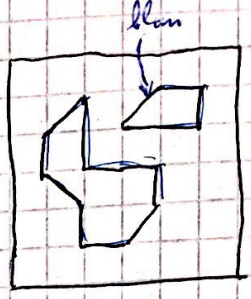
Mesh-Generierung mit Quadtrees:

Vereinfachung: diskrete Koord $[0, U] \in \mathbb{N}$;
ist z.B. für VLSI-Layout ok.

Eingabe: Quadrat $(0,0 - U,U)$ $[0,U]^2$
und Anzahl Polygonzüge.

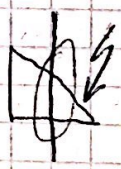


Vereinfachung: alle Ecken sind Integerkoord.,
nur Winkel $\in \{0°, 45°, 90°, 135°\}$.

Ziel: Triangulierung, und zwar so, daß kleine Dreiecke "in der Nähe" der Polygonzüge, und große weiter weg.
Außerdem hätte man gerne ein paar wünschenswerte Eigenschaften:

1. Alle Dreiecke sollen "angepaßt" sein ("conforming mesh") [manchmal auch "konsistent"]
   = keine T-Ecken = keine Ecke eines Dreiecks im Inneren einer Kante eines anderen Dreiecks.

2. Mesh soll Eingabe respektieren ("constrained mesh") nö
   = kein Dreieck liegt auf zwei Seiten einer Eingabekante
   = Kanten der Eingabe $\subseteq$ Vereinigung aller Kanten des Meshes.



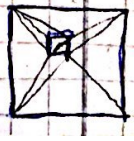
3. Alle Dreiecke wohlgeformt ("well-shaped mesh")
   = keine zu langen-schmalen Dreiecke, keine zu spitzen Winkel
   = hier genauer: alle Winkel zwischen 45° und 90°

4. Mesh soll nicht-uniform sein.

Andere Methode der nicht-uniformen Triangulierung: Delaunay-Triangulierung

Probleme:

1. Man kann nicht einfach Delaunay auf Punktmenge machen;
   sonst: Mesh respektiert Eingabe nicht!
2. Also muß man Constrained Delaunay machen;
   leider: Dreiecke können lang und schmal sein!



Lösung: verwende Quadtrees

Ansatz: Quadtrees für Polygonzüge:

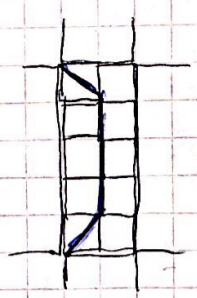
Früher Qu. für Punkte, jetzt Kantenmengen;

Unterschied: Abbruchkriterium;

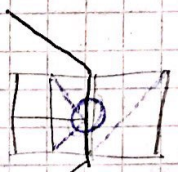
Abbruch früher: wenn < 2 Pkte,

Abbruch jetzt: wenn keine Kante (der Polylines) mehr schneidet oder berührt,

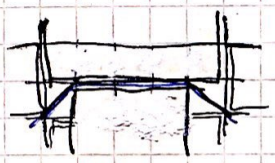
oder wenn Größe = 1×1.

⇒ 

Entlang der Kanten der Polygonzüge haben die Zellen Kantenlänge = 1



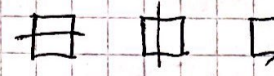
Man könnte Abbr. krit. modifizieren, ohne "oder berührt"

⇒ 

Welches Problem entsteht dann?
Das daraus generierte Mesh ist aber evtl. non-conforming!
(T-Ecken)

Aus Abbruchkriterium ⇒ jedes Blatt des Q-tree wird von Polygonzug entweder...

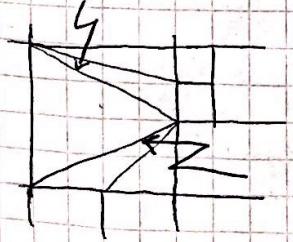
- nicht geschnitten / berührt
- entlang seiner Seite berührt
- diagonal geschnitten

 kann nicht passieren

Jetzt einfach Diagonalen hinzufügen klappt nicht:

 in so einem Fall entsteht T-Ecke!

