QuadTrees

Quadtrees are trees used to efficiently store data of points on a two-dimensional space. In this tree, each node has at most four children. We can construct a quadtree from a two-dimensional area using the following steps:

1. Divide the current two dimensional space into four boxes.
2. If a box contains one or more points in it, create a child object, storing in it the two dimensional space of the box
3. If a box does not contain any points, do not create a child for it

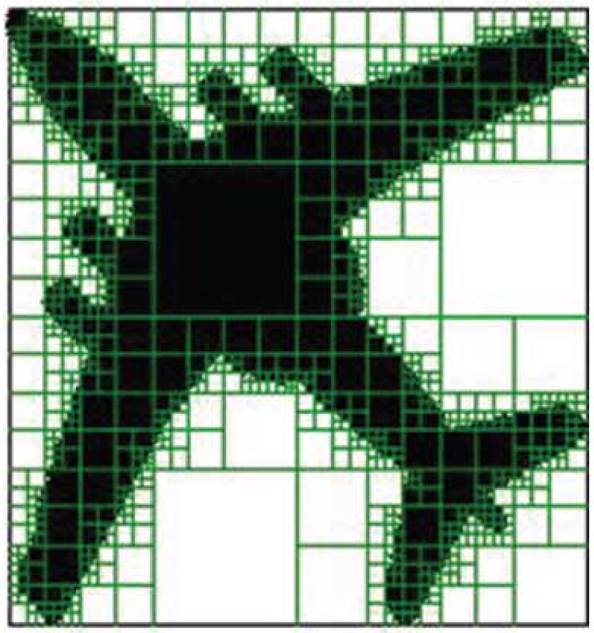
   Quadtrees are used in image compression, where each node contains the average colour of each of its children. The deeper you traverse in the tree, the more the detail of the image. Quadtrees are also used in searching for nodes in a two-

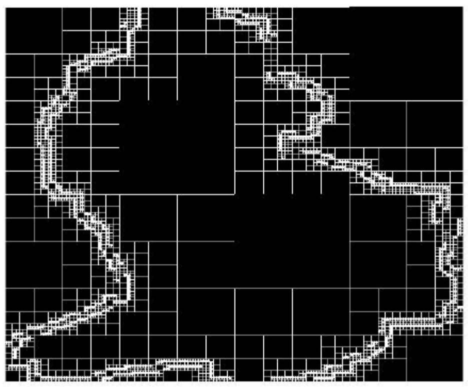
> dimensional area. For instance, if you wanted to find the closest point to given coordinates, you can do it using quadtrees.
>
> 4. Recurse for each of the children.
>
> (https://www.geeksforgeeks.org/quad-tree/)

https://en.wikipedia.org/wiki/Quadtree#/media/File:Quadtree_compression_of_an_image.gif

The problem we want to solve

We have a polyghon in 2d or 3d and we want to find a triangulation wich has some "nice" properties, so we want to be able to identify easily the location of some points in space





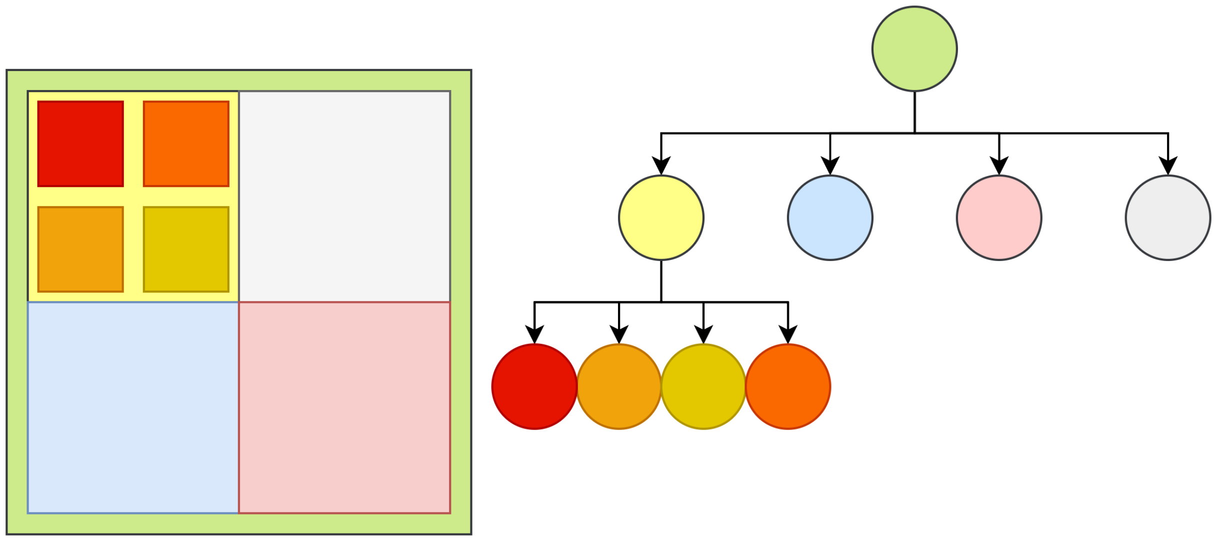
A cool structure that comes helpful is a tree, expecially one that divides the space is four equal regions.

