Πανεπιστημιο Κρητης

Σχολη Θετικων Επιστημων

Διατμηματικο Μεταπτυχιακο Προγραμμα Σπουδων
«Μαθηματικα και Εφαρμογες τους»

Μεταπτυχιακη Εργασια

Κατασκευη Περιβαλλουσων Καμπυλων για Κυρτα Πολυγωνικα Αντικειμενα με τη Βοηθεια Διαγραμματων Voronoi

Κουτάκη - Παντερμάκη Ειρήνη, Αριστείδη

Επιβλέπων Καθηγητής: Καραβέλας Μενέλαος

Ηράκλειο, Ιούνιος 2009

University of Crete

School of Sciences

Inter-departmental Graduate Program
"Mathematics and its Applications"

Master of Science Thesis

Construction of Surrounding Curves for Convex Polygonal Objects using Voronoi Diagrams

Koutaki - Pantermaki Irini, Aristidis

Thesis Adviser: Karavelas Menelaos

Heraklion, June 2009

Επιτροπή Αξιολόγησης

- Π. Δ. Κακλής, Καθηγητής, Σχολή Ναυπηγών Μηχ/γων Μηχανικών, Εθνικό Μετσόβιο Πολυτεχνείο
- Ι. Γ. Τόλλης, Καθηγητής, Τμήμα Επιστήμης Υπολογιστών, Πανεπιστήμιο Κρήτης
- Μ. Ι. Καραβέλας (Επιβλέπων Καθηγητής), Επίκουρος Καθηγητής, Τμήμα Εφαρμοσμένων Μαθηματικών, Πανεπιστήμιο Κρήτης

Ευχαριστίες

Ευχαριστώ τον καθηγητή μου, κύριο Μενέλαο Καραβέλα, για την πολύτιμη βοήθεια και καθοδήγησή του, καθώς και τα άλλα δύο μέλη της επιτροπής, κύριο Παναγιώτη Κακλή και Ιωάννη Τόλλη, για την προσεκτική μελέτη της εργασίας και τις χρήσιμες παρατηρήσεις τους. Επίσης, ευχαριστώ τους γονείς μου που με βοήθησαν να φτάσω έως εδώ.

Περίληψη

Στην εργασία αυτή παρουσιάζεται η κατασκευή μίας καμπύλης, η οποία περιβάλλει ξένα ανά δύο κυρτά πολύγωνα στο επίπεδο. Το βασικό εργαλείο με το οποίο δημιουργείται αυτή η καμπύλη, είναι το διάγραμμα Voronoi των κορυφών των πολυγώνων. Η κατασκευή αυτή εξυπηρετεί την επίλυση του προβλήματος ανακατασκευής επιφανειών που παρεμβάλλουν οπές σε παράλληλα επίπεδα. Παρουσιάζονται δύο αλγόριθμοι που υλοποιήθηκαν για την κατασκευή της καμπύλης αυτής, οι οποίοι ξεκινώντας από την κυρτή θήκη των πολυγώνων, αναζητούν πολύγωνα στο εσωτερικό της και τα προσθέτουν στην ακολουθία των πολυγώνων που ορίζουν την τρέχουσα περιβάλλουσα καμπύλη. Ο ένας αλγόριθμος κάνει αναζήτηση περιφερειακά, μιμούμενος την Breadth First αναζήτηση που χρησιμοποιείται σε γραφήματα, ενώ ο δεύτερος κάνει αναζήτηση πρώτα εις βάθος, μιμούμενος την Depth First αναζήτηση. Στη συνέχεια παρατίθενται παραδείγματα εφαρμογής των αλγορίθμων μας, καθώς και το public interface των κλάσεων `C++` του κώδικα της υλοποιήσης μας. Τέλος καταγράφονται κάποιες ιδέες για μελλοντικές βελτιώσεις της εργασίας αυτής.

Λέξεις Κλειδιά: περιβάλλουσα καμπύλη, διάγραμμα Voronoi, τριγωνοποίηση Delaunay, κυρτό πολύγωνο, κυρτή θήκη

Summary

In this thesis, we present the construction of a curve that surrounds disjoint convex polygons on a plane. The basic tool that we use is the Voronoi diagram of the polygon vertices. The solution to this problem serves as an intermediate step in the construction of surfaces that interpolate contours on parallel cross sections. We present two algorithms that we have developed, which output the surrounding curve. These algorithms, start from the convex hull, and then search for polygons in its interior and insert them in the sequence of polygons currently comprising the surrounding curve. The first algorithm searches in a breadth first search manner, while the second searches in a depth first search tactic. We demostrate the applicability of our algorithms via examples, and present the public interface of the implemented `C++` classes of our implementation. Finally, we discuss some ideas for further reasearch and improvements.

Keywords: surrounding curve, Voronoi diagram, Delaunay triangulation, convex polygon, convex hull

Περιεχόμενα

Κεφάλαιο 1

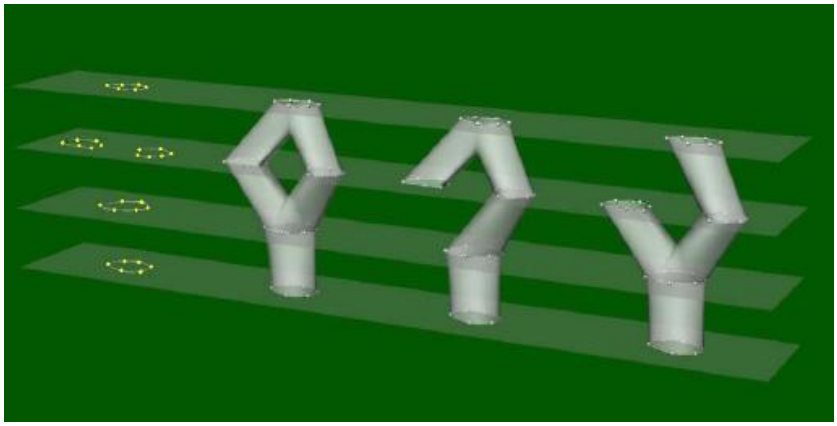
Εισαγωγή

Στην παρούσα εργασία εξετάζουμε την κατασκευή καμπύλης που περιβάλλει κυρτά πολυγωνικά αντικείμενα, τα οποία βρίσκονται στο ίδιο επίπεδο. Η κατασκευή αυτής της καμπύλης εξυπηρετεί τη λύση ενός μεγαλύτερου προβλήματος, αυτού της κατασκευής επιφανειών από παράλληλες τομές.

Το πρόβλημα της κατασκευής επιφανειών από παράλληλες τομές προέκυψε τη δεκαετία του 1970 στον τομέα της Βιοϊατρικής, όπου υπήρχε η ανάγκη αναπαράστασης των ανθρωπίνων οργάνων στον υπολογιστή. Αυτή η αναπαράσταση αντιστοιχούσε στην επιφάνεια, ενώ οι παράλληλες τομές αντιστοιχούσαν στο αποτέλεσμα της ακτινοβόλησης του αντίστοιχου οργάνου (CT Scans [13]). Άλλες εφαρμογές του συγκεκριμένου προβλήματος εμφανίζονται στον τομέα της Μηχανολογίας, Ναυπηγικής και Αεροναυπηγικής [44].

Θα αναφερθούμε περιληπτικά στο πρόβλημα, ώστε να γίνει εμφανής η χρησιμότητα της παρούσας εργασίας, ως τμήμα του προβλήματος αυτού.

Το πρόβλημα χωρίζεται σε δύο υποπροβλήματα:



Σχήμα 1.1: Ένα παράδειγμα δεδομένων, και οι πιθανές λύσεις του [44].

Πρόβλημα αντιστοίχησης περιγραμμάτων Εδώ εξετάζεται με ποια από τα περιγράμματα των παρακείμενων επιπέδων πρέπει να συνδεθεί με το κάθε περίγραμμα. Αν υπάρχει ένα περίγραμμα σε κάθε επίπεδο, τότε είναι εμφανής ο τρόπος σύνδεσής

τους. Αν εμφανιστούν όμως περισσότερα του ενός, όπως στο Σχήμα 1.1, τότε υπάρχουν και περισσότερες από μία πιθανές συνδέσεις. Η αναπαράσταση αυτών των συνδέσεων μπορεί να υλοποιηθεί με τη βοήθεια ενός γραφήματος $G(V, E)$ (όπου V είναι το σύνολο των κόμβων και E το σύνολο των ακμών του) στο οποίο οι κόμβοι αντιστοιχούν στα περιγράμματα όλων των τομών και οι ακμές στις συνδέσεις μεταξύ των περιγραμμάτων με επιφανειακά τμήματα. Τα σημαντικότερα χαρακτηριστικά του γραφήματος είναι:

- Περιγράμματα που ανήκουν στο ίδιο επίπεδο δε συνδέονται μεταξύ τους.
- Περιγράμματα που ανήκουν σε μη παρακείμενα επίπεδα δε συνδέονται μεταξύ τους.
- Υπάρχει πάντα τουλάχιστον μία σύνδεση μεταξύ δύο παρακείμενων επιπέδων.
- Μεταξύ 2 παρακείμενων επιπέδων H_{z_i} και $H_{z_{i+1}}$ μπορούν να εμφανιστούν ένα ή περισσότερα, ξένα μεταξύ τους, υπογραφήματα G_{ij} του G. Αν λοιπόν, το υπογράφημα G_{ij} του επιπέδου G ορίζει τις συνδέσεις μεταξύ M_i περιγραμμάτων του επιπέδου H_{z_i} και M_{i+1} περιγραμμάτων του επιπέδου $H_{z_{i+1}}$, τότε το σύνολο των ακμών του G_{ij} αποτελείται από όλους τους δυνατούς επιτρεπτούς συνδυασμούς των περιγραμμάτων αυτών.

Παραδείγματα μεθόδων επίλυσης του προβλήματος εύρεσης των ακμών του G_{ij} μπορεί κανείς να βρει στα [23, 35, 42].

Τοπικό πρόβλημα κατασκευής παρεμβάλλουσας επιφάνειας Χωρίς βλάβη της γενικότητας υποθέτουμε ότι όλες οι συνδέσεις μεταξύ των περιγραμμάτων των επιπέδων H_{z_i} και $H_{z_{i+1}}$ παριστάνονται από έναν μοναδικό υπογράφο G_l. Τότε η δομή του θα υπακούει σε μία από τις εξής τρεις περιπτώσεις:

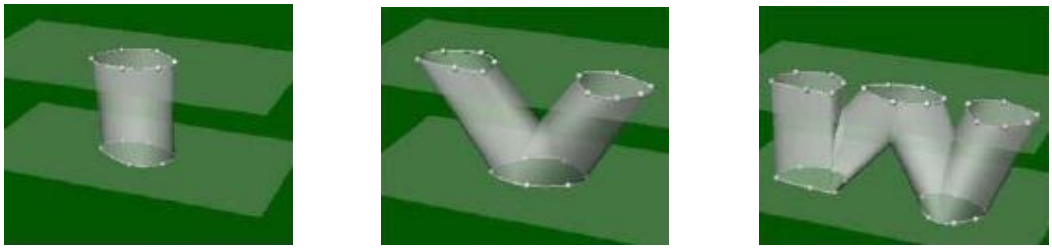
1. Το G_l έχει ένα κόμβο σε κάθε επίπεδο, δηλαδή μία μόνο ακμή. Άρα το G_l είναι *λίστα 2 στοιχείων*.
2. Το G_l έχει ένα κόμβο στο επίπεδο H_{z_i} και M κόμβους στο επίπεδο $H_{z_{i+1}}$. Τότε το G_l έχει M ακμές και είναι *δένδρο*.
3. Το G_l έχει M_i κόμβους στο επίπεδο H_{z_i} και M_{i+1} κόμβους στο επίπεδο $H_{z_{i+1}}$. Τότε το G_l έχει $M_i \times M_{i+1}$ ακμές.

Για κάθε μία από αυτές τις περιπτώσεις εμφανίζεται και ένα διαφορετικό *τοπικό πρόβλημα κατασκευής παρεμβάλλουσας επιφάνειας* (βλέπε Σχήμα 1.2).

(i) Πρόβλημα «ένα - προς - ένα».

(ii) Πρόβλημα «ένα - προς - M» ή πρόβλημα *διακλάδωσης*.

(iii) Πρόβλημα «M_i - προς - M_{i+1}» ή πρόβλημα *πολλαπλής διακλάδωσης*.



(α′) 1 - 1. (β′) 1 - M (γ′) M_i - M_{i+1}

Σχήμα 1.2: Τα υποπροβλήματα του προβλήματος κατασκευής επιφάνειας από παράλληλες τομές [44].

Στο [44] αναλύονται οι λόγοι για τους οποίους τα προβλήματα (ii) και (iii) δεν μπορούν να επιλυθούν άμεσα με κάποιο τοπικό επιφανειακό σχήμα παρεμβολής, ενώ το πρόβλημα (i) μπορεί. Στόχος λοιπόν είναι μετατρέψουμε κάθε ένα από τα δύο αυτά προβλήματα σε πρόβλημα τύπου (i).

Αν λοιπόν είμαστε στην περίπτωση (ii) και κατασκευαστεί στο επίπεδο H_{z_i} (αλλά και στο $H_{z_{i+1}}$, για την περίπτωση του (iii)) μία καμπύλη η οποία να περιβάλει όλα τα περιγράμματα, τότε θα έχουμε στο επίπεδο αυτό ένα υπερ-περίγραμμα το οποίο θα μας οδηγήσει να καταλήξουμε στο πρόβλημα «ένα - προς - ένα». Έτσι η επίλυση του «ένα - προς - ένα» οδηγεί και σε επίλυση των άλλων δύο προβλημάτων. Αυτήν την περιβάλλουσα καμπύλη θα επιχειρήσουμε να κατασκευάσουμε στην παρούσα εργασία.

1.1 Περιβάλλουσα καμπύλη

Έχοντας ως δεδομένα κάποια κυρτά πολυγωνικά αντικείμενα, θέλουμε να δημιουργήσουμε μία περιβάλλουσα καμπύλη. Δηλαδή μία καμπύλη η οποία δε θα είναι απλώς η κυρτή θήκη των σημείων των αντικειμένων αλλά κάτι πληρέστερο. Θα είναι μια καμπύλη η οποία θα περιέχεται στην κυρτή θήκη και θα ακουμπάει τα περισσότερα, ή και όλα τα πολυγωνικά αντικείμενα, αναλόγως με τις επιλογές του χρήστη. Η κάθε επιλογή που θα έχει στη διάθεσή του ο χρήστης θα του δίνει και διαφορετικό αποτέλεσμα καμπύλης. Περισσότερες λεπτομέρειες για τις διαφορετικές επιλογές του χρήστη αναλύονται στο Κεφάλαιο 3.

Την καμπύλη που κατασκευάσαμε τη μελετήσαμε κυρίως ως ένα συνδυαστικό αντικείμενο, διότι μας ενδιέφερε ποια αντικείμενα θα ακουμπήσει και με ποια σειρά. Δεν τη μελετήσαμε ως γεωμετρικό αντικείμενο, για να ασχοληθούμε με τις απαραίτητες συνθήκες λειότητας που θα έπρεπε να διαθέτει ως καμπύλη. Γι΄ αυτό, παρ΄ όλο που η περιβάλλουσα καμπύλη πρέπει να είναι ομοιομορφική με έναν κύκλο, αυτό δεν είναι ακόμα εφικτό, διότι δεν έχουν μπει τα κατάλληλα κριτήρια που να απαγορεύουν σε δύο ακμές της καμπύλης να εφάπτονται. Υπάρχουν όμως όλοι οι απαραίτητοι περιορισμοί, ώστε να μην τέμνει τον εαυτό της (βλέπε Ενότητα 3.3).