Quadrate mit allen Vertices triangulieren geht auch nicht:



zu schmale Dreiecke!
(spitze Winkel!)

Lösung: mache Quadtree balanciert.

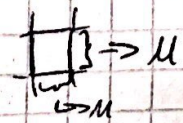
Algo:

Input = Polygonzüge S, mit bekannten Eigenschaften,

Output = Dreiecksmesh M mit bekannten Eigenschaften.

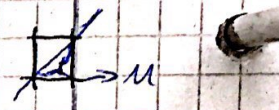
erzeuge Quadtree T von Polygonzüge;

balanciere $T \to Q$

initialisiere M mit Kanten induziert durch Q 

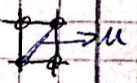
foreach Blatt $q \in Q$:

if q wird geschnitten von Kanten $\in S$ des Pgonzuges:

füge Kantenstück zu M dazu 

else

if q hat nur Vertices $\in S$ in den Ecken:

füge Diagonale zu M dazu 

else q hat Vertices auf ≥ 1 Seitenmitte:

füge Mittelpunkt von q ein,

verbinde M.punkt mit allen Vertices von q, 

füge diese Kanten zu M dazu

Dem.: warum überhaupt Triangulierung? Sonst T-Ecken! Geht bei FEM nicht. Auch sonst oft unangenehm.

Lemma:

Für jede Menge S von disjunkten Polygonzügen
in Quadrat $[0,U]^2$ mit o.g. Eigenschaften
gibt es ein Dreiecks-Mesh M mit o.g. Eigenschaften.
Es hat $O(p(S) \log U)$ Dreiecke
und kann in Zeit $O(p(S) \log^2 U)$ konstruiert werden.
Dabei ist $p(S) = \Sigma$ Länge aller Pgonzüge.

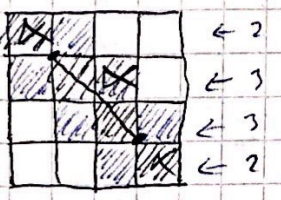
Bew.:

Eigenschaften: klar

Größe des unbalancierten Quadtrees: T

Zellen, die berührt / geschnitten werden, haben 1-Größe.

Strecke der Länge l kann max. $4+3\frac{l}{\sqrt{2}}$ unit cells 1-Zellen schneiden / berühren



$T = \frac{l}{\sqrt{2}} \cdot 3 + 4$

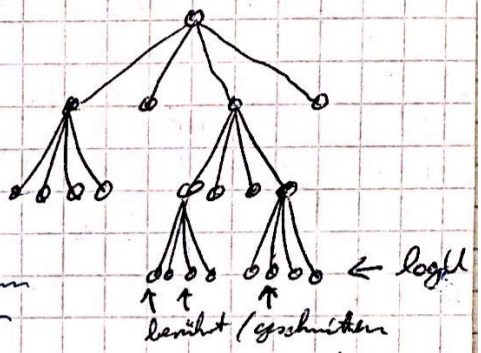
(Bug in 1st Edition)

$\Rightarrow$

Blätter, die berührt / geschnitten werden

$= O(p(S))$ auf bottom layer of T auf Q.-Ebene $\log U$

$\Rightarrow$ # Blätter $\in O(4 \cdot p(S))$



Je 4 Blätter haben 1 Vater, der max 3 Blätter als Brüder hat, die wiederum Kinder sein können

$\Rightarrow$ # Blätter insT $\in O(p(S) \log U)$ [gibt $\log U$ viele Stufen]

($\Rightarrow$ # Knoten $\in$ -"- Braucht man eigtl. nicht, da nur Blätter ins Hash eingehen)

($\Rightarrow$ balancierter Q-tree Q hat $O(p(S) \log U)$ viele Knoten)

$\Rightarrow$ # Dreiecke, denn jedes Blatt produziert max. 8 Dreiecke.