Definitions

Quad tree

A quad tree is a tree where each inner node has 4 children and each of them corresponds to a square. It is an unbalanced structure, meaning that every node must either be an inner node and have exactly 4 children, or a leaf node and have 0 children

The actual subdivision is made fom the top left corner spinning in anti-clockwise verse



We can define some terminology to use:

Vertex: one of the corners of a square

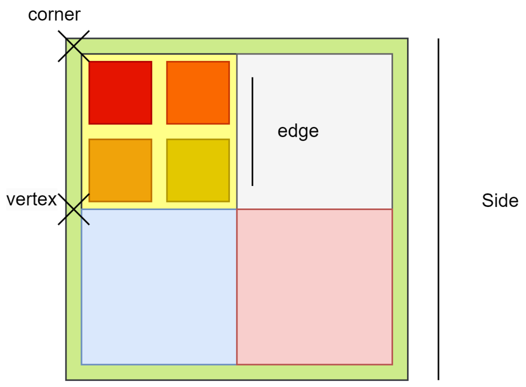
Edge: one of the sides of a square

Side: one of the edges of the root square

Corner: one of the corners of the root square

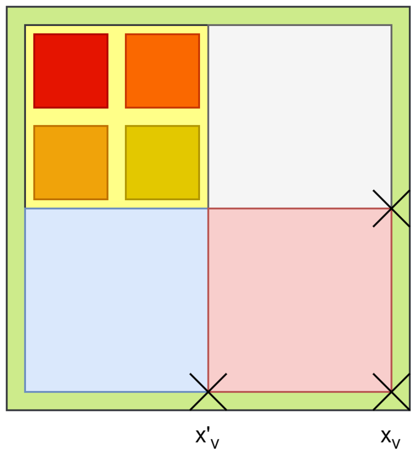
Neighbors nodes: two nodes sharing an edge

little note: $side \implies edge, corner \implies vertex$



Square of a node

Given a node v, $q(v) = [x_v, x'_v] \times [y_v, y'_v]$, so the square of a node is the actual region of the space inside the actual square formed by $[x_v, x'_v] \times [y_v, y'_v]$



Quad tree over a set of point

given a set of points $P \in \mathbb{R}^2$:

$$Q(P) := node\ v : \begin{cases} v\ is\ leave & if\ |P| \leq 1 \\ v\ is\ inner\ node & else \end{cases}$$

With childrens:

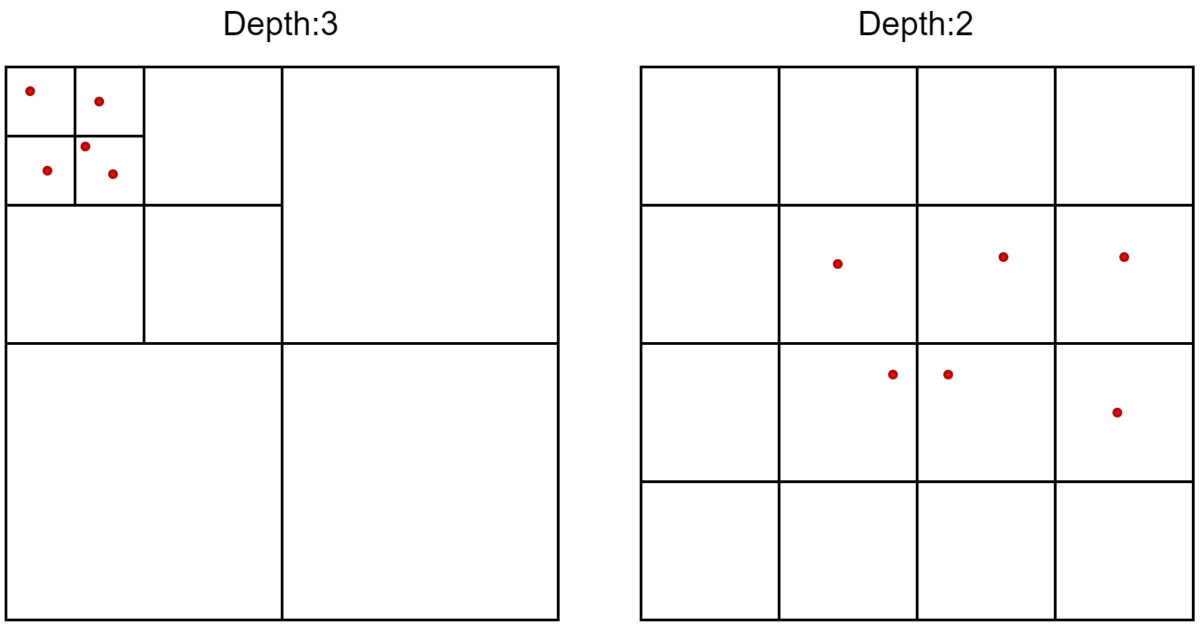
$$V_{UL} = \{p \in P : x_p < \frac{1}{2}(x_v - x'_v), y_p < \frac{1}{2}(y_v - y'_v)\}$$