Σημείο αναφοράς στην κατασκευή της καμπύλης είναι το διάγραμμα Voronoi, από το

οποίο αντλούμε τις πληροφορίες γειτνίασης που χρειαζόμαστε για να βρίσκουμε από πού θα περνάει η καμπύλη για να ανακαλύπτει νέα αντικείμενα στο εσωτερικό της. Περισσότερες λεπτομέρειες για τα διαγράμματα Voronoi και της χρησιμότητάς τους στην εργασία αυτή, αναλύονται στο Κεφάλαιο 2.

Μέχρι τώρα έχουν κατασκευαστεί στα [44, 25] περιβάλλουσες καμπύλες οι οποίες ακουμπούσαν και σε σημεία αντικειμένων τα οποία ήταν κοντά στη θήκη, αλλά όχι σε αντικείμενα στο εσωτερικό της. Επίσης οι υλοποιήσεις που έχουν γίνει έως τώρα αφορούν μόνο κύκλους, ενώ εδώ επεκτεινόμαστε σε γενικότερα πολυγωνικά κυρτά αντικείμενα, τα οποία έχουν διακριτοποιηθεί.

Κεφάλαιο 2

Διαγράμματα Voronoi

Πριν αρχίσουμε να δίνουμε μαθηματικούς ορισμούς του διαγράμματος, ας εξηγήσουμε με απλά λόγια τι είναι ένα διάγραμμα Voronoi, με τη βοήθεια του παρακάτω παραδείγματος. Ας φανταστούμε μία πόλη με σχολεία. Αν το κράτος ήθελε να φτιάξει ένα νέο σχολείο θα έπρεπε να υπολογίσει το πλήθος των μαθητών που αυτό θα προσέλκυε. Ένα σημαντικό κριτήριο με το οποίο οι μαθητές θα το επέλεγαν είναι η απόσταση. Δεδομένης λοιπόν της πόλης και των σχολείων, πρέπει με κάποιο τρόπο να αποφασίσουμε ποιες περιοχές εξυπηρετούνται από το κάθε σχολείο.

Τα σχολεία αντιστοιχούν στα κέντρα Voronoi. Κάθε μαθητής πηγαίνει στο πλησιέστερο σχολείο, οπότε η περιοχή που καλύπτει το καθένα αποτελείται από σημεία των οποίων η απόσταση από το συγκεκριμένο σχολείο είναι μικρότερη από τις αποστάσεις του από όλα τα υπόλοιπα. Η διαμέριση του συνόλου σε αυτές τις περιοχές αποτελεί το διάγραμμα Voronoi (βλέπε Σχήμα 2.1).

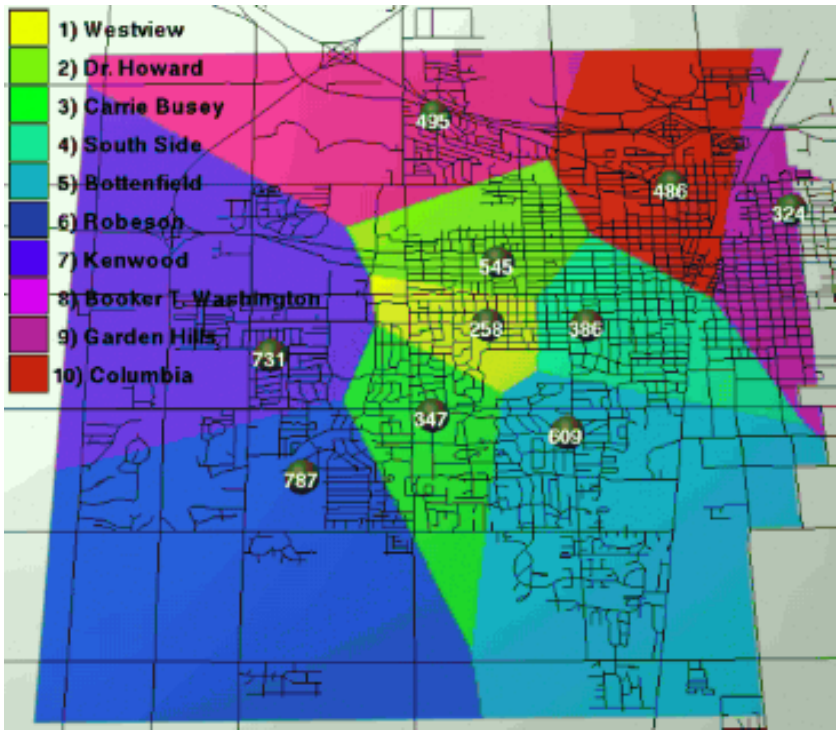
Το παραπάνω παράδειγμα είναι μία εφαρμογή του Voronoi διαγράμματος στον τομέα της γεωγραφίας, αλλά υπάρχουν εφαρμογές του σε πολλούς ακόμα τομείς, όπως η βιολογία, η χημεία, η κρυσταλλογραφία, η μετεωρολογία, η ρομποτική, η γραφική, η υπολογιστική όραση κ.ά.

Όπως θα εξηγηθεί και στη συνέχεια, το διάγραμμα Voronoi του παραδείγματος αναφέρεται σε σημεία. Υπάρχουν όμως πολλές ακόμα παραλλαγές, οι οποίες αναλύονται στο υπόλοιπο του Κεφαλαίου.

Παρακάτω θα ορίσουμε πρώτα το αφηρημένο (abstract) διάγραμμα Voronoi και στη συνέχεια θα καταγράψουμε διάφορες ειδικές περιπτώσεις του. Στο τέλος του Κεφαλαίου, θα αναφερθούμε και σε διαγράμματα Voronoi που δεν ανήκουν στην κατηγορία των αφηρημένων διαγραμμάτων Voronoi.

2.1 Αφηρημένο διάγραμμα Voronoi

Έστω για κάθε ζεύγος ακεραίων p, q το $D(p,q)$ το οποίο είναι είτε κενό, είτε ανοικτό μη φραγμένο υποσύνολο του $\mathbb{R}^2$, όπου $n \in \mathbb{N}$ και $1 \leq p \neq q < n$. Έστω ακόμα $J(p,q)$ το



Σχήμα 2.1: Voronoi διάγραμμα όπου τα κέντρα αντιστοιχούν σε σχολεία [45].

σύνορο του $D(p,q)$. Προκύπτουν τα εξής [32]:

1. $J(p,q) = J(q,p)$ και για κάθε p, q με $p \neq q$, οι περιοχές $D(p,q)$, $J(p,q)$ και $D(q,p)$ αποτελούν διαμέριση του $\mathbb{R}^2$ σε τρία ξένα σύνολα.

2. Αν $\varnothing \neq D(p,q) \neq \mathbb{R}^2$ τότε το $J(p,q)$ είναι ομοιομορφικό με το ανοικτό διάστημα $(0,1)$.

Το $J(p,q)$ το ονομάζουμε *διχοτόμο καμπύλη* για τα αντικείμενα p και q και το $D(p,q)$ *περιοχή κυριαρχίας* του p πάνω στο q. Έτσι, το αφηρημένο διάγραμμα Voronoi ορίζεται ως εξής:

Ορισμός 1. *Έστω* $S = \{1, \ldots, n-1\}$ *και*

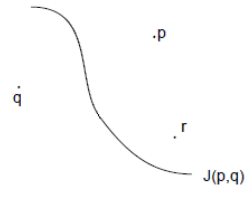
$$
\begin{aligned}
R(p,q) &= \begin{cases} D(p,q) \bigcup J(p,q) & \text{αν } p < q, \\ D(p,q) & \text{αν } p > q \end{cases} \\
EVR(p,S) &= \bigcap_{\substack{q \in S \\ q \neq p}} R(p,q) \\
VR(p,S) &= EVR^{\circ}(p,S) \\
V(S) &= \bigcup_{p \in S} \vartheta EVR(p,S)
\end{aligned}
\tag{2.1}
$$

Το $VR(p,S)$ ονομάζεται *περιοχή Voronoi* του p ή p-περιοχή ως προς το S. Το $EVR(p,S)$ ονομάζεται *γενικευμένη περιοχή Voronoi* του p ως προς το S και το $V(S)$

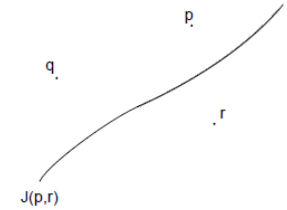
ονομάζεται *διάγραμμα Voronoi*. Τα στοιχεία του S ονομάζονται κέντρα.

Κάποιες από τις σημαντικότερες ιδιότητες του αφηρημένου διαγράμματος Voronoi είναι οι εξής:

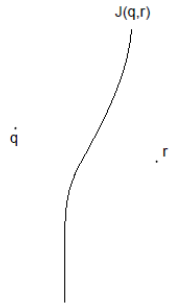
- Το αφηρημένο διάγραμμα Voronoi έχει γραμμικό μέγεθος, σε σχέση με το πλήθος των κέντρων του.
- Σε κάθε περιοχή Voronoi αντιστοιχεί ακριβώς ένα κέντρο.
- Οι περιοχές Voronoi του διαγράμματος είναι απλά συνεκτικά χωρία.
- Το διάγραμμα Voronoi είναι επίπεδο.



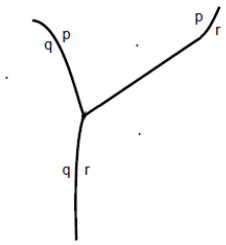
(α′) Διχοτόμος καμπύλη μεταξύ των p και q.



(β′) Διχοτόμος καμπύλη μεταξύ των p και r.



(γ′) Διχοτόμος καμπύλη μεταξύ των r και q.



(δ′) Αφηρημένο διάγραμμα Voronoi για τα p, q και r.

Σχήμα 2.2: Στάδια δημιουργίας του αφηρημένου διαγράμματος Voronoi για τα αντικείμενα p, q και r.

Στο Σχήμα 2.2 φαίνεται η κατασκευή ενός αφηρημένου διαγράμματος Voronoi από 3 αντικείμενα.

Μία μέθοδος κατασκευής για το αφηρημένο διάγραμμα Voronoi έχει προταθεί από τον Klein [38], η οποία βασίζεται στην τεχνική διαίρει και βασίλευε, ενώ μία δεύτερη από τους Klein, Mehlhorn και Meiser [32], η οποία χαρακτηρίζεται ως πιθανοθεωρητική αυξητική μέθοδος.

Η τεχνική «διαίρει και βασίλευε» έχει ως εξής: χωρίζεται το σύνολο αντικειμένων S, μέσω μίας διαχωριστικής γραμμής, σε δύο υποσύνολα L και R περίπου ιδίου μεγέθους.

Τα διαγράμματα Voronoi των L και R υπολογίζονται αναδρομικά. Οπότε τα σημαντικά μέρη του αλγορίθμου είναι ο υπολογισμός της διαχωριστικής γραμμής και η ένωση των δύο Voronoi διαγραμμάτων σε ένα, τα οποία γενικά χρειάζονται χρόνο $O(n)$. Λόγω όμως της αναδρομής, αν $T(n)$ ο συνολικός χρόνος του αλγορίθμου, τότε $T(n) = 2T(\frac{n}{2}) + O(n)$, το οποίο δίνει $T(n) = O(n \log n)$.

Η πιθανοθεωρητική αυξητική μέθοδος χρειάζεται αναμενόμενο χρόνο $O(n \log n)$. Στη μέθοδο αυτή, αντί να εισάγονται στο διάγραμμα Voronoi ένα προς ένα τα αντικείμενα, μέχρι αυτό να ολοκληρωθεί[1], τα αντικείμενα εισάγονται τυχαία. Η διαφορά μεταξύ της απλής αυξητικής μεθόδου και της πιθανοθεωρητικής αυξητικής μεθόδου μπορεί να γίνει εύκολα αντιληπτή, αν τις παραλληλίσουμε με τις μεθόδους ταξινόμησης insertion sort και quicksort, όπως εξηγούν και οι Clarkson και Shor [12]. Η πρώτη ταξινομεί τα στοιχεία συγκρίνοντας το καθένα με όλα τα προηγούμενα, με αποτέλεσμα να είναι αρκετά αργή για μεγάλα δεδομένα. Η δεύτερη επιλέγει τα στοιχεία που θα εισάγει με τυχαίο τρόπο και αυτό εξασφαλίζει, στην πλειοψηφία των περιπτώσεων, ότι τα δεδομένα βοηθάνε για να πετύχουμε τον αναμενόμενο χρόνο $O(n \log n)$.

Στις Υποενότητες 2.2 έως 2.7 γίνεται αναφορά σε διαφόρων ειδών διαγράμματα Voronoi τα οποία είναι υποπεριπτώσεις του αφηρημένου, και συνεπώς ο ορισμός τους ταυτίζεται με τον ορισμό του αφηρημένου διαγράμματος Voronoi.

2.2 Διάγραμμα Voronoi σημείων

Εκτός από τον αφηρημένο ορισμό, μπορούμε να ορίσουμε το διάγραμμα Voronoi, ειδικά για σημεία:

Έστω S ένα σύνολο n σημείων στο επίπεδο, $n \geq 3$. Για 2 σημεία $p = (p_1, p_2)$ και $x = (x_1, x_2)$ του επιπέδου, η Ευκλείδειά τους απόσταση ορίζεται ως:

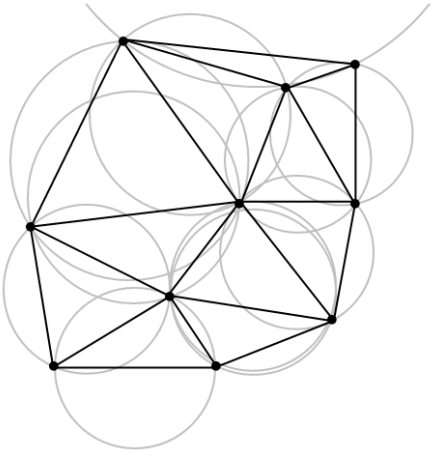
$$\delta(p, x) = \sqrt{(p_1 - x_1)^2 + (p_2 - x_2)^2} \tag{2.2}$$

Εξ' ορισμού κάθε Voronoi περιοχή $VR(p, S)$ είναι η τομή των $n - 1$ ημιεπιπέδων που περιέχουν το σημείο p. Συνεπώς η $VR(p, S)$ είναι ανοικτό σύνολο, κυρτή και δεν επικαλύπτεται με καμία από τις υπόλοιπες Voronoi περιοχές.

Θεωρούμε το σύνορο των περιοχών Voronoi, το οποίο αποτελεί το σκελετό Voronoi. Το κοινό σύνορο δύο περιοχών Voronoi, το οποίο περιέχει δύο ακριβώς κέντρα Voronoi ονομάζεται *ακμή Voronoi*, ενώ αν περιέχει περισσότερα από δύο, ονομάζεται *σημείο Voronoi*.

Παρακάτω αναφέρονται ορισμένες ιδιότητες των διαγραμμάτων Voronoi, ενός συνόλου S που αποτελείται από n το πλήθος σημεία στο επίπεδο.

[1] Έτσι δρα η αυξητική μέθοδος και χρειάζεται χρόνο $O(n^2)$.



Σχήμα 2.3: Τριγωνοποίηση Delaunay στην οποία φαίνονται οι περιγεγραμμένοι κύκλοι κάθε τριγώνου [20].

1. Το σύνολο που αποτελείται από την ένωση των περιοχών Voronoi με τις ακμές και τα σημεία, αποτελεί διαμέριση του επιπέδου.

2. Αν και τα n σημεία του S είναι συγγραμμικά, τότε το διάγραμμα Voronoi αποτελείται από $n-1$ παράλληλες γραμμές. Αλλιώς, ο σκελετός του διαγράμματος Voronoi του S είναι συνεκτικός και οι ακμές του είναι είτε ευθύγραμμα τμήματα ή ημιευθείες.

