³⁰ verwendet werden, in welchem analog zum Pseudo Quadtree die Un-

²⁸Für eine detaillierte Beschreibung dieses Verfahrens siehe [Ben75b], Seite 7

²⁹Siehe Kapitel 2.2

³⁰Vgl. [OL82], Seite 13ff

terteilung des Raumes nicht an den Koordinaten der Knoten, sondern an beliebigen Punkten vorgenommen wird, und so Daten nur in Blättern gespeichert werden.

3. Bereichsbasierte Quadrees

Wie im Kapitel 2 ausführlich erläutert wurde, erfolgt die räumliche Dekomposition bei punkt-basierten Quadrees immer anhand der gegebenen Punkte.³¹ Dieses Kapitel beschäftigt sich mit Quadrees, deren Form vordergründig a priori durch die Struktur des Raumes determiniert wird, was in etwa mit dem Prinzip von Tries vergleichbar ist.³²

Weiters wird in diesem Abschnitt nur insofern auf die verschiedenen Operationen eingegangen, als sie sich von punkt-basierten Quadrees unterscheiden.

3.1. MX Quadtree

Der MX Quadtree zeichnet durch seine Art der Raumunterteilung aus: Der Bereich, in dem die Datenpunkte liegen, wird unabhängig von den konkreten Daten unterteilt, und zwar rekursiv in vier gleichgroße Quadranten, bis auf der untersten Ebene 1x1 große Felder erreicht werden, welche die Daten beinhalten.³³

Diese Art der Dekomposition setzt folgendes voraus:

- (1) **Konstanter Bereich.** Die Region, in welcher sich Daten befinden, muss a priori bekannt sein und während des gesamten Programmablaufs konstant bleiben.
- (2) **Diskrete Koordinaten.** Die Schlüssel dürfen nur diskrete Werte annehmen, ähnlich den Einträgen einer Matrix, woher sich auch der Name „MX“ ableitet.³⁴
- (3) **Die Fläche des Raumes ist quadratisch mit einer Zweierpotenz als Seitenlänge.** Da der Raum aus praktischen Gründen in 4 gleichgroße Teile partitioniert wird, entsteht eine Gesamtgröße von $2^n \times 2^n$. Bei Datenregionen, welche nicht diese

³¹Bei Point bzw. k -d Trees gilt dies direkt; bei Pseudo Quadrees hingegen besteht die Abhängigkeit vielmehr indirekt: die Aufteilung stellt eine Funktion der Punkte dar.

³²Vgl. [Sam90], Seite 85

³³Vgl. [SW85], Seite 1f

³⁴Vgl. [Sam90], Seite 86

Ausdehnung besitzen, können Leereinträge verwendet werden, um diese Größe zu erreichen.

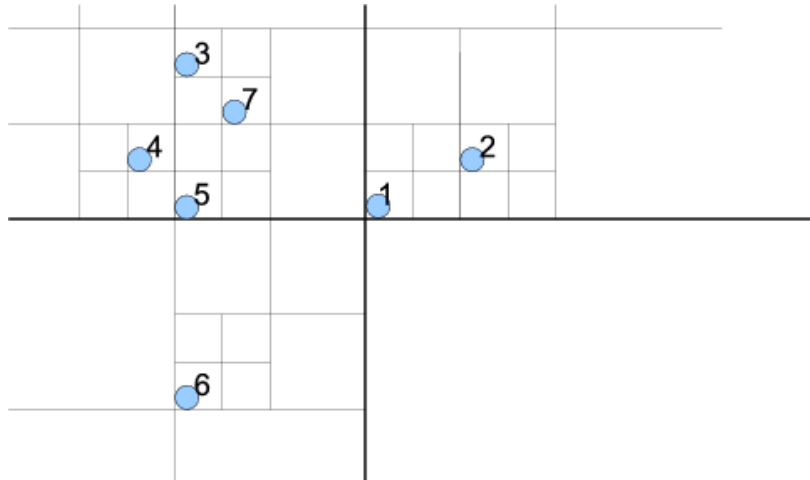


Abbildung 6: MX Quadtree. Diese Instanz ist eher spärlich besetzt.