$$V_{UR} = \{p \in P : x_p > \frac{1}{2}(x_v - x'_v), y_p < \frac{1}{2}(y_v - y'_v)\}$$

$$V_{BL} = \{p \in P : x_p < \tfrac{1}{2}(x_v - x'_v), y_p > \tfrac{1}{2}(y_v - y'_v)\}$$
$$V_{BR} = \{p \in P : x_p > \tfrac{1}{2}(x_v - x'_v), y_p > \tfrac{1}{2}(y_v - y'_v)\}$$

Depth of a quad tree

the depth of a quad tree depends mostly on the points in the set P we want to triangulate, as an example let's consider the following 2 sets of points:



In the second case it's possible to use a less deep tree, since the points are more uniformely scattered around, in the first case it's impossible to locate the position of a point without using almost a depth 3

Lemma

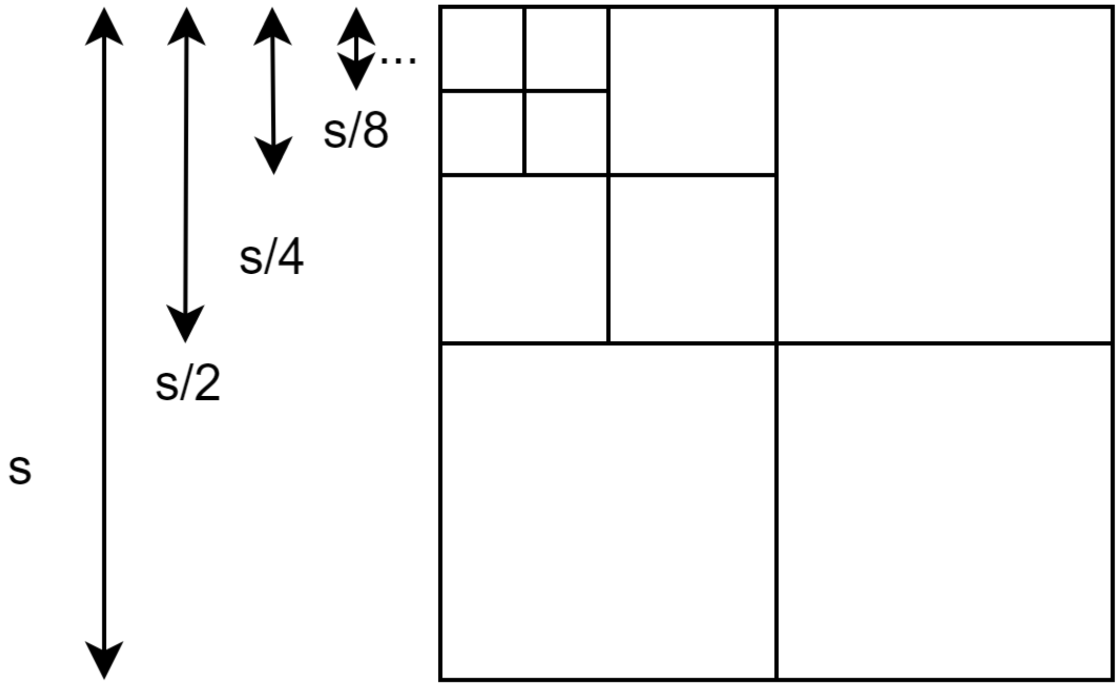
Let P be a set of points in $\mathbb{R}^2$, s be the side length of q(Root), and c $\in \mathbb{R}$ s.t.:

$c = \min\{\|p_i - p_j\| : p_i, p_j \in P, i \neq j\}$, so c is the maximum distance between 2 points in a node, or also the distance between the 2 closest points, then, it follows for the depth d of the tree:

$d \leq \log_2(\frac{s}{c}) + \frac{3}{2}$

Proof

if we split in half a square with side lenght s a number i of times, the side of each sub square is long exactly $\frac{s}{2^i}$



so we can say that the maximum distance between two points in the same square occours when they are in the deepest division(last level)and they are on the opposite vertices of the square, this distance is the diagonal of the square, having size $\sqrt{2} \cdot edgeLen$, and since edgeLen is $\frac{s}{2^i}$, we obtain $\frac{\sqrt{2}s}{2^i}$. if we now take every other disctance between points c' we can say that