Konstruktionszeit:

von früher: Quadtree T erzeugen kostet $O(\log U \cdot$ # Blätter$)$ Zeit;

Q. tree balancieren kostet nochmal $O(\log U \cdot$ # Knoten$)$;

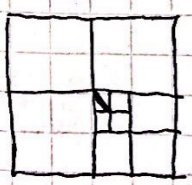
Dreiecke erzeugen kostet $O($# Blätter$)$ Zeit

$\Rightarrow$ Beh.

Q. tree hat $p(S) \log U = n$ Knoten; Zeit für balancierten Qtree ist $O(n \cdot \log U)$

Bem.: Schranke wird angenommen

Bsp.:



$p(S) \equiv$ const

Knoten $= 4 \cdot \log U + 1$

Samet: Hier. Data Struct. and Algos; CACM, Aug '88

Meshing für beliebige Pgonzüge:

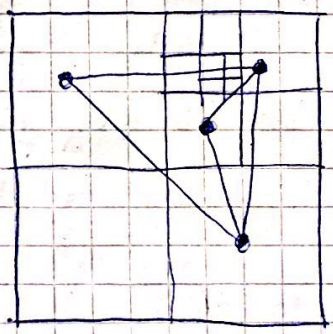
Keine Integer-Koord., beliebige Winkel.

→ gibt verschiedene Blattknoten:

Eckenknoten, Kantenknoten.

Blattkriterium (Unterteilungskriterium):

- Max.-Tiefe
- leer
- genau ein Kantenstück
- genau ein Input-Vertex v, und alle Input-Kanten sind inzident zu diesem Vertex v.



Beliebige Winkel im Pgonzug → Bedingung an Triangulierung abschwächen; jetzt nur noch "möglichst schöne" Dreiecke.

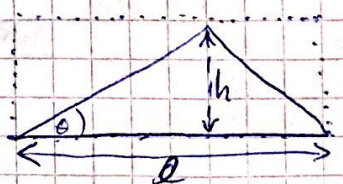
[de Berg, Overmars, et al.: Comp. Geom.]

Def.: "aspect ratio" eines Dreiecks

l := Länge der längsten Seite,

h := Höhe;

aspect ratio $\alpha = \frac{l}{h}$



Bem.: große aspect-ratio = schlechtes Dreieck

$\alpha_{Min} = \frac{2}{\sqrt{3}} \approx 1.15$ = gleichseitiges Dreieck

Wenn θ kleinster Winkel im Dreieck,

dann $\frac{1}{\sin\theta} \leq \alpha \leq \frac{2}{\sin\theta}$.

Ziel beim Mesh-Generation: Aspect-Ratio nahe am Opt.

[Schul, Dutton, Noell: "Grid and Geometry Techniques for Multi-Layer Process Sim.";]

Modifikation der Mesh-Generierung:

1. Balancierten Quadtree erzeugen

2. Triangulieren: 3 Fälle, for each square consider...

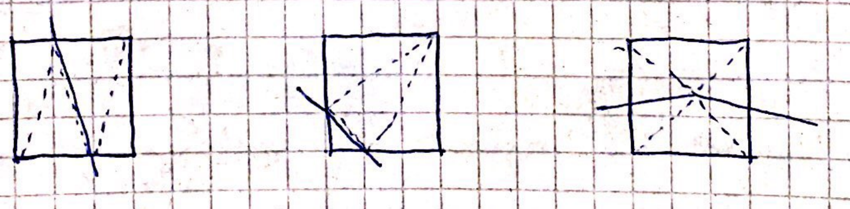
[auch Mare & Verner; Graphics Gems III; nennen es "grid snapping"]

