**3D R-tree**

Παραδοσιακά 2D R-trees χρησιμοποιούν rectangles για να δεικτοδοτήσουν σημεία. Σε 3D R-trees επομένως γίνεται χρήση κύβων δηλ. boxes.

Στα πλαίσια της εργασίας ένα δεικτοδοτούμενο σημείο της μορφής (x,y,t) αναπαριστά ένα σημείο στον τρισδιάστατο χώρο. Επίσης για να δεικτοδοτήσουμε ένα τέτοιο σημείο μέσα στο R-tree υποθέτουμε ότι είναι ένα box με μηδενικό όγκο στις συντεταγμένες x,y,t. Η υλοποίηση ενός σημείου δηλώνεται στο αρχείο point.py.

**class** Point:

**def** **\_\_init\_\_**(*self*, *x*: **float**, *y*: **float**, *t*: **float**) -> "Point":

**self**.x **=** x

**self**.y **=** y

**self**.t **=** t

# MBB = Minimum Bounding Box

**self**.mbb **=** Box(x, y, t, x, y, t)

**def** **\_\_str\_\_**(*self*) -> **str**:

**return** **f**"x:{**self**.x} y:{**self**.y} t:{**self**.t}"

Απαραίτητη είναι και η υλοποίηση των boxes η οποία βρίσκεται στο αρχέιο box.py

**class** Box:

**def** **\_\_init\_\_**(

*self*,

*x1*: **float**,

*y1*: **float**,

*z1*: **float**,

*x2*: **float**,

*y2*: **float**,

*z2*: **float**,

) -> "Box":

**self**.x1 **=** x1

**self**.y1 **=** y1

**self**.z1 **=** z1

**self**.x2 **=** x2

**self**.y2 **=** y2

**self**.z2 **=** z2

**def** **\_\_str\_\_**(*self*) -> **str**:

**return** **f**"x:({**self**.x1, **self**.x2} x:({**self**.y1, **self**.y2} z:({**self**.z1, **self**.z2}"

**def** getVolume(*self*) -> **float**:

**return** fabs((**self**.x2 **-** **self**.x1) **\*** (**self**.y2 **-** **self**.y1) **\*** (**self**.z2 **-** **self**.z1))

**def** getCoordinates(*self*):

**return** **self**.x1, **self**.y1, **self**.z1, **self**.x2, **self**.y2, **self**.z2

**def** overlaps(*self*, *box*: "Box") -> **bool**:

a, b **=** **self**, box

x1 **=** **max**(**min**(a.x1, a.x2), **min**(b.x1, b.x2))

y1 **=** **max**(**min**(a.y1, a.y2), **min**(b.y1, b.y2))

x2 **=** **min**(**max**(a.x1, a.x2), **max**(b.x1, b.x2))

y2 **=** **min**(**max**(a.y1, a.y2), **max**(b.y1, b.y2))

z1 **=** **max**(**min**(a.z1, a.z2), **min**(b.z1, b.z2))

z2 **=** **min**(**max**(a.z1, a.z2), **max**(b.z1, b.z2))

**return** x1 **<=** x2 **and** y1 **<=** y2 **and** z1 **<=** z2

-getVolume => επιστρέφει τον όγκο ενός box

-overlaps => επιστρέφει true ή false ανάλογα αν 2 box επικαλύπτονται ή εμπεριέχεται το ένα στο άλλο. Χρήση για την υποβολή ερωτημάτων στο R-tree

Κάθε κόμβος του δέντρου περιέχει μια λίστα από entries. Το entry περιέχει τα εξής δεδομένα: 1) Box => Δηλώνει τις συντεταγμένες του box στον χώρο που περιέχει ένα data point ή έναν ολόκληρο κόμβο.

2) Child => Δηλώνει αν το entry αυτο δείχνει σε έναν άλλο κόμβο. Κόμβοι που τα entries τους δείχνουν σε έναν άλλο κόμβο λέγονται ενδιάμεσοι κόμβοι (intermediate nodes).