$\frac{\sqrt{2}s}{2^i} \geq c' \geq c \Rightarrow \frac{\sqrt{2}s}{c} \geq 2^i \Rightarrow i \leq \log_2 \frac{\sqrt{2}s}{c} \Rightarrow i \leq \log_2 \sqrt{\frac{s}{c}} + \log_2 \sqrt{2} \Rightarrow i \leq \log_2 \frac{s}{c} + \frac{1}{2}$

this is true for inner nodes, but in the leaves we need to add another subdivision to i, bringing the depth to $i \leq \log_2 \frac{s}{c} + \frac{1}{2} + 1 = \log_2 \frac{s}{c} + \frac{3}{2}$

Balanced Quad Trees

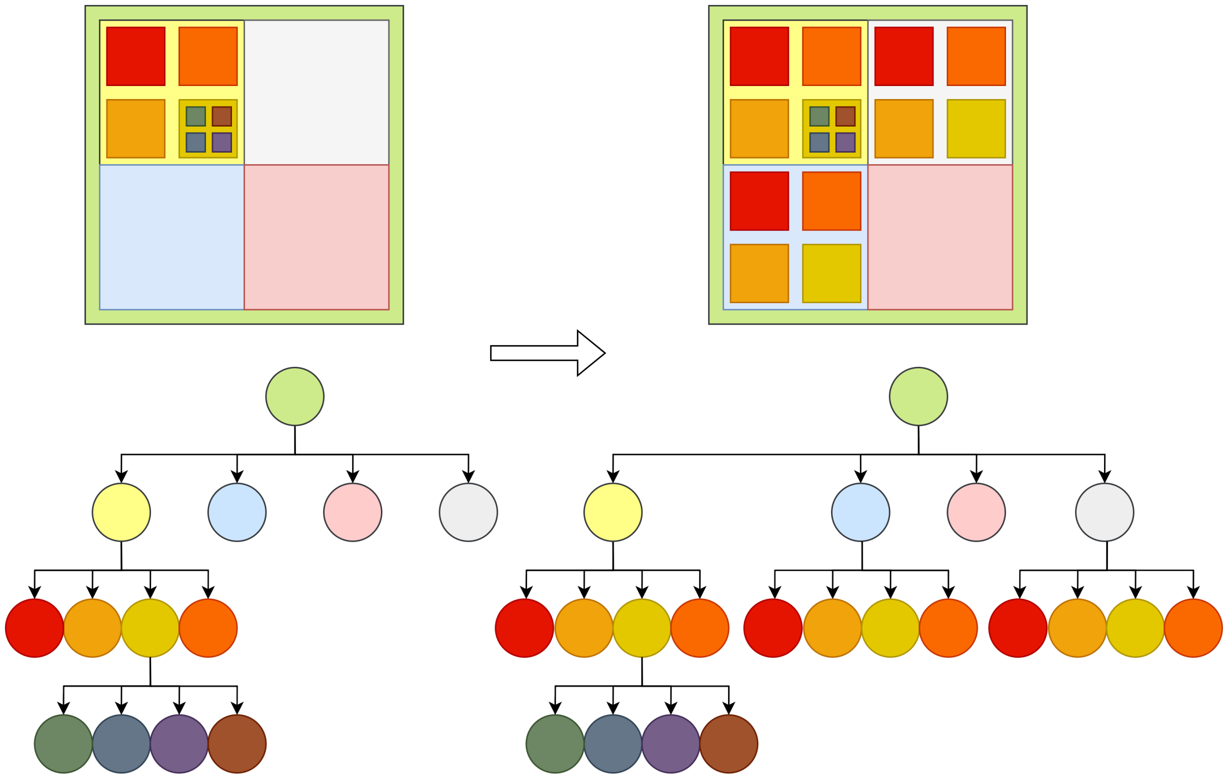
A quad tree is a balanced Quad Tree $\hat{Q}$ when the size of the neighbors differ maximum by a factor 2:

$$\forall v, v' \text{ s.t. } v' \text{ is a neighbor of } v, |depth(v') - depth(v)| \leq 1$$

turning a general quad tree to a balanced one can be done in the same time of building a quad tree, so the conversion can be done also after the creation, since

$O(X + X) = O(2X) = O(X)$.

to balance a quad tree is necessary to insert new nodes, as in the examle:



In order to make a tree balanced we need to check for every leaf if it needs to be split, comparing its depth with the neighbors depth, then check if its neighbors should be split too and in case apply the split

Lemma

Let Q be a quad tree with m nodes, $\hat{Q}$ is the balanced quad tree for Q, then $\hat{Q}$ has O(m) nodes that could be constructed in time O(m(d+1))

