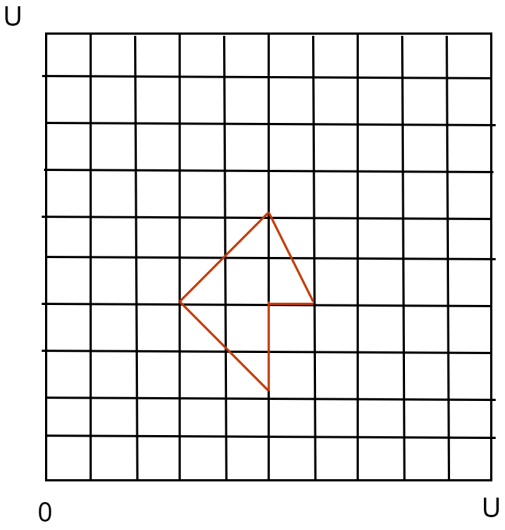
Dynamic meshing

Domain

What we want to achieve is a dynamic construction of a mesh, with the possibility to place a set of polylines over the domain.



for the sake of simplicity we will define:

input $\in [0, U]^2$

only integer coordinates

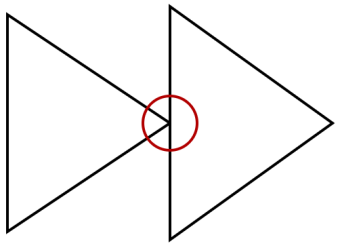
only angles $\in \{0^\circ, 45^\circ, 90^\circ, 135^\circ\}$

Properties

now we can define the “nice properties” we discussed in the QuadTree introduction for our triangulation:

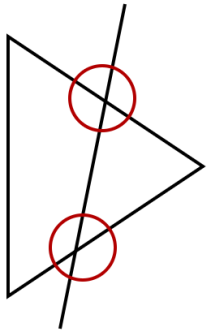
Consistent mesh

the meshes must have no t-edges, being when the edges of a triangle touches the side of another, otherwise we will be introducing edges



Constrained mesh

the polylinesmust be parts of the edges, we do not want interception of polyghones, otherwise we will be introducing edges

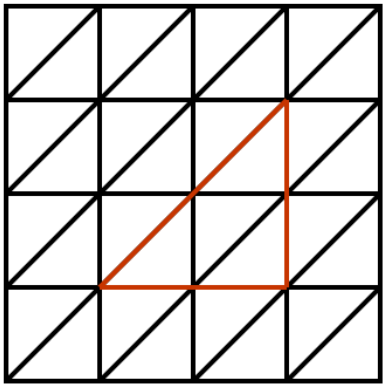


Well shaped mesh

we do not want really acute angles, called “**slivers**”, they could be source of problems

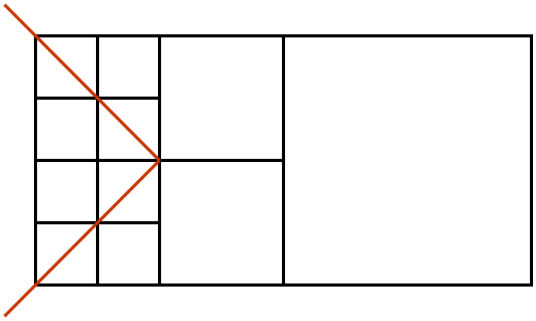
Adaptive mesh

we do not want to introduce too many triangles when they are useless, otherwise wi whould be wasting triangles and computational power



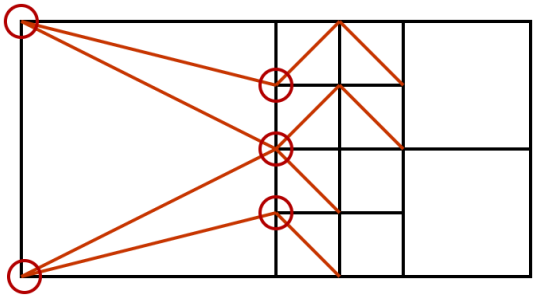
Solution

Obviously our solution is the use of quadtrees, we need to build a quadtree over a set of polylines. the algorhitm is similiar to the one where we build a quadtree over a set of points, but we have a different stopping criterion:

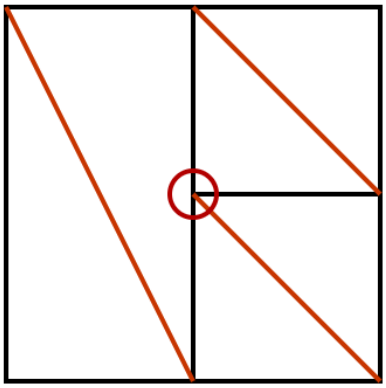


- size of nodes = 1x1
- no edges intercepts or touches the node

but some problems could still happen, like for example:



some slivers

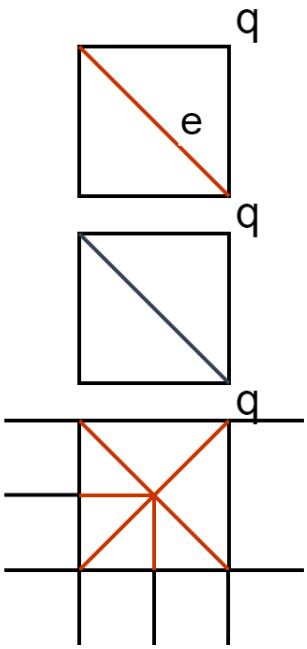


a t-edge

the solution is easy: we need to balance the quad tree!

A sketch of the solution

```
input: S = polyghon with "properties"
output: M = mesh with "nice properties"
balance T->Q
initialize M with edges induced by Q
for each leaf q in Q:
  if q is intercepted by edge e in S:
    add edge segment to M
  else if q has only vertices in corners:
    add diagonal to M
  else #Q has a vertex inside
    add center point to M
    triangulate with te vertices
```



Lemma

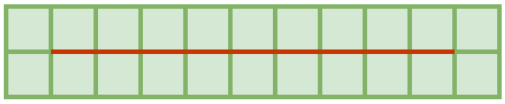
Question: how many triangules can the mesh have?

Let s be a polyline with above properties in $[0, U]^2$, there is a triangularar mesh that has $O(log_2(U)p(s))$ triangles that can be constructed in $O(p(s)log_2^2 U)$ time

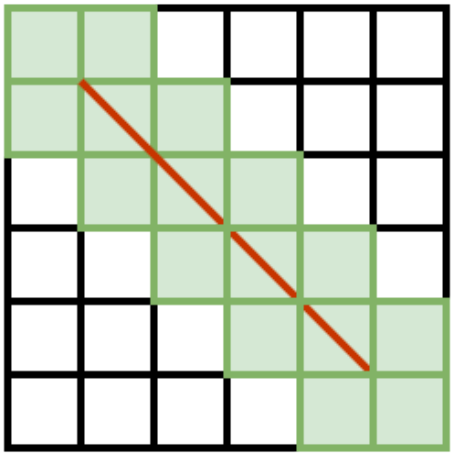
where $p(s)$ is the sum of lenght of all polylines

Proof

cells that gets touched or intercepted has size 1x1, due to stopping condition, so it means that an edge touched or intercepted by at most $4 + 3\frac{l}{\sqrt{2}}$ cells, 4 for the corners and 3 for each unit of lenght(the worst case are the diagonal lines)



in this case we have $2(l + 2)$ touching/intercepting nodes, for each touched square



in this case we have 5l+1 touching/intercepting nodes

the number of touched/intercepted leaves must be in the order of the total lenght of the polylines $O(p(s))$, so the leaves in the last layer must be in the order of $O(4p(s))$, and since the depth of the tree is $log_2(U)$ we get a maximum number of childs in the order of $O(p(s)log_2(U))$ and since every leaf can produce at maximum 4 trinagles(4 diagonal edges and 4 horizontal) we have that as a complexity

the time compexity now depends just from the time needed to build the triangles, we need a $O(log_2 U)$ time to reach a leaf node and a $O(1)$ time to build the triangle, leaving us with a $O(p(s)log_2^2(U))$ time complexity

Meshing for arbitrary polylines

we assumed the lines always has only integer coordinates and only angles in the range $\{0^\circ, 45^\circ, 90^\circ, 135^\circ\}$

now we want to see what happens if thoose boundaries are violated. to do so we need to introduce some different leaves nodes:

- edge nodes
- vertex nodes

and different stopping criterions:

- max depth reached(good practice)
- the node is empty(no vertex or edge in the node)
- the node has exactly one edge or part of edge inside, so no vertex
- exactly one vertex and no edges intercepting the node incident to the vertex

in order to do the triangulation we need to introduce the concept of **aspect ratio of a triangle**

given

- l the longest side of the triangle
- h the height

the aspect ratio of a triange is $\alpha = \frac{l}{h}$

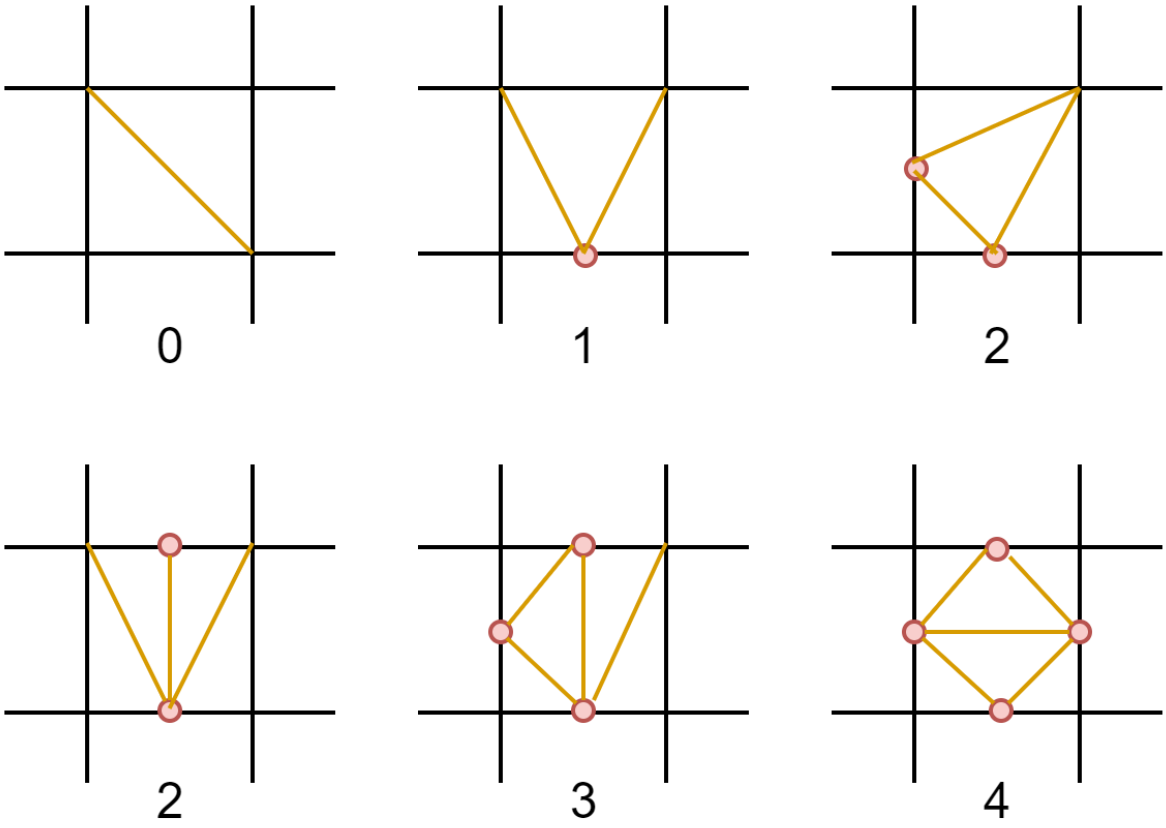
note that the minimum value is $\alpha \geq \frac{2}{\sqrt{3}} \approx 1.15$

if θ is the smallest angle of the triangle we always have

our objective is now to create triangules with an aspect rateo close to the optimum, we need to modify the alghorithm modified with the new stopping criterion, then we need to triangulate using 3 cases:

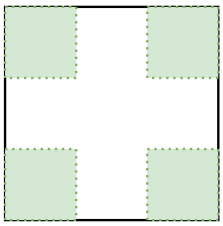
case 1:empty leaf

we need to triangulate using some templates, the node only has a vertex on the edges, so we introduce new edges to create triangules

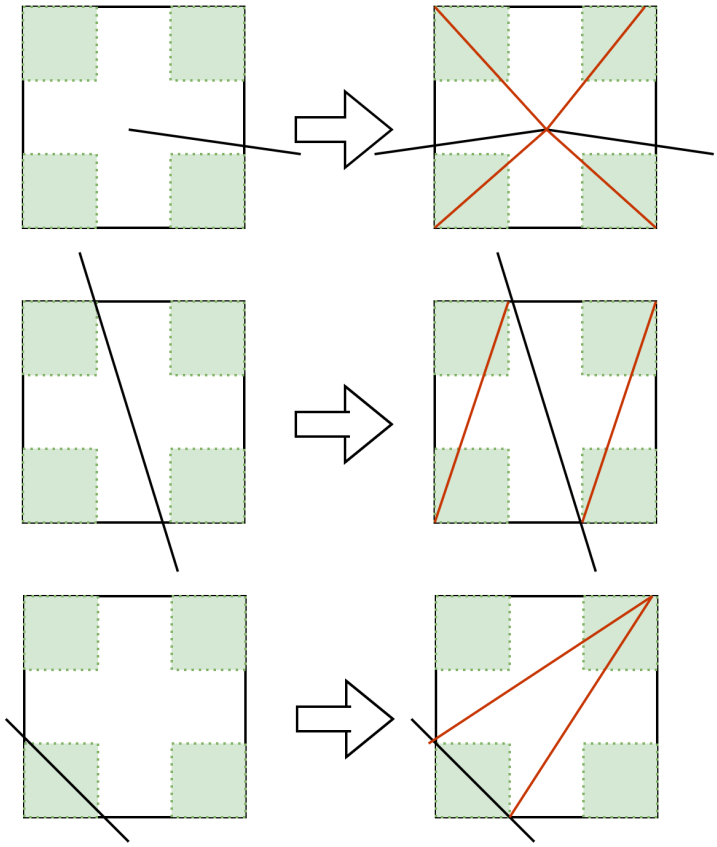


Case 2:leaf containing an edge that intercepts the edge

in this case we can do something if the interceptions happens in the inner thirds of the node, doin in this way we are not changing too mutch the aspect ratio of the triangules

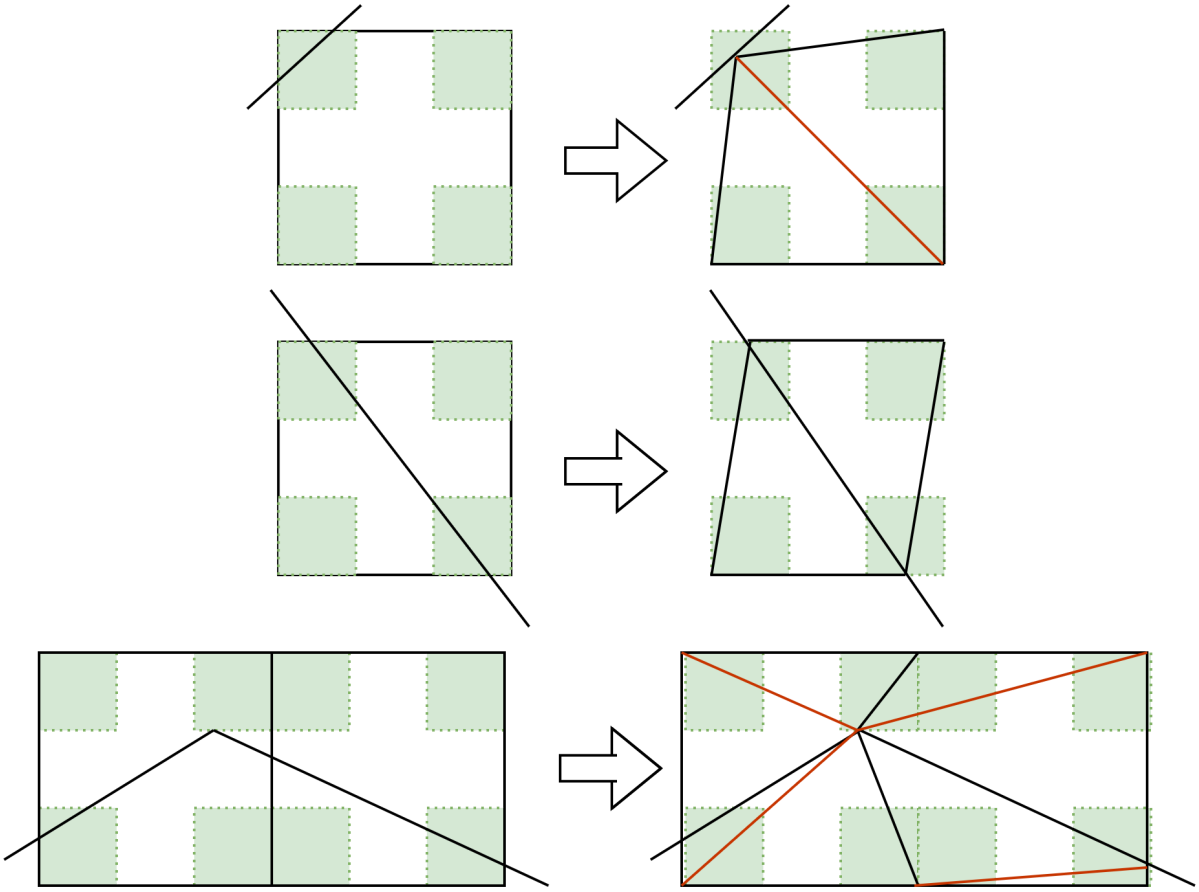


in this case we can introduce vertexes in the side or in the middle and edges to keep the nice triangulation



Case 3: in any other cases

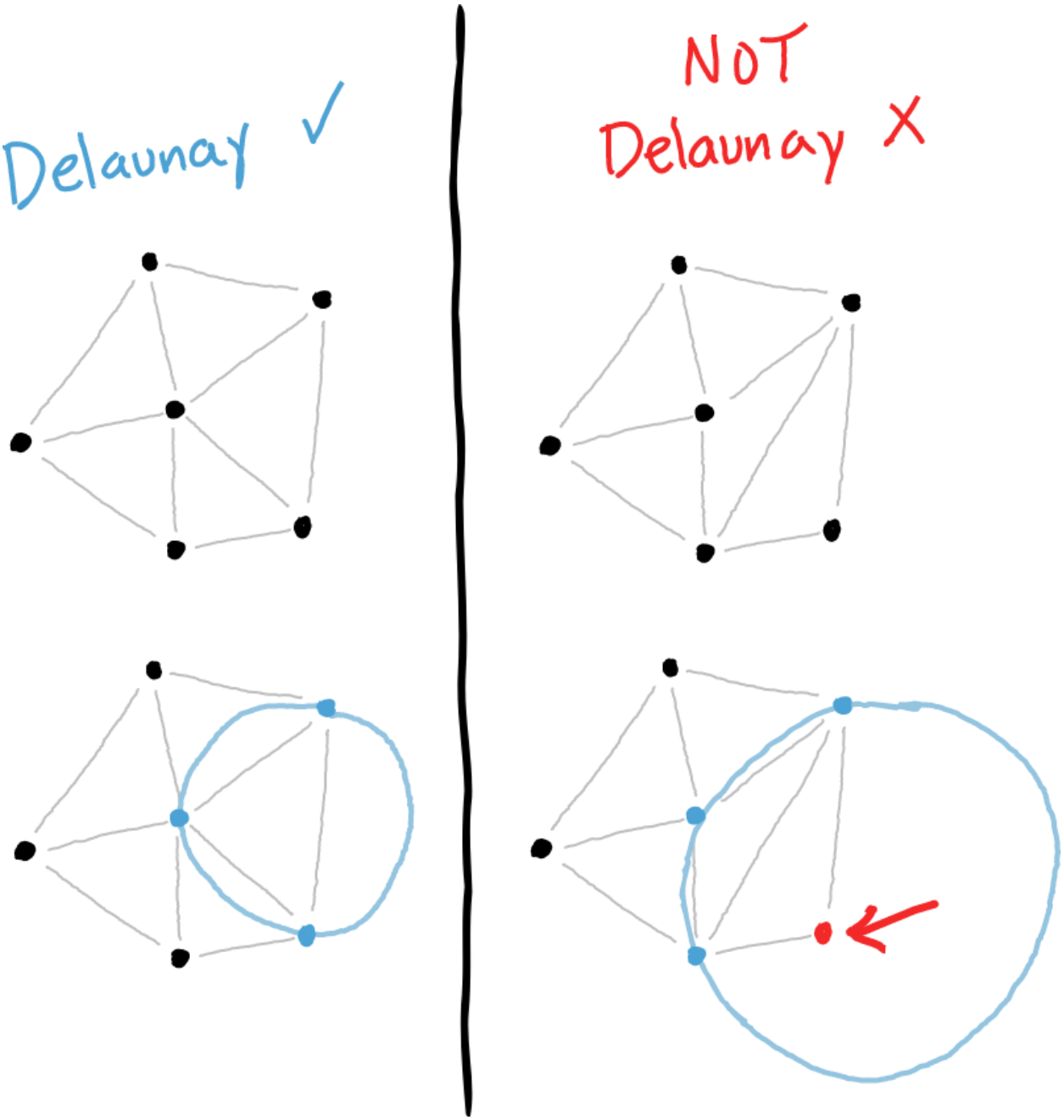
we need to “**warp**” the square to fit the edges



Delaunay triangulation

this is a nice way to improve our triangulation.

A triangule is a Delauney triangule $\iff$ its circumcicle does not contains other verteces from the mesh we can perform an operation called **edge flip** where we rotate inner edges inside a mesh in order to respect delauney rule



Cool material

https://shwestrick.github.io/2021/12/18/delaunay-viz.html