3) Data => Δηλώνει αν το entry δείχνει σε ένα data point. Αυτοι οι κόμβοι λέγονται leaf nodes ή leaves και περιέχουν τα δεδομένα του δέντρου. Τα leaves δεν δείχνουν ποτέ σε κάποιον άλλο κόμβο και είναι όλα στο ίδιο επίπεδο.

Κάθε κόμβος περιέχει ένα minimum και ένα maximum number of entries.

Η υλοποίηση των entries και nodes περιέχονται στο αρχείο rtree.py

Entry:

**class** Entry:

"""

Represents the data of each node. If the node is a leaf it contains a trajectory instance (x,y,t) else a pointer to a child node.

"""

**def** **\_\_init\_\_**(*self*, *child*: "Node" **=** None, *data*: "Point" **=** None) -> "Entry":

**self**.child **=** child

**self**.data **=** data

**self**.box **=** **self**.getEntryBox()

**def** **\_\_str\_\_**(*self*) -> **str**:

**return** **f**"Entry Box: {**self**.box}"

**def** getEntryBox(*self*) -> "Box":

"""Sets the size of the minimum bounding box of the entry based on child node or data."""

**if** **self**.child **==** None:

*# box size = point mbb*

**return** **self**.data.mbb

**return** **self**.child.getMBB()

Node:

**class** Node:

**def** **\_\_init\_\_**(*self*, *parent*: "Node" **=** None, *entries*: "Entry" **=** []) -> "Node":

**self**.parent **=** parent

**self**.entries **=** entries

**self**.assignParent()

**def** isRoot(*self*) -> **bool**:

**return** **self**.parent **==** None

**def** isLeaf(*self*) -> **bool**:

**if** **self**.isRoot():

**return** **self**.entries **==** [] **or** **self**.entries[0].data **!=** None

**return** **self**.entries[0].data **!=** None

**def** isFull(*self*) -> **bool**:

**return** **len**(**self**.entries) **>=** MAX\_ENTRIES

**def** getMBB(*self*) -> "Box":

"""Returns the minimum bounding box for this node based on the entries."""

boxes **=** [entry.box **for** entry **in** **self**.entries]

**return** MBB(boxes)

**def** insertEntry(*self*, *entry*: "Entry") -> None:

**self**.entries.append(entry)

**self**.assignParent()

**def** parentEntry(*self*):

**if** **self**.parent **is** **not** None:

**return** **next**(entry **for** entry **in** **self**.parent.entries **if** entry.child **is** **self**)

**return** None

**def** assignParent(*self*) -> None:

"""Assign parent to child nodes"""

**if** **not** **self**.isLeaf():

**for** entry **in** **self**.entries:

entry.child.parent **=** **self**

**def** getLeastVolumeEnlargement(*self*, *new\_entry*: "Entry") -> "Node":

"""Returns the node that will be enlarged the least after the new entry insertion"""

enlargement\_list **=** []

**for** entry **in** **self**.entries:

volumeEnlargement **=** getVolumeEnlargement(new\_entry.box, entry.box)

enlargement\_list.append([entry.child, volumeEnlargement])

*# Get the Node that requires least enlargement to fit new entry.*

**return** **min**(enlargement\_list, *key***=lambda** *e*: e[1])[0]

Τέλος οι αλγόριθμοι για insert και query πανω στο R-tree υλοποιούνται όπως περιγράφονται στο paper :

R-TREES : A DYNAMIC INDEX STRUCTURE FOR SPATIAL SEARCHING

του A. Guttman, 1984

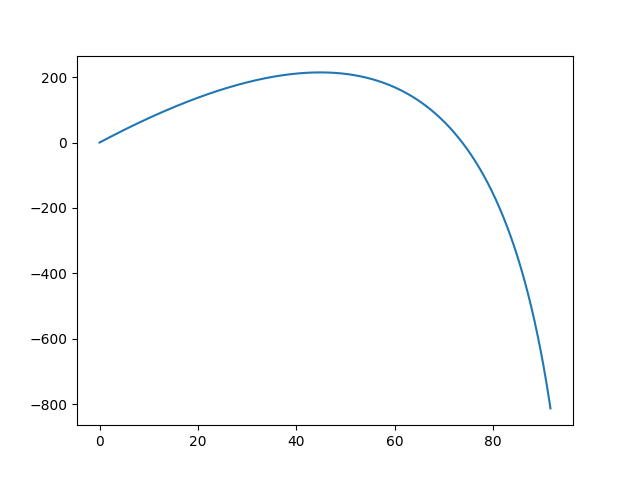
Insert:

Για την εισαγωγή νέων δεδομένων στο δέντρο ξεκινάμε από το root. Στην συνέχεια για κάθε child entry υπολογίζουμε πως θα επηρεάσει τον τελικό όγκο του box child μετα την εισαγωγή. Διαλέγουμε κάθε φορά τον κόμβο που θα έχει ως αποτέλεσμα την μικρότερη αύξηση σε όγκο, μέχρι να φτάσουμε σε ενα leaf node και να προσθέσουμε εκεί το data point. Σε περίπτωση ωστόσο που o leaf node έχει ήδη το μέγιστο αριθμό entries, πρέπει να χωρίσουμε τον κόμβο στα 2 δηλαδή να κάνουμε node split. Ιδανικά το node split θα δημιουργήσει 2 boxes τα οποία θα περιέχουν όλα τα δεδομένα με τον μικρότερο wasted όγκο. Υλοποιούμε τον αλγόριθμο quadratic node split, όπως περιγράφεται στο paper, που δεν εγγυάται το βέλτιστο αποτέλεσμα αλλα ένα πολύ καλό σε γρήγορο χρόνο. Μετά το node split προσαρμόζουμε και το υπόλοιπο δέντρο “προς τα πάνω” με τον ίδιο τρόπο. Οι συναρτήσεις για την υλοποίηση του insert υπάρχουν στο rtree.py.

Δεδομένα για το insert:

(Artificial Synthetic Dataset)

Για τα δεδομένα μας χρειαζόμαστε δεδομένα τροχιάς ενός αντικειμένου. Στο αρχείο data-gen.py γίνεται προσομοίωση ρίψης μίας σφαίρας στον αέρα με δειγματοληψία θέσης x,y και χρόνου t κάθε dt. Η προσομοίωση περιγράφει την εξής τροχιά:



και παράγει περίπου 2.500 data points της μορφής (x,y,t) και τα αποθηκεύει στο αρχείο trajectory-data.csv.

Bulk Insert:

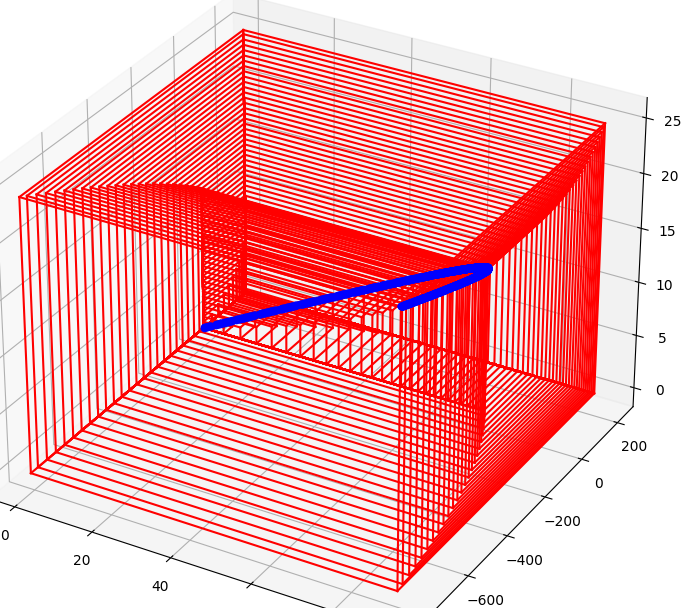
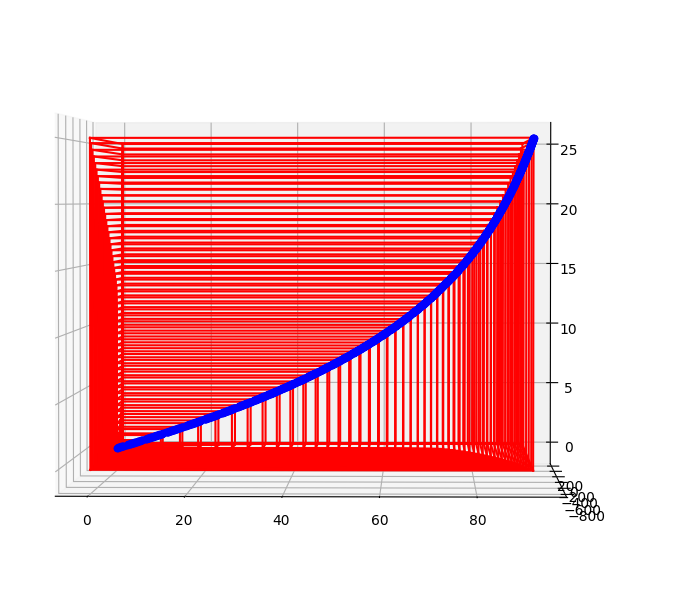
BulkInsert function στο demo.py τοποθετεί όλα τα δεδομένα στο 3D Rtree