Proof

being D(v) the depth of the subtree with root the node v, we can assume that $D(leafNode) = 0$. For each node with $D(v) \geq 2$ we need to split every neighbor node with $depth < 2$ only one time. the splitting will always be propagated in the direction of the node we mentioned, so we are never propagating the splitting operation through the not neighbors nodes, resultingin a maximum number of splits of m.

for the time complexity we can affirm that every “old node” that gets split introduces a number of “new nodes”$\leq 8 \cdot 4$, where 4 is the number of children per node, and 8 is the maximm number of neighbors a node can have. ath this point the number of splits depends from the depth of the tree, including the number of leaves, so d+1, so the worst possible case is when we have to traverse for every node the whole tree, bringing us to a complexity of O(m(d+1))

Complexity

A quad tree with depth d and n points has a time complexity of **O((d+1)n)** and a space complexity of **O((d+1)n)**

Proof

Space complexity

In the worst case we will build a tree with 4 children for each node, until the leaves, and have all the leaves balanced on the same level. let’s consider the number of nodes for each level

level	nodes in level	total nodes
0	1	1
1	4	1+4=5
2	16	5+16=21
3	64	21+64=85
…	…	…
d-1	4^{d-1}	$\sum_{i=0}^{d-1} 4^i$

In general we can affirm that

$\#nodesInLeveli = 3 \cdot \#sumOfNodesUntilLevel(i-1)+1$

and then going through all the levels until level d-1 we obtain that $\#nodes \leq n(d-1)+2n$, where the 2n is because the quadruplet to exist needs to have minimum 2 levels with almost a point inside each

Time complexity

Given m points inside the square q(v) of a node v the time to reach the node T(v)=O(m)

and since the number of points in each level must be $\leq n$, being n the total number of points, and we have d-1 levels the time complexity is O((d-1)n).

Neighbors finding

Given a node v we want to find one neighbor north, south, west or east v' of v, fordoing so we search for a neighbor such that $depth(v\text{'}) \leq depth(v)$

```
def northNeighbor(v):
    if v is root:  #base step
        return null
#at this point we know the parent of v exist
    if parent(v).LLChild == v:
        return parent(v).ULChild
    if parent(v).LRChild == v:
        return parent(v).URChild
#at this point we know that the neighbor of v is not child of the same parent
    u = northNeighbor(parent(v))
    if u is null or isLeaf(u):
        return u #will eventually reach the base step if v is on the top edge
    if parent(v).ULChild == v:
        return u.LLChid
    if parent(v).URChild == v:
        return u.LRChid
#this method does not have a return for every path, also if programmatically incorrect,
#it's almost impossible to reach this point
```

```
def southNeighbor(v):
    if v is root:  #base step
        return null
#at this point we know the parent of v exist
    if parent(v).ULChild == v:
        return parent(v).LLChild
    if parent(v).URChild == v:
        return parent(v).LRChild
#at this point we know that the neighbor of v is not child of the same parent
    u = southNeighbor(parent(v))
    if u is null or isLeaf(u):
        return u #will eventually reach the base step if v is on the top edge
    if parent(v).LLChild == v:
        return u.ULChid
    if parent(v).LRChild == v:
        return u.URChid
#this method does not have a return for every path, also if programmatically incorrect,
#it's almost impossible to reach this point
```

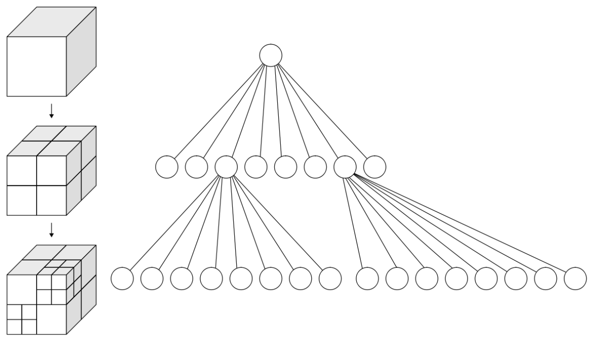
```
def eastNeighbor(v):
    if v is root:  #base step
        return null
#at this point we know the parent of v exist
    if parent(v).LLChild == v:
        return parent(v).LRChild
    if parent(v).ULChild == v:
        return parent(v).URChild
#at this point we know that the neighbor of v is not child of the same parent
    u = eastNeighbor(parent(v))
    if u is null or isLeaf(u):
        return u #will eventually reach the base step if v is on the top edge
    if parent(v).URChild == v:
        return u.ULChid
    if parent(v).LRChild == v:
        return u.LLChid
#this method does not have a return for every path, also if programmatically incorrect,
#it's almost impossible to reach this point
```

```
def westNeighbor(v):
    if v is root:  #base step
        return null
#at this point we know the parent of v exist
    if parent(v).LRChild == v:
        return parent(v).LLChild
    if parent(v).URChild == v:
        return parent(v).ULChild
#at this point we know that the neighbor of v is not child of the same parent
    u = westNeighbor(parent(v))
    if u is null or isLeaf(u):
        return u #will eventually reach the base step if v is on the top edge
    if parent(v).ULChild == v:
        return u.URChid
    if parent(v).LLChild == v:
        return u.LRChid
```

```
#this method does not have a return for every path, also if programmatically incorrect,
#it's almost impossible to reach this point
```

