Meshing

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

Simplification:
1. Only integer coords
2. Only angles in $\{0^\circ, 45^\circ, 90^\circ, 135^\circ\}$

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 original input crosses a triangle of $M$
2. \( \mu \) is well-shaped:
no angles other than 45° or 90°
(Cute, no long thin triangles)

4. \( \mu \) is not uniform:
not all triangles are same size
(if input permits)

\[ \text{Uniform mesh} \]

**Quadtree over polylines:**

Stopping criterion: no segment of the input intersects the node
or square has size 1x1

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

Drawing diagonals

in the simple quadtree could lead to

\[ \text{no} \]
Algorithm:

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

Init $M$ with all edges induced by $Q$ for each leaf $q \in Q$:
- if $q$ is intersected by segment $s \in S$:
  insert diagonal into $M$
- else
  if $q$ has vertices on its sides:
    connect it with the vertices on the sides

Lemma:

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

$M$ has $O(ps \log w)$ many triangles.
$M$ can be constructed in time $O(ps \log^2 w)$.
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 $4 + 3 \frac{L}{2}$ many cells:
   \[
   2 \left( e + 2 \right)
   \]
   \[
   \Rightarrow \text{leaves intersect by polylines } \in O \left( p(s) \right)
   \Rightarrow \# \text{ leaves on bottom layers of } T \in O \left( \log U \cdot p(s) \right) = O \left( p(s) \right)
   \]
   Each 4 leaves share one parent, which can have at most 3 leaf siblings:
   \[
   \# \text{ leaves in } T \in O \left( p(s) \cdot \log U \right)
   \]
   \[
   \Rightarrow \# \text{ leaves in } T \in O \left( p(s) \cdot \log U \right)
   \]
   \[
   \Rightarrow \# \text{ triangles in } M \in O \left( p(s) \cdot \log U \right)
   \]

2. Construct Time:
   Observe that $\# \text{ nodes in } T \in O \left( p(s) \cdot \log U \right)$
   \[
   \Rightarrow \text{ balanced tree } Q \text{ has } O \left( p(s) \cdot \log U \right) \text{ nodes:}
   \]
   Constructing $Q$ costs $O \left( \log U \cdot \# \text{ nodes} \right) = O \left( p(s) \cdot \log^2 U \right)$
   Constructing triangles costs $O \left( \# \text{ leaves} \right) = O \left( p(s) \cdot \log^2 U \right)$ time

Node: The bound is tight. Example
Meshing for arbitrary 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:

Modify mesh generation:

Consider 3 cases for triangulation:

1. Empty node → triangulate according to the following templates:

   - Case 1
   - Case 2
   - Case 3
   - Case 4

   $4 = \# \text{sides with neighbors at smaller size}$

2. Node with one segment or one vertex like this:

   - Then triangulate like:

   | Yellow input |
   | Pink part of output |
3 else deform square "little"

Generalizations:

1) d-dim octree
2) Bin-tree: only split along one axis, in round robin fashion
3) Triangle quadtree
4) Oblique quadtree/vantage quadtree
5) Exact octree

Nodes store geometry; different types of nodes:
- Black = inside the obj
- White = outside
- Vertex node
- Edge node
- Face node

Advantage: works nicely for spheres
Point Location Problem

Given $x, y \in [0, 1]^2$
Find the leaf containing $(x, y) = r$

Algo:

let $m = m(r)$ Morton code
start at root: $c = root$
branchbit = $1 << (d-1)$ // 'Shift 1 d-1 places'
bitnum = $(d-2)$ depth of QT
while $c$ has children
    child index = ($m \&$ branchbit) $\gg$ bitnum
    child i. = branchbit $\gg$ 1
    child i. t = ($m \&$ branchbit) $\gg$ (bitnum - 1)
    $c = children[\text{child index}]
    bitnum -= 1
    branchbit = branchbit $\gg$ 1
end while