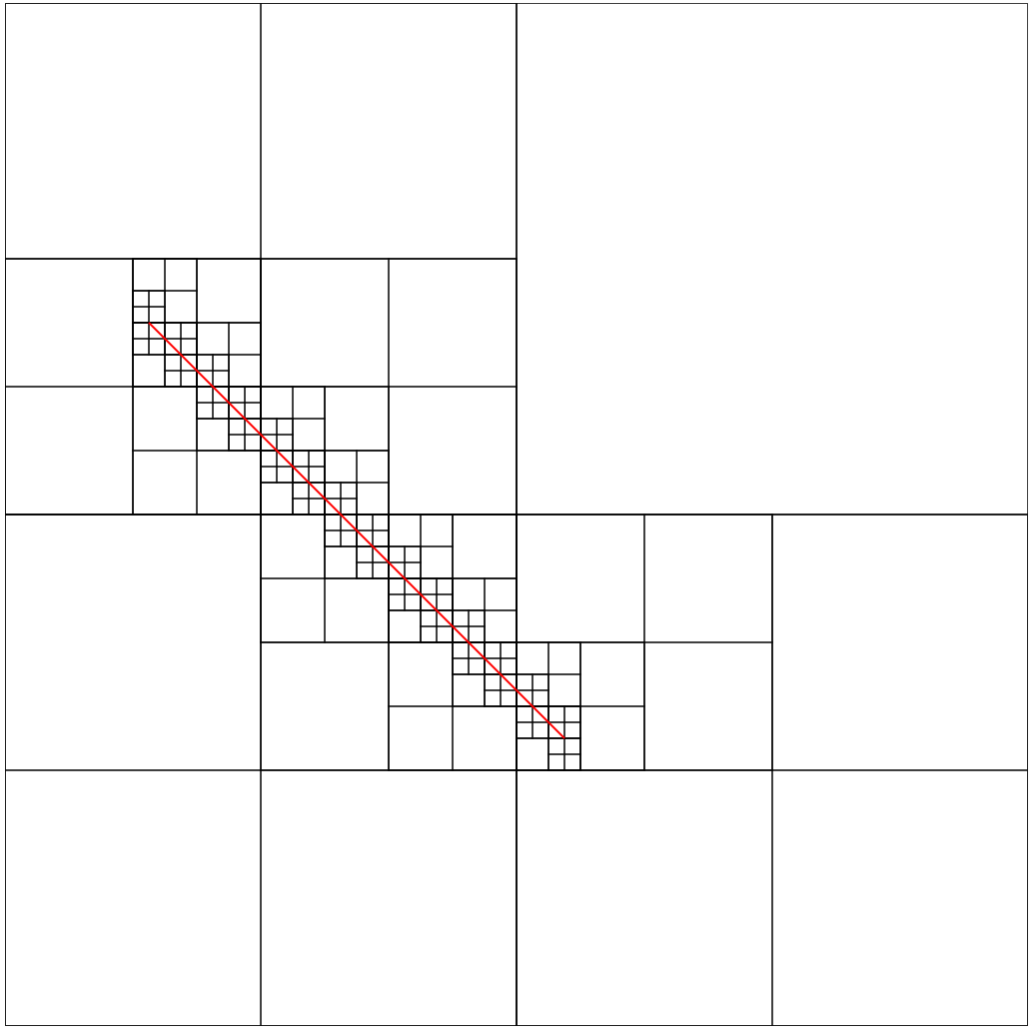
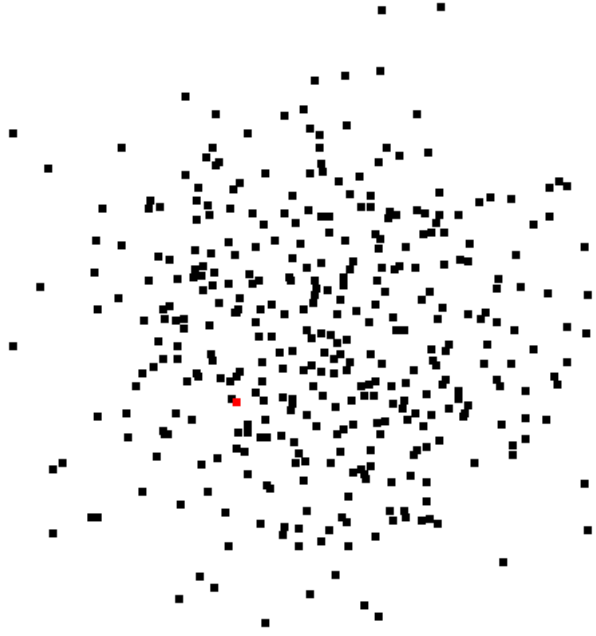
Quadtrees

