

Programmierkonzepte und Algorithmen: Quad-/Octrees

Spatiale Suche und Kollisionserkennung

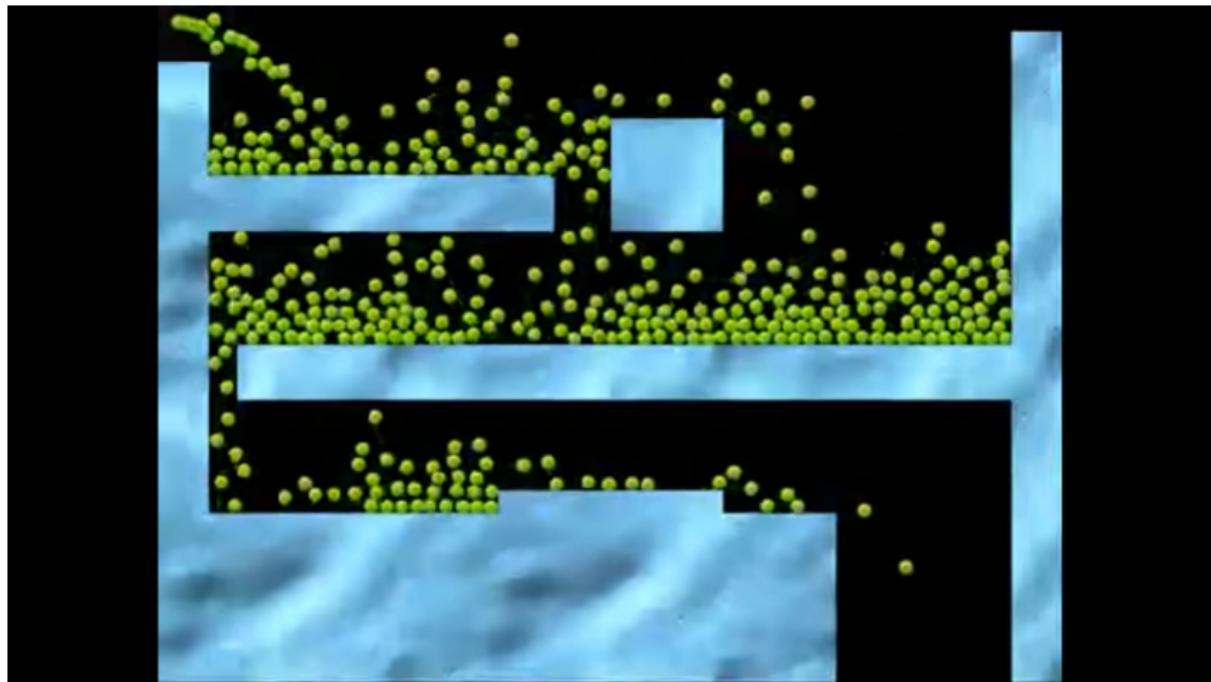
Fadi Moukayed (539502)
Eugen Kinder (539723)

ProgKo WS2016/17 – HTW Berlin AI(M)

- 1 Problematik
- 2 Grundlagen
- 3 Quaternärbaume
- 4 Algorithmen
- 5 Octrees
- 6 Komplexität
- 7 Abschluss

Spieleprogrammierung

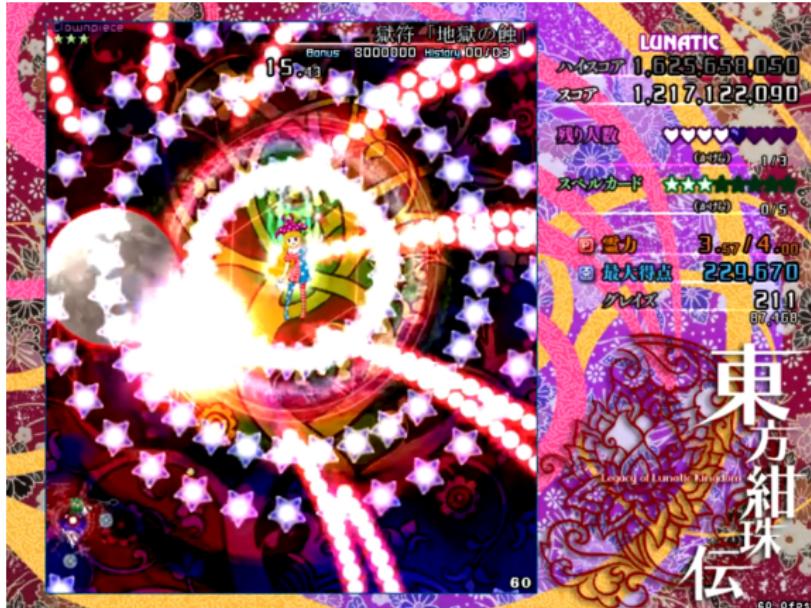
Fast real-time collision detection – TheDigitalSorceress/YouTube



Fast real-time collision detection – TheDigitalSorceress/YouTube

Spieleprogrammierung

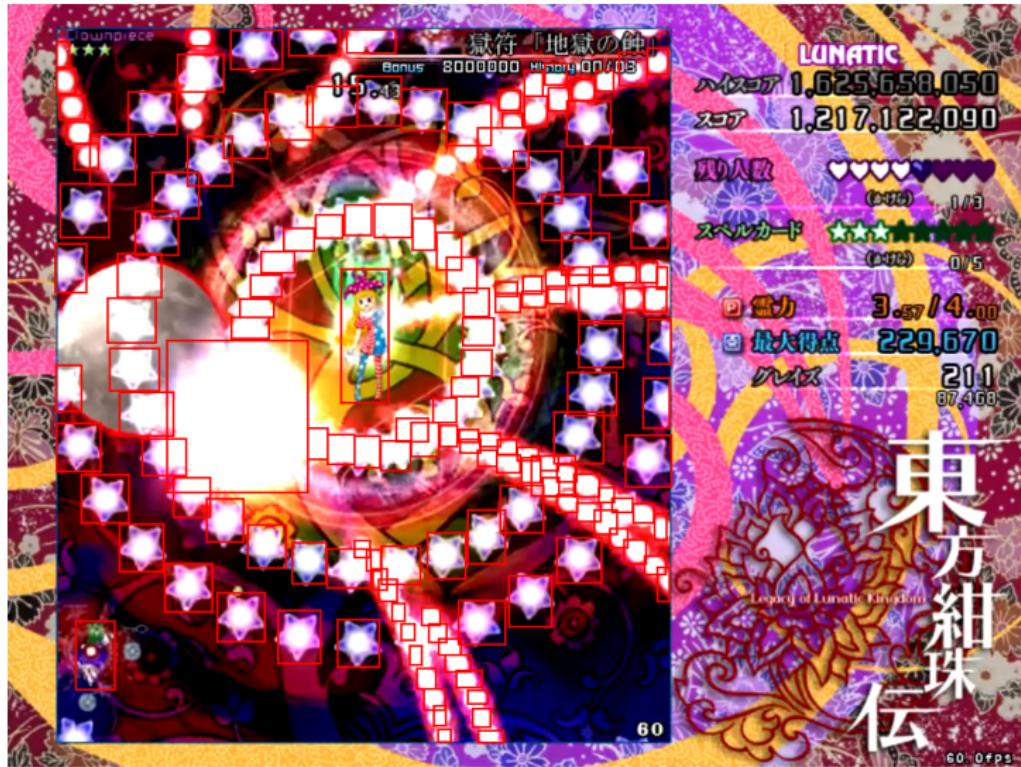
Touhou 15 LoLK: Legacy of the Luncatic Kingdom (2015)



Touhou 15 LoLK: Legacy of the Lunatic Kingdom (2015)

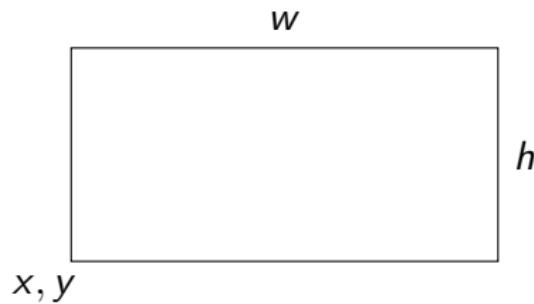
Spieleprogrammierung

Touhou 15 LoLK: Legacy of the Luncatic Kingdom (2015) - Bounding Boxes



Grundlagen

Bounding Boxes - Geometrie

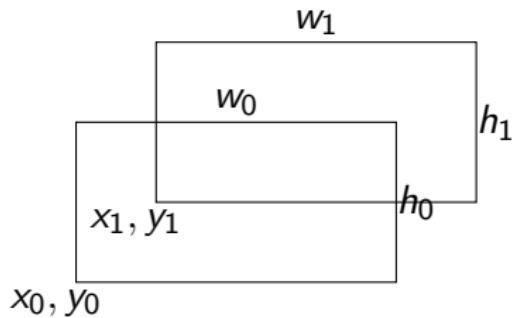


Def. Quadrat

- $(x, y, w, h) \in \mathbb{R}^4$
 - $x, y \in \mathbb{R}$
 - $w \in \mathbb{R}$ (En.: Width / Breite)
 - $h \in \mathbb{R}$ (En.: Height / Höhe)