a) leere Knoten → Triangulieren gemäß Templates

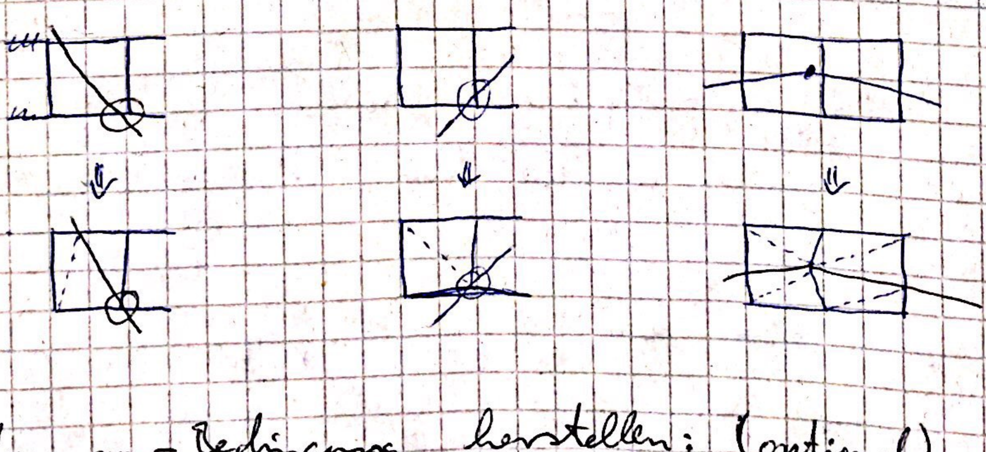
b) Polygonzug schneidet Quadrat & Schn.punkte sind ≥ $\frac{1}{3} l$ von Ecken entfernt & und evtl. Polygonzugecke ist ≥ $\frac{1}{3} l$ von Quadratkanten (und Q.-ecken) entfernt → trianguliere gemäß folgendem Schemas:

(das alles ist Bed. für diesen Fall)

(können jetzt beliebige Winkel sein!)



c) Sonst: deformiere Quadrat (Warping):



3. Delaunay-Bedingung herstellen: (optional)

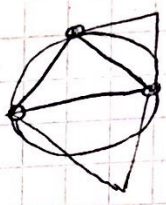
~~s. 4 Seiten später.~~

3. Delaunay - Bedingung nach Triangulierung ...

[Tao Ju's Page, gehört zu "A vector based control kit..."]

D. - Bedingung sagt:

Umkreis um Dreieck, def. durch die 3 Ecken des Dreiecks, darf keine weiteren Ecke der Triangulierung enthalten.



Alle Dreiecke der Triangulierung erfüllen Delaunay - Bed.

⇒ Triangulierung heißt Delaunay - Triang.

[de Berg, Overmars, Comp. Geom.

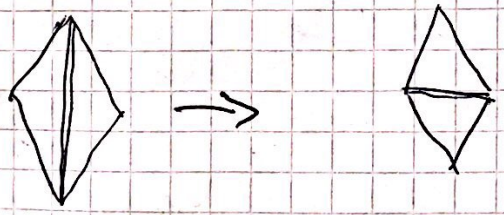
Gilt:

Zwei beliebige Triangulierungen über dieselbe Punktmenge können durch edge swapping ineinander überführt werden. (Nur in 2D!) *

Versuche also, Dreiecke, die Delaunay - Bed. verletzen, durch edge swapping in Delaunay - Dreiecke zu verwandeln.

Delaunay - Triangulierung liefert "beste" mögliche Dreiecke zu gegebenem Quadtree, denn:

Delaunay - Triangulierung maximiert den min. Winkel über alle möglichen Triangulierungen.



Gilt in 3D genauso: aber evtl. kann man keine Delaunay - Tr. herstellen

(*) Im 3D haben nicht alle Triangulierungen die selbe Anzahl Tetraeder (im 2D ist das so). Deswegen kann man nicht jede Triangulierung in eine Delaunay - Tr. überführen, da ein "edge flip" die Anzahl gleich läßt.

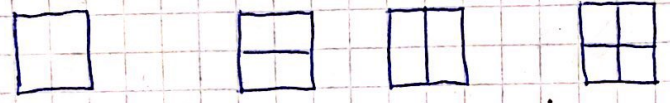
Varianten / Generalisierungen: (Navazo, Hybla, Brunet weiter hinten)

0. Octree (3D), bzw. d-dim Octree in $\mathbb{R}^d$

I. Generalisierter Quadtree:

Unterteile Quadranten nicht immer in 4 Sub-Q.;

a)



(k)ein Pkt | alle Pkte in einer Hälfte, diese ist "voll" besetzt | sonst, d.h., 1 od. 3 Quadrate besetzt, oder 2 schräg gegenüberliegende.

b) nicht immer in Mitte teilen:

analog für Pgonzüge.