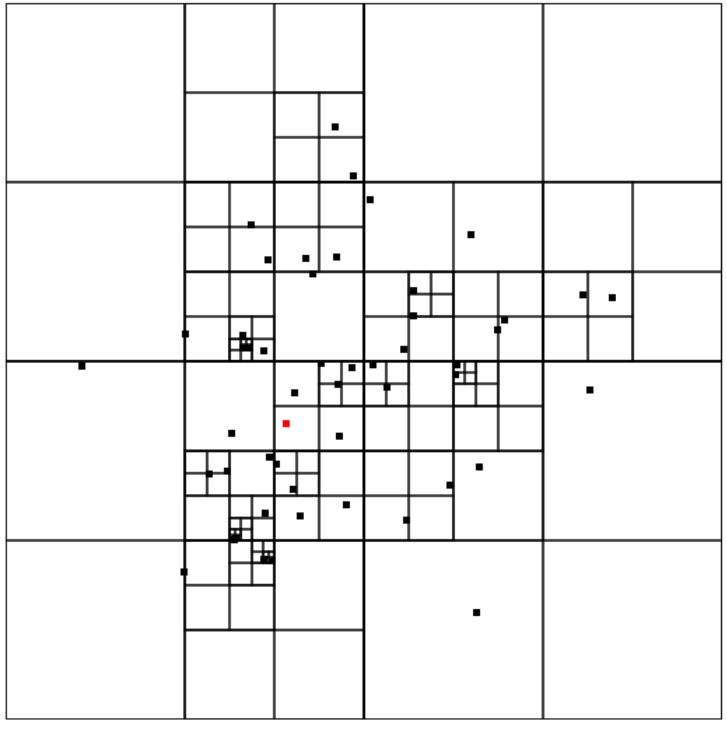


Motivation

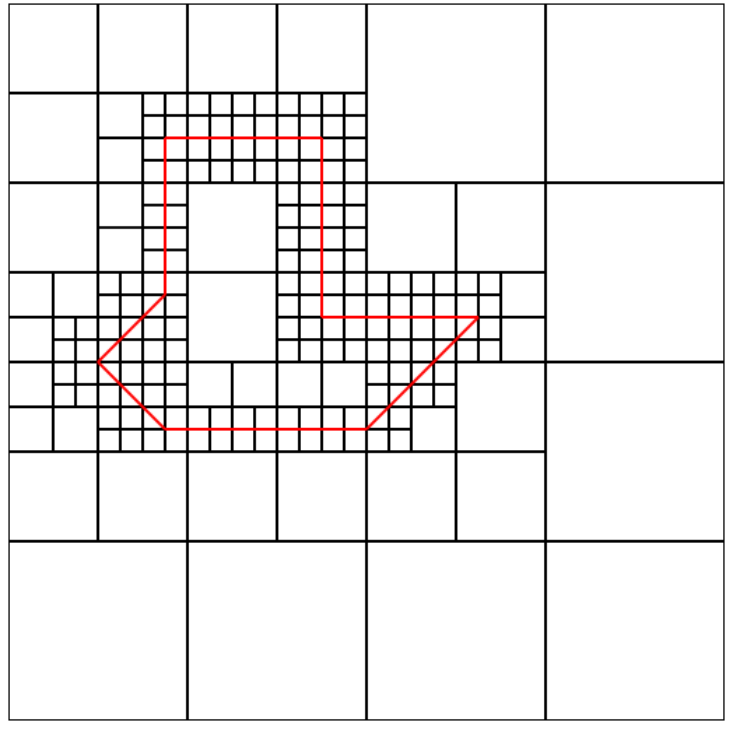


Wie findet man *schnell* die Nachbarn zum roten Punkt

Übersicht

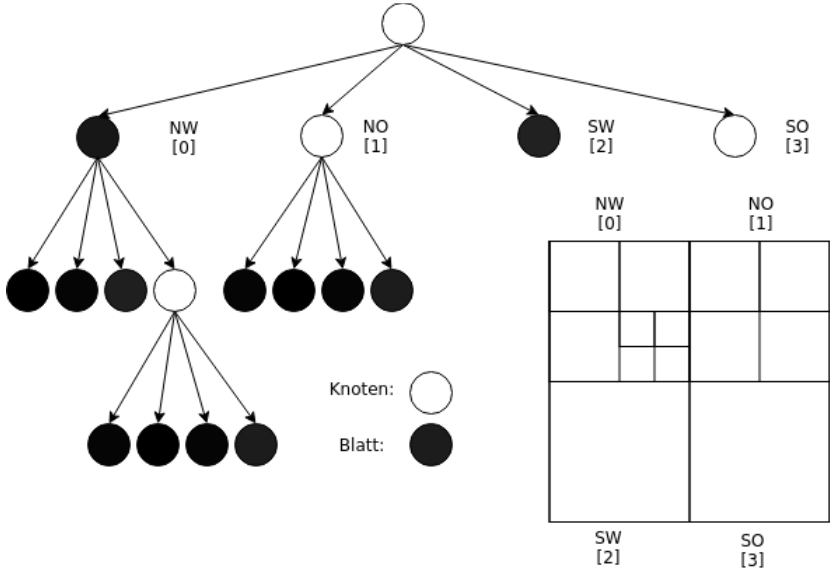


Punktverwaltung

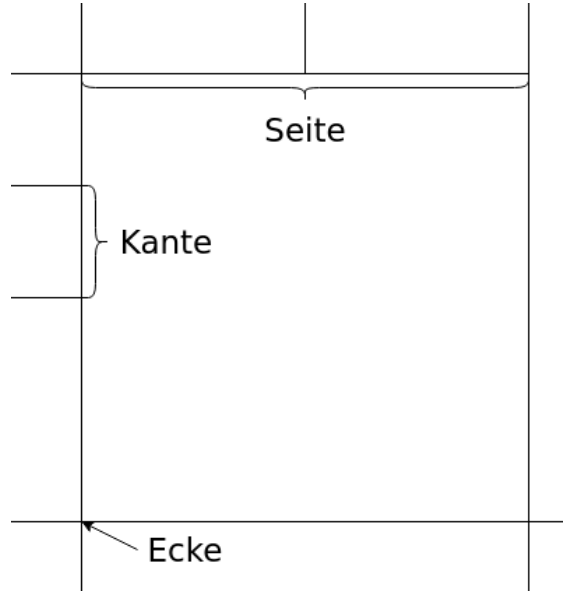


Geometrie

Aufbau eines Quadtree



Rekursive Baumstruktur



Begriffs-Konventionen

Punkte einfügen

Wurzelknoten σ_{root}, allgemein: $\sigma := [x_\sigma : x'_\sigma] \times [x_\sigma : x'_\sigma]$