Grundlagen

Bounding Boxes - Überschneidung



Def. Intersect

$\text{intersect} : \mathbb{R}^4 \times \mathbb{R}^4 \rightarrow \mathbb{B}$

$$(x_0, y_0, w_0, h_0), (x_1, y_1, w_1, h_1) \mapsto (x_0 + w_0 \geq x_1) \wedge (y_0 + h_0 \geq y_1) \wedge (x_0 \leq x_1 + w_1) \wedge (y_0 \leq y_1 + h_1)$$

Naive Kollisionserkennung

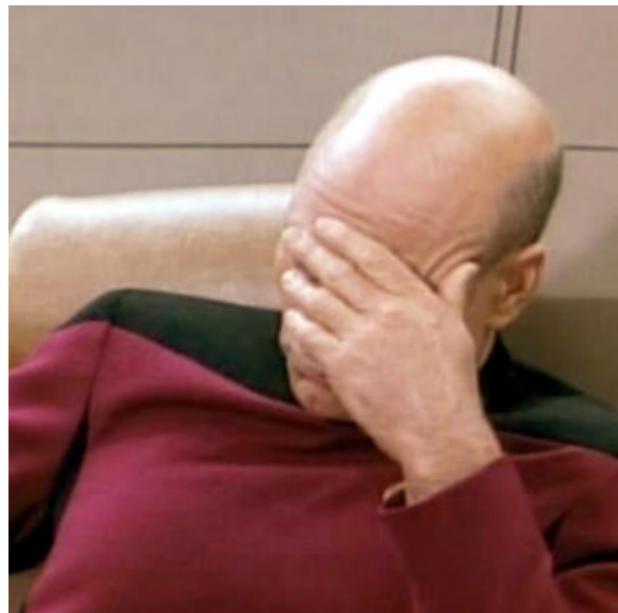
Komplexität

- Menge der Objekte $O = \{o_1, o_2, \dots, o_n\}$
 - $O \subseteq \mathbb{R}^4$
 - $|O| = n$
- Für o_1
 - $\text{intersect}(o_1, o_2) \vee \text{intersect}(o_1, o_3) \vee \dots \vee \text{intersect}(o_1, o_n)$
 - $(n - 1)$ intersect-Operationen
- Für alle Objekte in O
 - $n(n - 1)$ intersect-Operationen

Naive Kollisionserkennung

Komplexität

- Naive Kollisionserkennung
 - Komplexität pro Objekt: $O(n)$
 - Komplexität für alle Objekte: $O(n^2)$



Picard Facepalm, Deja Q: Star Trek TNG

Naive Kollisionserkennung

Komplexität

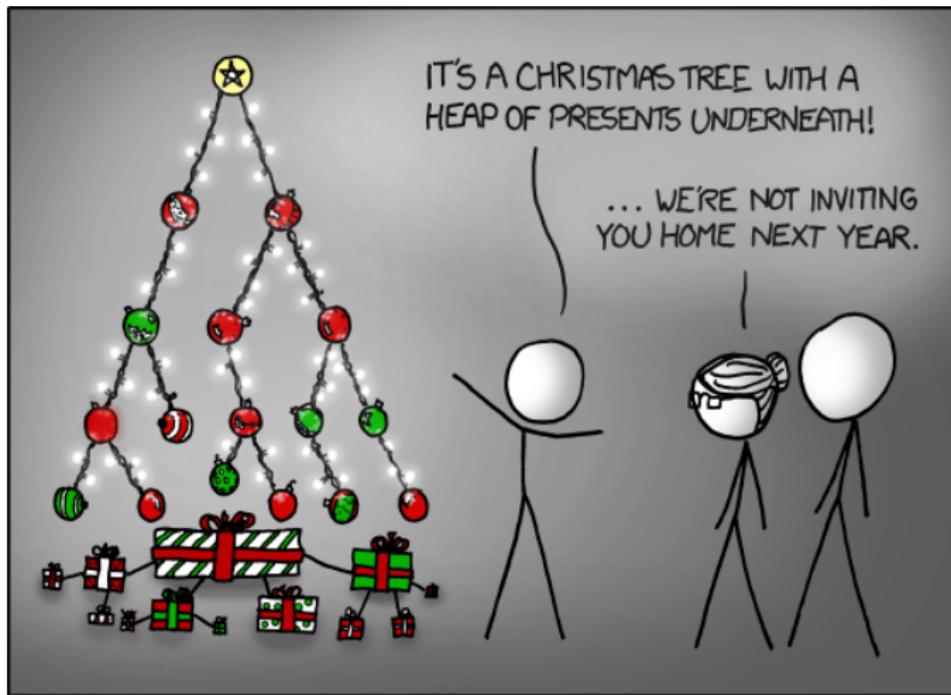
- Es gibt alternativen
- Trees!



J. L. Picard: Star Trek TNG

Bäume

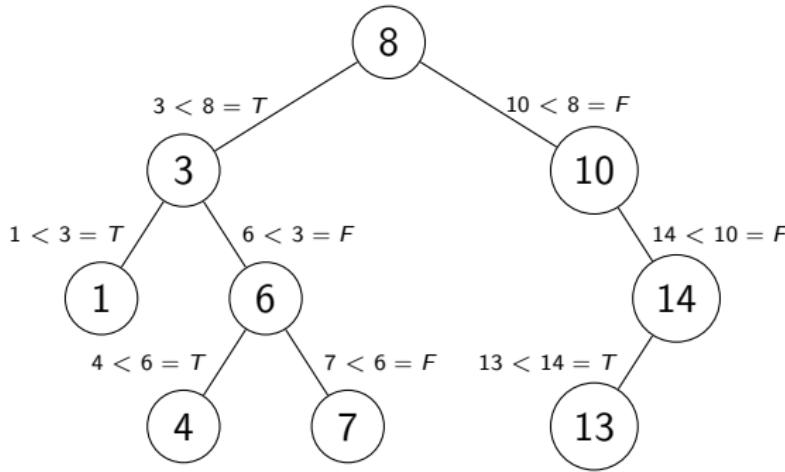
Recap



XKCD #835: Tree

Bäume

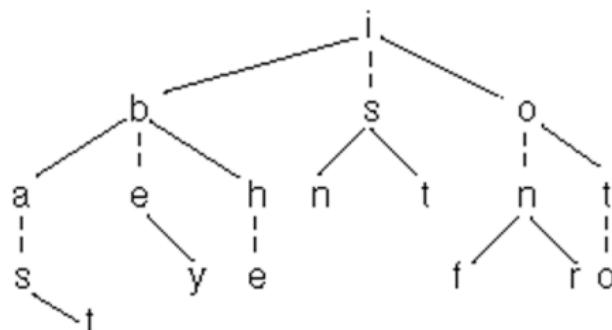
Binär



- Binary Search Tree (BST)
 - Binäre Ordnungsrelation „ $<$ “
 - $<: \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{B}$

Bäume

Ternär



as at be by he in is it of on or to

Ternary search tree for 12 words, J. Bentley, B. Sedgewick, Dr. Dobbs (1998)

- Ternary Search Tree (TST)

- Ternäre Ordnungsrelation cmp
- $cmp : \Sigma \times \Sigma \rightarrow \{lo, eq, hi\}$