Bei Granvertbildung: bestimme Fall anhand Min./Max. in den Zellen.

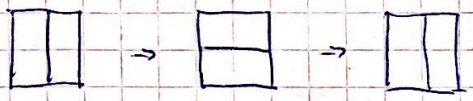
["A Vector Level Control Function for Generalized Octree Mesh Generation", Ken. Johnson, Dutton.]

II. Bintree (spezielle Variante des kd-tree)

splitte immer nur in 2 Hälften, [Samet, Grid A-Paper]

und immer zyklisch entlang xy-, dann xz-, dann yz-Ebene.

Bis Blattkriterium erfüllt.

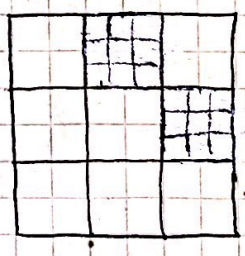


III. N^2-tree:

unterteile Quadrate in N^2 Quadranten. [Siggraph '99: Six DOF Haptic Rendering Using Voxel Sampling]

$N=2$ → Quadtree.

Vorteil: nicht so tief; und braucht evtl. weniger Speicher für größere N – hängt von Dichte der Geometrie ab.



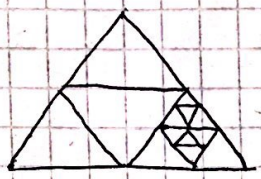
$N=3$

Damit hat man eine fast Kontinuum zw. Quadtree u. Gitter

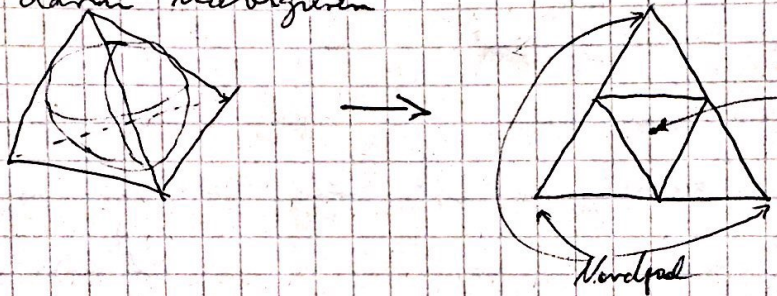
IV. Triangle quadtree:

unterteile nicht in Quadrate, sondern Δ.

[Lee & Samet: Navigating through triangle meshes... CAR-TR-687, 1993.]



Damit kann man schön ein Mesh auf der Kugel generieren: starte mit Tetraeder um Kugel herum und darin navigieren

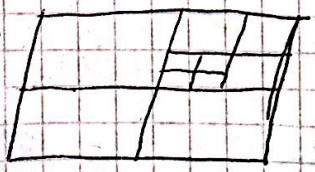


Besser noch starte mit Ikosaeder um Kugel herum, dann sind Verzerrungen kleiner. Ergibt Menge von 20 Quadtrees.

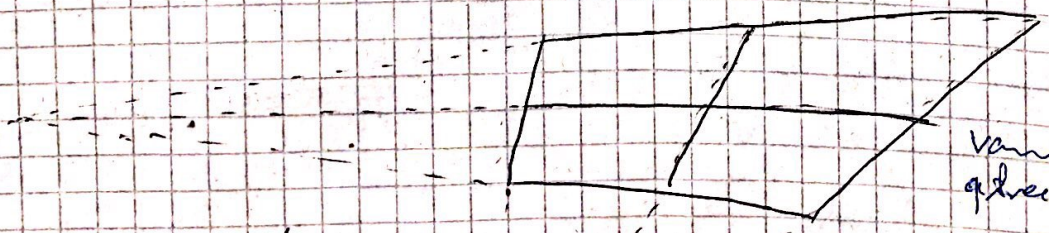
V. ... n-dimensional: Octree, heißt manchmal auch "multi-dim. k-arg search tree" (MKST)

V. Schiefe Quadtrees / vantage quadtrees

Unterteilung nicht ~~gerade~~ mittels rechtwinkliger Ebenen.



