

Εργασία Εξαμήνου: Υπολογιστική Γεωμετρία

Γ. Πρόκο

Π16119, Τμήμα Πληροφορικής
Πανεπιστήμιο Πειραιώς

Τελευταία Τροποποίηση:
30 Μαρτίου 2022

Περίληψη

Η παρούσα εργασία, αποτελεί υλοποίηση της υπολογιστικής εργασίας του μαθήματος **Υπολογιστική Γεωμετρία**, του 7ου εξαμήνου σπουδών του τμήματος Πληροφορικής, Πανεπιστήμιο Πειραιώς (ακαδημαϊκό έτος 2021 - 2022, υπό την επίβλεψη του Δρ. Κ. Μανέ).

Στην εργασία αυτή, καλούμαστε να χρησιμοποιήσουμε τη βιβλιοθήκη CGAL για τη δημιουργία προγράμματος εντοπισμού θέσης σημείου σε μια διαμέριση του επιπέδου A , η οποία ορίζεται από n ευθύγραμμα τμήματα, καθώς επίσης και την εύρεση του κυρτού περιβλήματος των σημείων της A . Η είσοδος του προγράμματος είναι ένα σύνολο Σ από n ευθύγραμμα τμήματα και ένα σύνολο P από k σημεία. Η έξοδος είναι η έδρα (ή ακμή, ή κορυφή) της παραγόμενης διαμέρισης στην οποία περιέχεται κάθε σημείο του P .

Η δομή του παρόντος, ακολουθεί την τμηματική επίλυση των ερωτημάτων που ζητούνται προς υλοποίηση της εργασίας, συγκεκριμένα:

- Εισαγωγή, σε μια αρχικά κενή διαμέριση A , n ευθύγραμμων τμημάτων του επιπέδου, τα οποία είτε α) παράγονται τυχαία ή β) διαβάζονται από αρχείο κειμένου text
- Εύρεση του κυρτού περιβλήματος των σημείων της A
- Εντοπισμός των k σημείων του P , δηλαδή εύρεση της έδρας της A στην οποία ανήκει κάθε δοσμένο σημείο, με κάθε έναν από τους 4 αλγόριθμους που αναφέρονται παρακάτω.
- Καταγραφή του χρόνου εκτέλεσης για κάθε ένα από τα βήματα 1, 2, 3, για διάφορες τιμές των n και k και για κάθε έναν από τους 4 αλγόριθμους αναφέρονται παρακάτω.
- Δυνατότητα αποθήκευσης του A σε αρχείο.
- Δυνατότητα φόρτωσης του A από αρχείο, για νέο εντοπισμό θέσης.

Οι κλάσεις της βιβλιοθήκης CGAL για τον εντοπισμό σημείου (που αντιστοιχούν σε διαφορετικές αλγοριθμικές τεχνικές) και θα μελετηθούν, είναι οι:

- `Arr_naive_point_location<Arrangement>`
- `Arr_walk_along_a_line_point_location<Arrangement>`
- `Arr_landmarks_point_location<Arrangement, Generator>`
- `Arr_trapezoid_ric_point_location<Arrangement>`

Η γλώσσα προγραμματισμού που θα χρησιμοποιηθεί είναι η C++. Η υλοποίηση όλων των ορισμένων συναρτήσεων που θα ορίσουμε για την επίλυση της εργασίας, θα παρατεθεί μέσω git συνδέσμων, στο τέλος του παρόντος.

Περιεχόμενα

1	Εισαγωγή, k σημείων και n ευθύγραμμων τμημάτων στο επίπεδο	3
1.1	Παραγωγή τυχαίων σημείων και ευθύγραμμων τμημάτων	3
1.1.1	GeneratePoints2DInstance(int minBound, int maxBound, int nrOfElements) . . .	3
1.1.2	GenerateLineSegments2DInstance(int minBound, int maxBound, int nrOfElements)	4
1.1.3	DisplayPoints(Vector_Point_2D vector, int precission)	4
1.1.4	DisplayLineSegments(Vector_Line_Segment_2D LineSegments, int precission) . . .	4
1.1.5	Στιγμιότυπο: Παραγωγή τυχαίων σημείων και ευθύγραμμων τμημάτων	4
1.2	Ανάγνωση στιγμιότυπου από αρχείου	4
1.2.1	ConvertSegmentsFromFile(Vector_Point_2D vector)	5
1.2.2	ReadPointsFromFile(String path)	5
1.2.3	ParseLineToPoint(String data)	5
1.2.4	Στιγμιότυπο: Ανάγνωση στιγμιότυπου από αρχεία	5
2	Εύρεση Κυρτού Περιβλήματος Σημείων της A	7
2.1	Απαραίτητες βιβλιοθήκες	8
2.2	Εγγραφή του αποτελέσματος σε αρχείο	8
3	Κατασκευή Διαμέρισης A	9
3.1	Απαραίτητες βιβλιοθήκες	9
3.2	ConstructArrangment(Vector_Line_Segment_2D segmentVector)	10
3.3	DisplayFacesOfArrangment(Arrangement_2D arr)	10
3.4	Στιγμιότυπο Διαμέρισης	10
4	Ερωτήματα Εντοπισμού Σημείου	12
4.1	Αφελής Εντοπισμός Σημείου	12
4.2	Εντοπισμός Σημείου Μέσω Ανάτρεξης Σε Ευθύγραμμο Τμήμα	13
4.3	Εντοπισμός Σημείου Μέσω Υποβοηθούμενων Σημείων	14
4.4	Εντοπισμός Σημείου Μέσω Τραπεζοειδούς Χάρτη	14
5	Μελέτη Χρονικής Απόδοσης	16
5.1	Παράθεση Πίνακα Αποτελεσμάτων	16
6	Αποθήκευση και Ανάκληση Διαμέρισης A	18
6.1	Αποθήκευση Διαμέρισης A	19
6.2	Ανάκληση της Διαμέρισης από το αρχείο	20
7	Υπερσύνδεση Αποθετηρίου Αναλυτικού Κώδικα	21

1 Εισαγωγή, k σημείων και n ευθύγραμμων τμημάτων στο επίπεδο

Το πρώτο βήμα το οποίο καλούμαστε να υλοποιήσουμε, είναι η εισαγωγή n ευθύγραμμων τμημάτων σε μια αρχικά κενή διαμέριση A του επιπέδου. Τα n ευθύγραμμα τμήματα τα οποία θα προστεθούν, θα καθορίσουν μια καινούρια διαμέριση του επιπέδου, A' , την οποία ωστόσο χωρίς βλάβη της γενικότητας θα την συμβολίζουμε με A :

$$A = A \cup A'$$

Η βιβλιοθήκη CGAL, εμπεριέχει κάποιες προκαθορισμένες δομές δεδομένων/αναπαραστάσεις για συγκεκριμένα γεωμετρικά σχήματα ή/και γεωμετρικές έννοιες, τις οποίες και καλούμαστε να χρησιμοποιήσουμε. Συγκεκριμένα, για να αναπαρασταθεί ένα σημείο στο επίπεδο, χρησιμοποιείται η τάξη: `Point_2` ενώ για να αναπαρασταθεί ένα ευθύγραμμο τμήμα¹, χρησιμοποιείται η τάξη: `Segment_2`.

Αρχικά θα εξετάσουμε πως μπορούμε να παράγουμε ένα σύνολο τυχαίων ευθύγραμμων σημείων, καθώς και με ποιον τρόπο μπορούμε να διαβάσουμε ένα σύνολο ευθύγραμμων τμημάτων απο αρχείο.

1.1 Παραγωγή τυχαίων σημείων και ευθύγραμμων τμημάτων

Για την παραγωγή τυχαίων σημείων, καλούμαστε να κάνουμε χρήση των κάτωθι βιβλιοθηκών:

- `#include <vector>`: Δομή δεδομένων η οποία συνιστά συνεχόμενη διάταξη, της οποίας το μέγεθος δύναται να μεταβληθεί.
- `#include <random>`: Βιβλιοθήκη η οποία εμπεριέχει συναρτήσεις παραγωγής ψευδο-τυχαίων αριθμών.
- `#include <iomanip>`: Βιβλιοθήκη η οποία χρησιμοποιείται για την δομή παρουσίασης αριθμών υψηλής ακρίβειας.
- `#include <CGAL/Exact_predicates_exact_constructions_kernel.h>`: Βιβλιοθήκη (πυρήνας), η οποία εμπεριέχει δομές δεδομένων και ορισμούς γεωμετρικών οντοτήτων. Συγκεκριμένα, χρησιμοποιεί καρτεσιανή αναπαράσταση, δίνει την δυνατότητα κατασκευής σημείων δοθέντων καρτεσιανών συντεταγμένων $\in R^n$, και τέλος εμπεριέχει ακριβείς ορισμούς γεωμετρικών οντοτήτων.

Στην συνέχεια, θα ορίσουμε μερικές ονομαστικές συμβάσεις που κάνουμε, προκειμένου ο πηγαίος κώδικας μας να είναι ποιο ευανάγνωστος:

```
typedef CGAL::Exact_predicates_exact_constructions_kernel Kernel;  
typedef Kernel::Point_2 Point_2D;  
typedef Kernel::Segment_2 Line_Segment_2D;  
typedef std::vector<Point_2D> Vector_Point_2D;  
typedef std::vector<Line_Segment_2D> Vector_Line_Segment_2D;
```

1.1.1 `GeneratePoints2DInstance(int minBound, int maxBound, int nrOfElements)`

Η συνάρτηση αυτή είναι υπεύθυνη για την παραγωγή ενός τυχαίου συνόλου σημείων $\in R^2$ κάνοντας χρήση της ορθογώνιας (ομοιόμορφης) κατανομής. Λαμβάνει ως είσοδο τις παραμέτρους κάτω όριο, άνω όριο καθώς και τον επιθυμητό αριθμό σημείων που θέλουμε να εξαγάγουμε, και επιστρέφει διάνυσμα πλήθους επιθυμητών σημείων $\in R^2 \cap [a, b]$.

¹Με την έννοια ευθύγραμμο τμήμα, αναφερόμαστε σε μη - άπειρη ευθεία η οποία καθορίζεται μοναδικά απο δύο διακριτά σημεία στο επίπεδο: $\overline{pq} : p, q \in R^2, [p, q]$

1.1.2 GenerateLineSegments2DInstance(int minBound, int maxBound, int nrOfElements)

Η συνάρτηση αυτή είναι υπεύθυνη για την παραγωγή ενός τυχαίου συνόλου ευθύγραμμων τμημάτων $\overline{p, q}$, κάνοντας χρήση της ορθογώνιας (ομοιόμορφης κατανομής). Λαμβάνει ως είσοδο τις παραμέτρους κάτω όριο σημείων, άνω όριο σημείων, καθώς και τον επιθυμητό αριθμό ευθύγραμμων τμημάτων, και επιστρέφει διάνυσμα πλήθους επιθυμητών ευθύγραμμων τμημάτων $\in ((R^2 \times R^2) \cap ([a, b] \times [a, b]))$.

1.1.3 DisplayPoints(Vector_Point_2D vector, int precision)

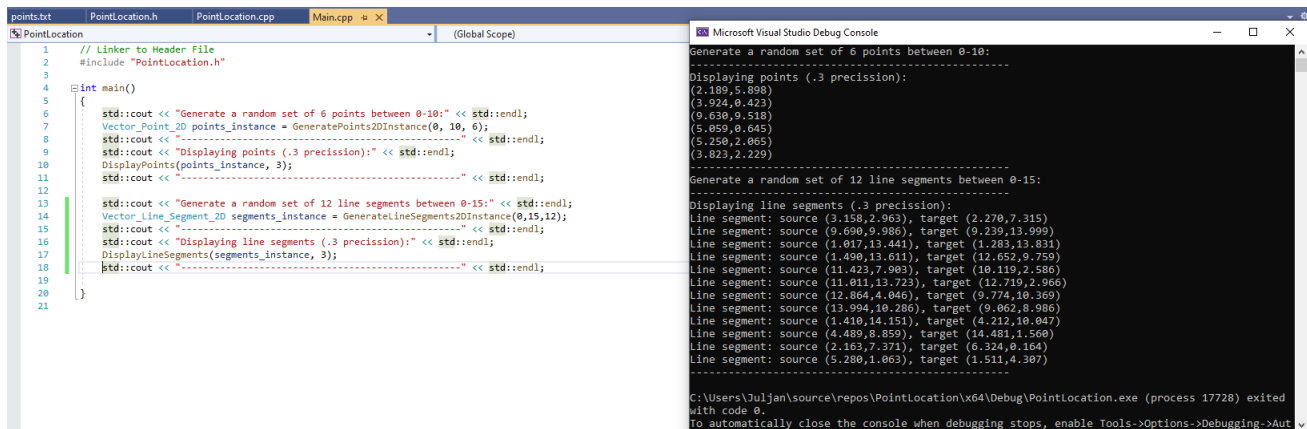
Η συνάρτηση αυτή είναι υπεύθυνη για την εμφάνιση στην οθόνη, μιας πλήρους λίστας δοσμένων σημείων στον R^2 με την εκάστοτε επιθυμητή ακρίβεια εμφάνισης.

1.1.4 DisplayLineSegments(Vector_Line_Segment_2D LineSegments, int precision)

Η συνάρτηση αυτή είναι υπεύθυνη για την εμφάνιση στην οθόνη, μιας πλήρους λίστας των δοσμένων ευθύγραμμων τμημάτων, συμπεριλαμβανομένου του σημείου έναρξης καθώς και του σημείου απόληξης του εκάστοτε ευθύγραμμου τμήματος, στην οθόνη. Τα σημεία αυτά εμφανίζονται με βάση την δοθείσα επιθυμητή ακρίβεια.

1.1.5 Στιγμιότυπο: Παραγωγή τυχαίων σημείων και ευθύγραμμων τμημάτων

Στην συνέχεια, θα μελετήσουμε ένα στιγμιότυπο εκτέλεσης των άνωθεν συναρτήσεων, κατά το οποίο παράγονται τυχαία κάποια σημεία στον χώρο, καθώς και κάποια ευθύγραμμα τμήματα, και εν συνεχεία εμφανίζονται στην οθόνη.



```
1 // Linker to Header File
2 #include "PointLocation.h"
3
4 int main()
5 {
6     std::cout << "Generate a random set of 6 points between 0-10:" << std::endl;
7     Vector_Point_2D points_instance = GeneratePoints2DInstance(0, 10, 6);
8     std::cout << "Displaying points (.3 precision):" << std::endl;
9     std::cout << "Displaying points (.3 precision):" << std::endl;
10    DisplayPoints(points_instance, 3);
11    std::cout << "-----" << std::endl;
12
13    std::cout << "Generate a random set of 12 line segments between 0-15:" << std::endl;
14    Vector_Line_Segment_2D segments_instance = GenerateLineSegments2DInstance(0, 15, 12);
15    std::cout << "Displaying line segments (.3 precision):" << std::endl;
16    std::cout << "Displaying line segments (.3 precision):" << std::endl;
17    DisplayLineSegments(segments_instance, 3);
18    std::cout << "-----" << std::endl;
19
20 }
21
```

```
Generate a random set of 6 points between 0-10:
-----
Displaying points (.3 precision):
(2.189,5.898)
(3.924,0.423)
(9.630,9.518)
(5.059,0.645)
(5.250,2.065)
(3.023,2.229)
-----
Generate a random set of 12 line segments between 0-15:
-----
Displaying line segments (.3 precision):
Line segment: source (0.158,2.963), target (2.270,7.315)
Line segment: source (0.690,9.986), target (9.239,13.999)
Line segment: source (1.017,13.441), target (1.283,13.831)
Line segment: source (1.490,13.611), target (12.652,9.759)
Line segment: source (11.423,7.983), target (10.119,2.586)
Line segment: source (11.011,13.723), target (12.719,2.966)
Line segment: source (12.864,4.046), target (9.774,10.369)
Line segment: source (13.994,10.286), target (9.062,0.986)
Line segment: source (1.410,14.151), target (4.212,10.047)
Line segment: source (4.489,8.859), target (14.481,1.500)
Line segment: source (2.163,7.371), target (6.324,0.164)
Line segment: source (5.280,1.063), target (1.511,4.307)
-----
C:\Users\Juljan\source\repos\PointLocation\Debug\PointLocation.exe (process 17728) exited
with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Aut
```

1.2 Ανάγνωση στιγμιότυπου απο αρχείου

Στην συνέχεια, θα μελετήσουμε και καθορίσουμε με ποιόν τρόπο μπορούμε να διαβάσουμε δοσμένα στιγμιότυπα απο αρχείο κειμένου .txt , προκειμένου να τα χρησιμοποιήσουμε στο πρόγραμμα μας. Για τον σκοπό αυτόν, χρησιμοποιούμε τις εξής βιβλιοθήκες:

- `#include <fstream>`: Βιβλιοθήκη υπεύθυνη για την ροή εισόδου και εξόδου δεδομένων μεταξύ αρχείων και του προγράμματος μας.
- `#include <string>`: Βιβλιοθήκη υπεύθυνη για την υποστήριξη ενεργειών σε συμβολοσειρές.²

Μια επιπρόσθετη ονομαστική σύμβαση που κάνουμε είναι η εξής:

```
typedef std::basic_string<char> String;
```

²Με την έννοια συμβολοσειρά, αναφερόμαστε σε μια δοθείσα ακολουθία χαρακτήρων.

1.2.1 ConvertSegmentsFromFile(Vector_Point_2D vector)

Η συνάρτηση αυτή είναι υπεύθυνη για την αντιστοίχιση σημείων σε ευθύγραμμα τμήματα. Αναλυτικότερα, λαμβάνει ως είσοδο ένα διάνυσμα $2n$ σημείων $\in R^2$ και επιστρέφει ένα διάνυσμα n ευθύγραμμων τμημάτων, το οποίο αντιστοιχεί στα σημεία. Συγκεκριμένα, τα ευθύγραμμα τμήματα αντιστοιχεί σε:

$$\forall i \in [n], i+ = 2, v_i, v_{i+1} \{v_i, v_{i+1} \in vector\}, return : \overline{v_i, v_{i+1}}$$

1.2.2 ReadPointsFromFile(String path)

Η συνάρτηση αυτή είναι υπεύθυνη για την ανάγνωση σημείων απο αρχείο .txt τα οποία εμπεριέχει σημεία. Η δομή του αρχείου είναι 1 σημείο ανά κάθε γραμμή. Η κάθε γραμμή αποτελείται εγγραφή της μορφής:

$x_1.x_2x_3...x_l, y_1.x_2y_3...y_m$

Εν συνεχεία, η συνάρτηση κάνει χρήση μιας άλλης συνάρτησης που έχει καθοριστεί απο εμάς (θα την αναφέρουμε στην επόμενη υποενότητα), και μετατρέπει μια δοθείσα εγγραφή σημείου αρχείου, σε σημείο Point_2D.

Τέλος η συνάρτηση επιστρέφει διάνυσμα k σημείων τα οποία αντιστοιχούν στα $\in R^2$ σημεία του αρχείου.

1.2.3 ParseLineToPoint(String data)

Η συνάρτηση αυτή είναι υπεύθυνη για την μετατροπή μιας δοθείσας εγγραφής αρχείου σε σημείο Point_2D . Αναλυτικότερα, αναγιγνώσκει την ροή των χαρακτήρων του αρχείου, δημιουργεί κατά την σάρωση τις απαραίτητες μεταβλητές x, y , μετατρέπει τις συμβολοσειρές σε πραγματικούς αριθμούς μέσω της:

`std::stod`

και τέλος επιστρέφει το σημείο που αντιστοιχεί.

1.2.4 Στιγμιότυπο: Ανάγνωση στιγμιότυπου απο αρχεία

Σε αυτήν την υποενότητα, θα μελετήσουμε ένα στιγμιότυπο το οποίο θα χρησιμεύσει και ως στιγμιότυπο αναφοράς στην συνέχεια της εργασίας. Αναλυτικότερα, όπως είδαμε και απο τις προηγουμένως καθορισμένες συναρτήσεις, μπορούμε να αναγνώσουμε σημεία απο ένα δοθέν αρχείο. Δοθέντος ότι μπορούμε να αναπαραστήσουμε και ευθύγραμμα τμήματα ως μια πλειάδα δύο σημείων (σημείο έναρξης και σημείο απόληξης), μπορούμε να αναγνώσουμε ευθύγραμμα τμήματα απο αρχείο.

Έστω αρχείο points.txt με τις εξής εγγραφές:

segments.txt	points.txt	PointLocation.h	PointLocation.cpp	Main.cpp
1	2.189,5.898			
2	3.924,0.423			
3	9.630,9.518			
4	5.059,0.645			
5	5.250,2.065			
6	3.823,2.229			

Καθώς και αρχείο segments.txt με τις εξής εγγραφές:

2.1 Απαραίτητες βιβλιοθήκες

Για τον υπολογισμό του κυρτού περιβλήματος βάση τον αλγόριθμο Graham - Andrews , χρησιμοποιούμε την εξής προσθήκη βιβλιοθήκης:

```
#include <CGAL/ch_graham_andrew.h>
```

Η βιβλιοθήκη αυτή, η οποία αποτελεί μέρος του πυρήνα: 2D Convex Hulls and Extreme Points έχει υλοποιημένο τον επιθυμητό αλγόριθμο. Ως αποτέλεσμα, επιστρέφονται τα σημεία που συστήνουν το κυρτό περίβλημα σε - αντιστροφή ωρολογιακή φορά - διάταξη.

2.2 Εγγραφή του αποτελέσματος σε αρχείο

Για σκοπούς που θα επεξηγηθούν παρακάτω, έχουμε υλοποιήσει μια επιπρόσθετη συνάρτηση, η οποία επιτελεί την εγγραφή του αποτελέσματος του αλγορίθμου εύρεσης του κυρτού περιβλήματος σε αρχείο convexHull.txt. Αναλυτικότερα, όπως διαφέρεται και από την εικόνα, η δομή εγγραφής ακολουθεί την δομή αποθήκευσης των ευθύγραμμων τμημάτων.

convexHull.txt	segments.txt	points.txt	PointLocation.h	PointLocation.cpp	Main.cpp
1	1.017,13.441				
2	1.511,4.307				
3	1.511,4.307				
4	3.924,0.423				
5	3.924,0.423				
6	6.324,0.164				
7	6.324,0.164				
8	14.481,1.56				
9	14.481,1.56				
10	13.994,10.286				
11	13.994,10.286				
12	11.011,13.723				
13	11.011,13.723				
14	9.239,13.999				
15	9.239,13.999				
16	1.41,14.151				
17	1.41,14.151				
18	1.017,13.441				

3 Κατασκευή Διαμέρισης A

Στην παρούσα ενότητα, καλούμαστε να κατασκευάσουμε μια διαμέριση του επιπέδου A, με βάση τα ευθύγραμμα τμήματα καθώς και τα σημεία που έχουμε διαβάσει μέσω των αρχείων.

Ορισμός: Ορίζουμε μια διαμέριση του επιπέδου ένα σύνολο επίπεδων υποδιαίρεσεων οι οποίες ορίζονται από επίπεδες εναποτυπώσεις γραφημάτων. Μια τέτοια διαμέριση θα την ονομάζουμε συνεκτική, αν το αντίστοιχο γράφημα είναι συνεκτικό. Το εναποτύπωμα ενός κόμβου του γραφήματος, λέγεται κορυφή. Το εναποτύπωμα ενός τόξου του γραφήματος, λέγεται ακμή. Έδρα της υποδιαίρεσης, είναι κάθε μεγιστιαίο συνεκτικό υποσύνολο του επιπέδου, το οποίο δεν έχει κοινά σημεία με καμία ακμή και κορυφή.

Για να αναπαραστήσουμε μια διαμέριση, χρησιμοποιούμε μια δομή δεδομένων με την ονομασία: Διπλοσυνδεδεμένος κατάλογος ακμών.

Ορισμός: Ορίζουμε έναν διπλοσυνδεδεμένο κατάλογο ακμών, ως δομή δεδομένων η οποία εμπεριέχει ένα δελτίο (μια εγγραφή), για κάθε έδρα, ακμή (στην πραγματικότητα σε έναν διπλοσυνδεδεμένο κατάλογο ακμών μια ακμή αντιπροσωπεύεται ως 2 ημιακμές) ή κορυφή της διαμέρισης που αναπαριστά. Τα δελτία αυτά, περιέχουν τις παρακάτω γεωμετρικές και τοπολογικές πληροφορίες:

- Το δελτίο για μια κορυφή v , περιέχει τις συντεταγμένες τις v . Περιέχει επίσης έναν δείκτη προς μια οποιαδήποτε ημιακμή έχει ως αφετηρία την v .
- Το δελτίο για μια έδρα f , περιέχει έναν δείκτη προς κάποια ημιακμή του εξωτερικού της συνόρου. Για μια μη - φραγμένη έδρα, η τιμή αυτού του δείκτη είναι κενή. Το δελτίο, περιέχει επίσης έναν κατάλογο δεικτών για κάθε οπή της έδρας. Ο κατάλογος αυτός περιέχει έναν δείκτη προς κάποια ημιακμή του συνόρου της οπής.
- Το δελτίο για μια ημιακμή \vec{e} , περιέχει έναν δείκτη προς την κορυφή αφετηρίας του, έναν δείκτη προς την δίδυμη ημιακμή, και έναν δείκτη προς την προσπίπτουσα έδρα που οριοθετεί. Κατά σύμβαση, επιλέγουμε μια ημιακμή να οριοθετεί την έδρα που προσπίπτει στα αριστερά της. Το δελτίο εμπεριέχει επιπρόσθετα και δύο δείκτες προς την επόμενη και την προηγούμενη, ημιακμή του συνόρου της έδρας που οριοθετεί.

3.1 Απαραίτητες βιβλιοθήκες

Για την κατασκευή μιας διαμέρισης A, (η οποία στην πραγματικότητα αναπαρίσταται από την δομή δεδομένων της διπλοσυνδεδεμένης λίστας ακμών), χρειαζόμαστε τις εξής βιβλιοθήκες:

```
#include <CGAL/Arr_segment_traits_2.h>
#include <CGAL/Arrangement_2.h>
```

Μέσω της πρώτης βιβλιοθήκης, εισαγάγουμε τα απαραίτητα συστατικά, οντότητες και συσχετίσεις που θα χρειαστεί η δεύτερη βιβλιοθήκη, η οποία στην πραγματικότητα εμπεριέχει την λειτουργικότητα για να δημιουργήσει και να χρησιμοποιήσει την δομή δεδομένων της διπλοσυνδεδεμένης λίστας ακμών.

Επιπρόσθετα, ορίζουμε τις κάτωθι ονομαστικές συμβάσεις:

```
typedef CGAL::Arr_segment_traits_2<Kernel> Arrangement_Traits_2D;
typedef CGAL::Arrangement_2<Arrangement_Traits_2D> Arrangement_2D;
typedef Arrangement_2D::Face_handle Face_handle;
```

```
typedef Arrangement_2D::Ccb_halfedge_const_circulator HalfEdge_circulator;
typedef Arrangement_2D::Face_iterator Face_iterator;
```

3.2 ConstructArrangement(Vector_Line_Segment_2D segmentVector)

Η συνάρτηση αυτή είναι υπεύθυνη για την κατασκευή μιας διαμέρισης επιπέδου, με βάση μια συλλογή απο ευθύγραμμα τμήματα. Κατά την κλήση της συνάρτησης και την δημιουργία της διαμέρισης, στην οθόνη εμφανίζεται κατάλληλο μήνυμα το οποίο αναφέρει το μέγεθος της διαμέρισης: δηλαδή το πλήθος των κορυφών, το πλήθος των ημιακμών καθώς και το πλήθος των εδρών.

3.3 DisplayFacesOfArrangment(Arrangement_2D arr)

Η συνάρτηση αυτή, είναι υπεύθυνη για την εμφάνιση των εδρών της διαμέρισης στην οθόνη. Αναλυτικότερα, για κάθε έδρα της διαμέρισης, εμφανίζεται ο κατάλογος με τις ημιακμές που την ορίζουν, δηλαδή κατάλογος με τις ημιακμές στις οποίες η έδρα προσπίπτει απο την αριστερή τους μεριά.

3.4 Στιγμιότυπο Διαμέρισης

Στην παρούσα υποενοότητα, θα παραθέσουμε μέσω στιγμιοτύπων, των προαναφερθεισών συναρτήσεων, καθώς και οπτική αναπαράσταση των αποτελεσμάτων(Σημειώνεται ότι στο πλήθος των ευθύγραμμων τμημάτων που καθορίζουν την διαμέριση, έχουν προστεθεί και τα ευθύγραμμα τμήματα τα οποία καθορίζουν το κυρτό περίβλημα της διαμέρισης, με σκοπό τον εμπλουτισμό του στιγμιοτύπου):

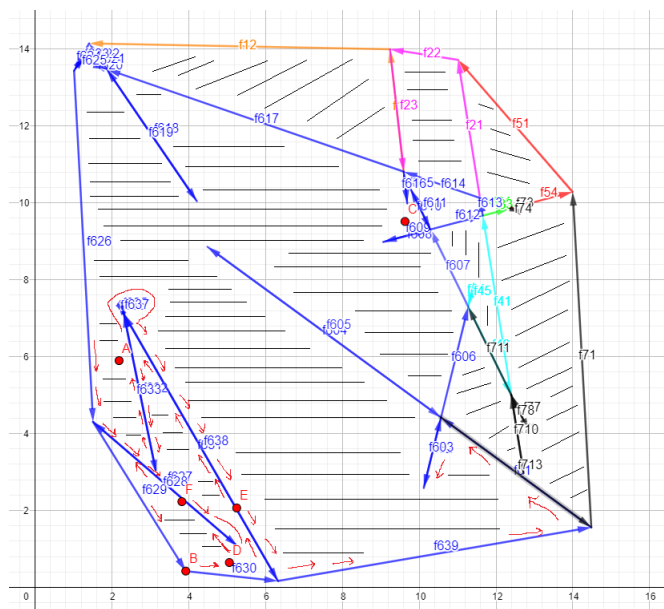
```
convex_hull.cpp  segments.txt  points.txt  PointLocation.h  PointLocation.cpp  Main.cpp  -  Microsoft Visual Studio Debug Console
(Global Scope)
6 {
7   std::cout << "Reading points from file 'points.txt':" << std::endl;
8   Vector_Point_2D file_points = ReadPointsFromFile("points.txt");
9   std::cout << "-----" << std::endl;
10  std::cout << "Displaying points (.3 precision):" << std::endl;
11  DisplayPoints(file_points, 3);
12  std::cout << "-----" << std::endl;
13
14  std::cout << "Reading line segment points from 'segments.txt':" << std::endl;
15  Vector_Point_2D file_segment_points = ReadPointsFromFile("segments.txt");
16  std::cout << "-----" << std::endl;
17  std::cout << "Converting segment points to segments:" << std::endl;
18  Vector_Line_Segment_2D file_line_segments = ConvertSegmentsFromFile(file_segment_points);
19  std::cout << "-----" << std::endl;
20  std::cout << "Displaying line segments (.3 precision):" << std::endl;
21  DisplayLineSegments(file_line_segments, 3);
22  std::cout << "-----" << std::endl;
23
24  std::cout << "Merge all points of A:" << std::endl;
25  Vector_Point_2D points_of_A = file_points;
26  points_of_A.insert(points_of_A.end(), file_segment_points.begin(), file_segment_points.end());
27  std::cout << "-----" << std::endl;
28  std::cout << "Calculating convex hull via Graham Andrew Algorithm:" << std::endl;
29  Vector_Point_2D convex_hull = GrahamAndrew(points_of_A);
30  std::cout << "-----" << std::endl;
31  std::cout << "Displaying convex hull (.3 precision):" << std::endl;
32  DisplayPoints(convex_hull, 3);
33  WriteConvexHullSegmentsToFile(convex_hull);
34  std::cout << "-----" << std::endl;
35
36  std::cout << "Reading line segment points from 'convex_hull.txt':" << std::endl;
37  Vector_Point_2D convex_segment_points = ReadPointsFromFile("convex_hull.txt");
38  std::cout << "-----" << std::endl;
39  std::cout << "Converting segment points to segments:" << std::endl;
40  Vector_Line_Segment_2D convex_line_segments = ConvertSegmentsFromFile(convex_segment_points);
41  std::cout << "-----" << std::endl;
42  std::cout << "Displaying Convex Hull line segments (.3 precision):" << std::endl;
43  DisplayLineSegments(convex_line_segments, 3);
44  std::cout << "-----" << std::endl;
45
46  Vector_Line_Segment_2D total_line_segments = file_line_segments;
47  total_line_segments.insert(total_line_segments.end(), convex_line_segments.begin(), convex_line_segments.end());
48
49  std::cout << "Creating the corresponding arrangement:" << std::endl;
50  Arrangement_2D arr = ConstructArrangement(total_line_segments);
51  std::cout << "-----" << std::endl;
52
53  std::cout << "Displaying faces:" << std::endl;
54  DisplayFacesOfArrangement(arr);
55  std::cout << "-----" << std::endl;
56 }
```

```
Reading line segment points from 'segments.txt':
Converting segment points to segments:
Displaying line segments (.3 precision):
Line segment: source (3.158,2.863), target (2.278,7.315)
Line segment: source (9.698,9.886), target (9.239,13.999)
Line segment: source (1.817,13.441), target (1.283,13.831)
Line segment: source (1.408,13.513), target (12.652,9.759)
Line segment: source (11.423,7.993), target (19.119,2.588)
Line segment: source (11.811,13.723), target (12.749,2.968)
Line segment: source (12.864,4.846), target (9.774,10.369)
Line segment: source (13.994,10.286), target (9.882,8.988)
Line segment: source (1.418,14.153), target (6.212,10.847)
Line segment: source (4.489,8.859), target (14.481,1.568)
Line segment: source (2.617,7.371), target (6.324,0.164)
Line segment: source (5.288,1.481), target (1.511,4.387)
Merge all points of A:
Calculating convex hull via Graham Andrew Algorithm.
Displaying convex hull (.3 precision):
(1.817,13.441)
(1.511,4.387)
(3.924,0.423)
(6.324,0.164)
(14.481,1.568)
(13.994,10.286)
(11.811,13.723)
(9.239,13.999)
(1.418,14.151)
Reading line segment points from 'convex_hull.txt':
Converting segment points to segments:
Displaying Convex Hull line segments (.3 precision):
Line segment: source (1.817,13.441), target (1.511,4.387)
Line segment: source (1.511,4.387), target (3.924,0.423)
Line segment: source (3.924,0.423), target (6.324,0.164)
Line segment: source (6.324,0.164), target (14.481,1.568)
Line segment: source (14.481,1.568), target (13.994,10.286)
Line segment: source (13.994,10.286), target (11.811,13.723)
Line segment: source (11.811,13.723), target (9.239,13.999)
Line segment: source (9.239,13.999), target (1.418,14.151)
Line segment: source (1.418,14.151), target (1.817,13.441)
Creating the corresponding arrangement:
Displaying arrangement line
Vertices : 35, Edges : 41, Faces : 8
```

```

6 {
7     std::cout << "Reading points from file 'points.txt'" << std::endl;
8     vector<point_2D> file_points = ReadPointsFromFile("points.txt");
9     std::cout << "-----" << std::endl;
10    std::cout << "Displaying points (.3 precision)" << std::endl;
11    DisplayPoints(file_points, 3);
12    std::cout << "-----" << std::endl;
13
14    std::cout << "Reading line segment points from 'segments.txt'" << std::endl;
15    vector<point_2D> file_line_segments = ReadPointsFromFile("segments.txt");
16    std::cout << "-----" << std::endl;
17    std::cout << "Converting segment points to segments" << std::endl;
18    vector<Line_Segment_2D> file_line_segments = ConvertSegmentsFromFile(file_line_segments);
19    std::cout << "-----" << std::endl;
20    std::cout << "Displaying line segments (.3 precision)" << std::endl;
21    DisplayLineSegments(file_line_segments, 3);
22    std::cout << "-----" << std::endl;
23
24    std::cout << "Merge all points of A" << std::endl;
25    vector<point_2D> points_of_A = file_points;
26    points_of_A.insert(points_of_A.end(), file_line_segments.begin(), file_line_segments.end());
27    std::cout << "-----" << std::endl;
28    std::cout << "Calculating convex hull via Graham Andrew Algorithm" << std::endl;
29    vector<point_2D> convex_hull = GrahamAndrew(points_of_A);
30    std::cout << "-----" << std::endl;
31    std::cout << "Displaying convex hull (.3 precision)" << std::endl;
32    DisplayPoints(convex_hull, 3);
33    WriteConvexHullSegmentsToFile(convex_hull);
34    std::cout << "-----" << std::endl;
35
36    std::cout << "Reading line segment points from 'convexhull.txt'" << std::endl;
37    vector<point_2D> convex_line_segments = ReadPointsFromFile("convexhull.txt");
38    std::cout << "-----" << std::endl;
39    std::cout << "Converting segment points to segments" << std::endl;
40    vector<Line_Segment_2D> convex_line_segments = ConvertSegmentsFromFile(convex_line_segments);
41    std::cout << "-----" << std::endl;
42    std::cout << "Displaying Convex Hull line segments (.3 precision)" << std::endl;
43    DisplayLineSegments(convex_line_segments, 3);
44    std::cout << "-----" << std::endl;
45
46    vector<Line_Segment_2D> totalLineSegments = file_line_segments;
47    totalLineSegments.insert(totalLineSegments.end(), convex_line_segments.begin(), convex_line_segments.end());
48
49    std::cout << "Creating the corresponding arrangement" << std::endl;
50    arrangement_2D arr = ConstructArrangement(totalLineSegments);
51    std::cout << "-----" << std::endl;
52    std::cout << "Displaying faces" << std::endl;
53    DisplayFacesOfArrangement(arr);
54    std::cout << "-----" << std::endl;
55
56 }

```



Το πλήθος των εδρών που αντιστοιχούν στο υπο - μελέτη στιγμιότυπο είναι 8, όπου 1 έδρα θεωρείται η μη - φραγμένη καθώς επίσης και 7 έδρες, θεωρούνται οι φραγμένες, κατά αναπαράσταση του προηγούμενου στιγμιότυπου.

4 Ερωτήματα Εντοπισμού Σημείου

Στην παρούσα ενότητα, θα μελετήσουμε αλγοριθμικές προσεγγίσεις, μέσω των οποίων μπορούμε να εντοπίσουμε κάποιο δοθέν σημείο ως προς κάποια ορισμένη διαμέριση επιπέδου. Συγκεκριμένα θα μελετήσουμε 4 αλγοριθμικές τεχνικές:

- Αφελής Εντοπισμός Σημείου
- Εντοπισμός Σημείου Μέσω Ανάτρεξης Σε Ευθύγραμμο Τμήμα
- Εντοπισμός Σημείου Μέσω Υποβοηθούμενων Σημείων
- Εντοπισμός Σημείου Μέσω Τραπεζοειδούς Χάρτη

Για κάθε μια από τις προαναφερθείσες αλγοριθμικές τεχνικές, θα παραθέσουμε και τις σχετικές βιβλιοθηκές μαζί με τις εκάστοτε ορισμένες συναρτήσεις.

4.1 Αφελής Εντοπισμός Σημείου

Ο Αφελής Εντοπισμός Σημείου, υλοποιεί ένα απλό αλγόριθμο διάσχισης της διαμέρισης, κατά όλες τις κορυφές καθώς και όλες τις ημιακμές, προκειμένου να απαντήσει στην επιθυμητή ερώτηση εντοπισμού σημείου. Ο χρόνος απάντησης κάποιου δοθέντος σημείου συνεπώς είναι γραμμικός ως προς την πολυπλοκότητα της διαμέρισης, ο οποίος στην πραγματικότητα δύναται να είναι ιδιαίτερα μεγάλος για ογκώδεις διαμερίσεις.

Η απαραίτητη βιβλιοθήκη προς υλοποίηση είναι η:

```
#include <CGAL/Arr_naive_point_location.h>
```

ενώ η ονομαστική σύμβαση που ορίζουμε είναι η:

```
typedef CGAL::Arr_naive_point_location<Arrangement_2D> Naive_Point_Location;
```

Η συνάρτηση που υλοποιούμε για τον υπολογισμό σειράς ερωτημάτων ενός ενός διανύσματος σημείων σε μια διαμέριση, ορίζεται ως:

```
LocateAndDisplayPointNaive(Arrangement_2D arr, Vector_Point_2D points)
```

Ακολουθεί στιγμιότυπο εκτέλεσης:

```

// PointLocation.cpp
37 Vector_Point_2D convex_segment_points = ReadPointsFromFile("convexHull.txt");
38 std::cout << "-----" << std::endl;
39 std::cout << "Converting segment points to segments:" << std::endl;
40 Vector_Line_Segment_2D convex_line_segments = ConvertSegmentsFromFile(convex_segment_points);
41 std::cout << "-----" << std::endl;
42 //std::cout << "Displaying Convex Hull line segments (.3 precision):\n" << std::endl;
43 //DisplayLineSegments(convex_line_segments, 3);
44 //std::cout << "-----" << std::endl;
45
46 Vector_Line_Segment_2D total_line_segments = file_line_segments;
47 total_line_segments.insert(total_line_segments.end(), convex_line_segments.begin(), convex_line_segments.end());
48
49 std::cout << "Creating the corresponding arrangement:" << std::endl;
50 Arrangement_2D arr = ConstructArrangement(total_line_segments);
51 std::cout << "-----" << std::endl;
52
53 //std::cout << "Displaying faces:" << std::endl;
54 //DisplayFacesOfArrangement(arr);
55 //std::cout << "-----" << std::endl;
56
57 //std::chrono::steady_clock::time_point begin;
58 //std::chrono::steady_clock::time_point end;
59
60 std::cout << "=== Naive Point Location ===" << std::endl;
61 //begin = std::chrono::steady_clock::now();
62 LocateAndDisplayPointNaive(arr, file_points);
63 //end = std::chrono::steady_clock::now();
64 //std::cout << "Time difference = " << std::chrono::duration_cast<std::chrono::milliseconds>(end - begin).count()
65 //std::cout << "-----" << std::endl;
66 //
67 std::cout << "=== Walk Along Line Point Location ===" << std::endl;
68 begin = std::chrono::steady_clock::now();
69 LocateAndDisplayPointWalkAlongLine(arr, file_points);
70 end = std::chrono::steady_clock::now();
71 std::cout << "Time difference = " << std::chrono::duration_cast<std::chrono::milliseconds>(end - begin).count()
72 std::cout << "-----" << std::endl;
73
74 std::cout << "=== Landmarks Point Location ===" << std::endl;
75 begin = std::chrono::steady_clock::now();
76 LocateAndDisplayPointLandmarks(arr, file_points);
77 end = std::chrono::steady_clock::now();
78 std::cout << "Time difference = " << std::chrono::duration_cast<std::chrono::milliseconds>(end - begin).count()
79 std::cout << "-----" << std::endl;
80
81 std::cout << "=== Trapezoid Point Location ===" << std::endl;
82 begin = std::chrono::steady_clock::now();
83 LocateAndDisplayPointTrapezoid(arr, file_points);
84 end = std::chrono::steady_clock::now();

```

```

Microsoft Visual Studio Debug Console
Converting segment points to segments:
Creating the corresponding arrangement:
Displaying arrangement size:
Vertices : 35, Edges : 41, Faces : 8
-----
=== Naive Point Location ===
Point:(2.189,5.898) was located inside a face.
Face details:
Outer boundary:
vector((14.481, 1.56), (10.5683, 4.41814))
vector((10.5683, 4.41814), (10.119, 2.586))
vector((10.119, 2.586), (10.5683, 4.41814))
vector((10.5683, 4.41814), (4.489, 8.859))
vector((4.489, 8.859), (10.5683, 4.41814))
vector((10.5683, 4.41814), (11.2747, 7.2982))
vector((11.2747, 7.2982), (10.2915, 9.31007))
vector((10.2915, 9.31007), (9.062, 8.986))
vector((9.062, 8.986), (10.2915, 9.31007))
vector((10.2915, 9.31007), (9.774, 10.369))
vector((9.774, 10.369), (10.2915, 9.31007))
vector((10.2915, 9.31007), (11.6546, 9.66938))
vector((11.6546, 9.66938), (11.5818, 10.1283))
vector((11.5818, 10.1283), (9.59703, 10.8133))
vector((9.59703, 10.8133), (0.69, 9.986))
vector((0.69, 9.986), (9.59703, 10.8133))
vector((9.59703, 10.8133), (1.86767, 13.4807))
vector((1.86767, 13.4807), (4.212, 10.847))
vector((4.212, 10.847), (1.86767, 13.4807))
vector((1.86767, 13.4807), (1.49, 13.611))
vector((1.49, 13.611), (1.86767, 13.4807))
vector((1.86767, 13.4807), (1.41, 14.151))
vector((1.41, 14.151), (1.017, 13.441))
vector((1.017, 13.441), (1.283, 13.831))
vector((1.283, 13.831), (1.017, 13.441))
vector((1.017, 13.441), (1.511, 4.307))
vector((1.511, 4.307), (5.28, 1.063))
vector((5.28, 1.063), (1.511, 4.307))
vector((1.511, 4.307), (3.924, 0.423))
vector((3.924, 0.423), (6.324, 0.164))
vector((6.324, 0.164), (2.31081, 7.11498))
vector((2.31081, 7.11498), (3.158, 2.963))
vector((3.158, 2.963), (2.31081, 7.11498))
vector((2.31081, 7.11498), (2.163, 7.371))
vector((2.163, 7.371), (2.31081, 7.11498))
vector((2.31081, 7.11498), (2.27, 7.315))
vector((2.27, 7.315), (2.31081, 7.11498))
vector((2.31081, 7.11498), (6.324, 0.164))
vector((6.324, 0.164), (14.481, 1.56))
Degenerate Case: Point:(3.924, 0.423) was located on/as a vertex.
Vertex details:3.924 0.423
-----
Point:(9.63,9.518) was located inside a face.
Face details:
Outer boundary:
vector((14.481, 1.56), (10.5683, 4.41814))
vector((10.5683, 4.41814), (10.119, 2.586))
vector((10.119, 2.586), (10.5683, 4.41814))
vector((10.5683, 4.41814), (4.489, 8.859))
vector((4.489, 8.859), (10.5683, 4.41814))
vector((10.5683, 4.41814), (11.2747, 7.2982))

```

Όπως παρατηρούμε και από το στιγμιότυπο, για κάθε δοθέν σημείο ο αλγόριθμος εντοπίζει την θέση, και στην συνέχεια αναφέρει όλες τις ημιακμές που ορίζουν την έδρα.

4.2 Εντοπισμός Σημείου Μέσω Ανάτρεξης Σε Ευθύγραμμο Τμήμα

Ο εντοπισμός σημείου μέσω ανάτρεξης σε ευθύγραμμο τμήμα, ορίζεται ως μια αλγοριθμική τεχνική η οποία κατασκευάζει μια κάθετη γραμμή από το σημείο αναζήτησης. Διασχίζοντας την κάθετη γραμμή από την μη - φραγμένη έδρα μέχρι και την έδρα στην οποία εμπεριέχεται το σημείο, διαπερνάει μικρότερο πλήθος εδρών από την αφελή προσέγγιση, και ως εκ τούτου αποτελεί βελτίωση της.

Όπως η αφελής προσέγγισης έτσι και ο εντοπισμός σημείου μέσω ανάτρεξης σε ευθύγραμμο τμήμα, δεν χρησιμοποιούν κάποια επιπρόσθετη δομή δεδομένων και ως εκ τούτου η συσχέτιση τους με μια δοθείσα διαμέριση, συνιστά στανθερό χρόνο.

Η απαραίτητη βιβλιοθήκη προς υλοποίηση είναι η:

```
#include <CGAL/Arr_walk_along_line_point_location.h>
```

ενώ η ονομαστική σύμβαση που ορίζουμε είναι η:

```
typedef CGAL::Arr_walk_along_line_point_location; Arrangement_2D; Walk_Along_Line_Point_Location;
```

Η συνάρτηση που υλοποιούμε για τον υπολογισμό σειράς ερωτημάτων ενός ενός διανύσματος σημείων σε μια διαμέριση, ορίζεται ως:

```
LocateAndDisplayPointWalkAlongside(Arrangement_2D arr, Vector_Point_2D points)
```

Δοθέντος ότι όλες οι αλγοριθμικές προσεγγίσεις που υλοποιούμε, ακολουθούν το ίδιο μοτίβο παρουσίασης εξόδου αποτελεσμάτων, θα προσπεράσουμε αυτό το βήμα για την παρούσα και τις ερχόμενες προσεγγίσεις.

4.3 Εντοπισμός Σημείου Μέσω Υποβοηθούμενων Σημείων

Η αλγοριθμική τεχνική εντοπισμού σημείων μέσω υποβοηθούμενων σημείων, υλοποιεί μια προσέγγιση: κάνε άλμα και περπάτα. Η προσέγγιση αυτή υποδηλώνει ότι κάποια συγκεκριμένα σημεία, τα οποία αναφέρονται ως υποβοηθούμενα σημεία ή σημεία αναφοράς, επιλέγονται σε μια διαδικασία προ-επεξεργασίας, οι έδρες στις οποίες ανήκουν εντοπίζονται και εισαγάγονται σε μια δομή δεδομένων K-δ Δέντρων. Με βάση αυτήν την εξωτερική δομή, κάθε επιθυμητό ερώτημα εντοπισμού σημείου χρησιμοποιεί κατάλληλη διάσχιση.

Ενα επιπρόσθετο σημαντικό χαρακτηριστικό της παρούσας αλγοριθμικής τεχνικής, είναι η επιλογή των σημείων (στην διαδικασία της προεπεξεργασίας), η οποία ορίζει την κατασκευή του K-δ δέντρου διάσχισης. Αν και υπάρχουν αρκετές προσεγγίσεις, η προσέγγιση που θα ακολουθήσουμε εμείς, είναι η επιλογή των σημείων της διαμέρισης (δηλαδή των κορυφών που έχουν δημιουργηθεί από την διαμέριση).

Η απαραίτητη βιβλιοθήκη προς υλοποίηση είναι η:

```
#include <CGAL/Arr_landmarks_point_location.h>
```

ενώ η ονομαστική σύμβαση που ορίζουμε είναι η:

```
typedef CGAL::Arr_landmarks_point_location<Arrangement_2D> LandMarks_Point_Location;
```

Η συνάρτηση που υλοποιούμε για τον υπολογισμό σειράς ερωτημάτων ενός ενός διανύσματος σημείων σε μια διαμέριση, ορίζεται ως:

```
LocateAndDisplayPointLandmarks(Arrangement_2D arr, Vector_Point_2D points)
```

4.4 Εντοπισμός Σημείου Μέσω Τραπεζοειδούς Χάρτη

Η τελευταία προσέγγιση που θα μελετήσουμε για τους σκοπούς της παρούσας εργασίας, είναι ο εντοπισμός σημείου μέσω τραπεζοειδούς χάρτη. Η προσέγγιση αυτή, είναι η υλοποίηση της αλγοριθμικής τεχνικής που βρίσκουμε στο 6 κεφάλαιο του βιβλίου 'Υπολογιστική Γεωμετρία: ΑΛΓΟΡΙΘΜΟΙ ΚΑΙ ΕΦΑΡΜΟΓΕΣ'. Συνιστά έναν αυξητικό τυχαιοκρατικό αλγόριθμο, ο οποίος διαχωρίζει κάθε έδρα της διαμέρισης σε ψευτοπολυγωνικές έδρες (κυρτά πολύγωνα ή τρίγωνα), κατασκευάζοντας $2k$ (όπου k τα σημεία της διαμέρισης) κάθετες ακμές - μια με αφετηρία την κορυφή και κατεύθυνση προς τα πάνω - καθώς και μια με αφετηρία την κορυφή και κατεύθυνση προς τα κάτω. Η καινούριας ακμές προεκτείνονται μέχρις ότου να βρουν κάποια υπάρχουσα ακμή της διαμέρισης.

Για κάθε καινούρια εξαχθείσα ψευτο-πολυγωνική έδρα, διατηρείται μια εγγραφή σε μια δομή δεδομένων άκυκλου - κατευθυντού γραφήματος. Κάθε αναζήτηση σημείου συνεπώς λαμβάνει χρόνο $O(\log n)$. Παρόλο ωστόσο που ο χρόνος αναζήτησης είναι ιδιαίτερα αποδοτικός, η δυσκολία (και ως εκ τούτου η ανάγκη χρήσης της τυχαιοκρατικής ιδιότητας), έγκειται στον χώρο και στον χρόνο κατασκευής της βοηθητικής δομής δεδομένων του άκυκλου κατευθυντού γραφήματος.

Δοθέντος ότι ο αλγόριθμος είναι αυξητικός και κατασκευάζει την δομή μελετώντας κάθε ακμή της καινούριας διαμέρισης, μια κάθε φορά, ενδέχεται να υπάρχουν μεταθέσεις επιλογής ακμών οι οποίες αποφέρουν χρόνο κατασκευής της δομής τάξης $O(n^2)$. Η ιδιότητα αυτή εξαλείφεται προσθέτοντας την τυχαιοκρατικότητα, και ως εκ τούτου ο αναμενόμενος χρόνος εκτέλεσης της κατασκευής της εξωτερικής δομής, αναμένεται να είναι

$O(n \log n)$. Η ιδιότητα αυτή προκύπτει ως άμεσο αποτέλεσμα της παρατήρησης ότι σε μια τυχαία μετάθεση ακμών, κατά την μελέτη και προσθήκη εδρών που κατασκευάζονται στην εξωτερική δομή, δεν χρειάζεται να κάνουμε συχνές αλλαγές, τροποποιήσεις και ενημερώσεις.

Δοθέντος ότι η αλγοριθμική προσέγγιση απαιτεί την κατασκευή μιας εξωτερικής δομής δεδομένων, αναμένουμε να υπάρχει ένα επιπρόσθετο κόστος κατασκευής κατά την εκτέλεση, σε σχέση με τις άλλες τεχνικές, εντούτοις είναι βέβαιο ότι απαξ και η δομή δημιουργηθεί επιτυχώς, κάθε ερώτημα αναζήτησης σημείου θα αποδίδει εξαιρετικά γρήγορα.

Η απαραίτητη βιβλιοθήκη προς υλοποίηση είναι η:

```
#include <CGAL/Arr_trapezoid_ric_point_location.h>
```

ενώ η ονομαστική σύμβαση που ορίζουμε είναι η:

```
typedef CGAL::Arr_trapezoid_ric_point_location<Arrangement_2D> Trapezoid_Point_Location;
```

Η συνάρτηση που υλοποιούμε για τον υπολογισμό σειράς ερωτημάτων ενός ενός διανύσματος σημείων σε μια διαμέριση, ορίζεται ως:

```
LocateAndDisplayPointTrapezoid(Arrangement_2D arr, Vector_Point_2D points)
```

5 Μελέτη Χρονικής Απόδοσης

Σε αυτήν την ενότητα, θα μελετήσουμε τον χρόνο της ροής εκτέλεσης των προηγούμενων βημάτων, για διάφορες τιμές ευθύγραμμων τμημάτων διαμέρισης A n καθώς και διάφορες τιμές σημείων εντοπισμού k . Για τους σκοπούς της μελέτης μας, θα κάνουμε χρήση της βιβλιοθήκης:

```
#include <chrono>
```

η οποία αποτελεί βιβλιοθήκη διαχείρισης χρονικών στιγμιοτύπων.

5.1 Παράθεση Πίνακα Αποτελεσμάτων

Παρακάτω, παρουσιάζεται αναλυτικός πίνακας αποτελεσμάτων εκτέλεσης για διάφορες τιμές σημείων k και ευθύγραμμων τμημάτων n που ορίζουν μια τυχαία διαμέριση A στο επίπεδο. Επιπρόσθετα, ορίζονται το πλήθος κορυφών v , το πλήθος ημιακμών e καθώς και το πλήθος των εδρών f . Ο χρόνος εκτέλεσης μετρά τον χρόνο Εύρεσης του κυρτού περιβλήματος, καθώς και τον χρόνο εύρεσης και εμφάνισης της έδρας που εμπεριέχει το κάθε σημείο, για κάθε μια από τις 4 αλγοριθμικές προσεγγίσεις που έχουμε δει.

Χρόνοι Εκτέλεσης σε <i>milliseconds</i>					
Στιγμιότυπο	Εύρεση Κυρτού Περιβλήματος	Αφελής Προσέγγιση	Ανάτρεξη Σε Ευθ. Τμήμα	Υποβοηθούμενα Σημεία	Τραπεζοειδής Χάρτης
$k = 6$ $n = 12$ $v = 35$ $e = 41$ $f = 8$	1	≈ 331.0	≈ 328.0	≈ 330.6	≈ 492.6
$k = 12$ $n = 14$ $v = 48$ $e = 57$ $f = 12$	1	≈ 364.6	≈ 362.6	≈ 353.6	≈ 633.0
$k = 50$ $n = 100$ $v = 385$ $e = 624$ $f = 241$	2	$\approx 1,630.8$	$\approx 1,467.0$	$\approx 1,457.8$	$\approx 5,960.8$

$k = 289$ $n = 510$ $v = 32,333$ $e = 63,150$ $f = 30,819$	≈ 35.2	$\approx 109,830.0$	$\approx 9,625.0$	$\approx 9,591.0$	$\approx 541,004.0$
--	----------------	---------------------	-------------------	-------------------	---------------------

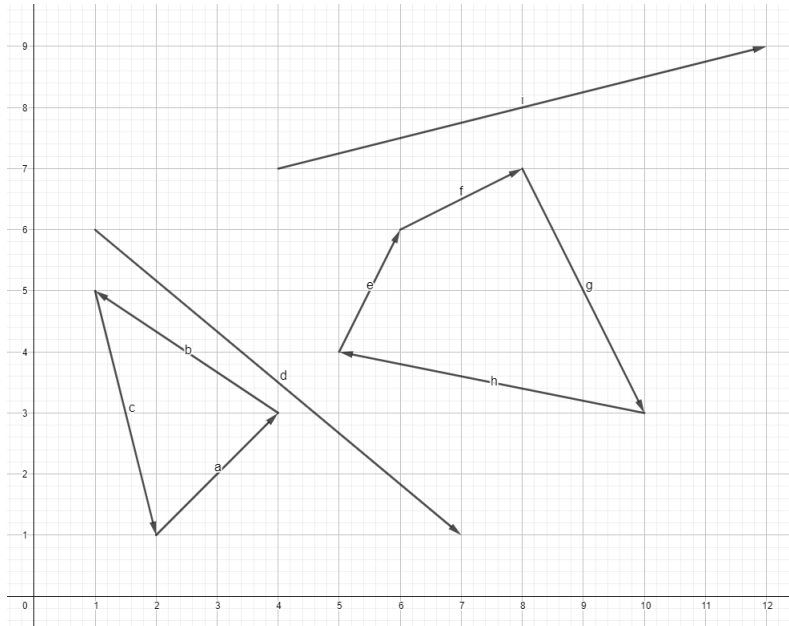
Απο τα παραπάνω στιγμιότυπα μπορούμε να επαληθεύσουμε την αναμενόμενη συμπεριφορά για κάθε αλγοριθμική προσέγγιση.

Αναλυτικότερα, παρατηρούμε ότι για μεγάλα στιγμιότυπα, ο αφελής αλγόριθμος είναι ιδιαίτερα μη - αποδοτικός. Η ανάτρεξη σε ευθύγραμμα τμήματα και η προσέγγιση των υποβοηθούμενων σημείων είναι εμφανώς ποιο αποδοτικά, με την προσέγγιση των υποβοηθούμενων σημείων να υπερτερεί ελαφρώς, ακόμα και εφόσον καλείται να κατασκευάσει μια επιπρόσθετη δομή δεδομένων. Αυτό μας υποδυκνύει ότι η αναζήτηση των σημείων είναι ιδιαίτερα ταχεία ενώ αν αποθηκεύσουμε την δομή σε κάποιο εξωτερικό μέσο, η αναζήτηση θα είναι εξαιρετική.

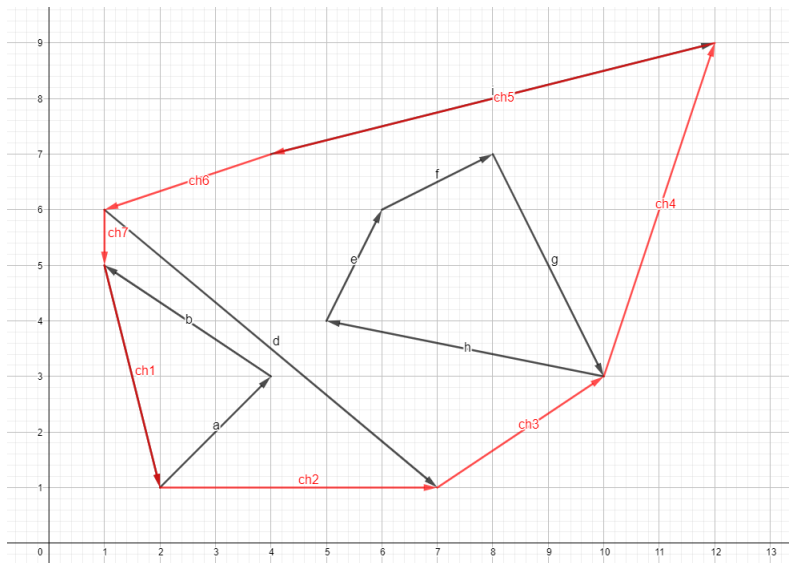
Ακριβώς το ίδιο συμπέρασμα μπορούμε να εξαγάγουμε και για την προσέγγιση του τραπεζοειδή χάρτη. Η συνολική χρονική απόδοση είναι εξαιρετικά αποθαυντική, καθώς ο χρόνος κατασκευής του χάρτη είναι στην μέση αναμενόμενη περίπτωση $O(n \log n)$ ενώ στην χειρότερη περίπτωση $O(n^2)$. Η προσέγγιση αυτή, κατά την άποψη του φοιτητή, θα πρέπει να χρησιμοποιείται αυστηρά για σχετικά μεγάλα στιγμιότυπα, ως εξωτερική αναφορά: δηλαδή να κατασκευαστεί ο χάρτης, να αποθηκευτεί σε εξωτερικό μέσο και πάνω σε αυτό να επιτελούνται ερωτήματα σε $O(\log n)$ χρόνο.

6 Αποθήκευση και Ανάκληση Διαμέρισης A

Στην παρούσα ενότητα, θα παραθέσουμε με ποιον τρόπο μπορούμε να αποθηκεύσουμε και να ανακαλέσουμε μια δοθείσα διαμέριση A, σε κάποιο εξωτερικό αρχείο *arrangment.txt*. Για τους σκοπούς επαλήθευσης της επιθυμητής λειτουργικότητας, θα παραθέσουμε ένα απλό παράδειγμα διαμέρισης. Τα ευθύγραμμα τμήματα που θα δοθούν ως είσοδο αποτυπώνονται στην επόμενη οπτική αναπαράσταση:



Το εμπλουτισμένο στιγμιότυπο το οποίο συμπεριλαμβάνει και τις ακμές του κυρτού περιβλήματος αποτυπώνεται απο την επόμενη οπτική αναπαράσταση:



Η διαμέριση του επιπέδου που διαμορφώνεται, εμπεριέχει όπως φαίνεται και απο την επόμενη εικόνα, σε: 11 κόμβους, 14 ακμές και 5 έδρες (συμπεριλαμβανομένου της εξωτερικής μη - φραγμένης).

```

22 Vector_Point_2D file_segment_points = ReadPointsFromFile("segments.txt");
23 std::cout << "Reading line segment points from 'segments.txt':\n" << std::endl;
24 std::cout << "Converting segment points to segments:" << std::endl;
25 Vector_Line_Segment_2D file_line_segments = ConvertSegmentsFromFile(file_segment_points);
26 std::cout << "Reading line segment points from 'convexHull.txt':\n" << std::endl;
27
28 //std::cout << "Displaying line segments (.3 precision):\n" << std::endl;
29 //DisplayLineSegments(file_line_segments, 3);
30 //std::cout << "-----" << std::endl;
31
32 std::cout << "Merge all points of A:" << std::endl;
33 Vector_Point_2D points_of_A = file_points;
34 points_of_A.insert(points_of_A.end(), file_segment_points.begin(), file_segment_points.end());
35 std::cout << "-----" << std::endl;
36
37 std::cout << "Calculating convex hull via Graham Andrew Algorithm." << std::endl;
38 begin = std::chrono::steady_clock::now();
39 Vector_Point_2D convexHull = GrahamAndrew(points_of_A);
40 end = std::chrono::steady_clock::now();
41 std::cout << "Time difference = " << std::chrono::duration_cast<std::chrono::milliseconds>(end - begin).count() << " ms." << std::endl;
42 WriteConvexHullSegmentsToFile(convexHull);
43 std::cout << "-----" << std::endl;
44
45 //std::cout << "Displaying convex hull (.3 precision):\n" << std::endl;
46 //DisplayPoints(convexHull, 3);
47 //std::cout << "-----" << std::endl;
48
49 std::cout << "Reading line segment points from 'convexHull.txt':\n" << std::endl;
50 Vector_Point_2D convex_segment_points = ReadPointsFromFile("convexHull.txt");
51 std::cout << "-----" << std::endl;
52
53 std::cout << "Converting segment points to segments:" << std::endl;
54 Vector_Line_Segment_2D convex_line_segments = ConvertSegmentsFromFile(convex_segment_points);
55 std::cout << "-----" << std::endl;
56
57 std::cout << "Displaying Convex Hull line segments (.3 precision):\n" << std::endl;
58 DisplayLineSegments(convex_line_segments, 3);
59 std::cout << "-----" << std::endl;
60
61 Vector_Line_Segment_2D totalLineSegments = file_line_segments;
62 totalLineSegments.insert(totalLineSegments.end(), convex_line_segments.begin(), convex_line_segments.end());
63
64 std::cout << "Creating the corresponding arrangement:" << std::endl;
65 Arrangement_2D arr = ConstructArrangement(totalLineSegments);
66 std::cout << "-----" << std::endl;
67
68 //std::cout << "Displaying faces:" << std::endl;

```

```

Microsoft Visual Studio Debug Console
Reading points from file 'points.txt':
-----
Reading line segment points from 'segments.txt':
Converting segment points to segments:
-----
Merge all points of A:
-----
Calculating convex hull via Graham Andrew Algorithm.
Time difference = 0 milliseconds
Reading line segment points from 'convexHull.txt':
Converting segment points to segments:
-----
Displaying Convex Hull line segments (.3 precision):
Line segment: source (1.000,5.000), target (2.000,1.000)
Line segment: source (2.000,1.000), target (7.000,1.000)
Line segment: source (7.000,1.000), target (10.000,3.000)
Line segment: source (10.000,3.000), target (12.000,9.000)
Line segment: source (12.000,9.000), target (4.000,7.000)
Line segment: source (4.000,7.000), target (1.000,6.000)
Line segment: source (1.000,6.000), target (1.000,5.000)
-----
Creating the corresponding arrangement:
Displaying arrangement size:
Vertices : 11, Edges : 14, Faces : 5
-----
C:\Users\Juljan\source\repos\PointLocation\64\Debug\PointLocation.exe (process 19020) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .

```

6.1 Αποθήκευση Διαμέρισης A

Για την αποθήκευση της διαμέρισης, αποθηκεύουμε κάθε μια από τις κορυφές της διαμέρισης, καθώς και κάθε ημιακή κάθε έδρας. Με αυτόν τον τρόπο, διασφαλίζουμε ότι δεν θα παραβλέψουμε κάποιον απομωνομένο κόμβο. Ακολουθεί στιγμιότυπο του αρχείου αποθήκευσης:

```

1 1,5
2 1,6
3 2,1
4 4,3
5 4,7
6 5,4
7 6,6
8 7,1
9 8,7
10 10,3
11 12,9
12 -----
13 Unbounded
14 -----
15 4, 3
16 1, 5
17 1, 5
18 2, 1
19 2, 1
20 4, 3
21 -----
22 7, 1
23 1, 6
24 1, 6
25 1, 5
26 1, 5
27 4, 3
28 4, 3
29 2, 1
30 2, 1
31 7, 1
32 -----
33 10, 3
34 8, 7
35 8, 7
36 6, 6
37 6, 6
38 5, 4
39 5, 4
40 10, 3
41 -----

```

Όπως παρατηρούμε, το αρχείο έχει χωριστεί σε κάποια επιμέρους κομμάτια. Το πρώτο κομμάτι, ορίζει όλες τις κορυφές της διαμέρισης, ενώ το κάτω κομμάτι, ορίζει όλες τις ημιακές της κάθε έδρας (συμπεριλαμβανομένης και της μη φραγμένης).

Η συνάρτηση η οποία είναι υπεύθυνη για την αποθήκευση της διαμέρισης, είναι η:

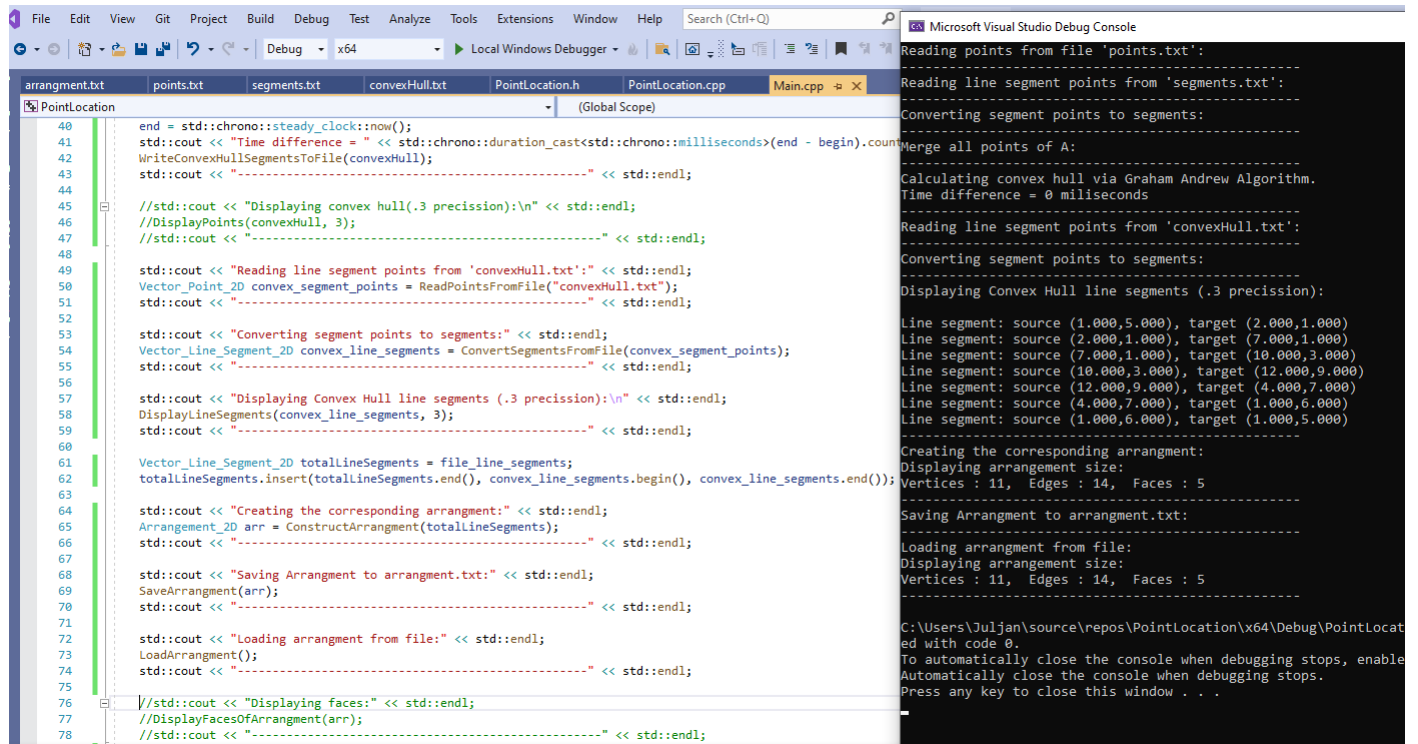
SaveArrangement(Arrangement_2D arr)

6.2 Ανάκληση της Διαμέρισης απο το αρχείο

Με βάση την δομή που μελετήσαμε στην προηγούμενη υποενότητα, κατασκευάζουμε συνάρτηση

`Arrangement_2D LoadArrangement()`

η οποία διαβάζει το αρχείο *arrangement.txt* και επιστρέφει μια διαμέριση. Παράλληλα, κατά την κλήση της, εκτυπώνεται και κατάλληλο μήνυμα του μεγέθους της διαμέρισης, όπως διαφάνεται και απο το επόμενο στιγμιότυπο:



The screenshot displays the Microsoft Visual Studio IDE with the `PointLocation.cpp` file open. The code is in C++ and includes various standard library headers and custom classes. It demonstrates the process of loading an arrangement from a file, calculating the convex hull, and displaying the resulting line segments and faces. The `LoadArrangement()` function is the focus of the section.

```
40 end = std::chrono::steady_clock::now();
41 std::cout << "Time difference = " << std::chrono::duration_cast<std::chrono::milliseconds>(end - begin).count() << " milliseconds\n";
42 WriteConvexHullSegmentsToFile(convexHull);
43 std::cout << "-----" << std::endl;
44
45 //std::cout << "Displaying convex hull(.3 precision):\n" << std::endl;
46 //DisplayPoints(convexHull, 3);
47 //std::cout << "-----" << std::endl;
48
49 std::cout << "Reading line segment points from 'convexHull.txt':" << std::endl;
50 Vector_Point_2D convex_segment_points = ReadPointsFromFile("convexHull.txt");
51 std::cout << "-----" << std::endl;
52
53 std::cout << "Converting segment points to segments:" << std::endl;
54 Vector_Line_Segment_2D convex_line_segments = ConvertSegmentsFromFile(convex_segment_points);
55 std::cout << "-----" << std::endl;
56
57 std::cout << "Displaying Convex Hull line segments (.3 precision):\n" << std::endl;
58 DisplayLineSegments(convex_line_segments, 3);
59 std::cout << "-----" << std::endl;
60
61 Vector_Line_Segment_2D totalLineSegments = file_line_segments;
62 totalLineSegments.insert(totalLineSegments.end(), convex_line_segments.begin(), convex_line_segments.end());
63
64 std::cout << "Creating the corresponding arrangement:" << std::endl;
65 Arrangement_2D arr = ConstructArrangement(totalLineSegments);
66 std::cout << "-----" << std::endl;
67
68 std::cout << "Saving Arrangement to arrangement.txt:" << std::endl;
69 SaveArrangement(arr);
70 std::cout << "-----" << std::endl;
71
72 std::cout << "Loading arrangement from file:" << std::endl;
73 LoadArrangement();
74 std::cout << "-----" << std::endl;
75
76 //std::cout << "Displaying faces:" << std::endl;
77 //DisplayFacesOfArrangement(arr);
78 //std::cout << "-----" << std::endl;
```

The Debug Console on the right shows the output of the program, including the time difference, the reading of line segment points, the conversion of segment points to segments, the display of convex hull line segments, the creation of the corresponding arrangement, the saving of the arrangement to a file, and the loading of the arrangement from a file. The output also includes the size of the arrangement (Vertices: 11, Edges: 14, Faces: 5).

7 Υπερσύνδεση Αποθετηρίου Αναλυτικού Κώδικα

Σε αυτήν την τελευταία ενότητα, παραθέτουμε υπερσυνδέσεις για το προσωπικό αποθετήριο του φοιτητή, όπου υπάρχει αναλυτικά όλος ο κώδικας (αυστηρά εμπλουτισμένος με σχόλια και παραθέσεις σε βιβλιοθήκες) για την υλοποίηση της εργασίας (Αποθετήριο). Αναλυτικότερα, ο αναγνώστης θα βρεί τα εξής αρχεία:

- PointLocation.h
- PointLocation.cpp
- Main.cpp
- arrangment.txt
- convexHull.txt
- segments.txt

Αναφορές

- [1] Υπολογιστική Γεωμετρία, Αλγόριθμοι και Εφαρμογές, M.D.Berg, O. Cheong, M.V.Kreveld, M.Overmars
Πανεπιστημιακές Εκδόσεις Κρήτης, 2011.