3. Ένα σημείο p του S βρίσκεται στην κυρτή θήκη του S αν και μόνο αν η Voronoi περιοχή του $VR(p, S)$ δεν είναι φραγμένη.

4. Το διάγραμμα Voronoi $V(S)$ έχει $O(n)$ το πλήθος ακμές και $O(n)$ το πλήθος κόμβους. Ο μέσος αριθμός ακμών που βρίσκονται στο σύνορο μίας περιοχής Voronoi είναι μικρότερος από 6.

5. Για $n \geq 3$, το πλήθος των κόμβων του διαγράμματος Voronoi του S είναι το πολύ $2n-5$ και το πλήθος των ακμών του είναι το πολύ $3n-6$.

6. Αν δεν υπάρχουν 4 συγγραμμικά σημεία στο S, τότε το δυϊκό $DT(S)$ του διαγράμματος Voronoi, είναι μία τριγωνοποίηση του S που ονομάζεται *τριγωνοποίηση Delaunay*. (Δυϊκό ενός γραφήματος G είναι το γράφημα το οποίο έχει ένα σημείο για κάθε χωρίο του G, και μία ακμή για κάθε ακμή του G που ενώνει δύο γειτονικά χωρία.) Στην τριγωνοποίηση αυτή, δύο σημεία του S ενώνονται από μία ακμή της αν και μόνο αν οι Voronoi περιοχές τους είναι γειτονικές.

7. Κανένα σημείο της τριγωνοποίησης Delaunay δεν περιέχεται σε κανένα από τους περιγεγραμμένους κύκλους των τριγώνων της τριγωνοποίησης (βλέπε Σχήμα 2.3).

8. Μεταξύ όλων των πιθανών τριγωνοποιήσεων του S, η Delaunay είναι αυτή που μεγιστοποιεί τις ελάχιστες γωνίες των τριγώνων (δηλαδή αποφεύγει τα πολύ «λεπτά» τρίγωνα).

Οι αποδείξεις των παραπάνω ιδιοτήτων βρίσκονται στο [4], εκτός από τις ιδιότητες (i) και (iv) των οποίων η απόδειξη βρίσκεται στο [14].

Εκτός από αυτές τις ιδιότητες, ισχύουν φυσικά και οι ιδιότητες που αναφέρθηκαν στα αφηρημένα διαγράμματα Voronoi.

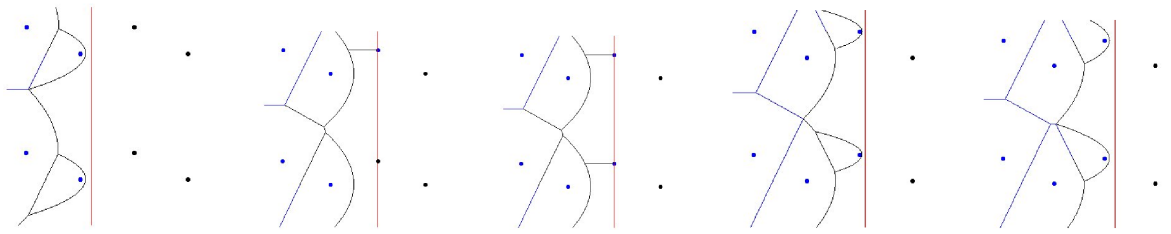
Πολλοί αλγόριθμοι έχουν παρουσιαστεί για την κατασκευή του Voronoi διαγράμματος και μπορούν να χωριστούν σε 5 κατηγορίες (βλέπε [4, 46]),αναλόγως της μεθόδου την οποία χρησιμοποιούν:

Αυξητική μέθοδος Κάθε φορά προκύπτει το $V(S)$ από το $V(S\backslash\{p\})$, εισάγοντας το σημείο p. Στις αυξητικές μεθόδους (και στις παραλλαγές τους, όπως οι πιθανοθεωρητικές αυξητικές μέθοδοι) η κεντρική ιδέα είναι ότι δεδομένου του νέου σημείου p ζητείται να βρεθεί το κομμάτι του διαγράμματος Voronoi που καταστρέφεται εξαιτίας του p ώστε στη συνέχεια να ανακατασκευαστεί το διάγραμμα γύρω από το p. Σε επίπεδο τριγωνοποίησης Delaunay, η εισαγωγή του νέου σημείου συνοδεύεται από καταστροφή τριγώνων που υπάρχουν στην τριγωνοποίηση Delaunay. Τα τρίγωνα που καταστρέφονται δημιουργούν μία πολυγωνική οπή στην τριγωνοποίηση, τα σημεία του οποίου συνδέονται με το νέο σημείο με ακμές δημιουργώντας έτσι νέα τρίγωνα και την καινούργια πλέον τριγωνοποίηση Delaunay. Το ζητούμενο στους αλγορίθμους αυτούς είναι ένα καλό σημείο εκκίνησης προκειμένου να ανακαλυφθούν όλα τα τρίγωνα της τριγωνοποίησης που πρέπει να καταστραφούν. Ένα τέτοιο τρίγωνο είναι το τρίγωνο που περιέχει το νέο σημείο p, ενώ αν $nn(p)$ ο κοντινότερος γείτονας του p στο σύνολο των σημείων που ήδη έχουν εισαχθεί στην τριγωνοποίηση, αποδεικνύεται ότι τουλάχιστον ένα εκ των τριγώνων που γειτνιάζει με το σημείο $nn(p)$ πρέπει να καταστραφεί λόγω της εισαγωγής του p. Οι διάφοροι αλγόριθμοι αυξητικού χαρακτήρα στηρίζονται σε αυτές ακριβώς τις παρατηρήσεις και διαφοροποιούνται ως προς το πώς βρίσκουν το σημείο εκκίνησης (και ενδεχομένως στο πώς κάνουν την ανακατασκευή της τριγωνοποιήσης). Ο χρόνος εκτέλεσης τέτοιων αλγορίθμων είναι $O(n^2)$. Πρώτοι οι Green και Sibson [27] μελέτησαν αυτή την ιδέα, η οποία στη συνέχεια βελτιώθηκε από άλλους (π.χ. Sugihara και Iri [43]).

Πιθανοθεωρητική αυξητική μέθοδος Ένας τέτοιος αλγόριθμος περιγράφηκε ήδη στην προηγούμενη υποενότητα στα πλαίσια των αφηρημένων διαγραμμάτων Voronoi. Τρέχει σε αναμενόμενο χρόνο $O(n \log n)$ ο οποίος είναι καλύτερος από τον αντίστοιχο της αυξητικής μεθόδου. Στον αλγόριθμο αυτό τα σημεία εισάγονται με τυχαία σειρά, και οι δομές δεδομένων που χρησιμοποιεί είναι απλούστερες. Λεπτομέρειες μπορεί κανείς να βρει στο [36]. Μία άλλη πιθανοθεωρητική μέθοδος είναι η jump-and-walk [17, 18]. Η μέθοδος αυτή επιλέγει με τυχαίο τρόπο ένα «μικρό» υποσύνολο των σημείων που ήδη έχουν εισαχθεί στο διάγραμμα. Προκειμένου να βρεθεί

ο κοντινότερος γείτονας του νέου σημείου, η μέθοδος ξεκινά διαλέγοντας (με brute-force τρόπο) τον κοντινότερο γείτονα μεταξύ των «λίγων» επιλεγμένων σημείων και στη συνέχεια από το σημείο που επελέγη μεταξύ των λίγων, περπατάμε από τρίγωνο σε τρίγωνο πάνω στην τριγωνοποίηση με κατεύθυνση τη θέση του νέου σημείου, ανανεώνοντας κάθε φορά την πληροφόρησή μας για το ποιος είναι ο κοντινότερος γείτονας. Η προσέγγιση αυτή οδηγεί σε αλγόριθμο που κατασκευάζει το διάγραμμα Voronoi σε αναμενόμενο χρόνο $O(n^{4/3})$, με πολύ καλή συμπεριφορά στην πράξη για δεδομένα μεσαίου μεγέθους. Στο [16] παρουσιάζεται μια παρεμφερής ιδέα: αντί να επιλέγουμε ένα «μικρό» σύνολο από βοηθητικά σημεία, η εύρεση του κοντινότερου γείτονα οργανώνεται σε πολλά επίπεδα. Σε κάθε επίπεδο υπάρχει η τριγωνοποίηση ενός ποσοστού των σημείων που υπήρχαν στο προηγούμενο επίπεδο (το αρχικό επίπεδο περιέχει όλα τα σημεία, ενώ το τελευταίο επίπεδο περιέχει σταθερό αριθμό από σημεία). Η αναζήτηση του κοντινότερου γείτονα του νέου σημείου γίνεται διαδοχικά σε κάθε επίπεδο, από το τελευταίο προς το πρώτο, ξεκινώντας από τον κοντινότερο γείτονα του προηγούμενου επιπέδου (ο έλεγχος στο τελευταίο επίπεδο γίνεται με brute-force τρόπο) και περπατώντας πάνω στα τρίγωνα της τριγωνοποίησης. Ο αλγόριθμος αυτός επιτυγχάνει τον υπολογισμό της τριγνωνοποίησης Delaunay σε αναμενόμενο χρόνο $O(n \log n)$.

Μέθοδος με γραμμή σάρωσης (sweepline) Υπάρχει μία κατακόρυφη (ή οριζόντια, αναλόγως τον αλγόριθμο) ευθεία γραμμή η οποία διατρέχει όλο το επίπεδο από τη μία άκρη έως την άλλη. Όσο προχωράει αυτή η γραμμή, συμβαίνουν αλλαγές κάθε φορά που συναντάται κάποιο σημείο, όπως δείχνουν οι εικόνες 2.4(α′)-2.4(γ′) και κάθε φορά που ένα τόξο εξαφανίζεται, όπως δείχνουν οι εικόνες 2.4(δ′), 2.4(ε′). Ο χρόνος που χρειάζεται αυτή η μέθοδος είναι $O(n \log n)$ (βλέπε Fortune [24]).



(α′) Αρχή της δημιουργίας του διαγράμματος.

(β′) Εμφάνιση νέου σημείου.

(γ′) Εμφάνιση και δεύτερου νέου σημείου (στην ίδια ευθεία).

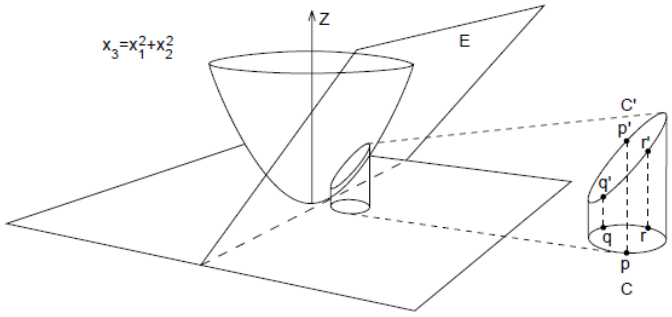
(δ′) Αντιμετωπίστηκε η εμφάνιση των δύο νέων σημείων.

(ε′) Ένα τόξο εξαφανίστηκε.

Σχήμα 2.4: Στάδια από τη δημιουργία ενός διαγράμματος Voronoi σημείων με τη μέθοδο sweepline (στιγμιότυπα από το applet [37]).

Διαίρει και βασίλευε Η διαδικασία περιγράφηκε στην προηγούμενη ενότητα (βλέπε [40, 28]). Ο αλγόριθμος αυτός, όπως και ο sweepline χρειάζεται $O(n \log n)$ χρόνο εκτέλεσης.

Μέθοδος gift wrapping Σε αυτή τη μέθοδο δημιουργείται η τριγωνοποίηση Delaunay, και από εκεί μπορεί να προκύψει προφανώς το διάγραμμα Voronoi. Κάθε νέο τρίγωνο δημιουργείται από μία ακμή ενός τριγώνου που έχει ήδη ανακαλυφθεί. Το κλειδί σε αυτή τη μέθοδο είναι να ανακαλυφθεί κάθε φορά το κατάλληλο τρίγωνο, ώστε να γίνονται οι έλεγχοι όσο το δυνατόν γρηγορότερα. Ο Dwyer [21] πρότεινε ένα σχετικό αλγόριθμο, ο οποίος έχει αναμενόμενο χρόνο εκτέλεσης $O(n)$.



Σχήμα 2.5: Μετασχηματισμός των κύκλων σε παραβολοειδή [4].

Μέθοδος μέσω μετασχηματισμού από το επίπεδο στο χώρο Ορίζεται ο μετασχηματισμός $\mathbb{R}^2 \to \mathbb{R}^3$ ο οποίος στέλνει κάθε σημείο (x_1, x_2) στο σημείο $(x_1, x_2, x_1^2 + x_2^2)$ (βλέπε Σχήμα 2.5). Όμως αν S' η εικόνα του S μέσω του παραπάνω μετασχηματισμού, τότε η τριγωνοποίηση Delaunay του S ισούται με την προβολή της κατώτερης κυρτής θήκης[2] του S' στο επίπεδο [4]. Ο υπολογισμός της κυρτής θήκης για 3 διαστάσεις υπολογίζεται σε αναμενόμενο χρόνο $O(n \log n)$ [11]. Στο [8] δίνεται ένας αλγόριθμος του οποίου ο χρόνος εξαρτάται από το πλήθος f των χωρίων που περικλείει η κυρτή θήκη, και ισούται με $O(nf)$, με χειρότερη περίπτωση σε $O(n^2)$.

Φυσικά το διάγραμμα Voronoi για σημεία, μπορεί να υπολογιστεί και για γενικότερες μετρικές εκτός από την Ευκλείδεια απόσταση L_2, όπως η L_p [33], όπου:

$$L_p(q, r) = (|q_1 - r_1|^p + |q_2 - r_2|^p)^{\frac{1}{p}}, \quad q = (q_1, q_2),\ r = (r_1, r_2),\ 1 \leq p < \infty \tag{2.3}$$

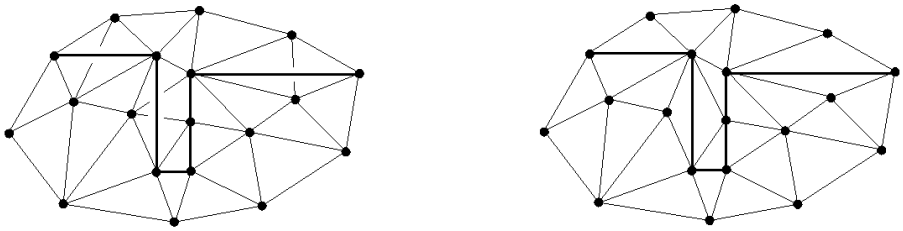
Μία γενίκευση του διαγράμματος Voronoi σημείων είναι το *k-τάξης* διάγραμμα Voronoi, όπου $k = \{1, 2, \ldots, n-1\}$. Το διάγραμμα αυτό εμφανίζει την ιδιότητα, τα σημεία κάθε περιοχής να απέχουν από τα ίδια k κέντρα [22]. Συνεπώς, για $k = 1$ έχουμε το κλασσικό διάγραμμα Voronoi σημείων.

Τα k-τάξης διαγράμματα στο επίπεδο μπορούν να κατασκευαστούν σε αναμενόμενο χρόνο $O(nk^2 \log n + nk \log^3 n)$ [5].

[2]Κατώτερη κυρτή θήκη του S' είναι το τμήμα της κυρτής θήκης το οποίο είναι ορατό από παρατηρητή ο οποίος βρίσκεται στο $-\infty$ του z-άξονα.

2.3 Περιορισμένο διάγραμμα Voronoi

Πολλές φορές κάποια σημεία παρόλο που είναι γεωγραφικώς γειτονικά, δεν μπορούν να θεωρηθούν γείτονες. Παραδείγματος χάριν στο χάρτη μίας χώρας αν υπάρχει ένα βουνό ανάμεσα σε δύο διπλανές πόλεις, δε θα σχεδιάσουμε τη διαδρομή ενός αυτοκινήτου να πηγαίνει απ΄ ευθείας από τη μία στην άλλη, διότι προφανώς αυτό δεν είναι εφικτό να γίνει. Όταν εμφανίζονται τέτοιου είδους περιορισμοί τότε υπάρχει η ανάγκη για το *περιορισμένο διάγραμμα Voronoi* [9]. Για να κατασκευαστεί δηλαδή αυτό το διάγραμμα, στην είσοδο, εκτός από τα σημεία, δίνονται και οι περιοριστικές ακμές ή πιο σωστά ένα επίπεδο γραμμικό γράφημα (planar straight-line graph). Χρησιμοποιώντας χωρίς ιδιαίτερες αλλαγές τη μέθοδο της γραμμής σάρωσης που περιγράφεται στην Ενότητα 2.2, η κατασκευή του γίνεται σε χρόνο $O(n \log n)$ [39].



Σχήμα 2.6: Τριγωνοποίηση Delaunay και περιοριστικές ακμές που δεν ανήκουν σε αυτήν (αριστερά) και περιορισμένη τριγωνοποίηση Delaunay (δεξιά).

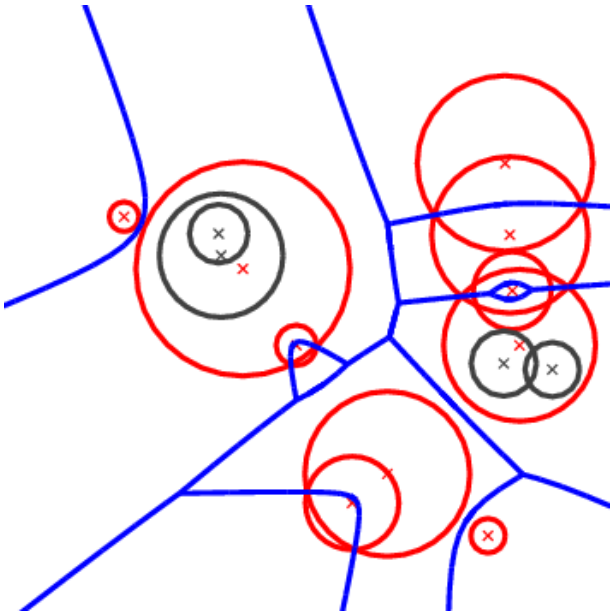
Στην ειδική περίπτωση που, αντί για επίπεδο γραμμικό γράφημα, ενδιαφερόμαστε να υπολογίσουμε την τριγωνοποίηση Delaunay στο εσωτερικό ενός πολυγώνου, οι ακμές του οποίου θεωρούνται ως περιορισμοί, είναι δυνατόν να κάνουμε κάτι καλύτερο σε σχέση με τη γενική περίπτωση. Στο [10] παρουσιάζεται ένας αλγόριθμος υπολογισμού της περιορισμένης τριγωνοποίησης Delaunay, ο οποίος σε χρόνο $O(\log n + k)$ ανανεώνει την τριγωνοποίηση προσθέτοντας τους k περιορισμούς. Ο αλγόριθμος αυτός βασίζεται στα αποτελέσματα του [1] στο οποίο προστίθενται ή αφαιρούνται k ακμές σε ένα διάγραμμα Voronoi σε χρόνο $O(\log n + k)$.

2.4 Το διάγραμμα του Απολλωνίου

Στο διάγραμμα του Απολλώνιου, το κάθε σημείο p του συνόλου S συνοδεύεται από ένα βάρος $w(p)$, το οποίο αντιπροσωπεύει την ικανότητα του σημείου p να επηρεάζει τη «γειτονιά» του. Αν το βάρος είναι θετικό, τότε μπορούμε να φανταστούμε το p, $w(p)$ ως έναν κύκλο, με κέντρο το σημείο p και ακτίνα $w(p)$ (βλέπε Σχήμα 2.7). Η μετρική που χρησιμοποιείται για την κατασκευή αυτού του διαγράμματος είναι η:

$$\delta(x, \{p, w(p)\}) = ||x - p|| - w(p) \tag{2.4}$$

Αν τα βάρη των σημείων του S έχουν όλα την ίδια τιμή, τότε το διάγραμμα δύναμης του S ταυτίζεται με το Voronoi διάγραμμα του S.



Σχήμα 2.7: Το διάγραμμα του Απολλώνιου φαίνεται με μπλε. Οι κύκλοι (κόκκινοι και μαύροι) συμβολίζουν τα σημεία (κέντρα των κύκλων) και τα αντίστοιχα βάρη (ακτίνες των κύκλων) [30]. Οι μαύροι κύκλοι αντιστοιχούν σε βεβαρημένα σημεία με κενή περιοχή Voronoi.

Στο [41] υπάρχει ένας αλγόριθμος που βασίζεται στην τεχνική «διαίρει και βασίλευε», ο οποίος κατασκευάζει το διάγραμμα του Απολλώνιου σε χρόνο $O(n \log^2 n)$. Στο [24] χρησιμοποιείται η τεχνική της γραμμής σάρωσης δίνοντας χρόνο $O(n \log n)$. Στο [29] κατασκευάζεται το διάγραμμα, με δυναμικό αλγόριθμο ο οποίος χρησιμοποιεί την τριγωνοποίηση Delaunay και αναμενόμενο χρόνο $O(n \log h)$, όπου h το πλήθος των βεβαρημένων σημείων που έχουν μη κενή περιοχή Voronoi.

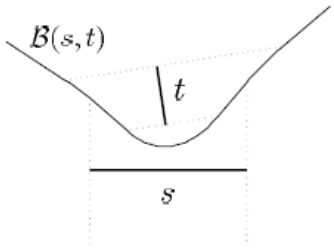
2.5 Διάγραμμα Voronoi ευθυγράμμων τμημάτων

Εκτός από σημεία, θα μπορούσαμε να σκεφτούμε και άλλα αντικείμενα που θα εκπροσωπούσαν τα κέντρα Voronoi. Σε αυτή την ενότητα αναφέρεται η περίπτωση των ευθυγράμμων τμημάτων. Έστω G το σύνολο των αντικειμένων, τα οποία αυτή τη φορά θα είναι ευθύγραμμα τμήματα και τα σημεία που βρίσκονται στα άκρα τους. Δύο ευθύγραμμα τμήματα δεν επιτρέπεται να τέμνονται στο εσωτερικό τους, αλλά μόνο να έχουν κοινό άκρο.

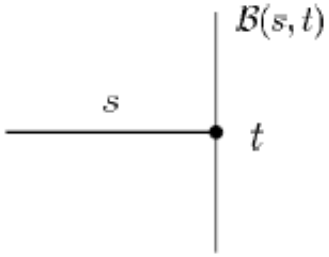
Αν τα s, t, δύο αντικείμενα που είναι είτε ευθύγραμμα τμήματα ή σημεία και έχουν Ευκλείδεια απόσταση δ, τότε διχοτόμος καμπύλη των s και t είναι η $B(s,t)$, όπου:

$$B(s,t) = B(t,s) = \{x \in \mathbb{R}^2 \,|\, \delta(x,s) = \delta(x,t)\} \tag{2.5}$$

Στο Σχήμα 2.8(α′) φαίνεται η διχοτόμος δύο ευθυγράμμων τμημάτων s και t. Ενώ αν



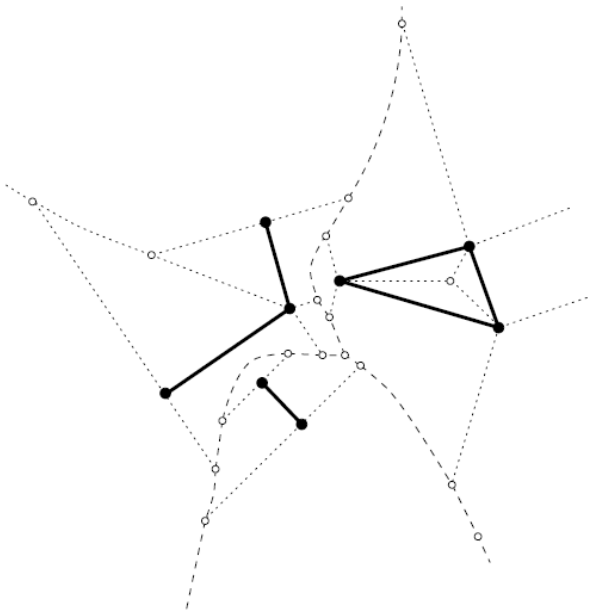
(α′) Διχοτόμος $B(s,t)$ των ευθυγράμμων τμημάτων s και t.



(β′) Διχοτόμος $B(s,t)$ του ευθυγράμμου τμήματος s και του σημείου t.

Σχήμα 2.8: Παραδείγματα διχοτόμου αντικειμένων s και t [6].

s είναι το άκρο του ευθυγράμμου τμήματος t, τότε η διχοτόμος τους φαίνεται στο Σχήμα 2.8(β′).



Σχήμα 2.9: Voronoi διάγραμμα ευθυγράμμων τμημάτων [4].

Για να μη δημιουργούνται διδιάστατες διχοτόμοι, τα άκρα κάθε ευθυγράμμου τμήματος θεωρούνται πάντα ότι είναι διαφορετικά αντικείμενα από το εσωτερικό του. Με τη θεώρηση αυτή το Ευκλείδειο διάγραμμα Voronoi ευθυγράμμων τμημάτων αποτελεί ειδική περίπτωση αφηρημένου διαγράμματος Voronoi.

Το $B(s,t)$ διαιρεί το επίπεδο σε δύο μη επικαλυπτόμενα ημιεπίπεδα, το ένα είναι το $D(s,t)$ που περιέχει το αντικείμενο s και το άλλο είναι το $D(t,s)$ που περιέχει το t, όπου:

$$D(s,t) = \{x \in \mathbb{R}^2 : \delta(x,s) < \delta(x,t)\} \tag{2.6}$$

Τα Voronoi σημεία και ευθείες ορίζονται με τον ίδιο τρόπο όπως και στο διάγραμμα Voronoi σημείων [6].

Και εδώ, όπως και στο διάγραμμα Voronoi σημείων, οι αλγόριθμοι που έχουν δημοσιευθεί είναι πολλοί, οπότε και πάλι μία ομαδοποίηση[3] τους, ως προς τη μέθοδο που χρησιμοποιούν θα ήταν πολύ χρήσιμη.

Πιθανοτική αυξητική μέθοδος Παρουσιάστηκε σε προηγούμενες ενότητες (βλέπε [32]) και χρειάζεται αναμενόμενο χρόνο $O(n \log n)$. Στο [31] παρουσιάζεται ένας αλγόριθμος, ο οποίος με χρήση πιθανοτικής αυξητικής μεθόδου κατασκευάζει το διάγραμμα Voronoi για ευθύγραμμα τμήματα, τα οποία μπορούν είτε να είναι ξένα, είτε να έχουν κοινά άκρα, είτε ακόμα και να τέμνονται στο εσωτερικό τους. Ο αναμενόμενος χρόνος κατασκευής του διαγράμματος είναι $O((n+m)\log^2 n)$, όπου m είναι το πλήθος των εσωτερικών σημείων τομής των ευθυγράμμων τμημάτων.

Μέθοδος με γραμμή σάρωσης (sweepline) Όμοια με την αντίστοιχη στο διάγραμμα σημείων (βλέπε Fortune [24]). Το διάγραμμα Voronoi των ευθυγράμμων τμημάτων υπολογίζεται σε χρόνο $O(n \log n)$.

Μέθοδος με κινούμενα σημεία αντικειμένων Κατασκευάζεται αρχικά το διάγραμμα Voronoi για σημεία, επιλέγοντας το ένα άκρο από κάθε ευθύγραμμο τμήμα, και μετά δημιουργείται το τελικό διάγραμμα επεκτείνοντας τα σημεία αυτά των άκρων, ένα προς ένα, προς τα αντίστοιχα ευθύγραμμα τμήματά τους Gold et al. [26]. Ο συγκεκριμένος αλγόριθμος στη χειρότερη περίπτωση απαιτεί χρόνο $O((n+t)\log n)$, όπου t είναι ο αριθμός των τοπολογικών αλλαγών που εμφανίζονται κατά τη διάρκεια των επεκτάσεων των σημείων σε ευθύγραμμα τμήματα, με $t = O(n^2)$. Στην πράξη ο παραπάνω αλγόριθμος φαίνεται να τρέχει σε σχεδόν γραμμικό χρόνο.

2.6 Διάγραμμα Voronoi ξένων ανά δύο κυρτών πολυγώνων

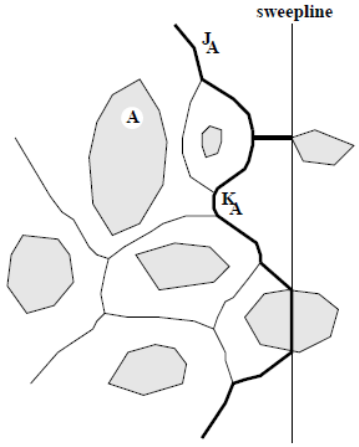
Αν αντί για σημεία ή ευθύγραμμα τμήματα, έχουμε κυρτά πολύγωνα, μπορούμε και πάλι να ορίσουμε το διάγραμμα Voronoi. Η ιδέα για κάτι τέτοιο προέκυψε από τον τομέα της Ρομποτικής, στο πρόβλημα σχεδίασης κίνησης (motion planning), όπου εκεί τα φυσικά εμπόδια συνήθως δεν είναι ούτε σημεία, ούτε ευθείες.

Έστω p ένα σημείο και A ένα φραγμένο κυρτό αντικείμενο στο Ευκλείδειο επίπεδο $\mathbb{E}^2$. Ορίζουμε την απόσταση $\delta(p, A)$ από το p στο A ως:

$$\delta(p,A) = \begin{cases} \min_{x \in \partial A} ||p-x||, & p \notin A \\ -\min_{x \in \partial A} ||p-x||, & p \in A \end{cases} \tag{2.7}$$

[3] Ο τρόπος δράσης των τριών πρώτων μεθόδων είναι αντίστοιχος με αυτούς που αναλύθηκαν στην Υποενότητα 1.2.2, γι΄ αυτό δίνονται μόνο αναφορές.

όπου ∂A το σύνορο του A και $||\cdot||$ η Ευκλείδεια νόρμα.



Σχήμα 2.10: Κατασκευή διαγράμματος Voronoi για πολύγωνα, με γραμμή σάρωσης [34].

Είχαν γίνει και πριν το 1996 κάποιες προσεγγίσεις για τέτοιου είδους διαγράμματα, αλλά ήταν κυρίως διαγράμματα αρκετά γενικευμένα, έτσι ώστε να καλύπτουν υπό προϋποθέσεις αυτήν την περίπτωση. Στο [34] παρουσιάζεται ένας αλγόριθμος φτιαγμένος για κυρτά πολυγωνικά αντικείμενα, ο οποίος λειτουργεί με τη μέθοδο της γραμμής σάρωσης σε χρόνο $\Theta(k \log n)$, όπου k το πλήθος των πολυγώνων (βλέπε Σχήμα 2.10).

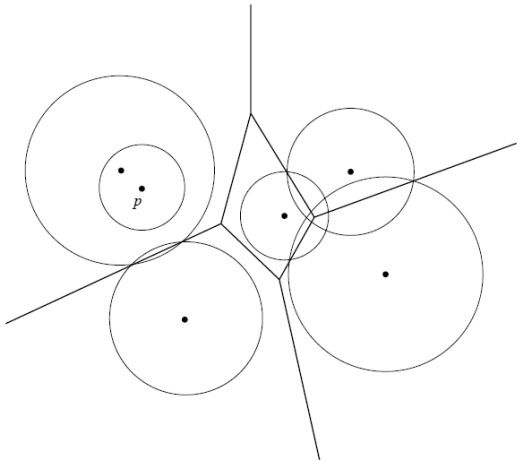
2.7 Διαγράμματα Voronoi με άλλες μετρικές απόστασης

2.7.1 Διάγραμμα Δύναμης

Μία άλλη παραλλαγή είναι το διάγραμμα δύναμης, το οποίο αναλύεται στο [3]. Σε αυτή την εργασία θα δώσουμε μία σύντομη περιγραφή του, σύμφωνα με το [4], αλλά θα περιοριστούμε στις 2 διαστάσεις, για να είμαστε συνεπείς με το αντικείμενο της εργασίας. Έστω ένα σύνολο S, n σημείων στο 2-διάστατο χώρο. Έστω ακόμα ότι κάθε σημείο p του S χαρακτηρίζεται από ένα βάρος $w(p)$, το οποίο αντιπροσωπεύει την ικανότητα του σημείου p να επηρεάζει τη «γειτονιά» του. Η συνάρτηση που εκφράζει πόσο ένα σημείο x του επιπέδου επηρεάζεται από ένα σημείο p του συνόλου S είναι η εξής:

$$pow(x, p) = ||x - p||^2 - w(p) \tag{2.8}$$

Όταν τα βάρη είναι θετικά, τότε θα μπορούσαμε να φανταστούμε το p ως έναν κύκλο, με κέντρο το σημείο p και ακτίνα $\sqrt{w(p)}$. Έστω ένα σημείο x εκτός του κύκλου, και l η εφαπτόμενη στον κύκλο ευθεία, που περνάει από το σημείο x. Τότε η $\sqrt{pow(s, p)}$, όπου $pow(w, p) > 0$, εκφράζει την απόσταση του x από το σημείο επαφής της l στον κύκλο. Πρόκειται δηλαδή για τη δύναμη του σημείου x ως προς τον κύκλο $\{p, \sqrt{w(p)}\}$.



Σχήμα 2.11: Διάγραμμα δύναμης σημείων στο επίπεδο. Οι κύκλοι αντιπροσωπεύουν τα βάρη των σημείων [4].

Ο γεωμετρικός τόπος των σημείων «ίσης δύναμης» (δηλαδή που ισαπέχουν από τα p και q, υπό την έννοια της (2.8)) από τα σημεία p και q του συνόλου S, είναι μία ευθεία, η οποία ονομάζεται *ευθεία δύναμης* των p και q. Αν $h(p,q)$ είναι το κλειστό χωρίο που φράσεται από το υπερεπίπεδο και περιέχει σημεία λιγότερης δύναμης ως προς το p, τότε η *δυναμική περιοχή* του p ορίζεται ως

$$VR(p) = \bigcap_{q \in S \setminus \{p\}} h(p,q) \tag{2.9}$$

και χωρίζει το επίπεδο σε κυρτά πολύγωνα τα οποία συνιστούν το *διάγραμμα δύναμης* $PD(S)$ του S. Στο Σχήμα 2.11 φαίνεται ένα διάγραμμα δύναμης. Αν τα βάρη των σημείων του S έχουν όλα την ίδια τιμή, τότε το διάγραμμα δύναμης του S ταυτίζεται με το Voronoi διάγραμμα του S.

Σύμφωνα με ένα Θεώρημα του [4], ο χρόνος που χρειάζεται για να υπολογιστεί το διάγραμμα δύναμης n σημείων στις 2 διαστάσεις ισούται με το χρόνο που χρειάζεται για να υπολογιστεί η κυρτή θήκη n σημείων στις 3 διαστάσεις. Αυτή υπολογίζεται σε αναμενόμενο χρόνο $O(n \log n)$ [11].

2.7.2 Airlift Voronoi διάγραμμα

Το διάγραμμα αυτό χρησιμοποιεί για τις αποστάσεις, τη μετρική m, η οποία είναι η παρακάτω:

$$m(p,q) = min \begin{cases} d(p,q) \\ d(p,a) + f(a,b) + d(b,q) \\ d(p,b) + f(b,a) + d(a,q) \end{cases} \tag{2.10}$$

Στην παραπάνω μετρική, η συνάρτηση f μπορούμε να φανταστούμε ότι αντιπροσωπεύει

κάποιο αεροπλάνο, ενώ η d κάποιο πιο αργό μεταφορικό μέσον, όπως το αυτοκίνητο. Έτσι λοιπόν, αν τα σημεία a και b συνδέονται με κάποια αεροπορική πτήση, τότε για να μεταφερθούμε από το σημείο p στο σημείο q, θα προτιμήσουμε να κάνουμε τη διαδρομή από το a στο b με αεροπλάνο, αν έτσι συντομεύουμε το χρόνο μας. (Η συνάρτηση d μετράει αποστάσεις μεταξύ σημείων με οποιαδήποτε L_p μετρική, $p \geq 1$.) Γι΄ αυτό και το Voronoi διάγραμμα που χρησιμοποιεί αυτή τη μετρική λέγεται *airlift*. Αν n το πλήθος των σημείων και c το πλήθος των αεροδρομίων, τότε το διάγραμμα αυτό μπορεί να υπολογιστεί σε χρόνο $O((n+c)\log(n+c))$ [38].

2.7.3 Voronoi διάγραμμα πόλης

Στο [2] παρουσιάζεται το Voronoi διάγραμμα πόλης και δίνεται ένας αλγόριθμος κατασκευής του. Η ιδέα είναι μία γενίκευση της ιδέας του airlift, δηλαδή ξεκίνησε από την ανάγκη να βρίσκουμε το συντομότερο μονοπάτι μεταξύ δύο σημείων σε μία πόλη, έχοντας στη διάθεσή μας τα μέσα μαζικής μεταφοράς, αλλά και την εναλλακτική δυνατότητα μετάβασης από ένα σημείο σε ένα άλλο περπατώντας. Εδώ να σημειώσουμε ότι τα μέσα μαζικής μεταφοράς ακολουθούν συγκεκριμένο δρομολόγιο και κάνουν συγκεκριμένες στάσεις, ενώ με τα πόδια μπορεί κάποιος να ακολουθήσει οποιαδήποτε διαδρομή επιθυμεί μέσα στην πόλη. Συνεπώς το διάγραμμα Voronoi θα προκύψει από το βέλτιστο συνδυασμό των παραπάνω.

Αν C γράφημα του δικτύου των μέσων μαζικής μεταφοράς της πόλης, με πλήθος σημείων c και S το σύνολο των σημείων του οποίου το Voronoi διάγραμμα $V_C(S)$ θέλουμε να υπολογίσουμε, τότε ονομάζουμε περιοχή Voronoi ενός σημείου $p \in S$ την:

$$reg(p) = \{x \,|\, d_C(x,p) < d_C(x,q), \forall q \in S \backslash \{p\}\} \tag{2.11}$$

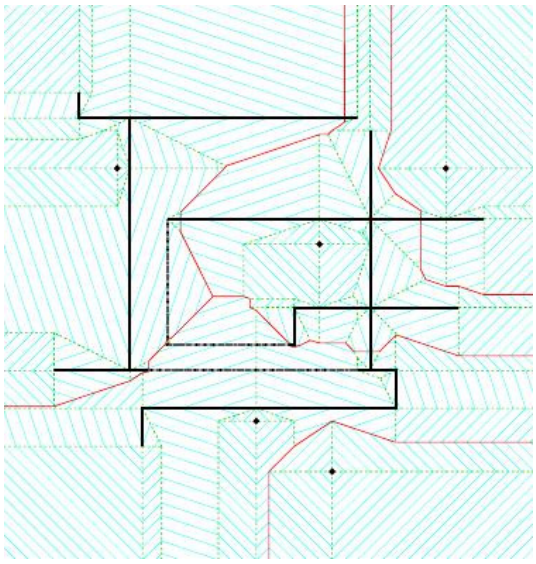
όπου η d_C είναι μία συνάρτηση απόστασης. Στο Σχήμα 2.12 φαίνεται το διάγραμμα αυτό.

Στο [2] το Voronoi διάγραμμα πόλης κατασκευάζεται σε χρόνο $O(n\log n + c^2 \log c)$ και χώρο $O(n+c)$.

2.8 Γενίκευση

Κλείνουμε αυτό το Κεφάλαιο αναφέροντας έναν όσο το δυνατόν πιο γενικό Ορισμό του διαγράμματος Voronoi [22], ο οποίος περικλείει μέσα του όλες τις παραπάνω υποπεριπτώσεις και γι΄ αυτό το λόγο αξίζει να αναφερθεί.

Ορισμός 2. *Έστω E ένα πεπερασμένο σύνολο n στοιχείων και $f_E = \{f_e \,|\, e \in E\}$ μία συλλογή από πραγματικές συναρτήσεις που έχουν ένα κοινό πεδίο ορισμού D. Έστω ακόμα*



Σχήμα 2.12: Voronoi διάγραμμα πόλης (με κόκκινο). Το δίκτυο των μέσων μαζικής μεταφοράς C είναι σημειωμένο με ευθείες μαύρες γραμμές [2].

R_f μία συνάρτηση από το D στο E, όπου:

$$R_f(x) = \{e \in E \mid f_e(x) = \min_{i \in E} f_i(x)\} \tag{2.12}$$

Το R_f ορίζει μία σχέση ισοδυναμίας ρ_r στο D, όπου για $x, y \in D$, $x \rho_f y \Leftrightarrow R_f(x) = R_f(y)$. Η διαμέριση του D που δημιουργείται από αυτή τη σχέση ισοδυναμίας ρ_f ονομάζεται διάγραμμα Voronoi του D ως προς το f_E. Οι κλάσεις ισοδυναμίας της διαμέρισης ονομάζονται περιοχές Voronoi.

Κεφάλαιο 3

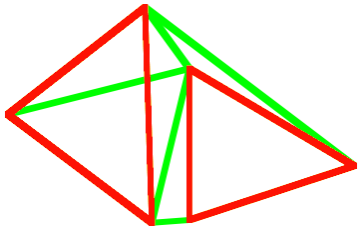
Κατασκευή της περιβάλλουσας καμπύλης

Στην ενότητα αυτή θα εξηγήσουμε πώς κατασκευάζουμε στον αλγόριθμό μας την περιβάλλουσα καμπύλη πολυγώνων του συνόλου. Επίσης θα αναφερθούμε, όπου αυτό καθίσταται απαραίτητο, και στην υλοποίηση των αλγορίθμων που παρουσιάζονται στο κεφάλαιο αυτό.

Αυτό που θέλουμε να πετύχουμε είναι να μην ακουμπάει η καμπύλη μόνο τα πολύγωνα που βρίσκονται κοντά στην κυρτή θήκη του συνόλου των πολυγώνων, αλλά και κάποια από αυτά που βρίσκονται στο εσωτερικό της, βάσει κριτηρίων που θα περιγράψουμε και θα αναλύσουμε. Η αναζήτηση με την οποία ανακαλύπτονται νέα πολύγωνα θα βασιστεί σε μία τριγωνοποίηση του συνόλου των σημείων των πολυγώνων. Η τριγωνοποίηση Delaunay παρέχει καλές πληροφορίες γειτνίασης, διότι στο διάγραμμα Voronoi (άρα και στην τριγωνοποίηση Delaunay) το πλησιέστερο πολύγωνο ενός πολυγώνου θα είναι και γείτονάς του. Γι΄ αυτό και επιλέχθηκε ως η τριγωνοποίηση που θα χρησιμοποιήσει ο αλγόριθμός μας.

Όπως έχουμε ήδη σημειώσει, η τριγωνοποίηση Delaunay είναι το δυϊκό γράφημα του διαγράμματος Voronoi. Για λόγους απλότητας επιλέγουμε να υπολογίσουμε το διάγραμμα Voronoi των σημείων των πολυγώνων, και όχι το διάγραμμα των ίδιων των πολυγώνων, αγνοώντας τις ακμές που δημιουργούνται μέσα στα πολύγωνα, αλλά και τις ακμές μεταξύ σημείων του ιδίου πολυγώνου. Η προϋπόθεση που πρέπει να ικανοποιείται, λόγω αυτής της απλούστευσης είναι ότι για οποιοδήποτε πολύγωνο, η μέγιστη απόσταση μεταξύ σημείων του θα πρέπει να είναι μικρότερη από την ελάχιστη απόσταση μεταξύ οποιονδήποτε πολυγώνων. Αλλιώς θα συμβεί κάτι σαν αυτό που φαίνεται στο Σχήμα 3.1. Με αυτήν την προϋπόθεση εξασφαλίζεται ότι δε θα τέμνουν οι ακμές της τριγωνοποίησης των σημείων, ακμές πολυγώνων.

Ξεκινώντας λοιπόν ο αλγόριθμος από την κυρτή θήκη, ελέγχει μία προς μία τις ακμές της, και είτε τις αφήνει ως έχουν, είτε τις αντικαθιστά με μία σειρά ακμών, έτσι ώστε η καμπύλη να ακουμπά ένα ή περισσότερα πολύγωνα που βρίσκονται στο εσωτερικό της.

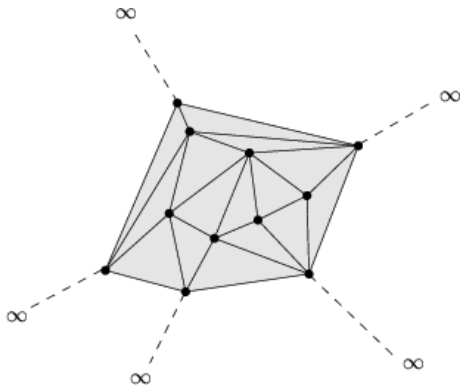


Σχήμα 3.1: Παράδειγμα στο οποίο υπάρχει απόσταση μεταξύ σημείων του ιδίου πολυγώνου που να είναι μεγαλύτερη από κάποια απόσταση του πολυγώνου αυτού με κάποιο άλλο. Με κόκκινο φαίνονται τα πολύγωνα και με πράσινο οι ακμές της τριγωνοποίησης των σημείων, κάποιες από τις οποίες τέμνουν τις ακμές των πολυγώνων

Η αναζήτηση γίνεται, όπως είπαμε, με περπάτημα πάνω στα τρίγωνα της τριγωνοποίησης Delaunay, ενώ οι γωνίες των τριγώνων, παίζουν καθοριστικό ρόλο στο αν η περιβάλλουσα καμπύλη θα ακουμπάει ένα πολύγωνο ή όχι. Λεπτομέρειες για το ρόλο των γωνιών των τριγώνων θα αναπτυχθούν παρακάτω σε αυτό το Κεφάλαιο.

3.1 Αναζήτηση πολυγώνων στο εσωτερικό της κυρτής θήκης

Λόγω του ότι παίρνουμε τις πληροφορίες μας από το διάγραμμα Voronoi σημείων και όχι πολυγώνων, πρέπει να γνωρίζουμε σε ποιο πολύγωνο ανήκει κάθε σημείο. Δημιουργούμε λοιπόν μία συνάρτηση αντιστοίχησης των σημείων σε θετικούς ακεραίους αριθμούς. Έτσι δύο σημεία αντιστοιχούν στον ίδιο αριθμό αν και μόνο αν ανήκουν στο ίδιο πολύγωνο.



Σχήμα 3.2: Τριγωνοποίηση όπου οι κόμβοι του συνόρου συνδέονται με τον άπειρο κόμβο [47].