Ebenen müssen auch nicht parallel sein:



verwende "Pencil" von Geraden / Ebenen.

Gut geeignet für Rekonstruktion aus Kameraaufnahmen.

Variante VI.: "exakte Quadtrees" (& Octrees)
a.k.a. SP-Octrees

Speichere in den Knoten Stücke der Geometrie.

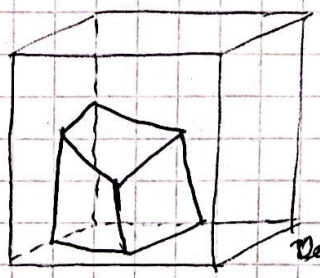
Blattkriterium wie beim Meshing.

5 Arten Knoten (and Blätter!):

- schwarz = innen } Blätter
- weiß = außen

- Vertex-Knoten
- Kanten-Kn. } Blätter
- Face-Kns.

bzw. "grau" bei inneren Knoten



Vertex-Knoten

[Fujimura & Kunii. Carlbom, Chakravarty & Vanderschel. Carlbom nennt es "Polytree". Navazo, Ayala, & Brunet. Samet nennt es "vector tree"]

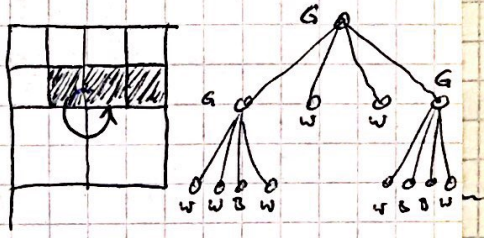
Implementierung:

[Samet, CGRA, May '88 Part I]

I. Pointers: ↖ oder Indizes
I½: R-Leiste
II. Treecode: 1 Zeiger pro Knoten, der auf den Block der 8 Kinder zeigt; alle 8 Kinder werden in diesem Block/Array gespeichert.

Stelle Quadtree als Folge von Knoten dar, in der Reihenfolge von depth-first-Traversierung.

Besonders geeignet für Bilder-Übertragung, und zum Abspeichern von allg. Q.trees.



→ G G W W B W W G W B B W

(d. aufgelöst)

III. Linearer Quadtree:

Repräsentiere Knoten v durch Lkt $(x,y)^{\ell}$ der linken unteren Ecke.

d = Tiefe des Baumes, dann

Kleinster Knoten induziert Gitter $[0,U]^2$, auf dem Quadtree lebt

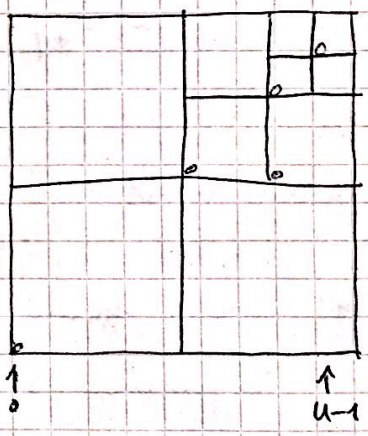
Knoten $v := (x, y, \ell)$, $x, y \in [0, \dots, U-1]$, $\ell \in [0, \dots, d]$

wobei $U = 2^d$.

→ $2d + \log d$ Bits / Knoten.

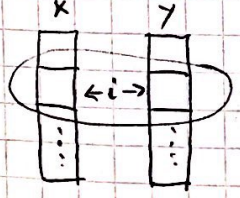
ℓ braucht man, weil Koord. nichts über Tiefe sagt.

(Eigtl. braucht man nur die ersten ℓ Bits von (x,y) speichern.)



(x,y) heißen auch "location codes".

Pfad von Wurzel zu v wird durch (x,y) beschrieben:



→ ergibt Richtung (LU=00, ..., RO=11), die Pfad von Knoten auf Level i zu Knoten auf Level i+1 nimmt.