$\sigma_{NW}, \sigma_{NO}, \sigma_{SW}, \sigma_{NO}$ sind Kinder von σ

Punkte $p \in P$ werden in den Kindern von σ gespiechert:

$x_{mid} := \frac{x_\sigma + x'_\sigma}{2} \quad y_{mid} := \frac{y_\sigma + y'_\sigma}{2}$

$$P_{NW} := \{p \mid p_x \leqslant x_{mid} \,\&\, p_y \leqslant y_{mid} \,\&\, p \in P\}$$
$$P_{NO} := \{p \mid p_x > x_{mid} \,\&\, p_y \leqslant y_{mid} \,\&\, p \in P\}$$
$$P_{SW} := \{p \mid p_x \leqslant x_{mid} \,\&\, p_y > y_{mid} \,\&\, p \in P\}$$
$$P_{SO} := \{p \mid p_x > x_{mid} \,\&\, p_y > y_{mid} \,\&\, p \in P\}$$

Punkte einfügen

Algorithmus

```
function insertPoint (Tree t, Point p) {
  if(t.size <= UNITSIZE && t.point != null) return false

  if(t.childs == null && t.point == null) {
    t.point = p
  } else if(t.childs == null && t.point != null) {
    t.createChilds()
    t.insertPointToChilds(p)
    t.insertPointToChilds(t.point)
  } else {
    t.insertPointToChilds(p)
  }
  return true
}
```

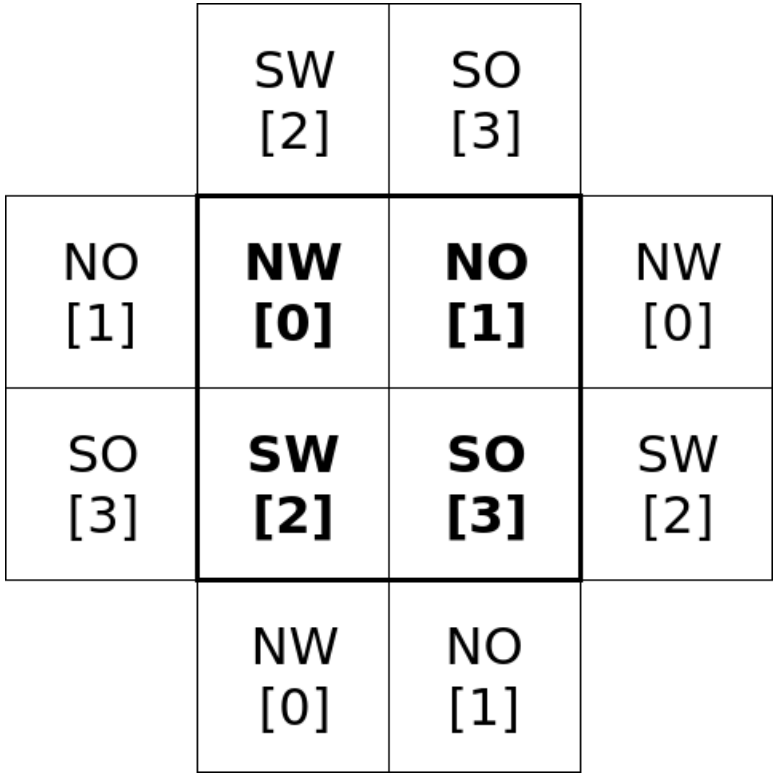
Punkte einfügen

Laufzeit

Satz

Eine Quadtree der Tiefe d , welcher n Punkte speichert, kann in der Zeit $O((d+1)n)$ erzeugt werden und hat $O((d+1)n)$ Knoten.

Nachbarn Finden



Nachbarn Finden

Richtung und Position zu Enum übersetzen:

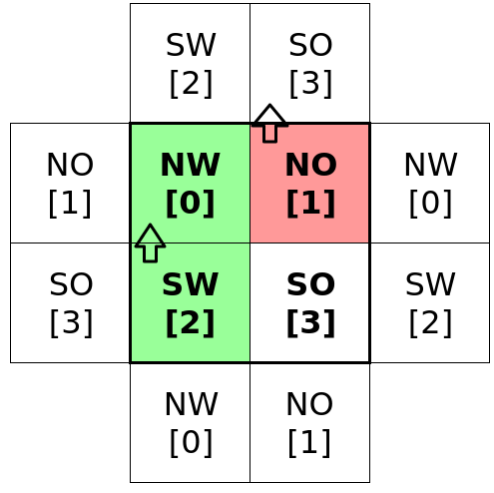
```
Enum Pos = {NW=0, NO=1, SW=2, SO=3}
Enum Direction = {NORTH=0, EAST/OST=1, SOUTH=2, WEST=3}

//getInsideNeighbour
function gINeighbour (Direction d, Position t) {
    return <correctDirection>
}
```

gINeighbour(NORTH, Pos.SW)

return Pos.NW

gINeighbour(NORTH, Pos.N0)

return -1 (no Neighbour found)



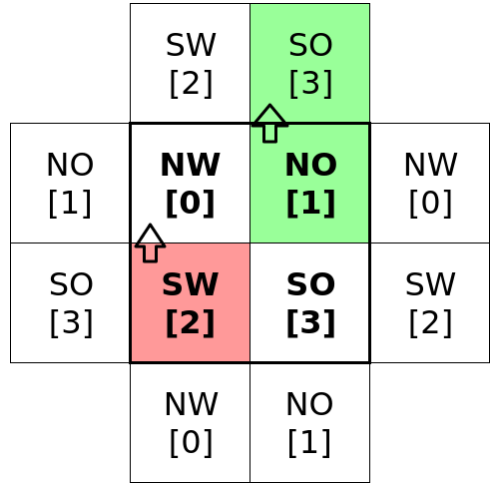
Nachbarn Finden

Richtung und Position zu Enum übersetzen:

```
Enum Pos = {NW=0, NO=1, SW=2, SO=3}
Enum Direction = {NORTH=0, EAST/OST=1, SOUTH=2, WEST=3}

//getOutsideNeighbour
function gONeighbour (Direction d, Position t) {
        return <correctDirection>
}
```

gONeighbour(NORTH, Pos.SW)
return -1 //no Neighbour found

gONeighbour(NORTH, Pos.N0)
return Pos.S0



Nachbarn Finden

Algorithmus

```
Enum Direction = {N = 0, O = 1, S = 2, W = 3}
Enum Pos = {NW = 0, NO = 1, SW = 2, SO = 3}

function findNeighbour(Tree t, Direction d) {
  if(t.parent == null) return null

  else if(gINeigbour(d, t.position) != -1) {
    return t.parent.childs[gINeigbour(d, t.position)
  } else {
    out = t.parent.getNeighbour(d)
    if(out == null || out.childs == null) {
      return out
    } else {
      return out.childs[gONeighbour(d, t.position)
} } }
```

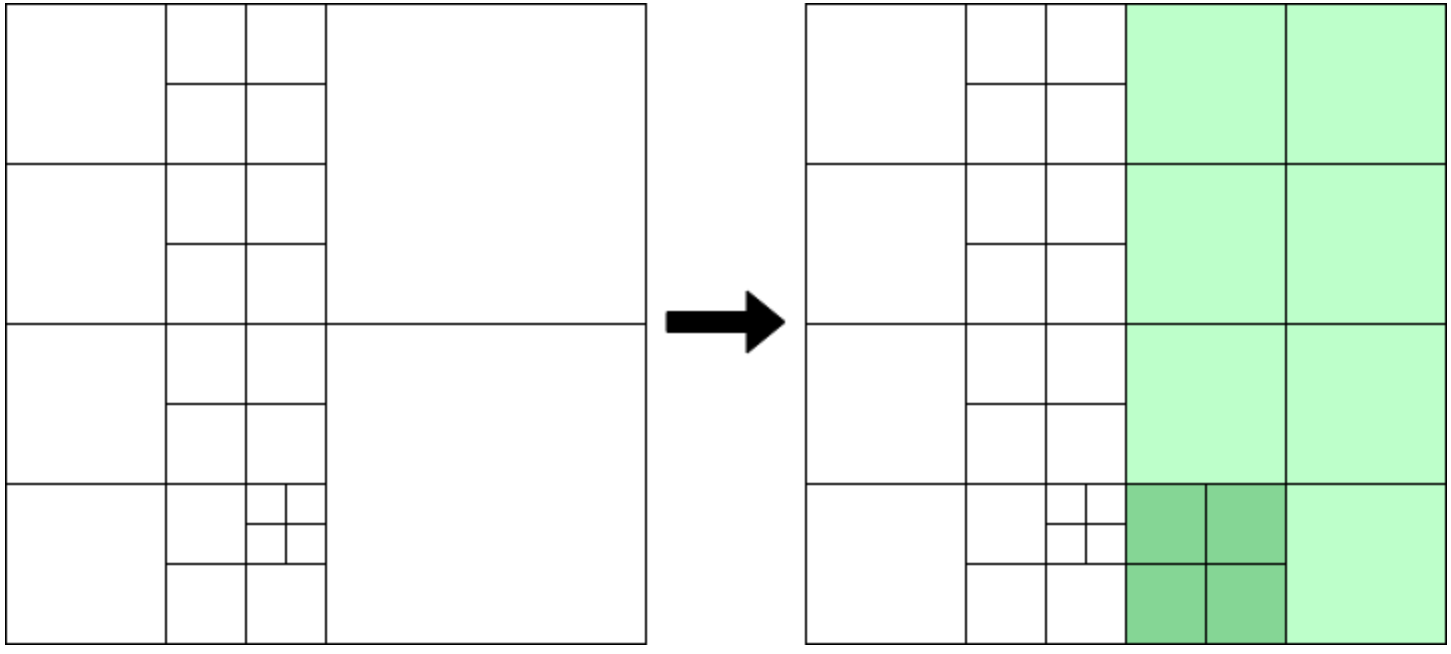
Nachbarn Finden

Laufzeit

Satz

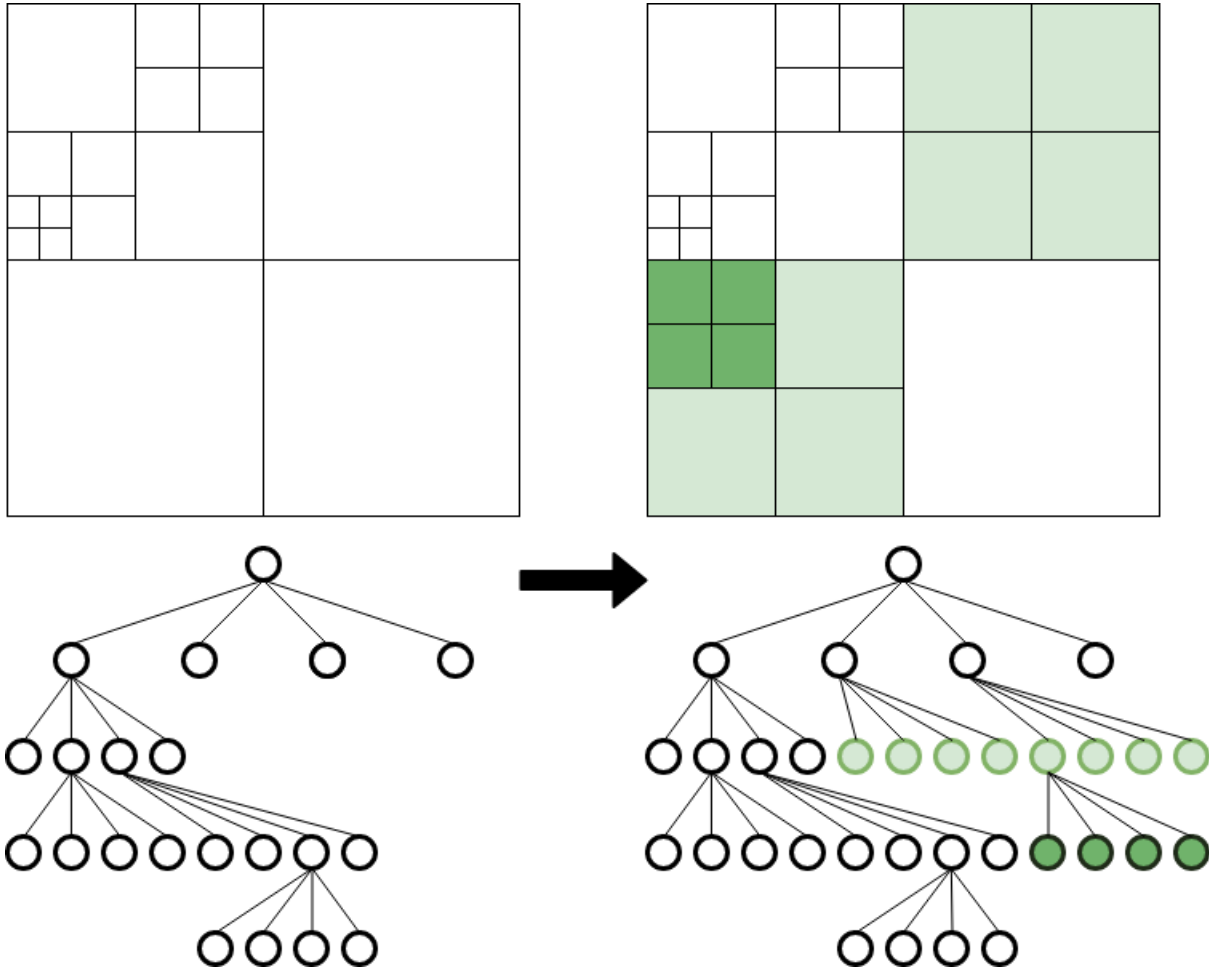
Sei T ein Quadtree der Tiefe d, so kann für einen Knoten v der Nachbar in gegebener Richtung in $O(d+1)$ gefunden werden.

Balancieren eines Quadtrees



Grafische Darstellung

Balancieren eines Quadtrees



Baumdarstellung

Balancieren eines Quadtrees

Algorithmus

```
function balance(QTree root) {
  List l = root.getLeavesRecursive();
  for (Qtree t in l){
    for(direction = 0; direction < 4, direction++) {
      neighbour = t.getNeighbour(direction)
      if(neighbour != null && neighbour.childs != null){
        if(smallChildsAdjacent(neighbour,direction)) {
          t.addChilds()
          l.append(t.getChilds())
          l.append(getNeighboursWithoutChilds(t))

          if(t.point != null)
            t.insertPointToChilds(t.point)
          break
} } } } }
```

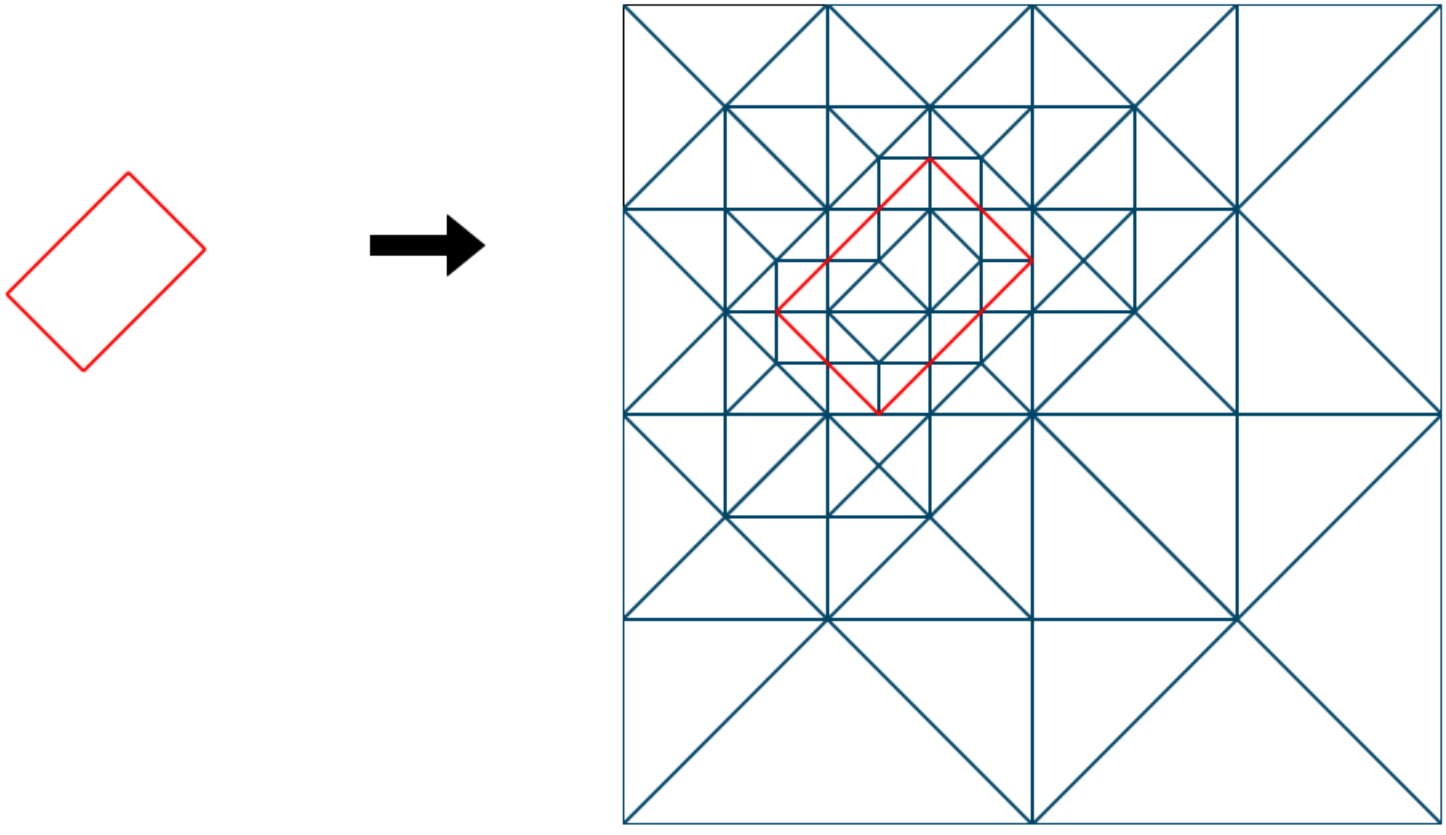
Balancieren eines Quadtrees

Laufzeit

Satz

Sei T ein Quadtree mit m Knoten, dann hat der balancierte Quadtree T' $O(m)$ Knoten und kann in $O((d+1)m)$ aus T konstruiert werden.

Netzkonstruktion mit Quadtrees



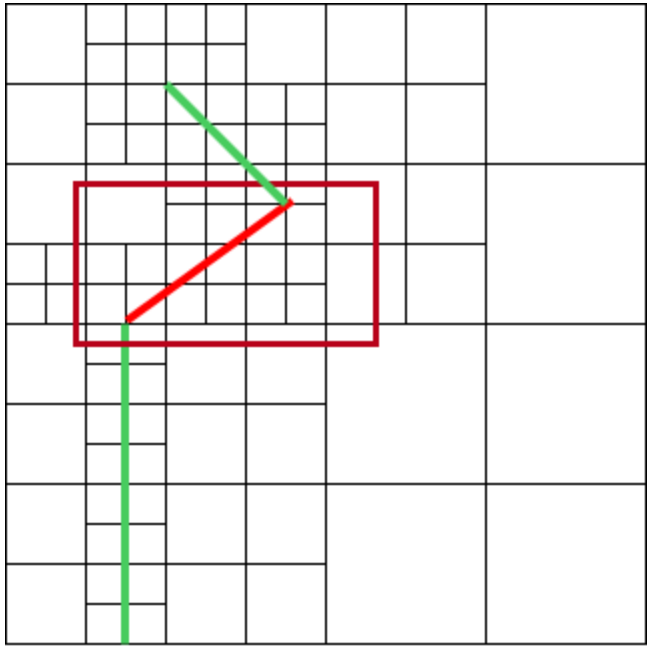
Netzkonstruktion mit Quadtrees

Bedingungen und Vorgaben:

- Polygon Ecken haben nur die Winkel 0°, 45°, 90° und 135°.
- Es wird, wenn möglich, ein nicht-uniformes Netz erzeugt
- Ein Quadtree, der eine Kante eines Polygons enthält, wird aufgeteilt, bis das Kind mit der Polygonkante Minimalgröße hat.
- Das gewünschte Netz muss aus einem Balancierten Graphen erzeugt werden.