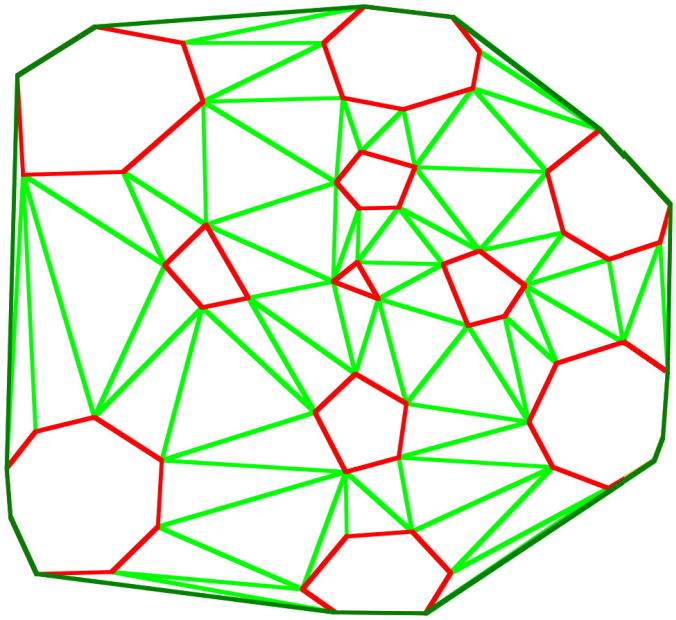
Στην τριγωνοποίηση Delaunay εμφανίζεται και ένα σημείο που αντιστοιχεί στο άπειρο και είναι γειτονικό με τα εξωτερικά σημεία της τριγωνοποίησης (βλέπε Σχήμα 3.2). Η συνάρτηση αντιστοίχησης σημείων στέλνει το σημείο αυτό στον ακέραιο -1.

Όπως είπαμε παραπάνω, η περιβάλλουσα καμπύλη αρχικά θα ταυτιστεί με την κυρτή θήκη της τριγωνοποίησης, και σταδιακά θα μεταβάλλεται ακουμπώντας και κάποια εσωτερικά

πολύγωνα.

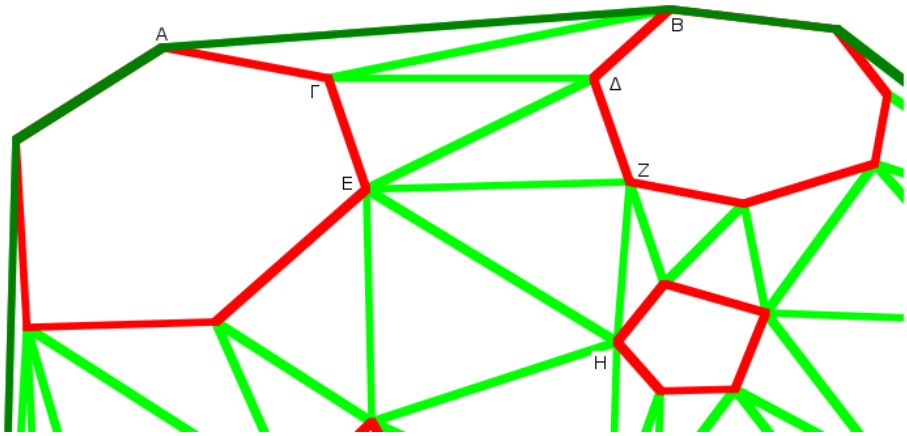
Για να βρεθεί ένα εσωτερικό πολύγωνο, αρκεί να βρούμε ένα τρίγωνο της τριγωνοποίησης Delaunay το οποίο να έχει τα άκρα του σε 3 διαφορετικά πολύγωνα. Αν είχαμε το διάγραμμα Voronoi των πολυγώνων, τότε απλά θα πηγαίναμε σε ένα εσωτερικό σημείο Voronoi και αυτό σίγουρα θα αντιστοιχούσε στο τρίγωνο που ακουμπάει σε τρία διαφορετικά πολύγωνα. Όμως επειδή χρησιμοποιούμε διάγραμμα Voronoi σημείων, για να βρούμε αυτό το τρίγωνο πρέπει να γίνει με κάποιο τρόπο αναζήτηση μέσα στην τριγωνοποίηση.

Ο τρόπος με τον οποίο κάνουμε αυτή την αναζήτηση των τρίγωνων είναι ο εξής. Αρχικά βάζουμε σε ένα διάνυσμα όλες τις ακμές της τριγωνοποίησης που ανήκουν στην κυρτή θήκη (είναι οι ακμές με το βαθύ πράσινο του Σχήματος 3.3) και για κάθε μία από αυτές ελέγχουμε αν το τρίγωνο στο οποίο ανήκει η ακμή έχει τα άκρα του σε τρία διαφορετικά πολύγωνα. Αν όχι, τότε επαναλαμβάνουμε τον έλεγχο, αλλά αυτή τη φορά αλλάζουμε την ακμή μας και χρησιμοποιούμε ως ακμή - οδηγό, αυτή που ακουμπούσε σε 2 διαφορετικά πολύγωνα. Αλλιώς έχουμε βρει το τρίγωνο που ψάχναμε.



Σχήμα 3.3: Με κόκκινο φαίνονται τα πολύγωνα, με σκούρο πράσινο η κυρτή θήκη και με ανοικτό πράσινο η τριγωνοποίηση Delaunay.

Για να γίνει πιο κατανοητή η μέθοδος, θα δώσουμε ένα παράδειγμα πάνω σε ένα μεγεθυμένο τμήμα του Σχήματος 3.3, το οποίο είναι το 3.4. Εδώ ας υποθέσουμε ότι ξεκινάμε την αναζήτηση από την ακμή AB που ανήκει στο τρίγωνο $AB\Gamma$. Όμως τα σημεία A και Γ, ανήκουν στο ίδιο πολύγωνο, άρα η ακμή AB δεν είναι η επιθυμητή και την αλλάζουμε με την $B\Gamma$ που ανήκει στο τρίγωνο $B\Gamma\Delta$ το οποίο και πάλι έχει 2 άκρα σε κοινό πολύγωνο, τα B και Δ. Ομοίως συνεχίζουμε με το τρίγωνο $\Gamma\Delta E$ και το ΔEZ, έως ότου καταλήξουμε τελικά στο EZH. Το τρίγωνο αυτό έχει πράγματι κάθε σημείο του σε διαφορετικό

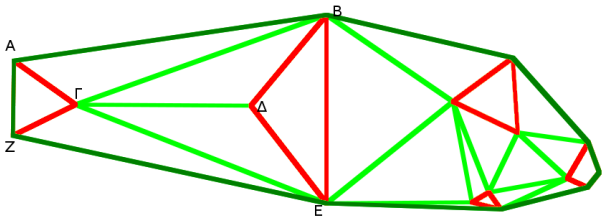


Σχήμα 3.4: Μεγεθυμένο τμήμα του Σχήματος 3.3

πολύγωνο, άρα είναι αυτό που ψάχναμε.

Στην περίπτωση που δεν υπάρχει κανένα εσωτερικό πολύγωνο, τότε σίγουρα η αναζήτηση θα καταλήξει στην απέναντι εξωτερική πλευρά της κυρτής θήκης και θα σταματήσει συναντώντας τον άπειρο κόμβο, χάρις στη συνάρτηση αντιστοίχησης πολυγώνων - σημείων που αντιστοιχεί το σημείο του απείρου στον αριθμό -1.

Ένα τέτοιο φαινόμενο παρουσιάζεται στο Σχήμα 3.5. Έστω ότι ξεκινάμε από την ακμή AB, που ανήκει στο τρίγωνο $AB\Gamma$. Όμως επειδή τα σημεία A και Γ ανήκουν στο ίδιο πολύγωνο, αλλάζουμε την ακμή AB με την $B\Gamma$ και ελέγχουμε το $B\Gamma\Delta$. Και πάλι, τα B και Δ ανήκουν στο ίδιο πολύγωνο, οπότε τώρα ελέγχουμε τη $\Gamma\Delta$. Τα Δ και E ανήκουν στο ίδιο πολύγωνο. Στη συνέχεια πάμε στην ΓE του τριγώνου ΓEZ και τέλος στην EZ, η οποία ανήκει στο τρίγωνο με άκρα τα E, Z και το σημείο που αντιστοιχεί στο άπειρο, οπότε και σταματάει η αναζήτηση. Έτσι στην περιβάλλουσα καμπύλη θα παραμείνει η αρχική ακμή AB. Η αντίστοιχη διαδικασία θα συμβεί όταν ελέγξουμε την ακμή ZE.

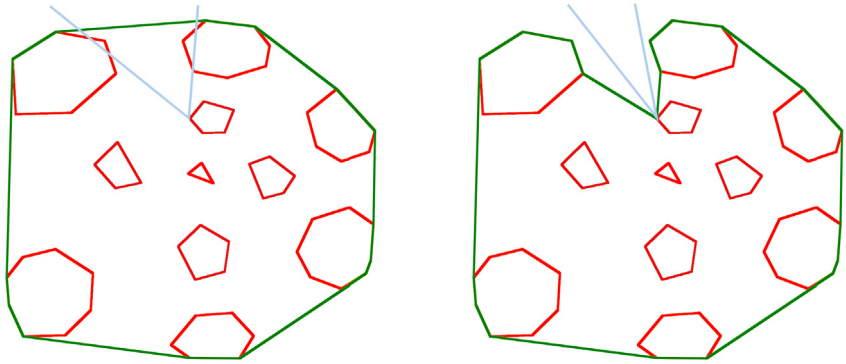


Σχήμα 3.5: Παράδειγμα όπου η αναζήτηση δε συναντάει ποτέ τρίγωνο με κόμβους σε 3 διαφορετικά πολύγωνα.

3.2 Κριτήρια αποδοχής

Στόχος μας, όπως ήδη περιγράψαμε, είναι να μπορέσει η καμπύλη να ακουμπάει εσωτερικά πολύγωνα. Κάθε φορά που περνάει η καμπύλη από ένα εσωτερικό πολύγωνο σχηματίζει μία γωνία, η οποία διαφοροποιείται αναλόγως με το πόσο κοντά είναι το πολύγωνο στην

κυρτή θήκη. Γι΄ αυτό χρησιμοποιούμε και εμείς γωνίες ως κριτήρια για τον αλγόριθμο. Ο χρήστης θέτει ως όριο μία γωνία η οποία θα είναι η μικρότερη γωνία που θα γίνεται αποδεκτή από τα κριτήρια. Τα κριτήρια διαφοροποιούνται ως προς τη γωνία που ελέγχουν.



(α΄) Η γωνία του κριτηρίου του τριγώνου είναι μικρότερη από τη γωνία που έχει καθορίσει ο χρήστης η οποία έχει σημειωθεί με γαλάζιο.

(β΄) Η γωνία του κριτηρίου του τριγώνου είναι μεγαλύτερη από τη γωνία που έχει καθορίσει ο χρήστης η οποία έχει σημειωθεί με γαλάζιο.

Σχήμα 3.6: Το αποτέλεσμα της περιβάλλουσας καμπύλης (για την ακμή AB) εξαρτάται από το όριο της γωνίας που θέτει ο χρήστης.

3.2.1 Κριτήριο τριγώνου

Στο κριτήριο αυτό, συγκρίνουμε τη γωνία του χρήστη με τη γωνία του τριγώνου που έχουμε βρει (που έχει τους κόμβους του σε τρία διαφορετικά πολύγωνα) η οποία είναι απέναντι από την ακμή που χρησιμοποιήσαμε ως οδηγό. Στο παράδειγμα του Σχήματος 3.4, η γωνία που είναι απέναντι από την EZ είναι η $Z\hat{H}E$. Αν αυτή η γωνία είναι μικρότερη από το όριο του χρήστη, τότε στην περιβάλλουσα καμπύλη θα ανήκει μόνο η αρχική ακμή από την οποία ξεκίνησε η αναζήτηση, όπως στο Σχήμα 3.6(α΄). Αλλιώς αντί της ακμής αυτής, θα προστεθούν όλες οι ακμές που προσπεράσαμε μέχρι να φτάσουμε στο συγκεκριμένο τρίγωνο, όπως δείχνει το Σχήμα 3.6(β΄).

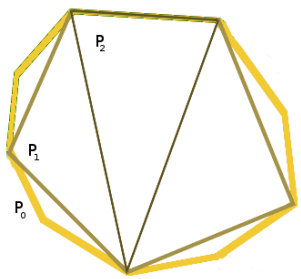
3.2.2 Κριτήριο κώνου

Έστω ότι η ακμή από την οποία ξεκινήσαμε για να ανακαλύψουμε το εσωτερικό πολύγωνο ακουμπούσε στα πολύγωνα A και B. Στο κριτήριο αυτό, αντί για τη γωνία του τριγώνου που βρισκόμαστε, παίρνουμε μία άλλη γωνία, η οποία είναι η εσωτερική γωνία δύο ημιευθειών. Οι ημιευθείες αυτές θα έχουν κοινή αρχή το σημείο του τριγώνου που ανήκει στο εσωτερικό πολύγωνο, ενώ καθεμία τους θα εφάπτεται είτε στο A είτε στο B, έτσι ώστε το πολύγωνο αυτό να μην περιέχεται στο σχηματιζόμενο κώνο. Δεν υπάρχει περιορισμός στο αν θα εμφανίζονται άλλα πολύγωνα στο εσωτερικό του κώνου αυτού.

Για να βρεθεί από ποιο σημείο του κάθε πολυγώνου θα περνάει η κάθε πλευρά της γωνίας, ελέγχουμε ένα προς ένα τα σημεία των δύο πολυγώνων. Αρχικά οι ακμές της γωνίας εσωκλείουν και τα δύο πολύγωνα. Επιλέγουμε τη μία ακμή και ελέγχουμε το διπλανό σημείο από αυτό που περνάει, αν μειώνει το άνοιγμα της γωνίας. Αν ναι, τότε αντικαθίσταται η αρχική ακμή από αυτήν την ακμή. Όσα σημεία του πολυγώνου δεν εσωκλείονται στην παρούσα ακμή δε θα ελεγχθούν ποτέ, αφού σίγουρα αυξάνουν την παρούσα γωνία. Η διαδικασία αυτή συνεχίζεται έως ότου να ελαχιστοποιηθεί η γωνία και κατόπιν γίνεται το ίδιο και για την άλλη ακμή της γωνίας.

Η μέθοδος αυτή δεν είναι η βέλτιστη, γι΄ αυτό και παραθέτουμε άλλη μία μέθοδο υπολογισμού της γωνίας του κώνου, η οποία δεν έχει υλοποιηθεί στο πρόγραμμα, και μπορεί να υπολογίσει τη γωνία του κώνου σε χρόνο $O(\log m)$, όπου m το πλήθος των σημείων του πολυγώνου που ακουμπάει η ακμή. Η μέθοδος αυτή προέρχεται από τον αλγόριθμο των Dobkin-Kirkpatrick [19].