Variants and generalization of QuadTrees

Quad-Trees in higher dimension

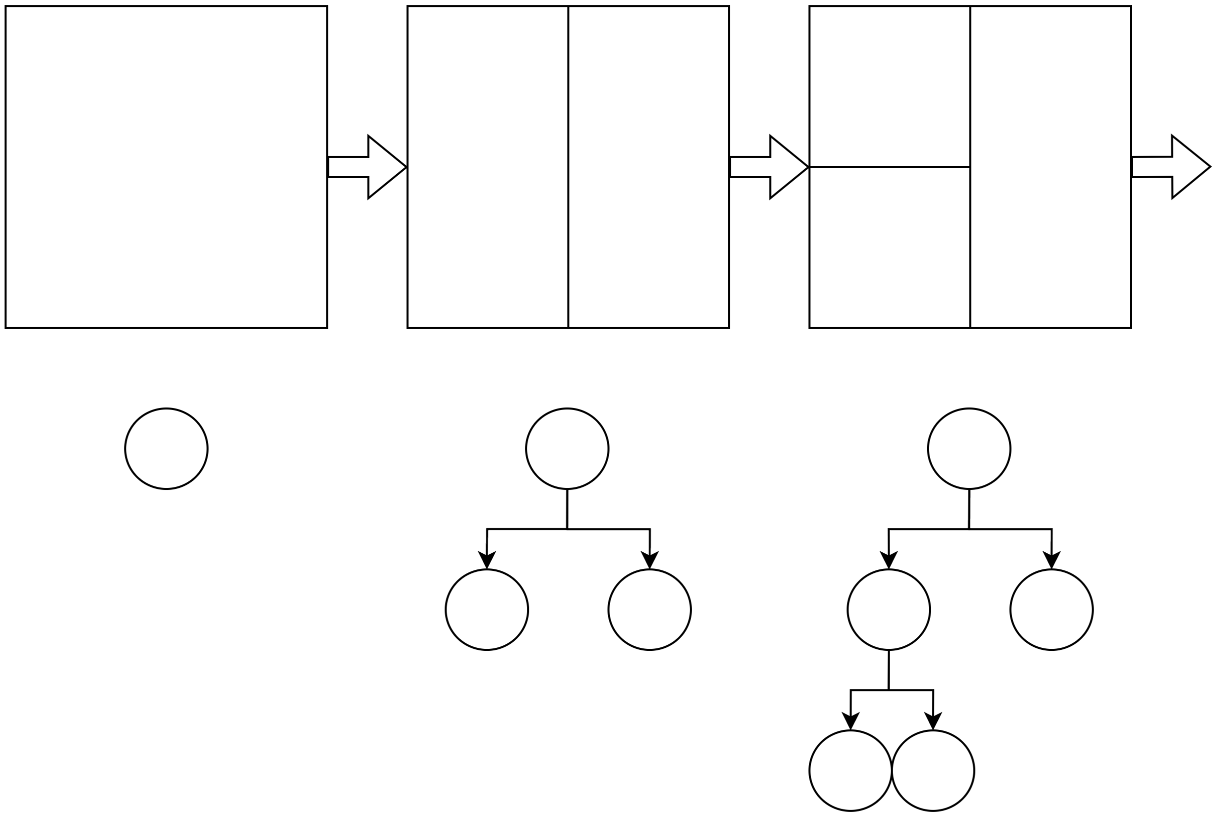


According to the dimension we can define:

- 2 dimensions: **QuadTree**
- 3 dimensions **OctTree**
- d>3 dimensions **d-dimensional OctTree**

