

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ - 2023

# ΠΟΛΥΔΙΑΣΤΑΤΕΣ ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ

IOANNIS LOUDAROS - CHRISTINA KRATIMENOU - PARIS PSERGIANNIS - ALEXIS LEKARAKOS

# Γενικές Πληροφορίες



Για να έχετε πρόσβαση στην τελευταία έκδοση των απαντήσεων μπορείτε να σκανάρετε το παραπάνω QR Code ή να χρησιμοποιήσετε το παρακάτω κουμπί.

**Πατήστε Εδώ**

Στις επόμενες σελίδες αναλύονται οι απαντήσεις της ομάδας μας στο Project του μαθήματος “Πολυδιάστατες Δομές Δεδομένων”. Σε αυτή την σελίδα έχετε πρόσβαση σε γενικές πληροφορίες γύρω από το Project αλλά και γύρω από τα μέλη της ομάδας μας.

Για την υλοποίηση του Project εργαστήκαμε σε μια ομάδα 4 ατόμων. Χωρίσαμε τα ερωτήματα του πρότζεκτ ώστε ο φόρτος εργασίας να καταμεριστεί. Όλοι είχαμε επίγνωση των εργασιών που πραγματοποιούν οι συνεργάτες μας και δίναμε feedback ο ένας στον άλλον με αποτέλεσμα να προχωράμε συνεκτικά. Η ομάδα αποτελείται από τα εξής άτομα:

Αλέξης Λεκαράκος (1069367)

Ιωάννης Λουδάρος (1067400)

Χριστίνα Κρατημένου (1067495)

Πάρης Σεργιάννης (1067467)

Παρακάτω υπάρχουν αναλυτικότερες πληροφορίες για τα μέλη της ομάδας μας.



Χριστίνα Κρατημένου  
1067495

up1067495@upnet.gr  
Φοιτήτρια 5ου έτους



Πάρης Σεργιάννης  
1067467

iloudaros@upnet.gr  
Φοιτητής 5ου έτους



Ιωάννης Λουδάρος  
1067400

iloudaros@upnet.gr  
Φοιτητής 5ου έτους



Αλέξης Λεκαράκος  
1069367

st1069367@ceid.upatras.gr  
Φοιτητής 5ου έτους



# Περιεχόμενα

<b>Απαντήσεις .....</b>	<b>1</b>
3D R-trees for Spatio-Temporal Queries σε ΒΔ τροχιών στο επίπεδο	2
Interval trees και Segment trees	7
Convex Hull	8
Line Segment Intersection	9
<b>Παράρτημα Α .....</b>	<b>11</b>
Ορισμός του Point	11
Ορισμός του Entry	11
Ορισμός του Box	13
Ορισμός του Node	14
<b>Παράρτημα Β .....</b>	<b>15</b>
<b>Παράρτημα Γ .....</b>	<b>16</b>
<b>Παράρτημα Δ .....</b>	<b>17</b>

# Απαντήσεις

## Σύντομη παρουσίαση Πρότζεκτ

Για την διευκόλυνση σας, έχει προετοιμαστεί ένα αρχείο python που παρουσιάζει γρήγορα το κάθε ερώτημα.

Μέσα στον κατάλογο contributions, τρέξτε το αρχείο python test.py. Τρέχοντας αυτό το script, μπορείτε μέσω μιας απλής διεπαφής να περιηγηθείτε στα παραδείγματα που έχουμε προετοιμάσει για το κάθε ερώτημα.

Παρακάτω παραθέτουμε screenshot από την εν λόγω διεπαφή.



Περισσότερες πληροφορίες για το κάθε ερώτημα μπορείτε να βρείτε στις επόμενες σελίδες.

# **3D R-trees for Spatio-Temporal Queries σε ΒΔ τροχιών στο επίπεδο**

Μπορείτε να βρείτε τον κώδικα που αντιστοιχεί στο ερώτημα στο Παράρτημα Α.

## **Εισαγωγή**

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.

## **Boxes**

Απαραίτητη είναι και η υλοποίηση των boxes. Ο κώδικας βρίσκεται στο αρχείο box.py

- 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

## **Entries**

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 αυτό 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. Αυτοί 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

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

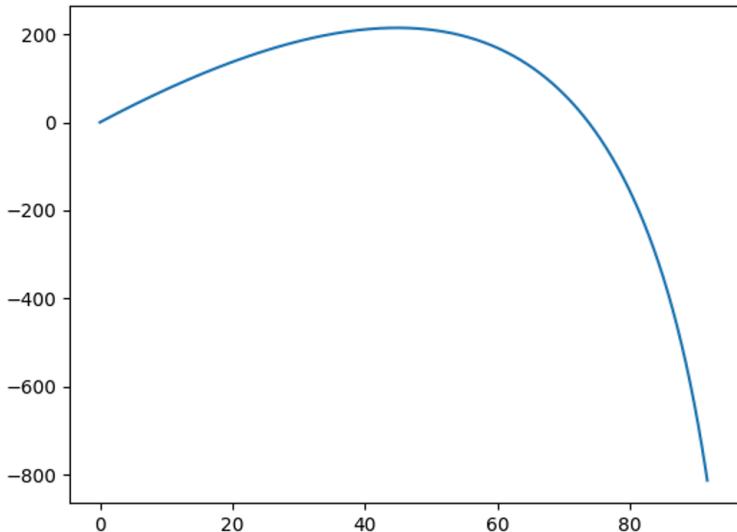
## Insert

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 ο 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 τροχιά που ακολουθεί:



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

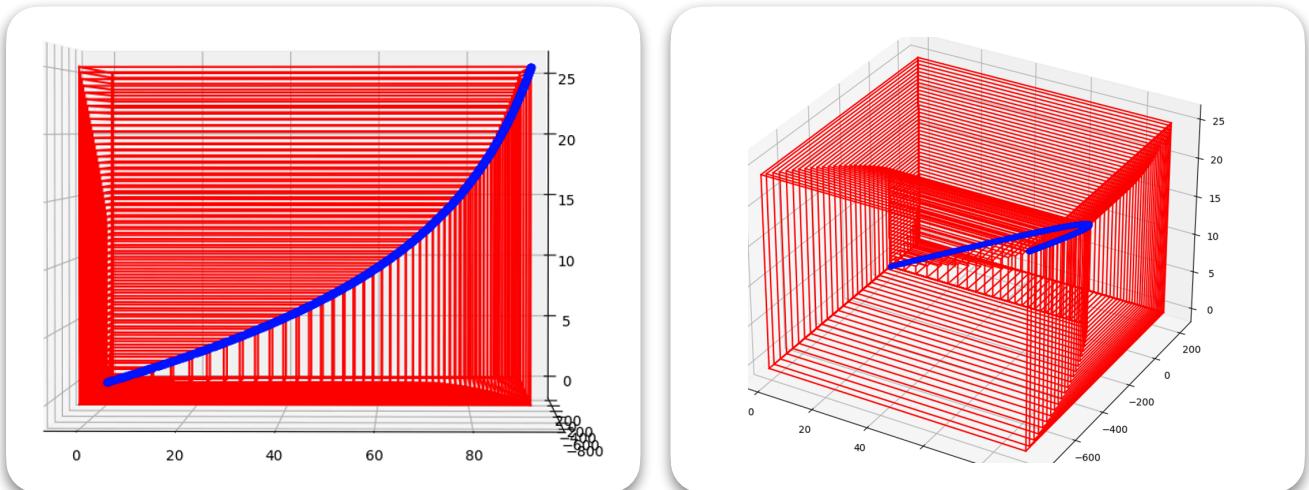
## Bulk Insert

BulkInsert function στο demo.py τοποθετεί όλα τα δεδομένα στο 3D Rtree σε περίπου 0.1 secs.

```
o → alekarakos git:(main) ✘ /usr/local/bin/python3 "/Users/iloudaros/Documents/Projects/My Projects/CEID/Multidimensional-DS-Project/Contributions/alekarakos/demo.py"  
Inserted all data points in 0.0011320114135742188 seconds.
```

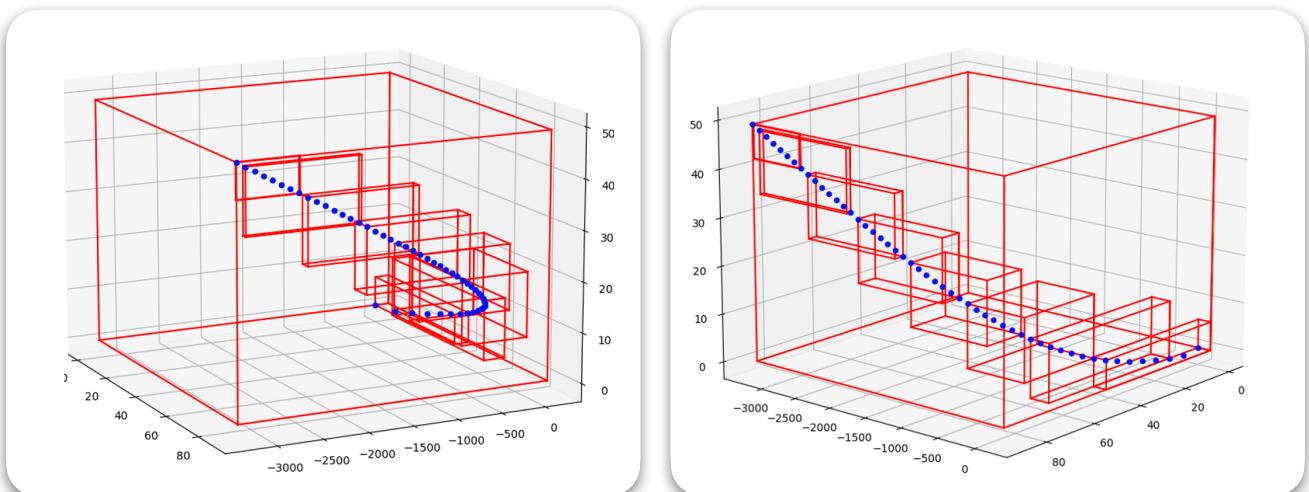
## VisualizeRtree

Συνάρτηση η οποία παρουσιάζει σε γραφικό κομμάτι όλα τα δεδομένα και παραγόμενα boxes του δέντρου. Μετά 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:



Μπορούμε να δούμε το nesting διαφορετικών μεγεθών boxes καθώς διαγράφεται ξεκάθαρα η τροχιά του αντικειμένου.

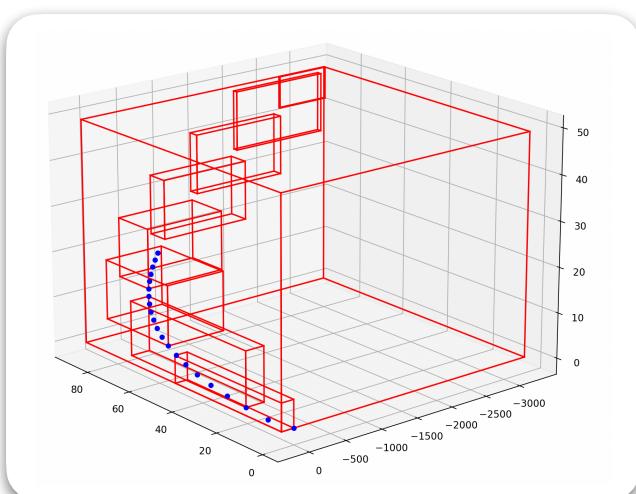
## Trajectory Queries

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  $t_0$  εώς ένα  $t_1 = t_0 + \Delta t$ .

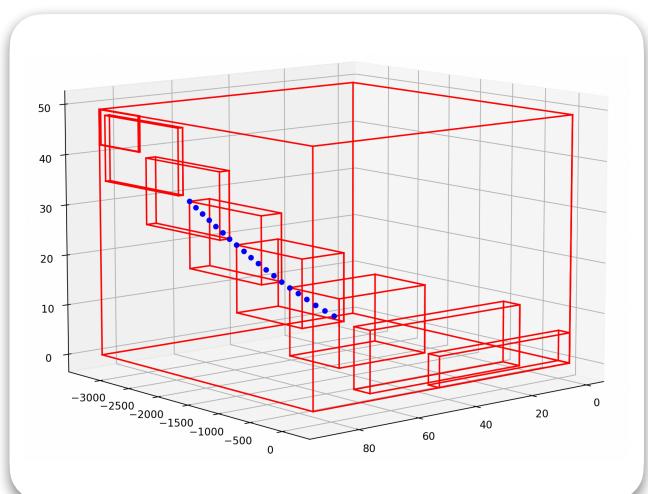
Ο τρόπος που γίνεται το query στο 3D R tree είναι δημιουργώντας ένα νέο box όπου η μία διάσταση που εκπροσωπεί τον χρόνο. Οι άλλες διαστάσεις εκπροσωπούν το x,y με  $x_1=x_{min}$ ,  $x_2=x_{max}$  και y αντίστοιχα (πληροφορίες που ήδη υπάρχουν στο root node συνεπώς δεν χρειάζεται κάποια αναζήτηση για 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



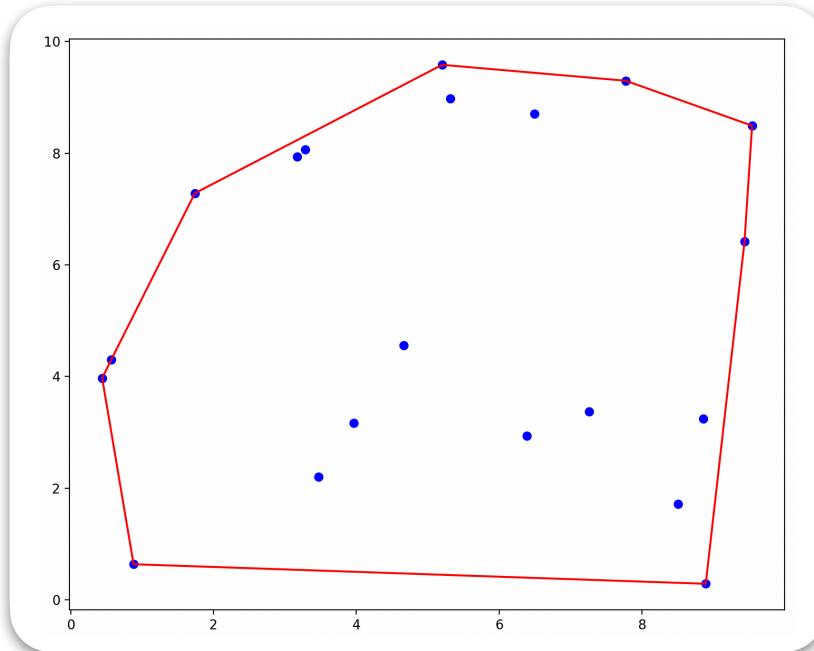
Query Timeframe 15-35



# **Interval trees και Segment trees**

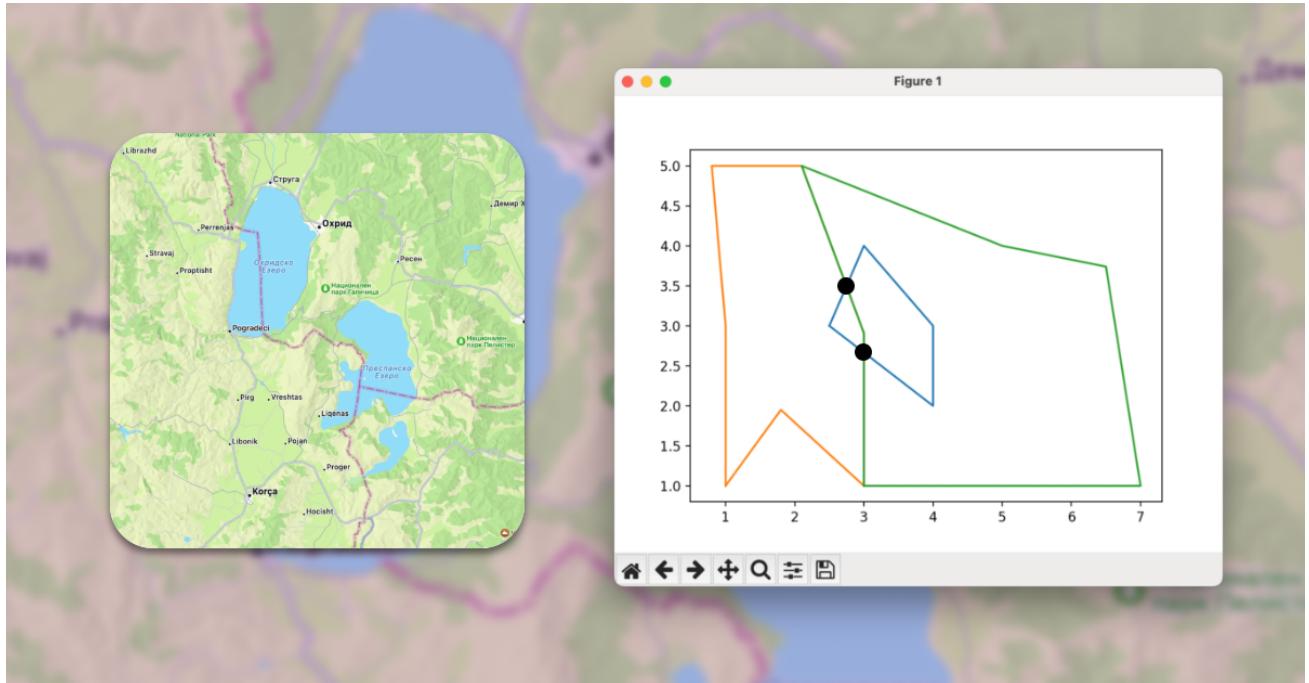
# Convex Hull

Το πρόγραμμα χρησιμοποιεί τον αλγόριθμο Graham scan για την υλοποίηση του convex hull. Η κύρια συνάρτηση του είναι η `convex_hull`, η οποία υπολογίζει το κυρτό περίβλημα. Ταξινομεί τα σημεία του επιπέδου λεξικογραφικώς, δηλαδή ξεκινάει με το αριστερότερο σημείο (το σημείο με την μικρότερη συντεταγμένη x) και στη συνέχεια προσθέτει σημεία στο περίβλημα αριστερόστροφα. Αυτό συμβαίνει ελέγχοντας το κάθε σημείο και κρατώντας μόνο αυτά που είναι αριστερά της γραμμής που σχηματίζεται από τα δύο τελευταία σημεία του περιβλήματος. Αν το σημείο υπό εξέταση βρίσκεται στα αριστερά, τότε απορρίπτεται καθώς δεν είναι μέρος του convex hull. Η διαδικασία επαναλαμβάνεται ώσπου να ελεγχθούν όλα τα σημεία. Για την κατασκευή του lower hull τα σημεία ελέγχονται με την σειρά που εμφανίζονται, ενώ για την κατασκευή του upper hull ελέγχονται με αντίθετη σειρά. Έπειτα, τα δύο κομμάτια ενώνονται και αφαιρείται το διπλότυπο σημείο και έτσι προκύπτει το τελικό περίβλημα.



Για την οτικοποίηση των αποτελεσμάτων γίνεται generate ένας αριθμός από τυχαία σημεία με την χρήση της βιβλιοθήκης random.

# Line Segment Intersection



Μπορείτε να βρείτε τον κώδικα που αντιστοιχεί στο ερώτημα στο [Παράρτημα Δ.](#)

Αυτό το ερώτημα εξερευνά αλγορίθμους εύρεσης τομών ευθυγράμμων τμημάτων με εφαρμογές στο επίπεδο που προκύπτουν από τα "σύνορα" πολυγωνικών περιοχών (όπως π.χ. σε περιπτώσεις υπερθέσεις χαρτών). Αναπτύξαμε δύο υλοποιήσεις αλγορίθμων. Αυτές είναι:

- Αφελής αλγόριθμος (`pol_interesect_naive()`)
- Αλγόριθμος βασισμένος σε γραμμή σάρωσης (`sweep_line()`)

Οι υλοποιήσεις των αλγορίθμων βρίσκονται στο αρχείο `line_segment_intersection.py`

## Δομές που Παράχθηκαν

Για τις ανάγκες των ερωτημάτων δημιουργήθηκαν οι δομές Line, Event και Polygon ώστε ο κώδικας να είναι ευανάγνωστος και να γίνεται αντιληπτή η έννοια που πραγματεύεται κάθε φορά.

## Τα δεδομένα και το Demo

Εμπνευστίκαμε το Demo από περιπτώσεις λιμνών ως φυσικών συνόρων μεταξύ χωρών. Έτσι λοιπόν, στο παράδειγμα μας έχουμε δύο πολύγωνα (country1 και country2) τα οποία μοιράζονται κάποιες πλευρές, και άλλο ένα πολύγωνο (lake). Τα δεδομένα μας είναι δημιουργημένα από εμάς για τις ανάγκες του ερωτήματος.

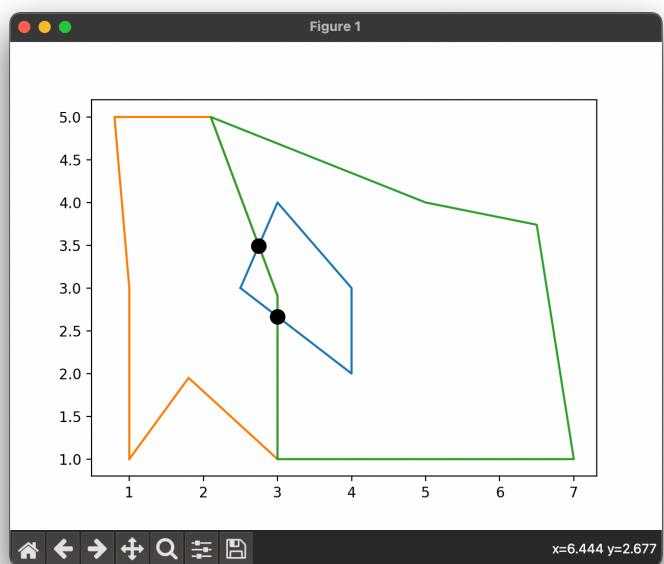
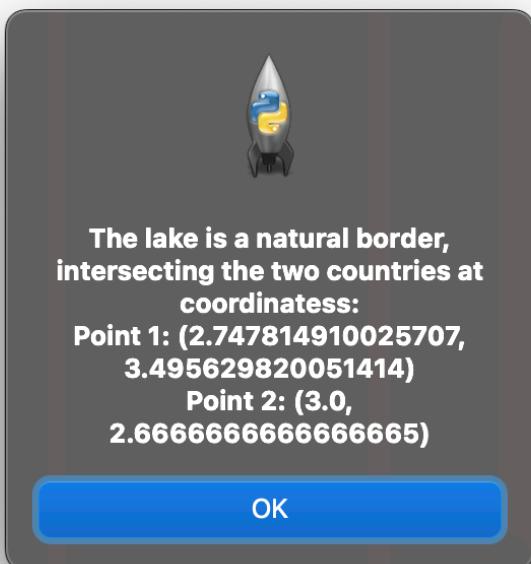
## Τρόπος Εκτέλεσης

Για να μπορέσετε να τρέξετε το demo θα χρειαστείτε τα πακέτα matplotlib και random.

Για να το εκτελέσετε αρκεί το : python lsi\_demo.py

## Αποτελέσματα

Ο κώδικας μας ελέγχει αν τα πολύγωνα τέμνονται. Ύστερα αν τέμνονται, μας ενημερώνει για τα σημεία και μας τα δείχνει γραφικά, όπως φαίνεται παρακάτω.



# Παράρτημα A

## Ορισμός του Point

```
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}"
```

## Ορισμός του 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()
```



# Ορισμός του Box

```
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
```

# Ορισμός του 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]
```

# Παράρτημα Β

# Παράρτημα Γ

# Παράρτημα Δ