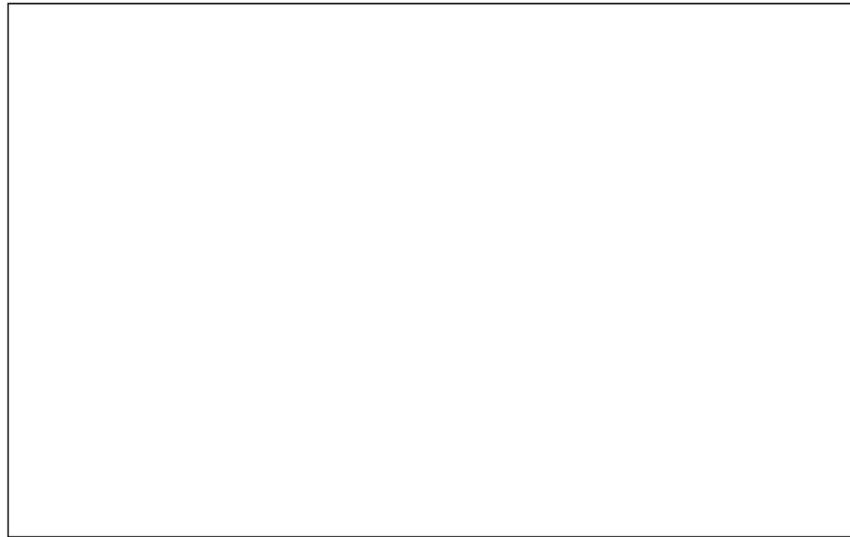
Bäume

Quaternär

- Quaternärbaum
- Quaternäre Ordnungsrelation
- $? : ? \times ? \rightarrow Y$
 - $|Y| = 4$

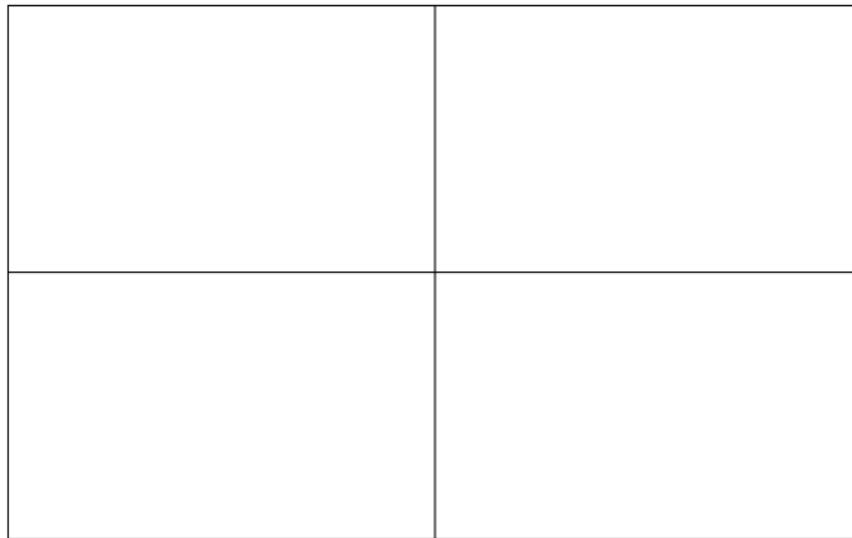
Quaternärbaume

Konzept



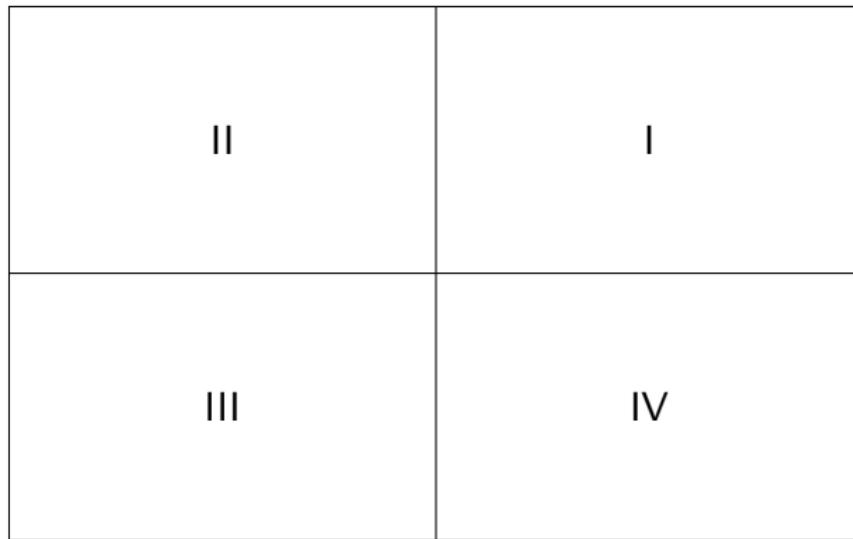
Quaternärbaume

Konzept



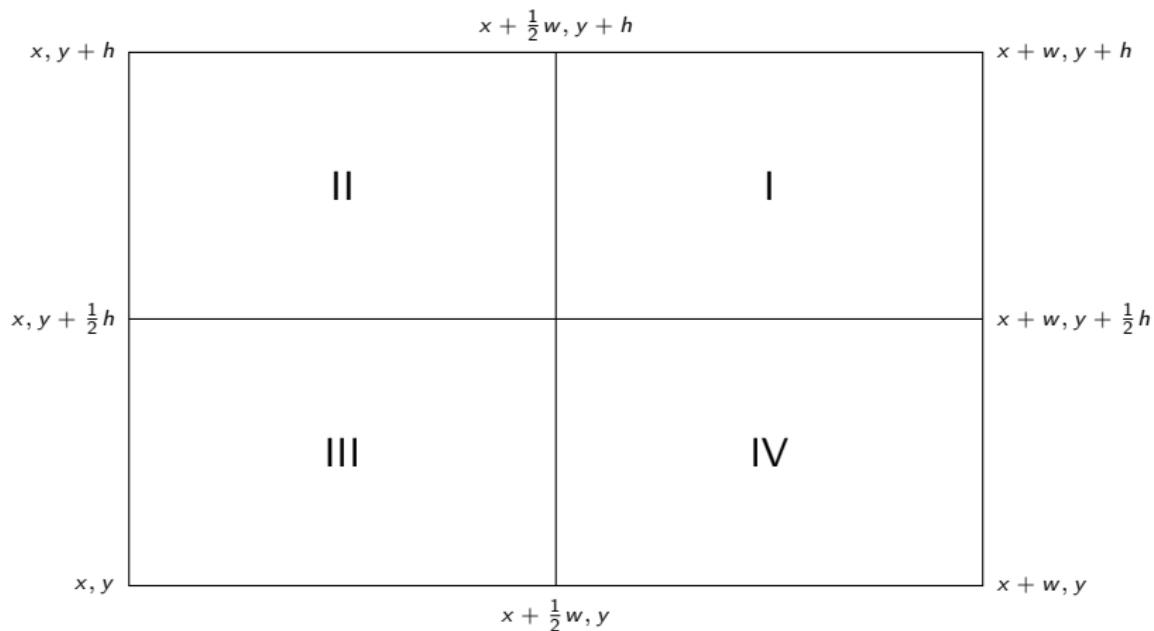
Quaternärbaume

Konzept



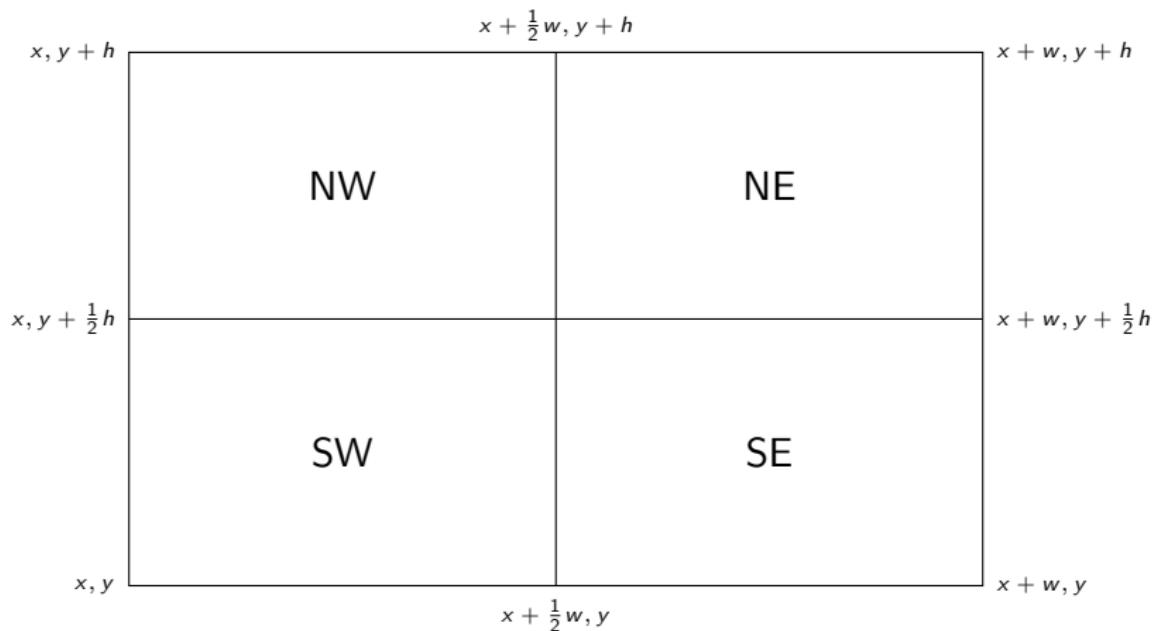
Quaternärbaume

Konzept



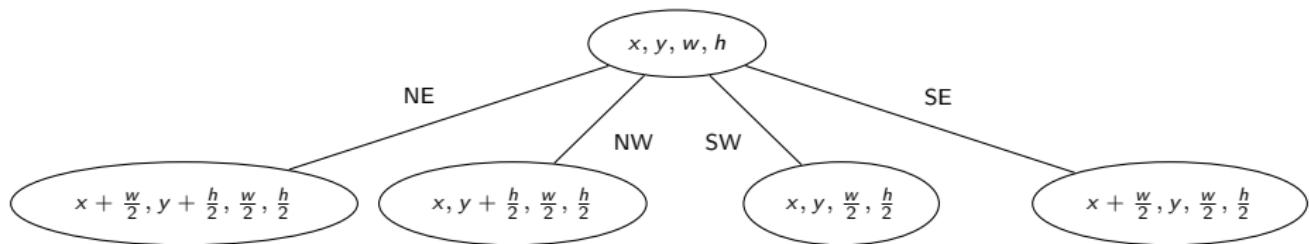
Quaternärbaume

Konzept



Quaternärbaume

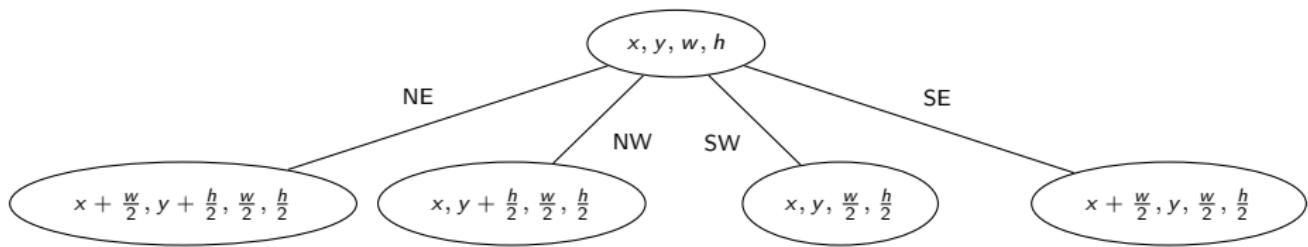
Datenstruktur



- Quaternäre Ordnungsrelation
- $quadrant : \mathbb{R}^4 \times \mathbb{R}^4 \rightarrow \{NE, NW, SW, SE\}$

Quaternärbaume

Datenstruktur



```

template <typename T> struct node {
    std :: vector<T> elements;
    double x, y, w, h;
    std :: array<std :: unique_ptr<node>, 4> children;
};
  
```

Quaternärbaume

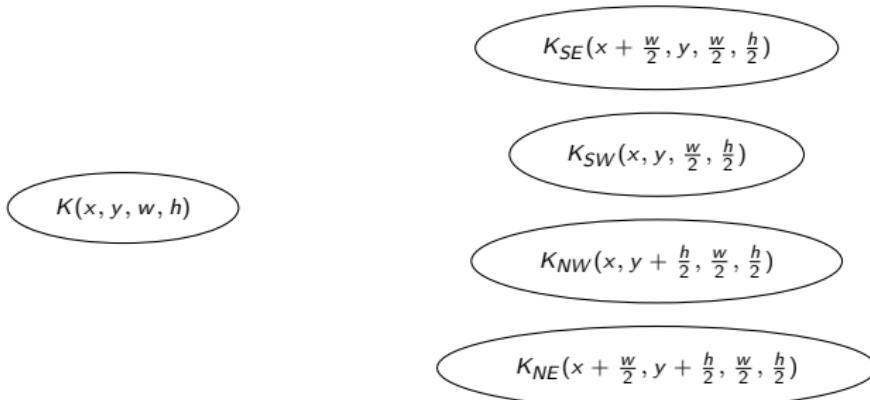
Algorithmen - Subdivision

$$K(x, y, w, h)$$

- ① Vaterknoten K auswählen

Quaternärbaume

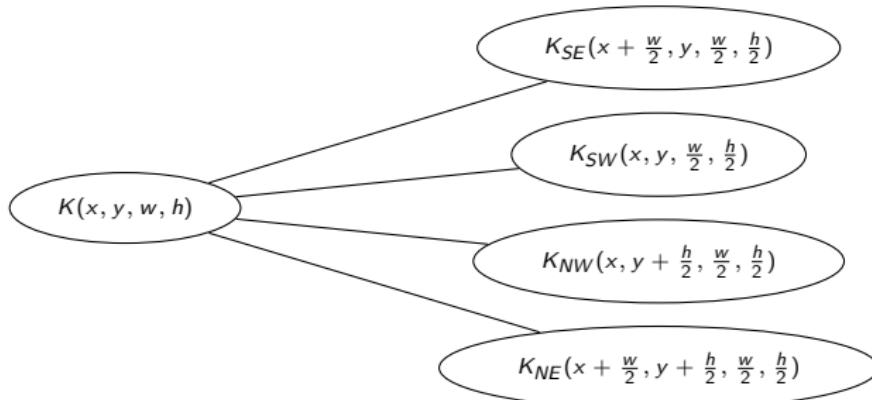
Algorithmen - Subdivision



- ① Vaterknoten K auswählen
- ② Vier Knoten $K_{NE}, K_{NW}, K_{SE}, K_{SW}$ erzeugen

Quaternärbaume

Algorithmen - Subdivision



- ① Vaterknoten K auswählen
- ② Vier Knoten $K_{NE}, K_{NW}, K_{SE}, K_{SW}$ erzeugen
- ③ $K_{NE}, K_{NW}, K_{SE}, K_{SW}$ als Kindknoten von K hinzufügen

Quaternärbaume

Algorithmen - Optimal Fit

- Optimal Fit: Suche den (räumlich) kleinsten Knoten, welches ein gegebenes Quadrat $Q = (x, y, w, h)$ umschließt
- Input: Wurzelknoten, Q
- Output: Knoten K
- Nachbedingung: $\text{contains}(K, Q) = T$

Def. *contains*

$\text{contains} : \mathbb{R}^4 \times \mathbb{R}^4 \rightarrow \mathbb{B}$

$$(x_0, y_0, w_0, h_0), (x_1, y_1, w_1, h_1) \mapsto (x_1 \geq x_0) \wedge (y_1 \geq y_0) \wedge (x_1 + w_1 \leq x_0 + w_0) \wedge (y_1 + h_1 \leq y_0 + h_0)$$

Quaternärbaume

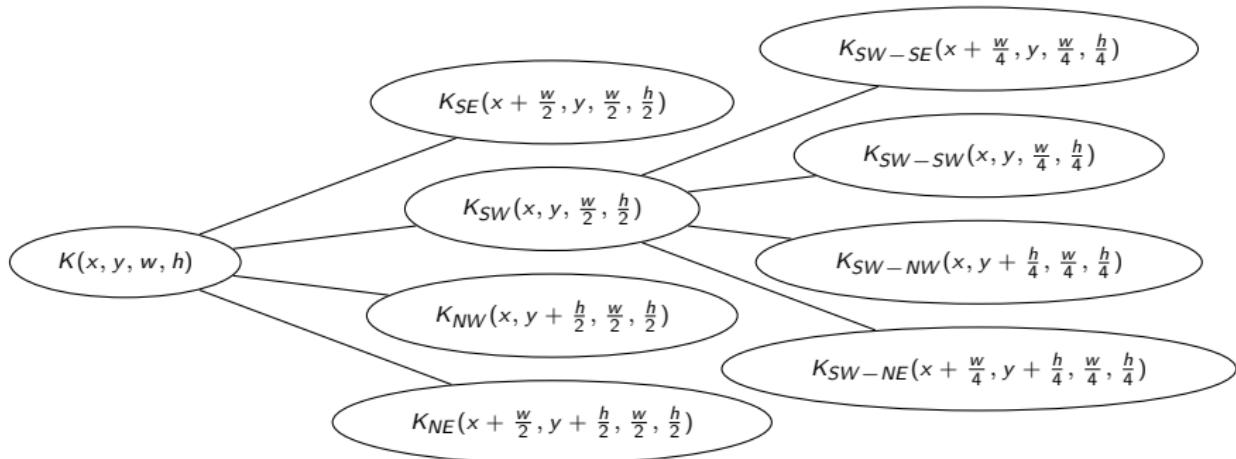
Algorithmen - Optimal Fit

Optimal-Fit Pseudocode

```
def optimalfit(K, Q):
    if contains(K, Q):
        if leaf(K):
            return K
        else:
            for child in (K.ne, K.nw, K.sw, K.se):
                if contains(child, Q):
                    return optimalfit(child, Q)
            return K
    else:
        return None
```

Quaternärbaume

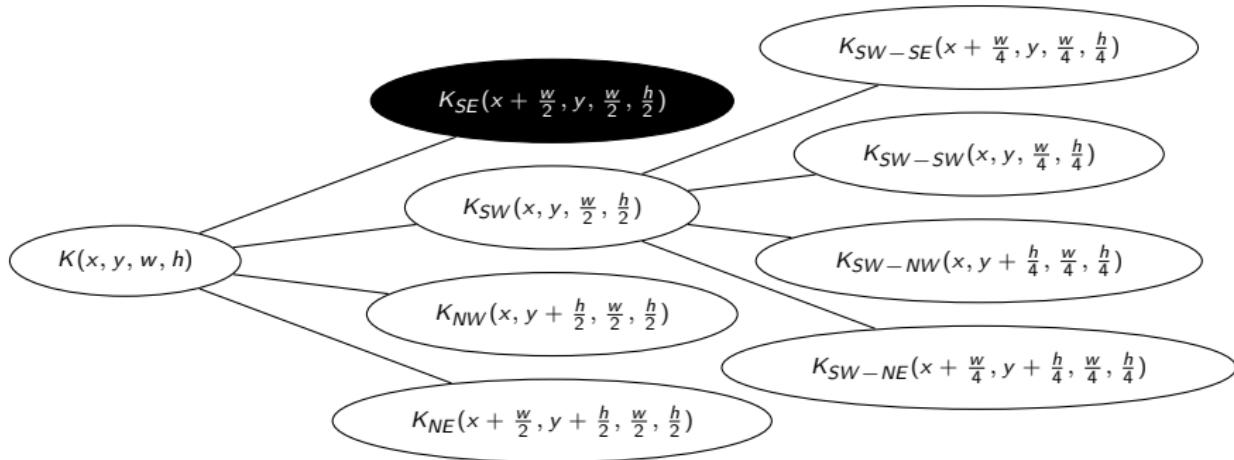
Algorithmen - Optimal Fit



- Gesucht: $Q = (x, y, \frac{w}{8}, \frac{h}{8})$

Quaternärbaume

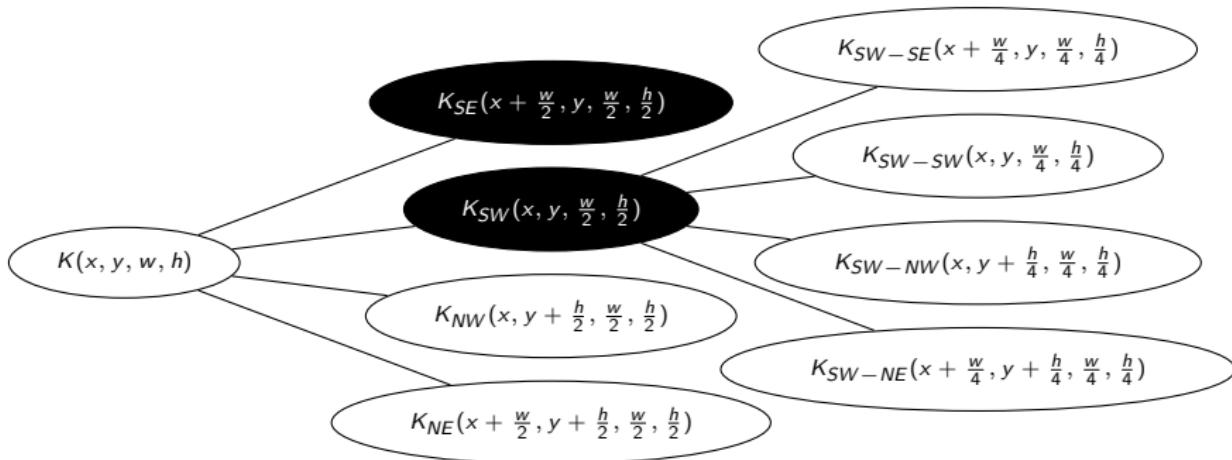
Algorithmen - Optimal Fit



- Gesucht: $Q = (x, y, \frac{w}{8}, \frac{h}{8})$
- $\text{contains}(K_{SE}, Q) = F$

Quaternärbaume

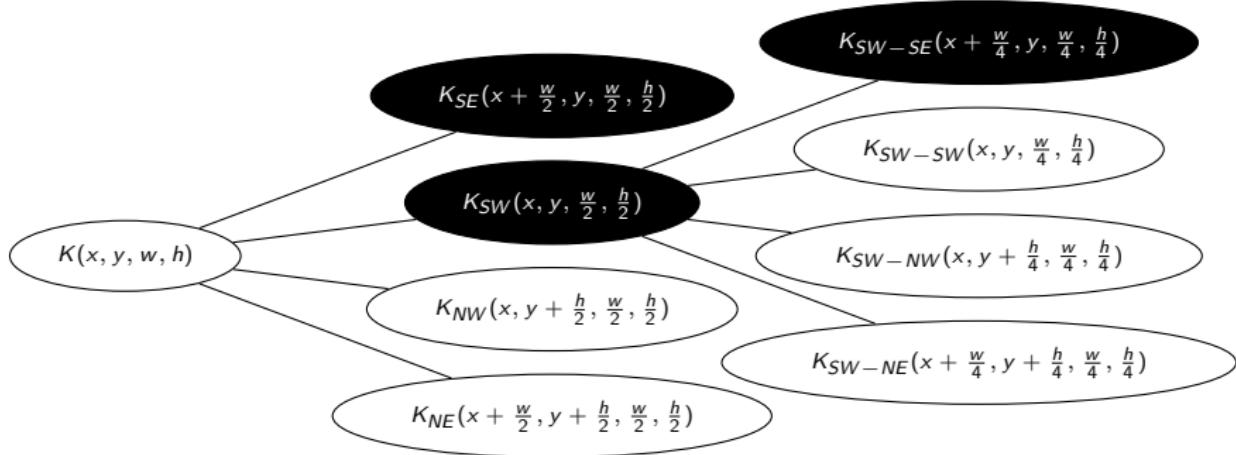
Algorithmen - Optimal Fit



- Gesucht: $Q = (x, y, \frac{w}{8}, \frac{h}{8})$
 - $\text{contains}(K_{SW}, Q) = T$
 - Wende den Optimal-Fit-Algorithmus an mit (K_{SW}, Q)

Quaternärbaume

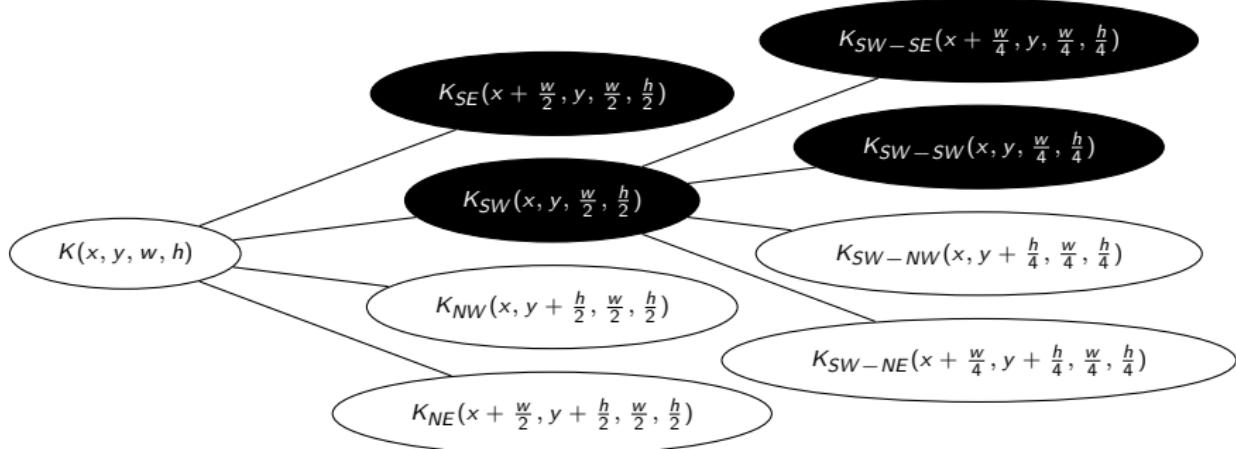
Algorithmen - Optimal Fit



- Gesucht: $Q = (x, y, \frac{w}{8}, \frac{h}{8})$
- $\text{contains}(K_{SW-SE}, Q) = F$