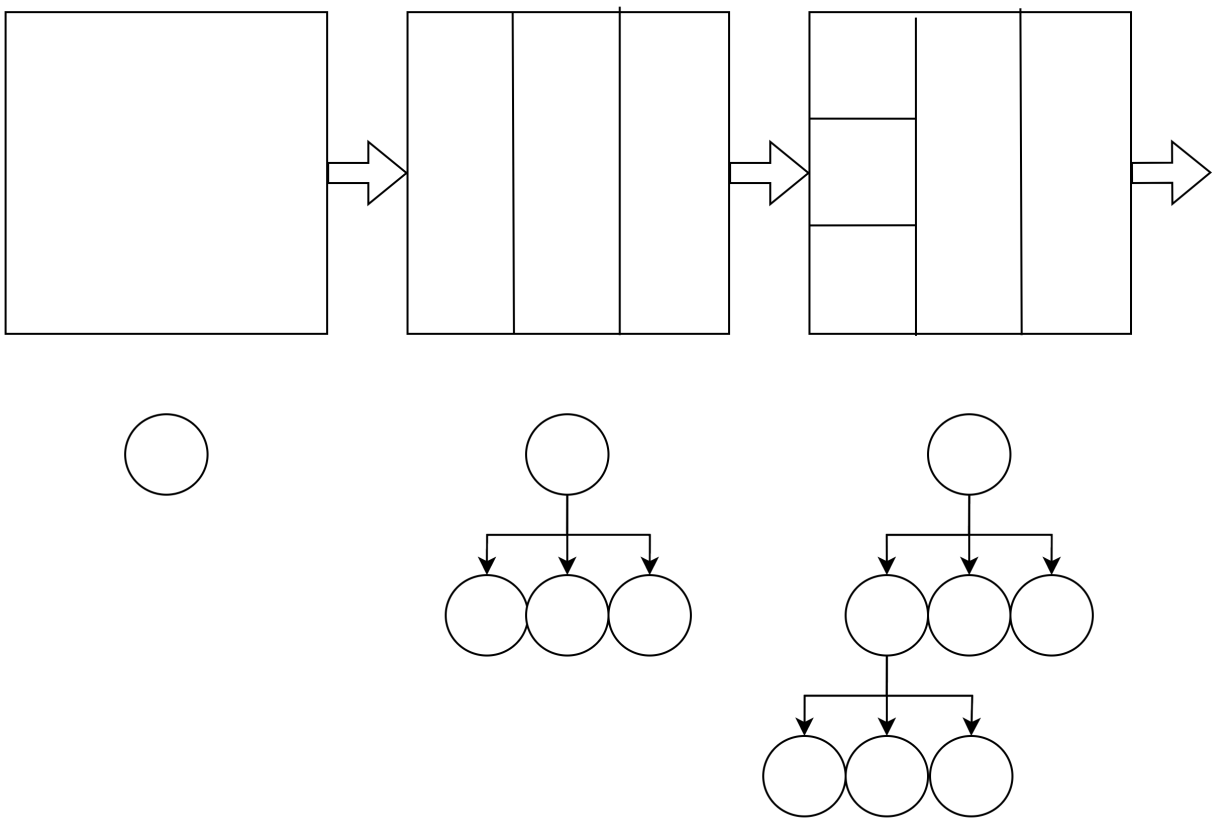
BinTree

every node has 2 children, so the tree is a binary tree. We split horizontally/vertically in an alternate manner, in this case for every 2 node layers we have a node in the original quadtree



N^2-Tree

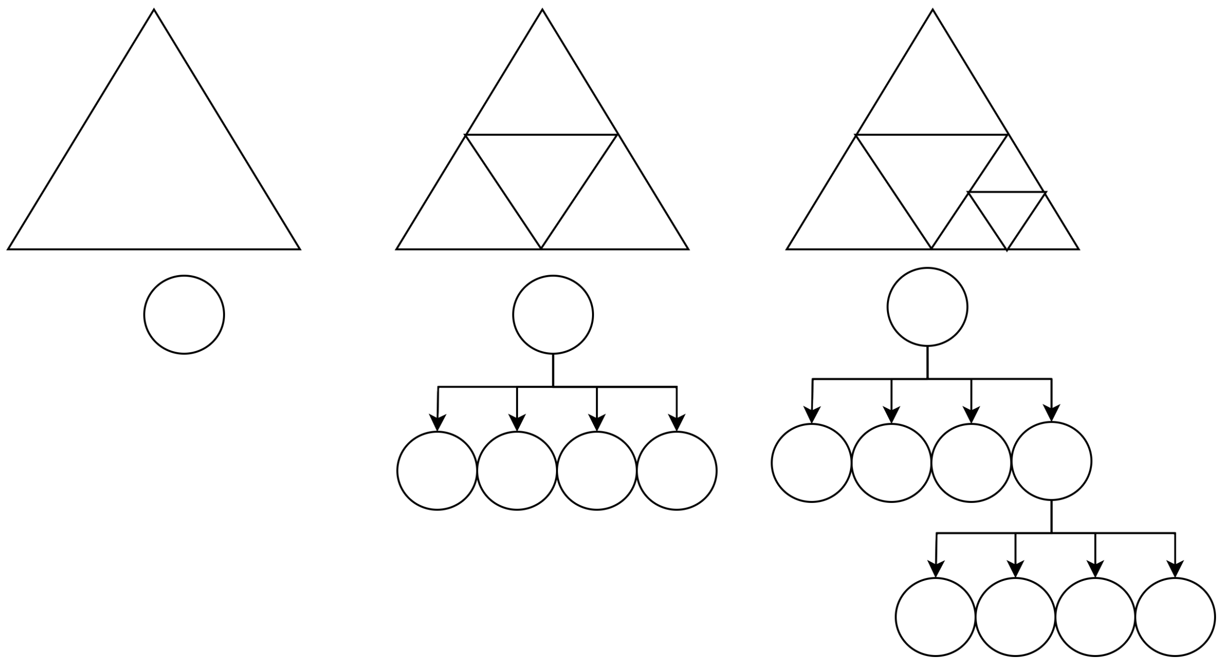
generalization of the binTree, evey time we split the tree in N equal children