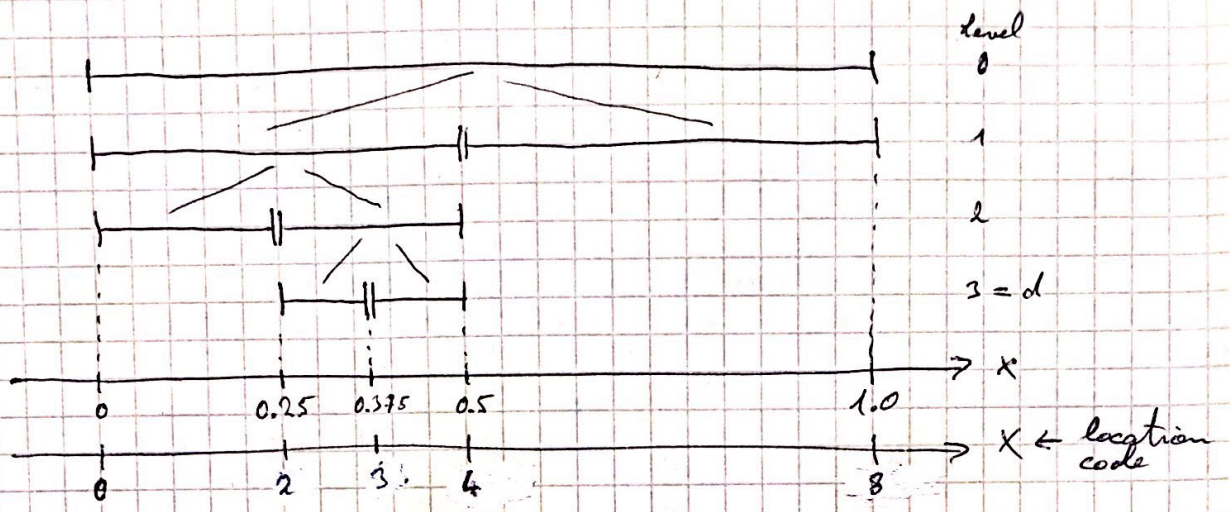
"Linear", weil man (x,y,ℓ) in Liste abspeichern kann, lexikographisch sortiert.

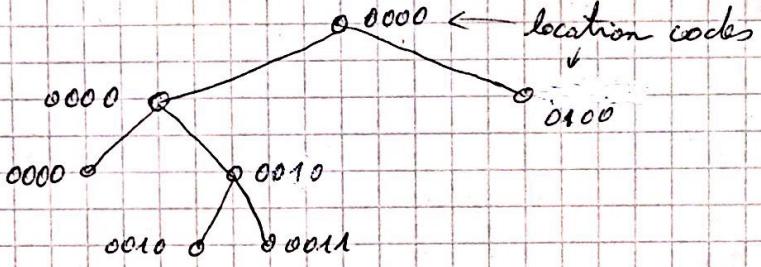
Evtl. auch Bits von x u. y interleaven in 1 Wort.

(Wenn man (x,y) "interleaved" speichert als $x_n y_n \dots x_0 y_0$ nennt man das auch Morton-Code.)

[Eisbeck & Peroy: Simple and Efficient Traversal Methods ... TR von MERL, Nov 2002]

Beispiel für Morton Location Codes in 1D (⇒ "bitree"):





Effizienter Algo. für Point Location in Pointer-basierten Octrees / Quadtrees:

(Bsp. wo Bit-Kniperei wirklich etwas bringt. Gut in 3D 3 Vergleiche)

Gegeben $x, y \in [0,1)$

Konvertiere in location code $X = \lfloor x \cdot 2^d \rfloor$, dito $Y = \ldots$

Traversiere Baum, gesteuert durch Bits der location codes:

```
cell = root
branchbit = 1 << (d-1)
bitnum = d-1
while cell->children ≠ NULL
    childindex = (X & branchbit) >> bitnum
    ~~bitnum --~~
    childindex += (Y & branchbit) >> (bitnum-1)
    cell = cell->children[childindex]
    bitnum -= 1; branchbit = branchbit >> 1;
```

(Ü-aufgabe: zeige, dass bitnum nicht < 0 werden kann. Vorausgesetzt, die übrigen Variablen u. der Quadtree sind konsistent)