Quaternärbaume

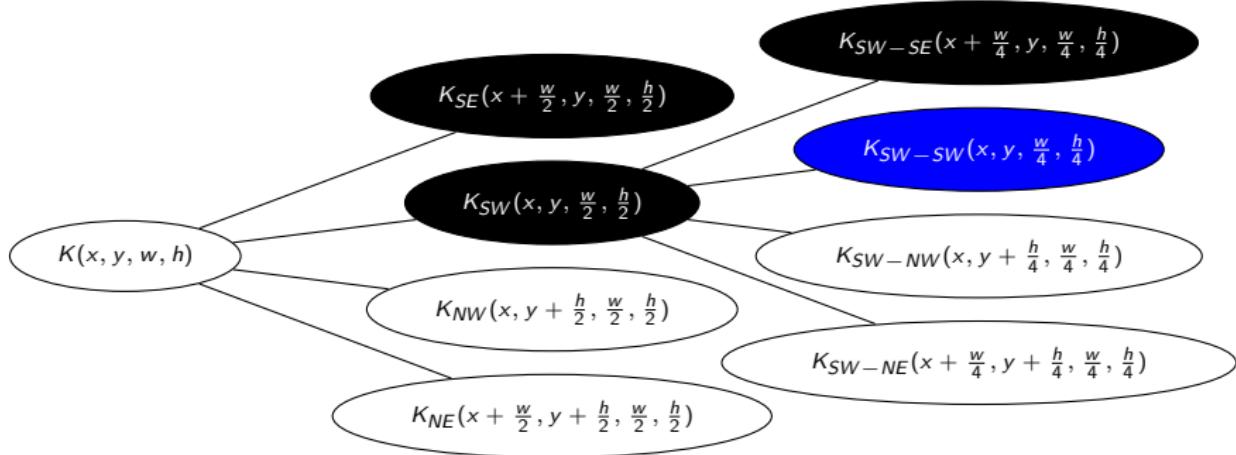
Algorithmen - Optimal Fit



- Gesucht: $Q = (x, y, \frac{w}{8}, \frac{h}{8})$
- $\text{contains}(K_{SW-SW}, Q) = T$
 - Wende den Optimal-Fit-Algorithmus an mit (K_{SW-SW}, Q)

Quaternärbaume

Algorithmen - Optimal Fit



- Gesucht: $Q = (x, y, \frac{w}{8}, \frac{h}{8})$
- K_{SW-SW} ist ein Blatt \rightarrow abbruch mit Ergebnis K_{SW-SW}

Quaternärbaume

Algorithmen - Insert

- Der Einfüge-Algorithmus Setzt sich aus aus *optimalfit* + *subdivide*
- Input: Knoten K , Quadratförmiges Element E , Schwellwert N
- Ablauf von $insert(K, E)$
 - Rufe $optimalfit(K, E)$ auf, speichere Ergebnis in O
 - Füge E in die Elementliste von O ein
 - Wenn die Elementliste von O mindestens N elemente enthält
 - Wende *subdivide* auf O an
 - Entferne alle Elemente aus Elementliste von O und füge alle in O neu ein

Quaternärbaume

Algorithmen - Demonstration

Demonstration

<http://github.com/kochab/tclquadtree>

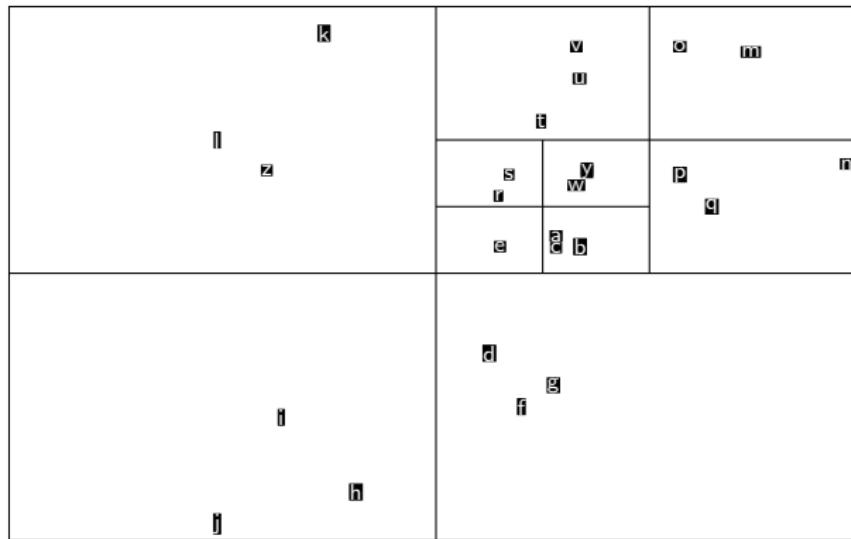
Quaternärbaume

Kollisionserkennung

- Verwendung von Quadtrees zur Kollisionserkennung
- Input: Knoten K , Quadratförmiges Element E
- Verfahren
 - Rufe $optimalfit(K, E)$ auf, speichere Ergebnis in O
 - Für alle Elemente X die in den Knoten des Pfades $O - ROOT$ enthalten sind
 - Prüfe auf Überschneidung (Kollision) mit $intersect(X, E)$

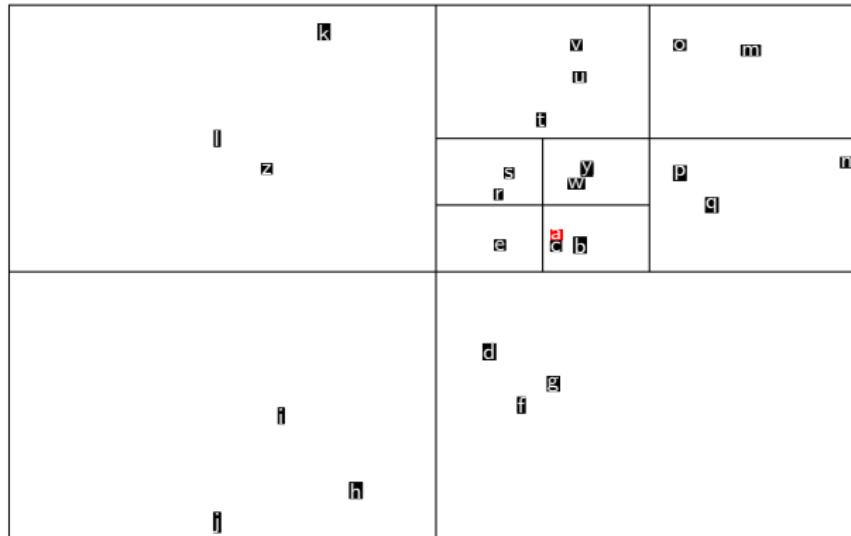
Quaternärbaume

Kollisionserkennung - Beispiel



Quaternärbaume

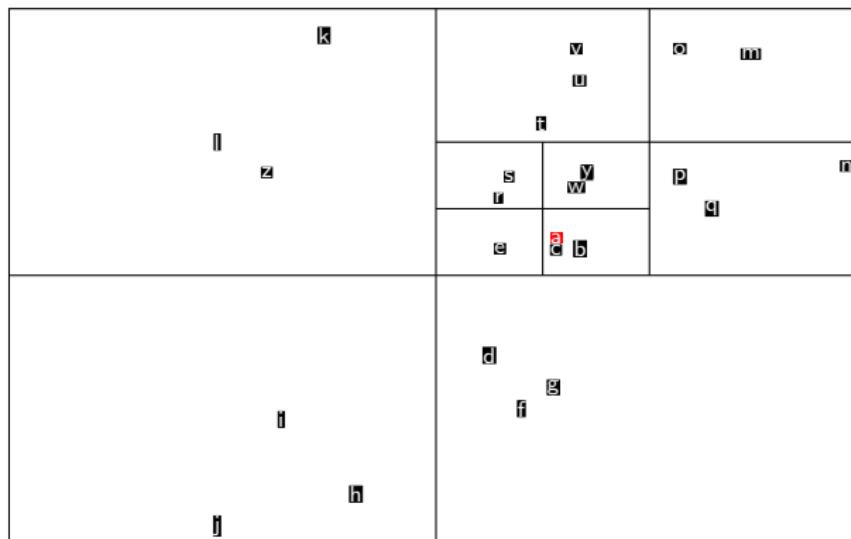
Kollisionserkennung - Beispiel



Kollisionskandidatensuche für a (Rot)

Quaternärbaume

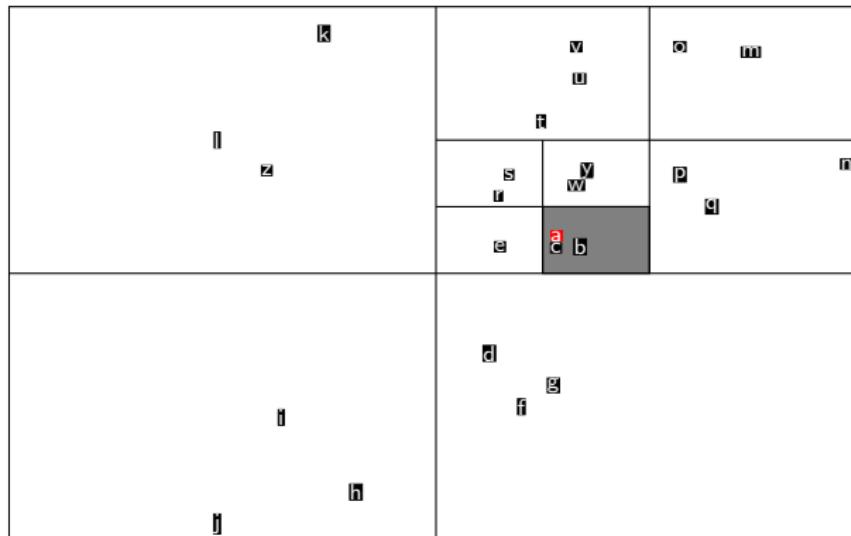
Kollisionserkennung - Beispiel



Kollisionskandidatensuche für a (Rot)
 $optimalfit(K, a)$

Quaternärbaume

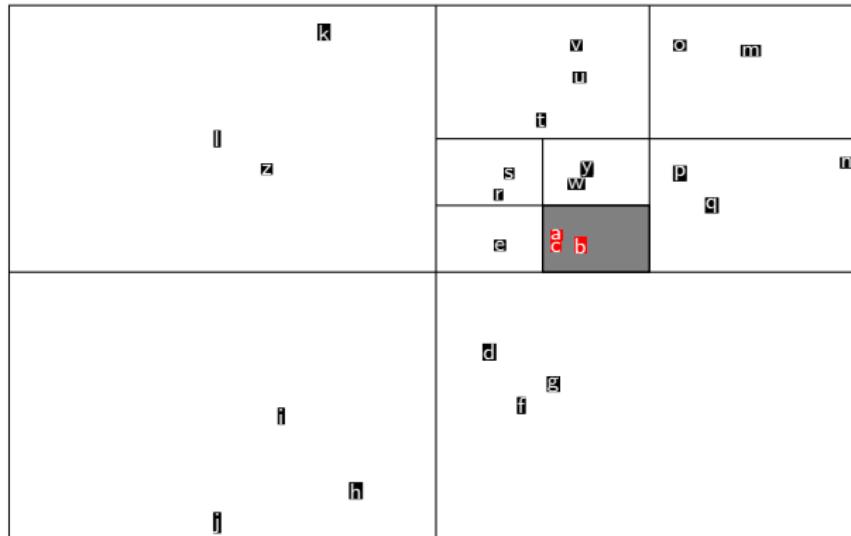
Kollisionserkennung - Beispiel



Kollisionskandidatensuche für a (Rot)
 $optimalfit(K, a) = K_{NE-SW-SE}$ (Grau)

Quaternärbaume

Kollisionserkennung - Beispiel

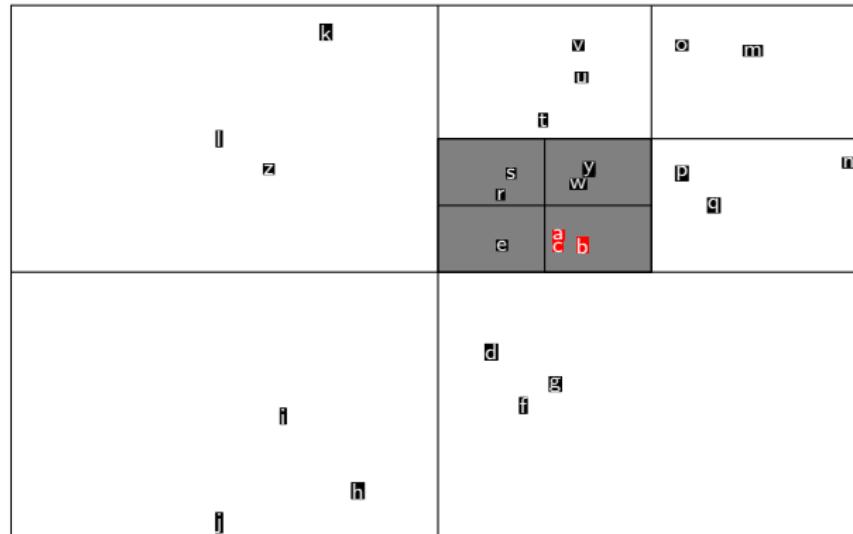


Kollisionskandidatensuche für a (Rot)

Objektlistenelemente von $K_{NE-SW-SE}$: $\{c, b\}$ als Kollisionskandidaten selektieren

Quaternärbaume

Kollisionserkennung - Beispiel

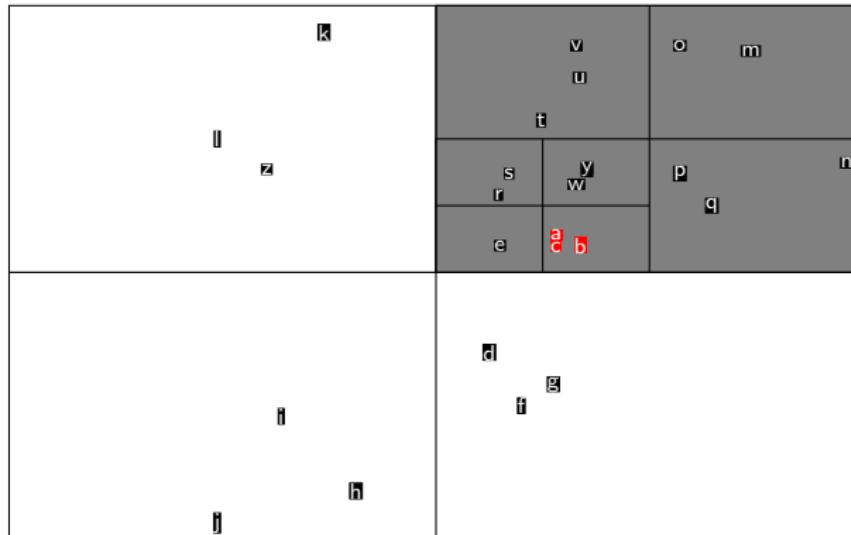


Kollisionskandidatensuche für a (Rot)

Objektlistenelemente von K_{NE-SW} : \emptyset als Kollisionskandidaten selektieren

Quaternärbaume

Kollisionserkennung - Beispiel

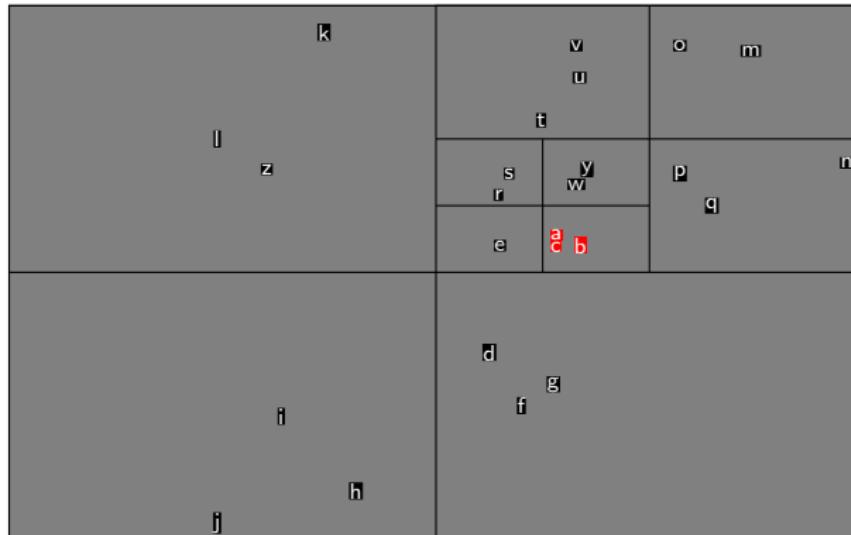


Kollisionskandidatensuche für a (Rot)