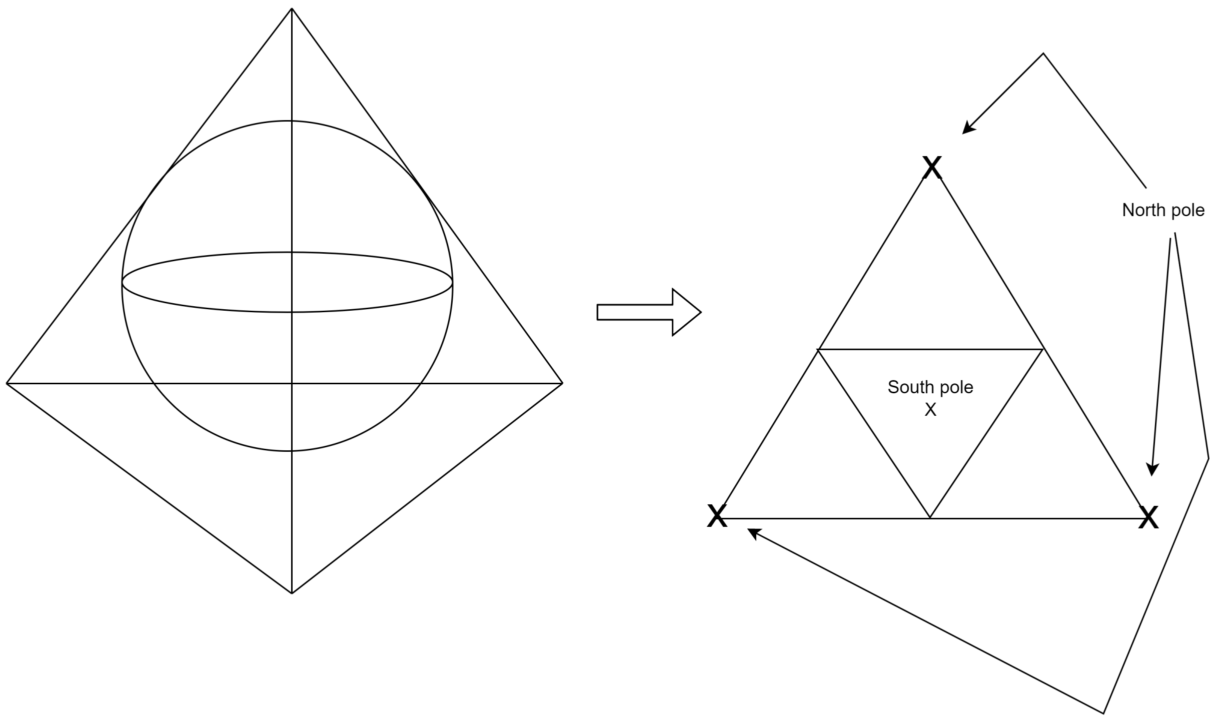
it could be considered as the missing ring between a quad tree and a full grid

Triangle quad tree

in this case we use a triangular domain and we subdivide it into triangle subnodes



it is really well suited to generate hierarchical partitioning of spheres in 3d, since we can inscribe a sphere in a tetrahedron, then project the sphere points on the tethraedron and we obtain a triangle quad tree and have the minimum distorsion possible



Storing of a QuadTree

The simplest way to store a quad tree is to use the bottom left vertex. we can store the coordinates of the point and its level, so for every node we will have: $v = (x, y, l), x, y \in [0, U]$, where U is the side lenght of the quare of the root

in this fashon we will need $2d + \log_2 d$ bits per each node

QuadTrees usages

🗓️ Image compression

🪆 Dynamic meshing

⚾ IsoSurfaces

Good resources

https://jimkang.com/quadtreevis/

https://www.geeksforgeeks.org/quad-tree/

https://graphics.stanford.edu/courses/cs468-06-fall/Slides/steve.pdf

https://i11www.iti.kit.edu/_media/teaching/winter2015/compgeom/algogeom-ws15-vl11-printable.pdf

https://www.jordansavant.com/book/algorithms/quadtree.md

https://iq.opengenus.org/quadtree/

https://personal.us.es/almar/cg/09quadtrees.pdf

https://ls11-www.cs.tu-dortmund.de/_media/buchin/teaching/akda_ws21/quadtrees.pdf