σε περίπου 0.1 secs.

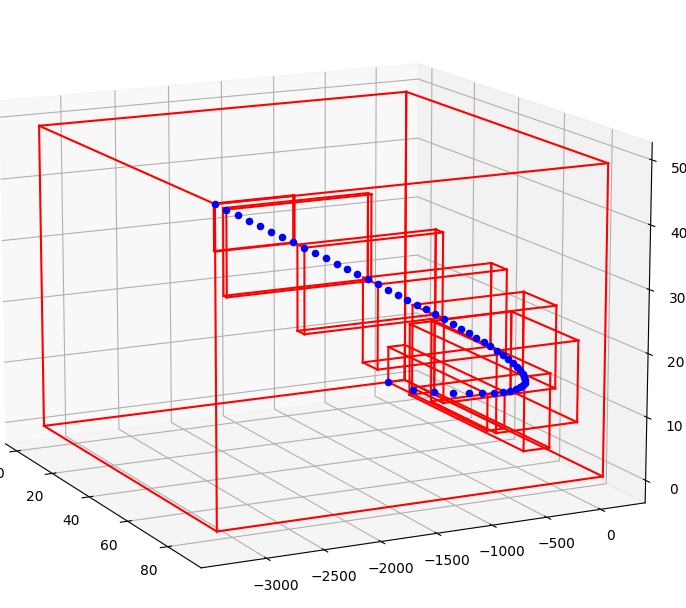


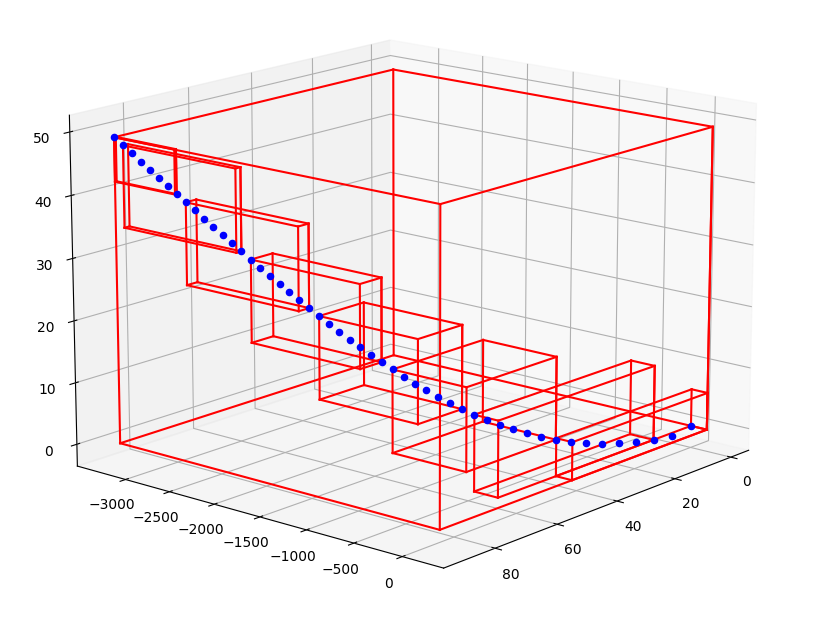
VisualizeRtree:

Συνάρτηση η οποία παρουσιάζει σε γραφικό κομμάτι όλα τα δεδομένα και παραγόμενα boxes του δέντρου. Μετα την εισαγωγή των δεδομένων το visualisation είναι το εξής:



Το οποίο είναι λογικό αποτέλεσμα καθώς η δειγματοληψία μας έχει πολύ μεγάλη ακρίβεια (μεταβλητή dt)

Για να φανεί καλύτερα η σωστή λειτουργία και ιδέα του 3D Rtree αν μειώσουμε την ακρίβεια της δειγματοληψίας (μεγαλύτερο dt) έχουμε το εξής αποτέλεσμα: 



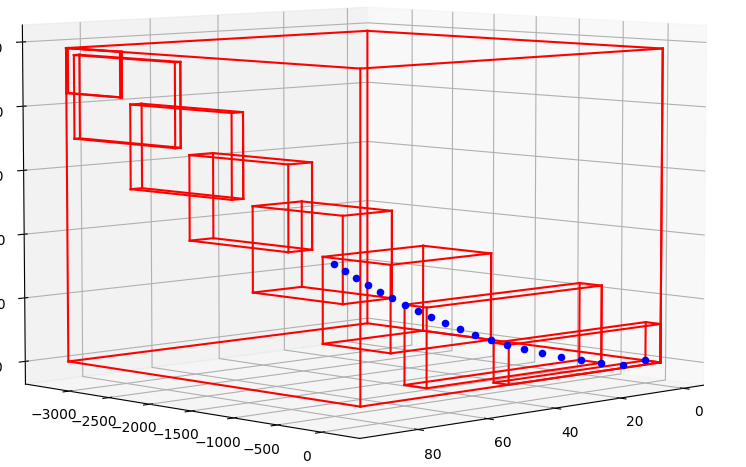
φαίνεται το nesting διαφορετικών μεγεθών boxes καθώς διαγράφεται ξεκάθαρα η τροχιά του αντικειμένου.

**Trajectory Queries:**

Ένα trajectory query μπορεί να έχει πολλές μορφές. Ερωτήματα με βάση την ταχύτητα ή την επιτάχυνση του αντικειμένου, βάση συντεταγμένων x,y ή συνδυασμό τους. Στην συγκεκριμένη υλοποίηση κάνουμε ένα trajectory query για ένα συγκεκριμένο timeframe, δηλαδή, επιστροφή των σημείων που ανήκουν στην τροχιά του αντικειμένου από ένα t0 εώς ενα t1 = to+Δt.

Ο τρόπος που γίνεται το query στο 3D R tree είναι δημιουργώντας ένα νέο box όπου η μία διάσταση που εκπροσωπεί τον χρόνο. Οι άλλες διαστάσεις εκπροσωπούν το x,y με x1=xmin, x2 =max και y αντίστοιχα (πληροφορίες που ήδη υπάρχουν στο root node συνεπώς δεν χρειάζεται κάποια αναζήτηση για να βρεθούν). Για την αναζήτηση βρίσκουμε ποια boxes και κατ’ επέκταση ποια datapoint ανήκουν στο box query καθώς τέμνονται σε αυτό. Τα υπόλοιπα είναι παράλληλα σε αυτό και δεν τα “ανακαλύπτουμε” ποτέ. Η μέθοδος αυτή μπορεί εύκολα να προσαρμοστεί για να καλύψει τις ανάγκες για τους άλλους τύπους trajectory queries που αναφέρθηκαν νωρίτερα.

Παραδείγματα για το sparse dataset:

Query timeframe 0-20:

Query timeframe 15-35: