24. April 2 Vorlesung

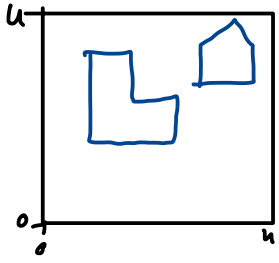
Meshing

Input: Set polylines S in square $\{0,u\}^2$

Simplification: 1 only integer coords

2 only angles in $\{0°, 45°, 90°, 135°\}$

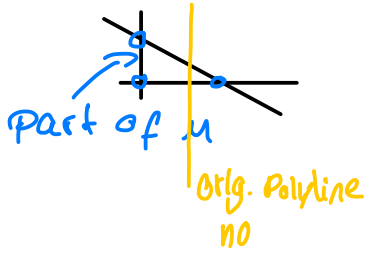
Goal: a triangulation M of the domain with the following properties



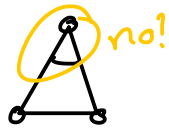
1 conforming mesh M

= no T vertices:

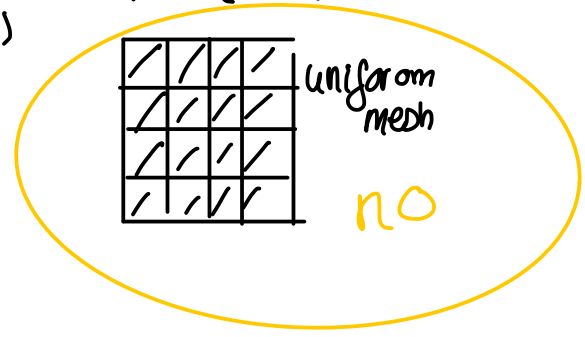
2. M is constrained mesh = no segment from orrginal input crossed a triangle of M



3. M is well-shaped =
no angles other than 45° or 90°
(Later: no long thin triangles)



4. M is not uniform =
not all triangles are same size
(if input permits)

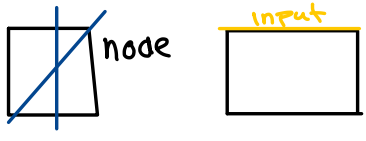


Hübscher aufschreiben

Quadtree over polylines:

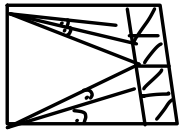
Stopping criterion: no segment of the input intersects* the node or square has size 1×1

*) Def "intersects": proper intersection or segment is contained in the side of the node



Drawing diagonals

in the simple quadtree could lead to

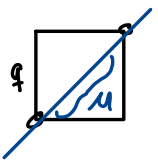


Algorithm:

Generate quadtree T over input polylines
balance $T \rightarrow$ quadtree Q

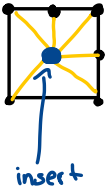
Init M with all edges induced by Q foreach leaf $q \in Q$:

if q is intersected by segment $\in S$:
insert diagonal into M



else

if q has vertices on its sides:
connect it with the vertices on the sides



Lemma:

done with Algo

Given polylines S with above properties in $[0,u]^2$, we can construct a triangle mesh M with above properties.

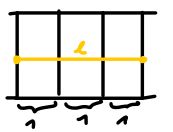
M has $O(p(S) \log u)$ many triangles.
M can be constructed in time $O(p(S) \log^2 u)$.
where $p(S)$ = sum of length of all segments in S.

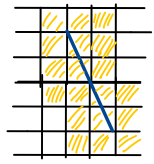
Proof:

1 Size of T (unbalanced).

Nodes that are intersected have size 1:

A segment of length ℓ can intersect at most $4+3\frac{\ell}{\sqrt{2}}$ many cells: (linear in length ℓ)





$\Rightarrow$ #leaves intersect by polylines $\in O(p(S))$

$\Rightarrow$ # leaves on bottom layers of $T \in O(4 \cdot p(S)) = O(p(S))$

(bottom layers: contains all unit cells)

Each 4 leaves share one parent, which can have at most 3 leaf siblings $\Rightarrow$

$$\#\text{leaves in } T \in O(p(S) \cdot \underbrace{\log u}_{\#\text{layers}})$$

Each node generated at most 8 triangles

$\Rightarrow$ #triangles in $M \in O(p(S) \log u)$

2 Construct Time:

Observe that #nodes in $T \in O(p(S) \log u)$

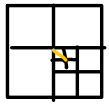
$\Rightarrow$ balanced atree Q has $O(p(S) \log u)$ nodes:

Constructing Q costs $\alpha \log u \cdot \#\text{nodes} = O(p(S) \log^2 u)$

Constructing triangles costs $O(\#\text{leaves}) \Rightarrow O(p(S) \log^2 u)$ time

---

Note: The bound is tight. Example



$p(S) = \text{const}$

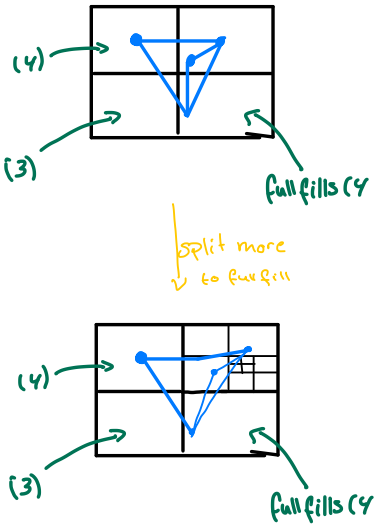
$\#\text{nodes} = 4 \cdot \log u + 1$ in T

Meshing for abitary input:

Leaf criterion:

1) max depth
2) empty (no intersects with polylines)
3) exactly one segment from input
4) exactly one vertex from input, and all segments in the node are incident to that vertex.

Example:



Def: The aspect ratio of a triangle

$\alpha := \frac{\ell}{h}$

where ℓ = longest side, h = height

Note: $\alpha \geq \frac{2}{\sqrt{3}} \approx 1.15$ (in case of equilateral tri)

Also: let θ = smallest angle in tri $\Rightarrow \frac{1}{\sin\theta} \leq \alpha \leq \frac{2}{\sin\theta}$

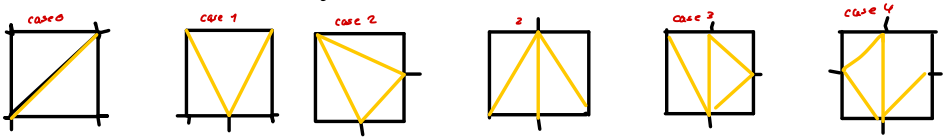
(gives a way to estimate α)

Small = better!

Modify mesh generation

Consider 3 cases for triangulation:

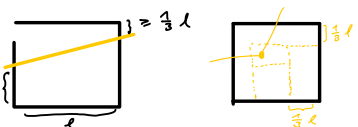
1 Empty node → triangulate acording to the following templates:



4 = #sides with neighbors at smaller size

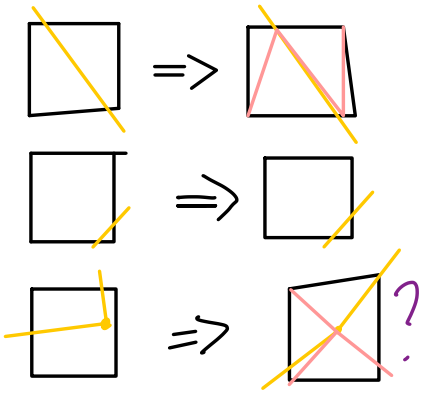
2 Node with one segment or one vetex like this:

- yellow input
- pink part of output



Then triangulate like:

3 else deform square "little"

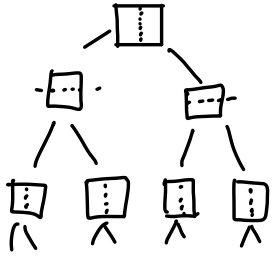


할 일 maybe some mistakes

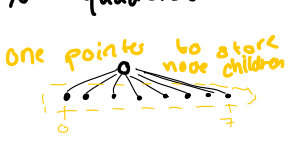
Generalizations:

0) d-dim octree

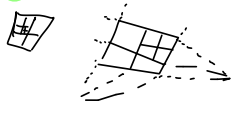
1) Bintree: only split along one axis, in round robin fashion



2) Split node by N^2 children (in 2-D)
"N^2-quadtree":



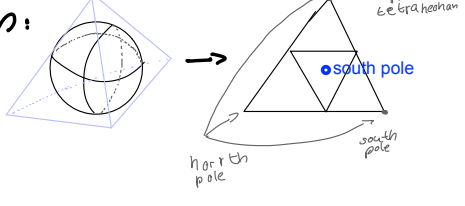
4) oblique quadtree/vantage quadtree



3)

← Triangle quadtree

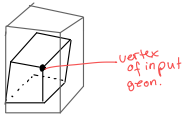
Advantage: works nicely for spheres



5) Exact octree

Nodes store geometry; different types of nodes:

- black = inside the obj
- white = outside -"-
- vertex node
- edge node
- face node



Point location problem

Given $x, y \in [0,1]$
Find the leaf containing $(x,y) = p$

Algo:

```
let m = m(p)  Morton code
start at root. c = root
branchbit = 1 << (d-1)        // "shift 1 d-1 places"
bitnum = (d-1)  depth of QT
 while c has children
   child index = (m & branchbit) >> bitnum
  child i. = branchbit >> 1
  child i.t = (m & branchbit) >> (bitnum-1)
  c = children [child index]
  bitnum -= 1
  branchbit = branchbit >> 1
end while
```