Abbildung 6 auf Seite 15 zeigt dieselben Daten wie in den vorhergegangenen Grafiken in Form eines MX Quadtrees. Dabei fällt auf, dass die Aufteilung des Raumes hier unabhängig von der Reihenfolge des Einfügens der Knoten ist, was eine Konsequenz aus der Partitionierung auf Basis der Raumstruktur ist. Dies führt auch zu einer konstanten Höhe h des Baumes ohne Bezug zu den konkret vorhandenen Datenblättern, wodurch wiederum nur die begrenzte Anzahl von Knoten $2^h * 2^h$ in einem MX Quadtree gespeichert werden können.³⁵ Diese Einschränkung ist jedoch durch die obigen Bedingungen gesichert.

Ein klassischer Anwendungsfall dieses Prinzip sind Rastergrafiken, welche ein geeignetes mentales Modell für MX Quadtrees darstellen, da hier jeder Datenpunkt (Pixel) offensichtlich genau 1×1 groß ist und einer Koordinate zugeordnet ist.³⁶ Davon abgeleitet spricht man in diesem Kontext von weißen, schwarzen und grauen Knoten³⁷:

Schwarze Knoten sind entweder Datenblätter oder Knoten mit vier schwarzen Kindern.

Graue Knoten stellen Knoten dar, welche mindestens ein graues Kind besitzen oder mindestens je ein schwarzes und ein weißes Kind.

³⁵Vgl [Sam90], Seite 88

³⁶Vgl. [HS79]

³⁷Vgl [Gar82], Seite 2

Weißer Knoten sind leere Blätter ohne Daten bzw. Knoten, die weder direkt noch indirekt mittels Kindverweise auf Daten verweisen. Diese werden im Allgemeinen nicht explizit repräsentiert.

Eine konkrete Instanz eines MX Quadtree enthält somit nur schwarze und graue Knoten. Das Nicht-Repräsentieren weißer Knoten führt dazu, dass die Struktur des Baumes erst beim Einfügen aufgebaut wird. Dies ist jedoch nur eine Optimierung; es wäre genauso möglich, den vollständigen Baum mit weißen Knoten bzw. Blättern aufzufüllen, welche später grau oder schwarz eingefärbt bzw. von schwarzen Blättern ersetzt werden.

Das Löschen von Knoten im MX Quadtree ist weiters trivial, da im Gegensatz zu Point Quadrees hier die Daten keinen Einfluss auf die Struktur des Baumes haben und nur in Blättern gespeichert werden. Nach dem Entfernen des eigentlichen Datenblattes muss nur die Färbung der Knoten am Pfad von der Wurzel zu diesem Blatt überprüft und gegebenenfalls angepasst werden.

3.2. PR Quadtree

Die Einschränkung durch diskrete Koordinaten bei MX Quadrees ist manchmal in der Praxis nicht tragbar. Nichtsdestoweniger ist die Eigenschaft, dass dieser Baumtyp nicht entarten kann (da die Einfügereihenfolge keine Rolle spielt), oft sehr sinnvoll.

Der PR Quadtree stellt eine Möglichkeit dar, diesen Zwiespalt zu lösen. Er ist eine Kombination von Point und Region Quadtree, wovon sich auch der Name ableitet (**P**oint **R**egion Quadtree).³⁸ Hier orientiert sich die Aufteilung des Raumes ähnlich den MX Quadrees wiederum an der rekursiven Dekomposition eines fixen Bereiches, diese kann jedoch beliebig tief werden.

Die Einfügeoperation läuft wie folgt ab: Der tiefste Quadrant, welcher die Koordinaten eines einzufügenden Knotens beinhaltet, wird gesucht. Ist dieser leer, kann der Knoten hier platziert werden. Andernfalls muss eine weitere Unterteilung vorgenommen werden, was durch Partitionierung des Quadranten in gleichgroße Teile geschieht. Der Knoten, welcher

³⁸Vgl. [Sam90], Seite 92f

sich in dem Quadranten befand, wird in die neue Unterteilung überführt, und es wird neuerlich versucht, den einzufügenden Knoten in dieser zu platzieren. Sollte es hier erneut zu Konflikten kommen, wird der Prozess der Partitionierung rekursiv so lange wiederholt, bis beide Knoten in verschiedene Quadranten eingeteilt werden. Dies ist trivialerweise garantiert, wenn die Koordinaten verschieden voneinander sind.

Bei diesem Vorgang ist die Dekompositionsstruktur offensichtlich unabhängig von den konkreten Schlüsseln, wodurch die Einfügereihenfolge irrelevant wird und der Baum nicht im Sinne von Point Quadrees entarten kann.³⁹

Diese Strategie kann jedoch problematisch werden, wenn Datenpunkte nahe aneinander liegen. Hier müssen viele Unterteilungen getroffen werden, bis sie in verschiedene Quadranten eingeteilt werden können. Diese Struktur ist somit mit dieser Einschränkung für alle Schlüsseltypen geeignet.⁴⁰

Das Löschen aus einem PR Quadtree ist wiederum unkompliziert, da sich die Daten nur in Blättern befinden. Durch das Entfernen eines Blattes können Unterteilungen überflüssig werden, wenn etwa ein Knoten nur ein Kind besitzt. In diesem Fall kann der entsprechende Knoten einfach durch das Kind ersetzt werden.

³⁹Vgl [Sam90], Seite 96

⁴⁰Vgl [Sam90], Seite 95f

Teil II.

Praktikum

Dieser Abschnitt ist ein work in progress.

4. Problemstellung

In dem Praktikum habe ich mich mit einer Problemstellung aus dem Spiel „Unknown Horizons“ beschäftigt. Um das Umfeld der Aufgabenstellung deutlich zu machen, werde ich zuerst auf dieses Programm näher eingehen.

4.1. Unknown Horizons

„Unknown Horizons“ ist ein klassisches Aufbaustrategiespiel, in welchem eine Inselgruppe besiedelt wird. Das Szenario ist historisch gesehen vergleichbar zur Besiedelung Amerikas durch europäische Siedler.

Ziel des Spieles ist es, eine florierende Wirtschaft aufzubauen, welche die Bevölkerung mit Waren versorgt, die wiederum mit Steuern die laufenden Kosten der Produktionsketten und den Bau neuer Gebäude finanzieren. Aus Spieler_innensicht geschieht der Aufbau der Siedlung durch das Errichten von Gebäuden, welche anschließend weitestgehend autonom arbeiten.

Das Programm ist unter der GPL-2⁴¹ lizenziert und somit freie, offene Software. Es befindet sich noch in einer relative frühen Phase der Entwicklung, obschon der bereits implementierte Teil „spielbar“ ist. Nähere Informationen sind unter <http://www.unknown-horizons.org> verfügbar.

4.2. Technische Details

Die Spiellogik ist in Version 2.6 der Sprache Python⁴² implementiert und baut auf die Engine FIFE⁴³ auf. Weiters läuft das Spiel in nur einem Thread ab, weswegen jede Berechnung den Programmablauf blockiert, und somit sehr effizient sein muss.

Die Spielwelt stellt eine zweidimensionale Fläche dar. Sie ist in 1x1 große Tiles unterteilt, welche daher die kleinste Größeneinheit sind. Die Zuordnung von Tiles zu Koordinaten ist bijektiv und über den ganzen Spielverlauf hindurch konstant. Diese Tiles (dt.: „Fliesen“) sind

⁴¹<http://www.gnu.org/licenses/gpl-2.0.html>, abgerufen am 01.03.2011

⁴²<http://www.python.org>, abgerufen am 01.03.2011

⁴³<http://fifengine.de>, abgerufen am 01.03.2011

in verschiedene geographische Typen unterteilt (unter anderem Wasser, Land, Küste), denen zur graphischen Darstellung ein entsprechendes Bild zugeordnet ist. Desweiteren enthalten sie einen Verweis auf etwaige Gebäude, welche den Platz an dieser Koordinate einnehmen und stellen so die Basis der räumlichen Verwaltung der Inseln dar.

Aus diesem System folgt, dass alle Objekte im Spiel nur ganzzahlige, diskrete Koordinaten aufweisen.

4.3. Problembeschreibung

Die Problemstellung, welche in diesem Praktikum behandelt wird, ist eine Bereichsabfrage:

Im Spiel hat jedes Gebäude einen Einflussbereich, innerhalb welchem gewisse Aktionen ausgeführt werden können. Ein Holzfäller etwa kann nur Bäume fällen, welche sich innerhalb dieses Radius befinden, eine Schule nur Einwohner dieses Bereiches betreuen. Abbildung 7



Abbildung 7: Screenshot aus „Unknown Horizons“. Abbgebildet ist hier ein Bootsbauer inklusive seinem Aktionsradius, welcher den Bereich beschreibt, aus welchem dieser Rohstoffe für den Bootsbau einsammeln kann.

auf Seite 20 zeigt ein Gebäude, dessen Einflussbereich eingefärbt wurde.

Dieses Gebiet umfasst alle Tiles, welche folgende Ungleichung erfüllen

$$\textit{Distanz}(\textit{Gebaeude}, \textit{Tile}) \leq \textit{Radius des Einflussbereichs} \quad (1)$$

sowie sich

(1) im Areal der Siedlung

(2) an markierbaren Stellen

befinden. Da die Fläche eines Gebäudes eine beliebige rechteckige Form darstellen kann, entsteht durch diese Gleichung ein Rechteck mit abgerundeten Ecken, welches durch Bedingungen 1 und 2 beliebig beschränkt sein kann.

Nachdem dieser Bereich beim Gebäudebau einen zentralen Faktor darstellt, muss er dargestellt werden, wenn die Maus über die Spielwelt bewegt wird, um eine geeignete Position für ein zu bauendes Gebäude zu suchen, was offensichtlicherweise auch sehr schnell geschehen muss, um hier Verzögerungen zu vermeiden. Die Einfärbungsroutine wird nur dann aufgerufen, wenn der Mauszeiger einen Moment über einer gewissen Koordinate verweilt, um „Ruckler“ zu vermeiden. Im allgemeinen Anwendungsfall wird anschließend auf die Darstellung des Bereiches gewartet, um sicher zu stellen, dass das Gebäude mit diesem Aktionsradius die gewünschte Funktion übernehmen kann.

Die ursprüngliche Implementierung dieser Aufgabe ist für einen flüssigen Spielablauf zu ineffizient, besonders erfahrene Spieler_innen stoßen hier durch ihr zügiges Spieltempo auf Verzögerungen. Wie später noch deutlich werden wird, entstehen die Verzögerungen im Wesentlichen durch Berechnung der zu markierenden Tiles, das Übergeben dieser an die Engine sowie das eigentliche Einfärben. In dieser Arbeit werde ich mich auf die Optimierung des ersten Teils beschränken, was für zufriedenstellende Laufzeitergebnisse alleine in der Praxis nicht ausreicht. Die Performance der Engine ist hier weiters schwer messbar, was verschiedene Gründe hat: Diese ist in einer anderen Programmiersprache als die Spiellogik implementiert, wodurch bei jedem Funktionsaufruf (etwa die Übergabe der einzelnen Tiles) Zeit im Wrapper verbraucht wird, welcher das Interface bereitstellt. Die Markierungsfunktion

erledigt weiters nur einen Teil der Arbeit – das eigentliche Rendern findet in periodischen Durchläufen statt, in welchen das gesamte Bild berechnet wird. In diesen Prozess fließen somit viele äußere Faktoren ein.

4.4. Ursprünglicher Lösungsansatz

Die bisherige Methode zur Speicherung der Tiles stellt eine Hashmap dar, in welcher die Koordinaten der Insel als Schlüssel auf die Tileobjekte verweisen. Es sei darauf hingewiesen, dass hier Nähebeziehungen nicht repräsentiert werden.

Der Lösungsansatz besteht im Wesentlichen aus drei Schritten:

- (1) Berechnung der Koordinaten im Einflussbereichs.
- (2) Lookup dieser Koordinaten in der Hashmap.
- (3) Überprüfung der Tiles bezüglich Siedlungszugehörigkeit und Markierbarkeit.

Schritt 1 kann äußerst schnell durchgeführt werden. Zusammengefasst berechnet der verwendete Algorithmus die Grenze eines Viertels des Bereiches, die restlichen Viertel werden gespiegelt. Diese Begrenzungen können schließlich einfach zeilenweise aufgefüllt werden. In dieser Form ist der Ansatz nicht optimal (die Berechnung eines Achtels würde etwa auch ausreichen), in der Praxis ist hingegen der Anteil an der gesamten Laufzeit so gering, dass weitere Optimierungen nicht gerechtfertigt werden könnten.

Der nächste Schritt kann für einzelne Elemente auch sehr schnell umgesetzt werden, bei großen Flächen entsteht hier jedoch durch die Anzahl der Tiles eine messbare Verzögerung. Es ist nicht möglich, mehrere angrenzende Tiles gleichzeitig anzusprechen, jedes muss einzeln aus der Hashmap abgefragt werden. Besonders wenn viele Tiles am Meer liegen, wie es etwa in Abbildung 7 auf Seite 20 der Fall ist, scheitern alle diese Lookups, denn die Koordinatenberechnung hat keine Informationen über die Struktur der Insel.

Der dritte Schritt ist notwendig, da die Hashmap der Insel für verschiedene andere Anwendungsfälle sämtliche Tiles der Insel beinhalten muss; eine spezielle Hashmap zur Markierung könnte dies vereinfachen.

5. Implementierung

Nachdem bei diesem Markierungsproblem die Lage der Tiles zueinander eine entscheidende Rolle spielt, ist es naheliegend, dieses mittels Quadrees zu lösen. In einem solchen kann die Struktur der Inseln auf eine Weise abgebildet werden, welches es der Suche ermöglicht, Gebiete am Meer bzw. außerhalb der Siedlung schnell zu ignorieren sowie größere, zusammenhängende Blöcke effizient als innerhalb des Bereiches liegend zu identifizieren.

Der Aufbau und die Wartung des Quadrees können zwar je nach Typ einen Mehraufwand darstellen, dieser ist jedoch nur beim Spielstart (währenddessen ohnehin mit einer Verzögerung gerechnet wird) bzw. seltener beim Bau verschiedener Gebäude, welche den Einflussbereich der Siedlung erweitern, erforderlich. Insofern ist es hier sinnvoll, diesen Overhead in Kauf zu nehmen, um Verzögerungen während des Baumodus zu vermeiden.

Mit einer schnelleren Laufzeit kann hier vor allem dann gerechnet werden, wenn viele Bereiche bei der Suche ausgeschlossen werden können. In Abbildung 7 auf Seite 20 liegen etwa die Hälfte der Tiles im Einflussgebiet des Gebäudes am Meer und sind somit im Quadtree nicht vorhanden. Bei einer günstigen Partitionierung des Raumes ist somit zu erwarten, dass auf relativ hoher Ebene gewisse Kinder von Knoten, welche diesen Bereich abdecken, in der Terminologie von Abschnitt 3.1 „weiß“ und somit durch *NULL* repräsentiert sind, wodurch der Suchalgorithmus die Information bekommt, dass dieses gesamte Gebiet nicht weiter berücksichtigt werden muss. In der ursprünglichen Implementierung ist diese Information nicht vorhanden, wodurch jede Koordinate in dieses Bereichs in der Hashmap unnötigerweise abfragt wird.

Da die Unterteilung des Raumes in Quadraten vorgenommen wird und der Einflussbereich der Gebäude ein Rechteck mit abgerundeten Ecken ist, muss der Algorithmus an den Rändern bis zur Blattebene absteigen, um die genaue Grenze zu bestimmen.

5.1. Wahl des Quadreetyp

Die Art der im Quadtree zu speichernden Daten sind ausschließlich 1x1 große Tiles. Da man diese ebenso als Punkte interpretieren kann, eignen sich für diese Anwendung punktbasierte

sowie bereichsbasierte Quadtreetypen gleichermaßen.

Weiters liegt hier eine fixe Raumaufteilung vor: Siedlungen sind durch die Insel begrenzt und wachsen im Allgemeinen auf diese Größe. Dies bedeutet, dass im Falle von punktbasierten Quadtrees, welche an sich entarten können, optimierte Bäume berechnet werden können, um diesem Problem zu entgegen. Dadurch würde die Struktur dieser jener der bereichsbasierten Varianten ähneln, da alle Koordinaten innerhalb einer Inseln im Quadtree repräsentiert werden müssen, und die effizienteste Möglichkeit, dies zu tun, ist eine Aufteilung eines Bereiches genau in der Mitte, was exakt der Strategie des MX Quadrees entspricht.

Aus diesen Gründen, sowie der Einfachheit des Löschens, habe ich mich dazu entschieden, die gestellte Aufgabe mit einem MX Quadtree zu lösen. Es ist jedoch anzumerken, dass im Prinzip jeder der hier vorgestellten Typen für diese Problemstellung verwendet werden kann.

Listing 1: Python-ähnlicher Pseudocode zur Lösung des Markierungsproblems

```
"""
Klasse der Knoten des MX Quadrtrees
"""
class Node(object):

    [...]

    """
    Rekursive Methode der Klasse Node, welche einen Callback auf alle
    Blaetter anwendet, die innerhalb der Bereichsspezifikation "area"
    liegen.
    """
    def visit_radius_tiles(self, area, callback):
        if self.is_leaf():
            # Auf Blattebene muss nur ueberprueft werden, ob der Callback
            # auf den aktuellen Knoten angewendet werden soll
            if area.includes(self):
                callback(self)
        else:
            quadrants = self.get_overlapping_quadrants(area)
            for quadrant in quadrants:
                # Optimierung: Sollte der ganze Quadrant im Bereich liegen,
                # muessen die einzelnen Subknoten nicht mehr einzeln
                # ueberprueft werden.
                if self.check_full_quadrant_included(quadrant, area):
                    quadrant.visit_all_tiles(callback)
                else:
                    quadrant.visit_radius_tiles(area, callback)
```

6. Analyse

7. Conclusio

Abbildungsverzeichnis

1.	Funktionsprinzip eines Quadtree	6
2.	Point Quadtree	7
3.	Entarteter Point Quadtree	9
4.	Pseudo Quadtree	11
5.	k -d Tree	13
6.	MX Quadtree	15
7.	Screenshot aus „Unknown Horizons“	20

Alle in dieser Arbeit verwendeten Bilder wurden vom Autor erstellt.

Listings

1.	Python-ähnlicher Pseudocode zur Lösung des Markierungsproblems	25
----	--	----

Literatur

- [Ben75a] BENTLEY, Jon L.: A survey of techniques for fixed radius near neighbor searching. Stanford, CA, USA : Stanford University, 1975 (333). – Forschungsbericht. – 444
- [Ben75b] BENTLEY, Jon L.: Multidimensional binary search trees used for associative searching. In: *Commun. ACM* 18 (1975), September, 509–517. <http://dx.doi.org/http://doi.acm.org/10.1145/361002.361007>. – DOI <http://doi.acm.org/10.1145/361002.361007>. – ISSN 0001–0782
- [BF79] BENTLEY, Jon L. ; FRIEDMAN, Jerome H.: Data Structures for Range Searching. In: *ACM Comput. Surv.* 11 (1979), December, 397–409. <http://dx.doi.org/http://doi.acm.org/10.1145/356789.356797>. – DOI <http://doi.acm.org/10.1145/356789.356797>. – ISSN 0360–0300
- [BKOS00] BERG, Mark de ; KREVELD, Marc van ; OVERMARS, Mark ; SCHWARZKOPF, Otfried: *Computational Geometry: Algorithms and Applications*. Second. Springer-Verlag, 2000. – 367 S. <http://www.cs.uu.nl/geobook/>
- [FB74] FINKEL, Raphael A. ; BENTLEY, Jon L.: Quad Trees: A Data Structure for Retrieval on Composite Keys. In: *Acta Inf.* 4 (1974), S. 1–9
- [FGPR91] FLAJOLET, Philippe ; GONNET, Gaston ; PUECH, Claude ; ROBSON, J. M.: The analysis of multidimensional searching in quad-trees. In: *Proceedings of the second annual ACM-SIAM symposium on Discrete algorithms*. Philadelphia, PA, USA : Society for Industrial and Applied Mathematics, 1991 (SODA '91). – ISBN 0–89791–376–0, 100–109
- [Fre60] FREDKIN, Edward: Trie memory. In: *Commun. ACM* 3 (1960), September, 490–499. <http://dx.doi.org/http://doi.acm.org/10.1145/367390.367400>. – DOI <http://doi.acm.org/10.1145/367390.367400>. – ISSN 0001–0782
- [Gar82] GARGANTINI, Irene: An Effective Way to Represent Quadtrees. In: *Commun. ACM* 25 (1982), Nr. 12, S. 905–910

- [HS79] HUNTER, G.M. ; STEIGLITZ, K.: Operations on Images Using Quad Trees. In: *PAMI* 1 (1979), April, Nr. 2, S. 145–153
- [Knu98] KNUTH, Donald E.: *The Art Of Computer Programming, Volume 3: Sorting and Searching*. Second. Redwood City, CA, USA : Addison Wesley Longman Publishing Co., Inc., 1998. – ISBN 0–201–89685–0
- [Lue78] LUEKER, George S.: A Data Structure for Orthogonal Range Queries. In: *FOCS*, 1978, S. 28–34
- [OL82] OVERMARS, Mark H. ; LEEUWEN, Jan van: Dynamic Multi-Dimensional Data Structures Based on Quad- and *K-D* Trees. In: *Acta Inf.* 17 (1982), S. 267–285
- [Sam80] SAMET, Hanan: Deletion in Two-Dimensional Quad Trees. In: *Commun. ACM* 23 (1980), Nr. 12, S. 703–710
- [Sam84] SAMET, Hanan: The Quadtree and Related Hierarchical Data Structures. In: *ACM Comput. Surv.* 16 (1984), Nr. 2, S. 187–260
- [Sam90] SAMET, Hanan: *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, 1990
- [SW85] SAMET, Hanan ; WEBBER, Robert E.: Storing a collection of polygons using quadtrees. In: *ACM Trans. Graph.* 4 (1985), July, 182–222. <http://dx.doi.org/http://doi.acm.org/10.1145/282957.282966>. – DOI <http://doi.acm.org/10.1145/282957.282966>. – ISSN 0730–0301