Objektlistenelemente von K_{NE} : \emptyset als Kollisionskandidaten selektieren

Quaternärbaume

Kollisionserkennung - Beispiel

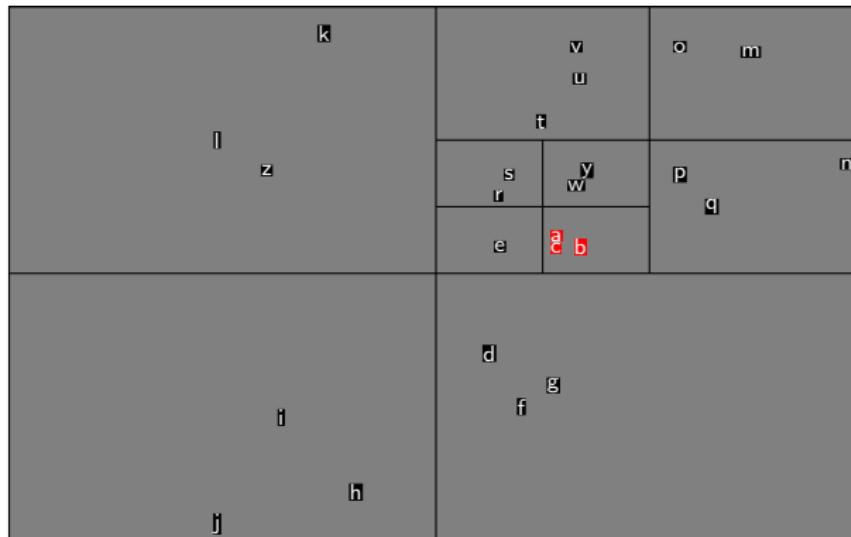


Kollisionskandidatensuche für *a* (Rot)

Objektlistenelemente von *K*: \emptyset als Kollisionskandidaten selektieren

Quaternärbaume

Kollisionserkennung - Beispiel

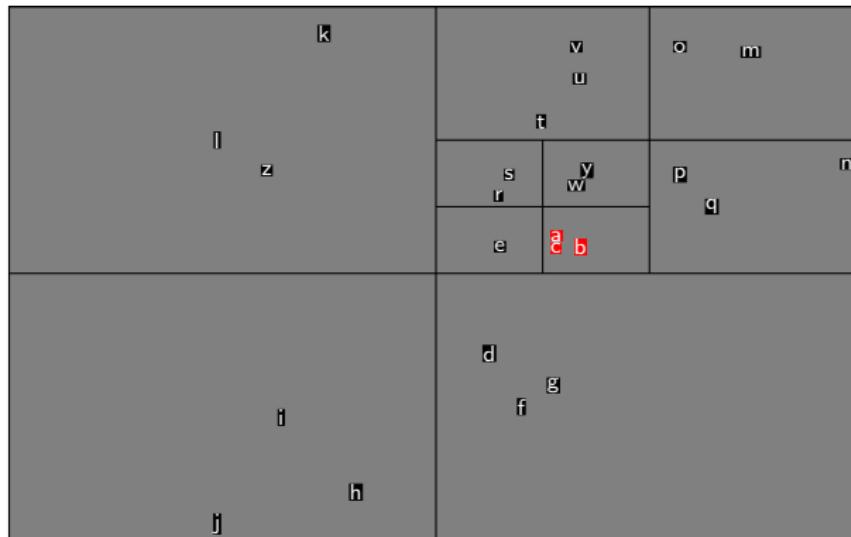


Kollisionskandidatensuche für a (Rot)

Ergebnis: a hat die Kollisionskandidaten $\{c, b\}$

Quaternärbaume

Kollisionserkennung - Beispiel



Kollisionskandidatensuche für a (Rot)

Zu evaluieren ist nur $\text{intersect}(a, c) \vee \text{intersect}(a, b)$

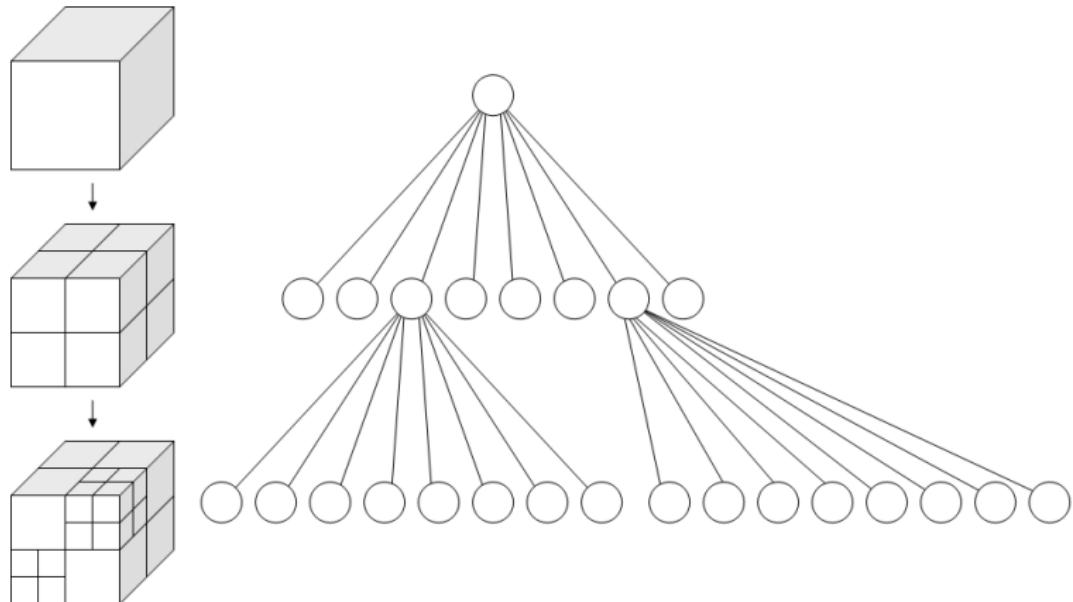
Octrees

Struktur

- Quadtrees sind für Flächen (2D)
- Octrees sind für Räume (3D)
- Konzeptuell sehr ähnlich

Octrees

Struktur



Octree - Nü, Wikimedia (License: GFDL)

Octrees

Struktur

- Subdivision in 8 Subquader
- Quader beschrieben durch $(x, y, z, w, h, d) \in \mathbb{R}^6$
 - d : depth
- $contains_{3D} : \mathbb{R}^6 \times \mathbb{R}^6 \rightarrow \mathbb{B}$
- $intersect_{3D} : \mathbb{R}^6 \times \mathbb{R}^6 \rightarrow \mathbb{B}$

Quad-/Octrees

Komplexität

- subdivide: ?

Quad-/Octrees

Komplexität

- subdivide: $O(1)$
- optimalfit: ?

Quad-/Octrees

Komplexität

- subdivide: $O(1)$
- optimalfit: $O(\log_4 n)$
- insert: ?

Quad-/Octrees

Komplexität

- subdivide: $O(1)$
- optimalfit: $O(\log_4 n)$
- insert: $O(\log_4 n)$

Quad-/Octrees

Komplexität

- subdivide: $O(1)$
- optimalfit: $O(\log_4 n)$
- insert: $O(\log_4 n)$
- collide: ?

Quad-/Octrees

Komplexität

- subdivide: $O(1)$
- optimalfit: $O(\log_4 n)$
- insert: $O(\log_4 n)$
- collide:
 - Worst case: $O(n^2)$
 - Empirisch: $O(\log_4 n)$

Quad-/Octrees

Videobeispiel



YouTube: Asteroids like Bullet Hell (Quadtrees finished) - Zimhey

Quad-/Octrees

Andere Anwendungen

- Punktemenge (Alt. zu Hashverfahren wie z.B. Morton-Code)
- Frustum Culling
- Sparse data storage (Spreadsheets/Matrizen)
- Maximum disjoint sets

Quad-/Octrees

Schlussfolie

Vielen Dank für die Aufmerksamkeit!

Fragen?