Netzkonstruktion mit Quadtrees

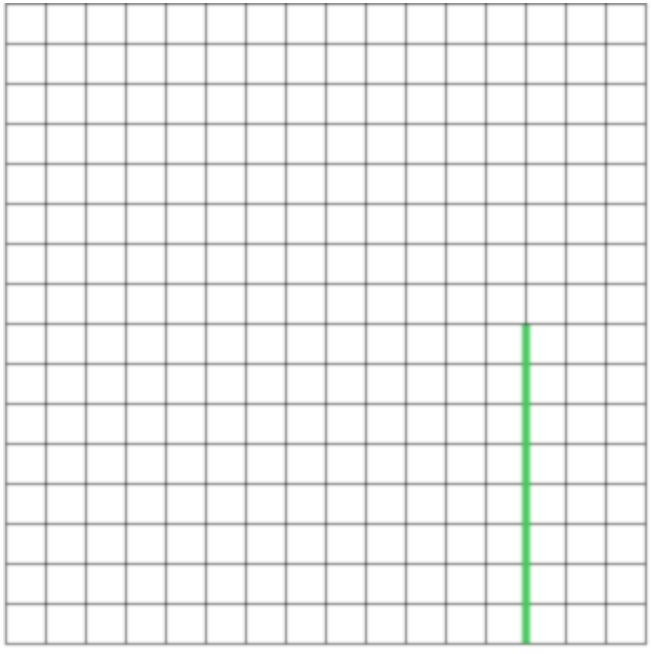
Polygone haben nur bestimmte Winkel

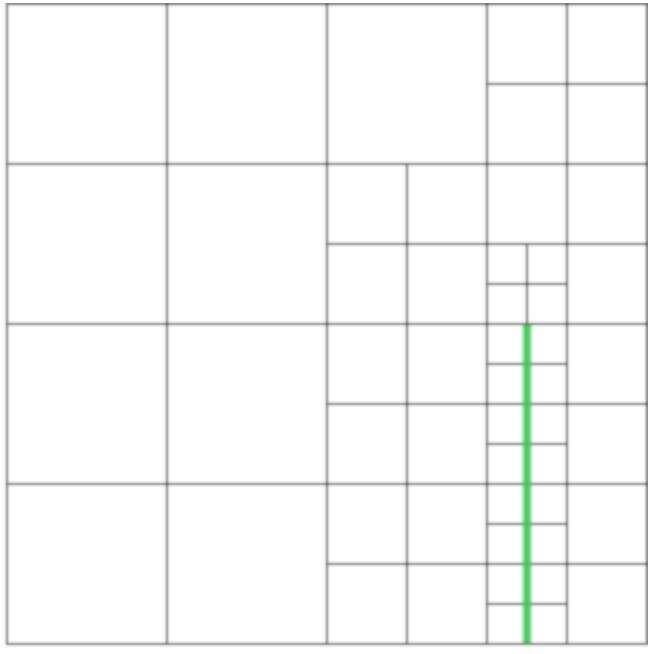


Polygonkante schneidet Quadrate an Seiten

Netzkonstruktion mit Quadtrees

Es wird ein nicht-uniformes Netz erzeugt

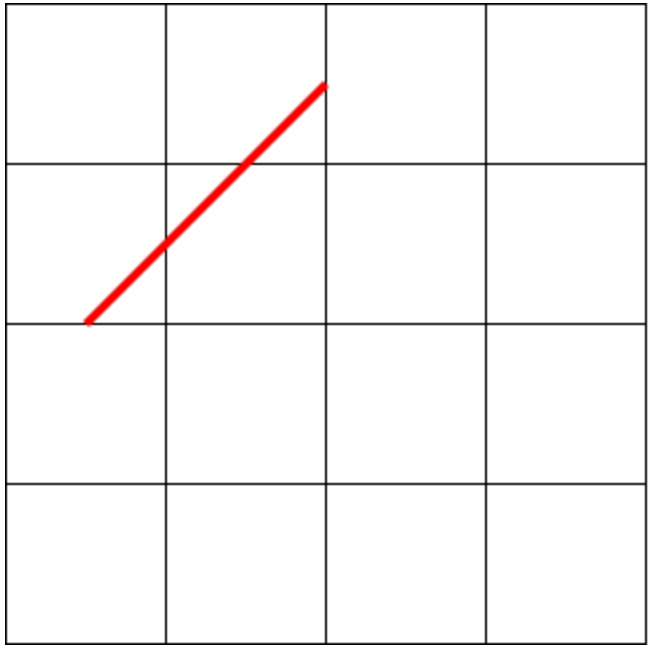


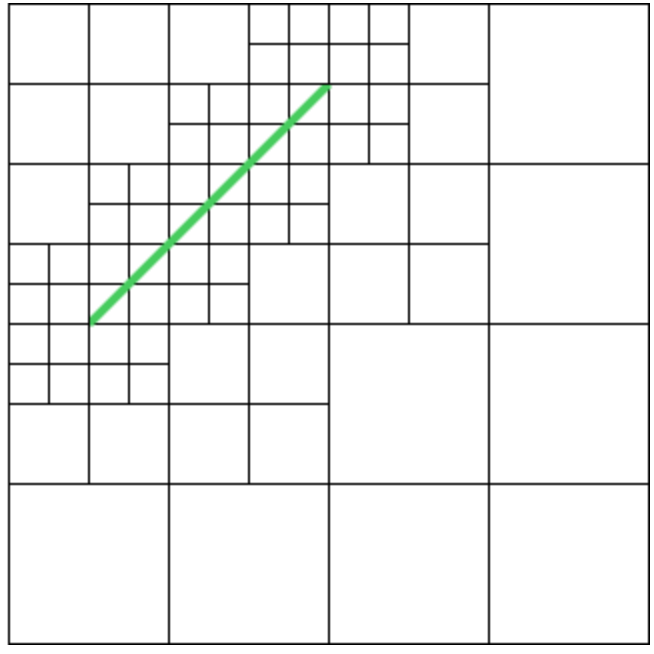


Nicht-uniformes Netz unnötig aufwändig

Netzkonstruktion mit Quadtrees

Es wird bis auf Minimalgröße aufgelöst

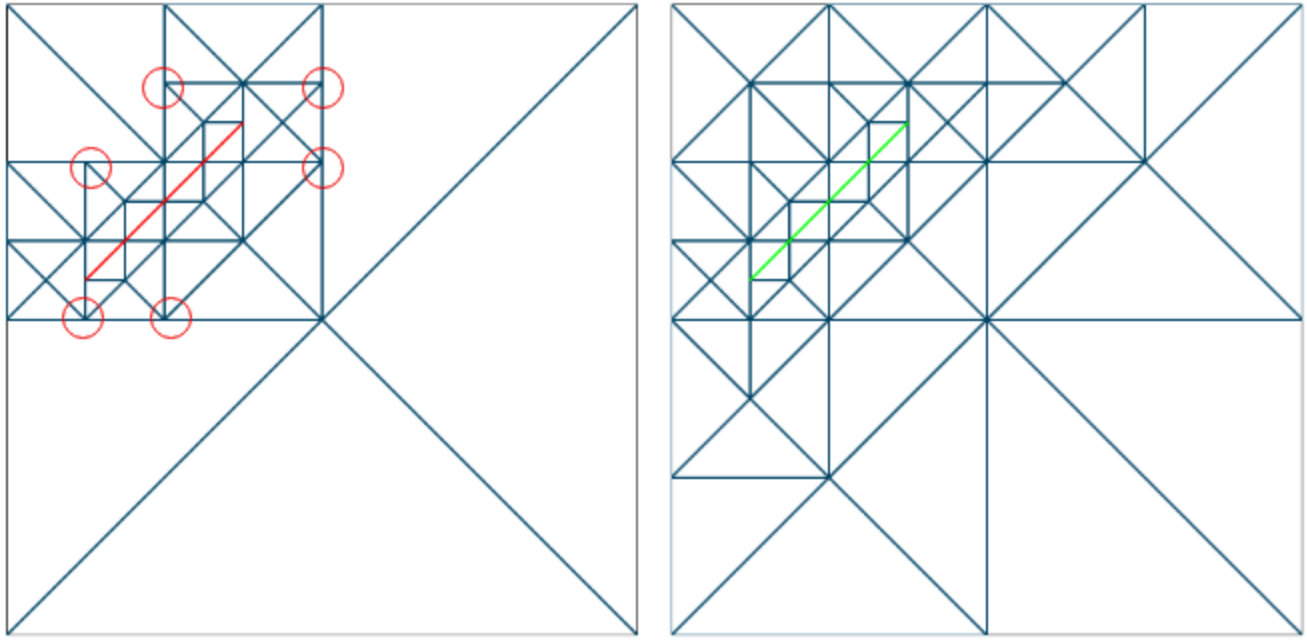




Polygonkante schneidet Quadrate an Seiten

Netzkonstruktion mit Quadtrees

Erzeugung aus balanciertem Graph



Unbalancierte Netzberechnung erzeugt unvollständiges Netz

Netzkonstruktion mit Quadtrees

Algorithmus

```
function insertLine (Qtree t, Line l) {
  if(t.size <= UNITSIZE) {
    if(lineCrossesSquare(t, l) || lineTouchesNorW(t,l))
      t.insertLineSegment(l)
    return
  }
  if(this.childs != null) {
    t.insertLineInChilds(l)
  } else if(lineIntersectsSquare(t,l) {
    t.addChilds()
    t.insertLineInChilds(l)
  }
}
```

Netzkonstruktion mit Quadtrees

Laufzeit

Satz

Sei M ein Menge disjunkter polygonaler Komponenten im Quadrat $[0 : U] \times [0 : U]$, mit den vorher genannten Anforderungen, so lässt sich ein Netz mit $O(p(S) \log U)$ Dreiecken für M erzeugen. Hierbei ist $p(S)$ die Summe der Perimeter der Komponenten von M, und das Netz kann in $O(p(S) \log^2 U)$ erzeugt werden.

Sources

Books

Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars (2008). Computational Geometry (3rd revised ed.). Springer-Verlag. ISBN 3-540-77973-6. 1st edition (1997): ISBN 3-540-61270-X

Algorythms

https://stackoverflow.com/questions/9043805/test-if-two-lines-intersect-javascript-function Line Intersection von Dan Fox

Images

created with https://www.draw.io/ (licence free)