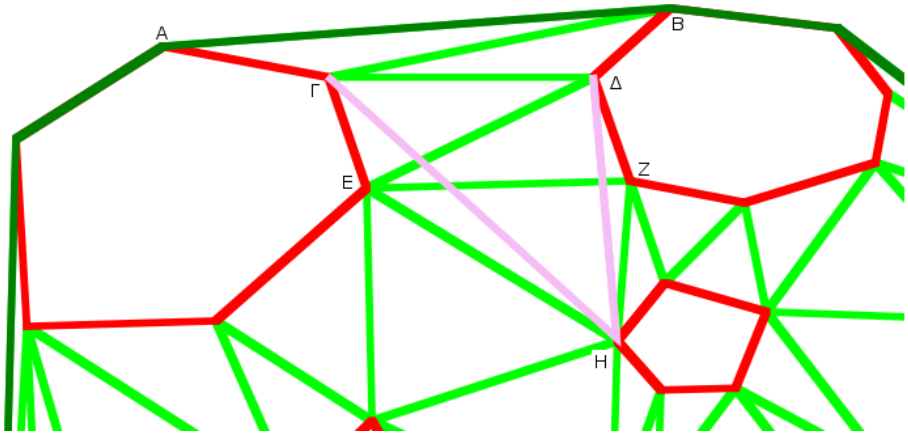
Για ένα κυρτό πολύγωνο P θεωρούμε την ακολουθία των κυρτών πολυγώνων $P = P_0$, $P_1, P_2, \ldots, P_k$, η οποία κατασκευάζεται ως εξής (βλέπε Σχήμα 3.7): Έστω ότι το πολύγωνο P έχει n κόμβους u_1, u_2, …, u_n. Επιλέγοντας κάθε δεύτερο κόμβο (δηλαδή τους u_1, u_3, u_5, …) κατασκευάζουμε το P_1. Επαναλαμβάνουμε την ίδια διαδικασία για να πάρουμε το P_2 κ.ο.κ έως ότου τερματίσουμε στο P_k το οποίο είναι τρίγωνο. Κάθε πολύγωνο P_i λέμε ότι αντιστοιχεί σε ένα επίπεδο αυτής της ιεραρχίας. Στην ακολουθία πολυγώνων P_0, P_1, …, P_k, που δημιουργήθηκε, έχουμε ότι $k = O(\log m)$.



Σχήμα 3.7: Κατασκευή της ιεραρχίας Dobkin-Kirkpatrick.

Αφού οριστεί λοιπόν αυτή η ιεραρχία, για να βρούμε το σημείο από το οποίο θα περνάει η ακμή της γωνίας, γίνονται τα εξής: Αρχικά ακουμπάει η ακμή σε κάποιο σημείο του πολυγώνου, το οποίο ανήκει στα επίπεδα των P_i, P_{i-1}, …, P_0. Αυτό είναι γειτονικό με άλλα το πολύ δύο σημεία, τα οποία ανήκουν στα επίπεδα των πολυγώνων από το P_{i-1} και κάτω. Από αυτά τα δύο σημεία (ή το ένα), επιλέγω και μετακινώ την ακμή σε αυτό που μειώνει τη γωνία. Αν και τα δύο σημεία μειώνουν τη γωνία, τότε επιλέγω τυχαία το ένα από αυτά, γιατί ανεξάρτητα από το ποιο θα διαλέξω, στο τέλος θα καταλήξω στο ίδιο αποτέλεσμα. Αν τη μειώνει λοιπόν, τότε η ακμή μετακινείται σε αυτό το σημείο και επαναλαμβάνεται ο έλεγχος μέχρι το επίπεδο του P_0. Αλλιώς, αν κανένα από τα δύο σημεία δε μειώνει τη γωνία, τότε η αναζήτηση σταματάει.

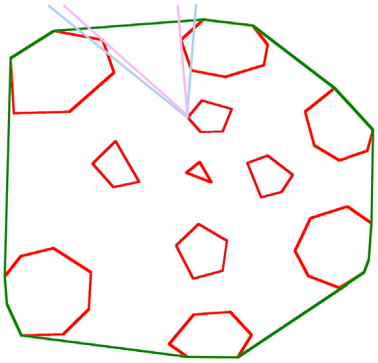
Αφού το k είναι μεγέθους $O(\log m)$ καταλαβαίνουμε ότι κάθε φορά η αναζήτηση της γωνίας του κώνου χρειάζεται χρόνο $O(\log m)$, άρα συνολικά ο αλγόριθμος για το κριτήριο του κώνου θα χρειαστεί χρόνο $O(n \log n)$, το οποίο βελτιώνει τα αποτελέσματα του αλγορίθμου που έχουμε υλοποιήσει, όπως θα δούμε και παρακάτω.



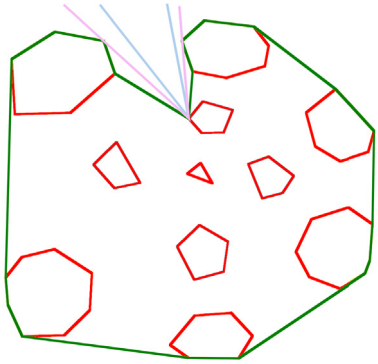
Σχήμα 3.8: Μεγεθυμένο τμήμα του Σχήματος 3.3, στο οποίο έχει σημειωθεί με ροζ η γωνία του κώνου που ξεκινάει από το σημείο H και επεκτείνεται προς την ακμή AB.

Συνεχίζοντας με το παράδειγμα του Σχήματος 3.4, όπου ξεκινήσαμε με ακμή οδηγό την AB, θα καταλήξουμε στη γωνία που σχηματίζουν οι δύο ροζ ακμές του Σχήματος 3.8.

Σε αυτό το σημείο δεν έχουμε παρά να ελέγξουμε και πάλι, αν η γωνία αυτή είναι μικρότερη από τη γωνία που έχει καθορίσει ο χρήστης. Αν ναι, τότε παραμένει στην περιβάλλουσα καμπύλη η αρχική ακμή οδηγός, όπως στο Σχήμα 3.9(α΄), αλλιώς στη θέση της μπαίνουν όλες οι ακμές που προσπεράσαμε μέχρι να φτάσουμε στο συγκεκριμένο τρίγωνο (βλέπε Σχήμα 3.9(β΄)).



(α΄) Η γωνία του κριτηρίου του κώνου (ροζ) είναι μικρότερη από τη γωνία που έχει καθορίσει ο χρήστης (γαλάζιο).



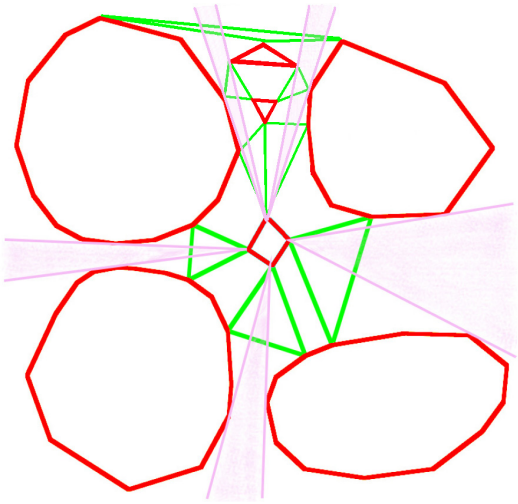
(β΄) Η γωνία του κριτηρίου του κώνου (ροζ) είναι μεγαλύτερη από τη γωνία που έχει καθορίσει ο χρήστης (γαλάζιο).

Σχήμα 3.9: Το αποτέλεσμα της περιβάλλουσας καμπύλης (για την ακμή AB) εξαρτάται από το όριο της γωνίας που θέτει ο χρήστης.

3.2.3 Κριτήριο κώνου με ορατότητα προς την κυρτή θήκη

Το κριτήριο αυτό διαφοροποιείται κάπως από το προηγούμενο. Και πάλι θα χρησιμοποιήσει τη γωνία ενός κώνου, ο οποίος θα έχει την κορυφή του στο σημείο του τριγώνου που ανήκει στο εσωτερικό πολύγωνο, αλλά τα σημεία από τα οποία θα περνάνε οι ημιευθείες του ίσως διαφοροποιηθούν. Στο προηγούμενο κριτήριο η κάθε ημιευθεία περνούσε από το σημείο ενός από τα άλλα δύο πολύγωνα στα οποία ανήκουν τα άλλα δύο σημεία του τριγώνου, ενώ τώρα η γωνία θα είναι η εξής:

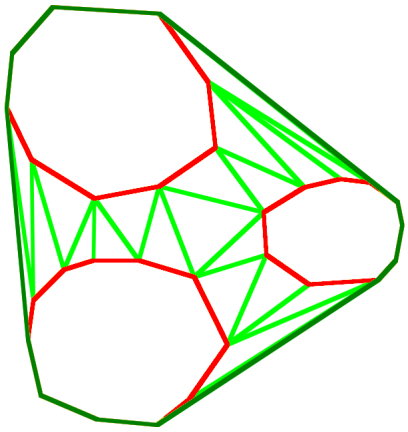
Ας φανταστούμε ότι από το σημείο του εσωτερικού πολυγώνου υπάρχει μία πηγή φωτός η οποία ρίχνει το φως της προς την ακμή της κυρτής θήκης την οποία συναντάμε αν προχωρήσουμε προς την κατεύθυνση στην οποία υπάρχει η ακμή από όπου ξεκίνησε η αναζήτηση του τριγώνου, όπως φαίνεται στο Σχήμα 3.10. Αν θεωρήσουμε ότι κάποιο πολύγωνο εμποδίζει την πορεία του φωτός, τότε ο τομέας φωτός που τελικά καταλήγει στην ακμή της κυρτής θήκης ενδεχομένως να είναι πολύ μικρότερος από τη γωνία του προηγούμενου κριτηρίου. Μπορεί ακόμα και να μην φτάνει καθόλου φως. Η γωνία αυτή είναι η γωνία την οποία θα συγκρίνουμε με το όριο του χρήστη.



Σχήμα 3.10: Με ροζ φαίνονται τα τμήματα από τα οποία βλέπει ένα σημείο του πιο εσωτερικού πολυγώνου την κυρτή θήκη. Αν εμφανίζονται περισσότερες από μία γωνίες, τότε επιλέγεται η μεγαλύτερη. Παρατηρείστε ότι αν δεν παρεμβάλλονται άλλα πολύγωνα, τότε η γωνία του κριτηρίου αυτού ταυτίζεται με τη γωνία του απλού κριτηρίου του κώνου. (Στο σχήμα με πράσινο φαίνονται μόνο τα τρίγωνα που έχουν άκρα σε 3 διαφορετικά πολύγωνα.)

3.3 Αντιμετώπιση αυτοτομών της περιβάλλουσας καμπύλης

Στον αλγόριθμό μας, αναζητώντας το τρίγωνο που ακουμπάει σε τρία διαφορετικά πολύγωνα, υπάρχει κίνδυνος να φτάσουμε σε ένα τρίγωνο από το οποίο έχει ξαναπεράσει

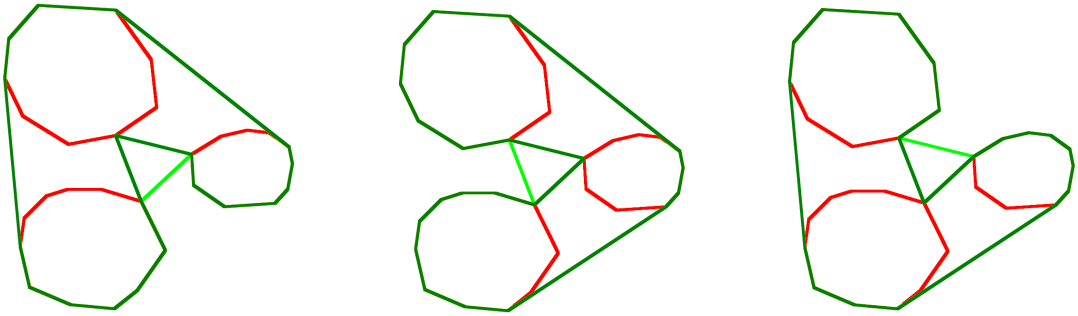


Σχήμα 3.11: Παράδειγμα πολυγώνων που η περιβάλλουσα καμπύλη θα μπορούσε να τμήσει τον εαυτό της. (Στο Σχήμα 3.12 φαίνεται πώς μπορεί να αυτοτμηθεί.)

η περιβάλλουσα καμπύλη και έτσι να δημιουργηθεί αυτοτομή της καμπύλης. Αυτό το φαινόμενο δεν επιτρέπεται, ειδικά όταν η καμπύλη αυτή χρησιμοποιηθεί για το πρόβλημα της κατασκευής παρεμβαλλουσών επιφανειών (βλέπε Κεφάλαιο 1).

Υπενθυμίζουμε ότι χρησιμοποιείται η περιβάλλουσα καμπύλη για να μετατρέψει τα M πολύγωνα του ενός επιπέδου σε 1 μόνο πολύγωνο, ώστε αντί να έχουμε το 1-M πρόβλημα, να έχουμε το 1-1. Άρα η καμπύλη θα αντιπροσωπεύει ένα πολύγωνο. Τα πολύγωνα όμως του προβλήματος δεν επιτρέπεται να τέμνουν τον εαυτό τους.

Στο Σχήμα 3.11 φαίνεται ένα τέτοιο παράδειγμα, όπου η καμπύλη κινδυνεύει να τμήσει τον εαυτό της όπως φαίνεται στα Σχήματα 3.12(α′)-3.12(γ′).



(α′) (β′) (γ′)

Σχήμα 3.12: Αν οι γωνίες του τριγώνου που ακουμπάει στα 3 πολύγωνα είναι και οι τρεις αποδεκτές, τότε η καμπύλη θα περνάει από αυτό και με τους 3 τρόπους που φαίνονται στις εικόνες, τέμνοντας τον εαυτό της. Αντίστοιχη τομή θα συμβεί, αν 2 από τις γωνίες του τριγώνου είναι αποδεκτές.

Παρακάτω παρουσιάζονται 3 λύσεις αυτού του προβλήματος, οι οποίες αξιολογούνται, και από αυτές επιλέγεται μία για τον αλγόριθμό μας.

1. Να μην περνάει ποτέ καμία σειρά ακμών από αυτό το τρίγωνο. Αυτή η λύση είναι εξαιρετικά απλή, αλλά το πολύγωνο που επρόκειτο να ακουμπήσει η καμπύλη

περνώντας από εκείνο το τρίγωνο ίσως να μην το ακουμπήσει τελικά.

2. Στο τρίγωνο θα παραμείνει η σειρά ακμών που πέρασε από αυτό πρώτη. Είναι μία αρκετά απλή στην υλοποίηση λύση, η οποία και εκτός από τη λύση του προβλήματος τομής, επιτρέπει τελικά στην καμπύλη να αγγίξει το εσωτερικό πολύγωνο στο οποίο δημιουργήθηκε το πρόβλημα. Όμως με τον τρόπο αυτό, η πλευρά από την οποία θα μπει η καμπύλη στο συγκεκριμένο τρίγωνο είναι τυχαία. Έτσι πολλαπλές εκτελέσεις του ίδιου αλγορίθμου δίνουν διαφορετικά αποτελέσματα.

3. Η λύση αυτή βασίζεται στον κανόνα που θέλει να επιλέγονται οι γωνίες που είναι μεγαλύτερες από το όριο του χρήστη. Αφού περισσότερες από μία γωνίες είναι μεγαλύτερες από το όριο, επιλέγεται πάντα η μεγαλύτερη από αυτές. Δηλαδή, αν η καμπύλη είχε περάσει από το τρίγωνο στο παρελθόν δημιουργώντας μικρότερη γωνία, τότε αυτές οι ακμές αφαιρούνται και αντικαθίστανται από την αρχική ακμή τους, και προστίθενται αυτές που δημιουργούν μεγαλύτερη γωνία.

Η λύση που επελέγη τελικά για τον αλγόριθμό μας είναι η τρίτη, γιατί παρόλο που δεν είναι τόσο απλή στην υλοποίηση, είναι πιο συνεπής στις αρχές του αλγορίθμου μας. Έτσι, στο παράδειγμα του Σχήματος 3.11 το αποτέλεσμα της καμπύλης θα είναι ένα από τα στιγμιότυπα του Σχήματος 3.12, αναλόγως της γωνίας που έχει θέσει ως όριο ο χρήστης.

Στο σημείο αυτό αξίζει να σημειωθεί ότι η λύση της αυτοτέμνουσας καμπύλης θα μπορούσε να προσφέρει έναν τρόπο καθορισμού του ορίου της γωνίας με αυτόματο τρόπο και όχι μέσω του χρήστη. Μπορεί να οριστεί δηλαδή, ως ελάχιστη γωνία, μία τέτοια γωνία ώστε να μη δημιουργούνται αυτοτομές της καμπύλης.

Η υλοποίηση του τρόπου αντιμετώπισης των αυτοτομών έχει γίνει ως εξής:
Έχουμε μία συνάρτηση αντιστοίχησης των χωρίων της τριγωνοποίησης σε λογικές μεταβλητές. Ένα χωρίο αντιστοιχεί στην τιμή *αληθής* αν και μόνο αν υπάρχουν δύο διαδοχικές ακμές της περιβάλλουσας καμπύλης που να ανήκουν σε αυτό το τρίγωνο. Συνεπώς κάθε φορά που ο αλγόριθμος πρόκειται να αντικαταστήσει μία ακμή της περιβάλλουσας καμπύλης με μία σειρά άλλων ακμών που καταλήγουν σε κάποιο τρίγωνο με άκρα σε 3 διαφορετικά πολύγωνα, ελέγχει αν το τρίγωνο αυτό αντιστοιχεί σε τιμή αληθή. Σε αυτήν την περίπτωση γίνεται ο έλεγχος των γωνιών που αναλύθηκε παραπάνω και, αν χρειάζεται, τοποθετούνται στην καμπύλη οι αντίστοιχες ακμές, ενώ αυτές που απορρίπτονται αντικαθιστώνται με την αρχική τους ακμή.

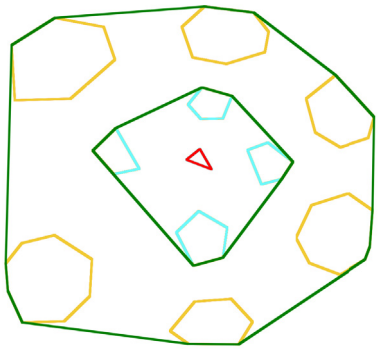
3.4 Κατασκευή περιβάλλουσας καμπύλης για τα πολύγωνα του πρώτου επιπέδου

Σε αυτήν την υποενότητα θα αρχίσουμε να εξηγούμε πώς κατασκευάζεται τελικά η περιβάλλουσα καμπύλη, αλλά είναι ανάγκη πριν από αυτό να εξηγήσουμε τι εννοούμε όταν

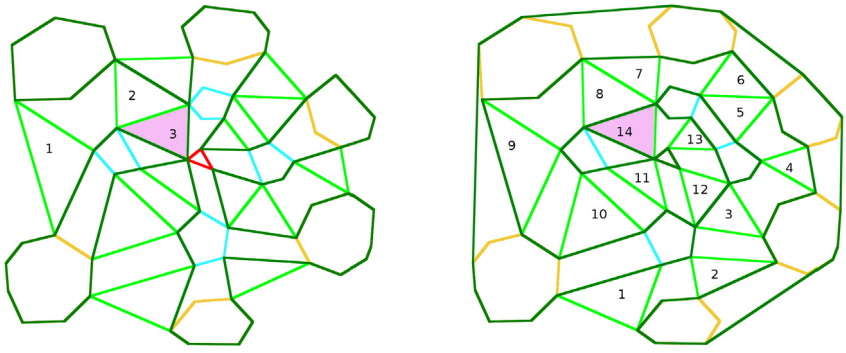
λέμε *πολύγωνο i-επιπέδου* καθώς και *βάθος τριγώνου*:

Τα πολύγωνα μηδενικού επιπέδου είναι αυτά που ακουμπούν στην κυρτή θήκη των πολυγώνων. Αν αφαιρέσουμε αυτά τα πολύγωνα από το σύνολο και θεωρήσουμε τη νέα κυρτή θήκη, τότε τα πολύγωνα που ακουμπάνε στη νέα κυρτή θήκη είναι τα πολύγωνα του πρώτου επιπέδου. Αντίστοιχα ορίζονται τα πολύγωνα του i-επιπέδου (βλέπε Σχήμα 3.13).

Τώρα θα ορίσουμε το *βάθος τριγώνου* (με άκρα σε τρία διαφορετικά πολύγωνα), το οποίο εξαρτάται από την ακμή που θα ξεκινήσει η διαδικασία για να ανακαλυφθεί το εν λόγω τρίγωνο, αλλά και από τον αλγόριθμο με τον οποίο κατασκευάζουμε την περιβάλλουσα καμπύλη. Το πλήθος των τριγώνων με άκρα σε τρία διαφορετικά πολύγωνα που θα προσπεράσει ο αλγόριθμος έως ότου ανακαλύψει το επιθυμητό τρίγωνο, συν ένα, ισούται με το βάθος του τριγώνου αυτού. Στο Σχήμα 3.14 φαίνεται το ίδιο τρίγωνο, πώς αποκτά διαφορετικό βάθος με χρήση δύο διαφορετικών αλγορίθμων (οι οποίοι περιγράφονται παρακάτω).



Σχήμα 3.13: Τα πολύγωνα 0-επιπέδου (πορτοκαλί) ακουμπάνε στην κυρτή θήκη (σκούρο πράσινο). Αφαιρώντας τα, ξανασχηματίζουμε την κυρτή θήκη και βρίσκουμε τα πολύγωνα 1-επιπέδου (γαλάζια). Τέλος με κόκκινο είναι τα 2-επιπέδου.



(α′) (β′)

Σχήμα 3.14: Το ροζ τρίγωνο αποκτά διαφορετικό βάθος, αναλόγως τη διαδρομή που ακολουθεί ο αλγόριθμος για να το ανακαλύψει. Η διαδρομή φαίνεται από τους σημειωμένους αριθμούς στα τρίγωνα που προσπέλασε κάθε φορά.

Αφού ορίστηκαν οι απαραίτητες έννοιες, και συνοψίζοντας όσα έχουμε πει έως τώρα,

μπορούμε να δώσουμε τα βήματα που ακολουθεί ο αλγόριθμός μας για να κατασκευάσει την περιβάλλουσα καμπύλη, η οποία θα ακουμπάει τα πολύγωνα μηδενικού και πρώτου επιπέδου.

1. Κατασκεύασε τριγωνοποίηση Delaunay των σημείων των πολυγώνων
2. Κατασκεύασε τη συνάρτηση αντιστοίχησης των σημείων σε ακεραίους, έτσι ώστε όλα τα σημεία ενός πολυγώνου να αντιστοιχούν σε κοινό ακέραιο (με διαφοροποίηση του «απείρου σημείου»)
3. Βρες τις ακμές που αντιστοιχούν στην κυρτή θήκη των πολυγώνων, και εισήγαγέ τες σε μία λίστα L
4. Για κάθε ακμή e που ανήκει στη λίστα και δεν είναι ακμή πολυγώνου:
5. Βρες το τρίγωνο T που έχει τα άκρα του σε 3 διαφορετικά πολύγωνα
6. Αν η γωνία που επιστρέφει το κριτήριο είναι μεγαλύτερη από το όριο του χρήστη, τότε
7. Αντικατέστησε την e με τη σειρά ακμών που ακολούθησε ο αλγόριθμος για να ανακαλύψει το T
8. Αν από αυτήν την αντικατάσταση δημιουργείται αυτοτομή της καμπύλης, αντιμετώπισε την
9. Επέστρεψε τη λίστα

Στην επόμενη υποενότητα, θα επεκτείνουμε αυτόν τον αλγόριθμο, ώστε να ακουμπήσει και πολύγωνα σε μεγαλύτερο βάθος.

3.5 Γενικότεροι αλγόριθμοι κατασκευής περιβάλλουσας καμπύλης

Αν θέλουμε να ανακαλύψουμε πολύγωνα σε μεγαλύτερο βάθος μπορούμε να επιλέξουμε μεταξύ τριών διαφορετικών αλγορίθμων. Ο πρώτος είναι άμεση επέκταση του αλγορίθμου που περιγράψαμε στην προηγούμενη ενότητα, ενώ ο δεύτερος και περισσότερο ο τρίτος διαφοροποιούνται αρκετά από την κατασκευή αυτή. Όμως η ιδέα για όλους τους αλγορίθμους ξεκίνησε από τον παραπάνω αλγόριθμο, γι΄ αυτό και τον θεωρήσαμε ως βασική αρχή για τους υπολοίπους. Ο κάθε αλγόριθμος ανακαλύπτει τα πολύγωνα με διαφορετική σειρά. Η διαφορά στη σειρά ανακάλυψης τους, ίσως αλλάξει και το σχήμα της περιβάλλουσας καμπύλης, αλλά όχι πάντα, όπως θα δούμε στο επόμενο κεφάλαιο. Αναλόγως λοιπόν του πώς θέλουμε να αγγίξουμε τα πολύγωνα, επιλέγουμε και διαφορετικό αλγόριθμο.

Στο πρόγραμμά μας έχουμε υλοποιήσει τους δύο πρώτους αλγορίθμους. Και στους δύο η αναζήτηση των πολυγώνων συνεχίζεται έως ότου να μην μπορούν να ανακαλυφθούν άλλα πολύγωνα, εκτός και αν ο χρήστης δηλώσει μέχρι τι βάθος θέλει να φτάσει η ανακάλυψη. Εδώ να σημειώσουμε ότι αν το βάθος ανακάλυψης οριστεί 0 τότε η περιβάλλουσα καμπύλη ταυτίζεται με την κυρτή θήκη.

Θεώρημα 1. *Ο αλγόριθμος κατασκευής περιβάλλουσας καμπύλης τερματίζει πάντα.*

Απόδειξη. Κάθε τρίγωνο μπορεί να έχει ως άκρα κόμβους που είτε θα ανήκουν σε 1, είτε σε 2, είτε σε 3 διαφορετικά πολύγωνα.

Αν όλα του τα άκρα ανήκουν σε ένα πολύγωνο, τότε είναι τρίγωνο το οποίο βρίσκεται στο εσωτερικό ενός πολυγώνου, και δεν πρόκειται ο αλγόριθμος να ασχοληθεί ποτέ μαζί του, διότι αγνοεί αυτού του είδους τα τρίγωνα.

Αν έχει τα άκρα του σε 2 διαφορετικά πολύγωνα, τότε ο αλγόριθμος αναζήτησης, το πολύ να περάσει από αυτό το τρίγωνο 2 φορές, μία από κάθε πλευρά που ακουμπάει τα 2 πολύγωνα. Από την πλευρά που έχει τα άκρα της στο ίδιο πολύγωνο, αποκλείεται να περάσει κάποια αναζήτηση. Επίσης δεν πρόκειται να περάσει περισσότερες από 2 φορές από την ίδια πλευρά, διότι κάθε πλευρά τη βλέπει ξεκινώντας από μία ακμή της αρχικής περιβάλλουσας καμπύλης, και δεν υπάρχει τρόπος, να καταλήξει στην ίδια πλευρά ξεκινώντας από δύο διαφορετικές ακμές της καμπύλης.

Αν έχει τα άκρα του σε 3 διαφορετικά πολύγωνα, τότε ο αλγόριθμος αναζήτησης, το πολύ να περάσει από αυτό το τρίγωνο 3 φορές, μία από κάθε πλευρά, για τον λόγο που εξηγήσαμε στην προηγούμενη παράγραφο.

Φυσικά το πλήθος των τριγώνων της τριγωνοποίησης Delaunay των σημείων των πολυγώνων είναι πεπερασμένο. Αφού λοιπόν έχω n τρίγωνα, και κάθε τρίγωνο προσπελαύνεται το πολύ 3 φορές, τότε σίγουρα η κατασκευή της περιβάλλουσας καμπύλης θα τερματίσει. □

3.5.1 Αναδρομική κατασκευή τύπου BFS

Σε αυτή την αναδρομική μέθοδο, η αναζήτηση γίνεται ομοιόμορφα. Αρχικά ο αλγόριθμος, ξεκινώντας από την κυρτή θήκη, παίρνει μία μία τις ακμές της και αναζητά για νέα πολύγωνα, αντικαθιστώντας κάποιες από αυτές με μία σειρά άλλων ακμών και επιστρέφει σαν αποτέλεσμα μία πρώτη εκδοχή της περιβάλλουσας καμπύλης. Με αυτή την αναδρομική μέθοδο, αφού επιστραφεί η σειρά ακμών της περιβάλλουσας καμπύλης, ξανατρέχει τον αλγόριθμο, ξεκινώντας αυτή τη φορά από αυτήν την καμπύλη, αντί της κυρτής θήκης και επαναλαμβάνει την αναζήτηση νέων πολυγώνων επιστρέφοντας μία δεύτερη εκδοχή της περιβάλλουσας καμπύλης, με μία σειρά άλλων ακμών. Αυτή η διαδικασία συνεχίζεται ξεκινώντας πάντα από την προηγούμενη περιβάλλουσα καμπύλη και επιστρέφοντας μία νέα, έως ότου η νέα καμπύλη να ταυτιστεί με την παλιά. Τότε ο αλγόριθμος τερματίζει. Ο αλγόριθμος αυτής της μεθόδου είναι ο εξής:

1. Κατασκεύασε τριγωνοποίηση Delaunay των σημείων των πολυγώνων
2. Κατασκεύασε τη δομή αντιστοίχησης των σημείων σε ακεραίους έτσι ώστε όλα τα σημεία ενός πολυγώνου να αντιστοιχούν σε κοινό ακέραιο (με διαφοροποίηση του «απείρου σημείου»)

3. Κατασκεύασε τη δομή αντιστοίχησης των χωρίων σε λογικές μεταβλητές
4. Βρες τις ακμές που αντιστοιχούν στην κυρτή θήκη των πολυγώνων, και εισήγαγε τες σε
5. μία ουρά προτεραιότητας και
6. μία λίστα L που θα περιέχει τις ακμές της περιβάλλουσας καμπύλης
7. Όσο η ουρά δεν είναι άδεια:
8. Βγάλε από την ουρά μία ακμή e
9. Ξεκινώντας από την e, βρες το τρίγωνο T που έχει τα άκρα του σε 3 διαφορετικά πολύγωνα
10. Αν η γωνία που επιστρέφει το κριτήριο είναι μεγαλύτερη από το όριο του χρήστη, τότε:
11. Αντικατέστησε την e στην L με τη σειρά ακμών που ακολούθησε ο αλγόριθμος για να ανακαλύψει το T
12. Αν από αυτήν την αντικατάσταση δημιουργείται αυτοτομή της καμπύλης, αντιμετώπισε την
13. Αν οι ακμές της περιβάλλουσας καμπύλης δεν είναι ίδιες με τις ακμές που ανήκαν αρχικά στη λίστα τότε:
14. Βάλε στο τέλος της ουράς ακμών τις ακμές που ανήκουν στην περιβάλλουσα καμπύλη
15. Επέστρεψε στο βήμα 4
16. Επέστρεψε τη λίστα

Αν ο χρήστης έχει καθορίσει μέχρι ποιο βάθος θα αναζητήσει ο αλγόριθμος, τότε η επανάληψη από το βήμα 4 έως το 14 γίνεται τόσες φορές όσο είναι το προκαθορισμένο βάθος.

Ανάλυση πολυπλοκότητας:

Έστω ότι έχουμε ως είσοδο k πολύγωνα και το σύνολο των σημείων όλων των πολυγώνων είναι n.

- Η κατασκευή της τριγωνοποίησης Delaunay χρειάζεται χρόνο $O(n \log n)$ [15].
- Η κατασκευή της δομής αντιστοίχησης σημείων χρειάζεται χρόνο $O(n \log n)$.
- Η κατασκευή της δομής αντιστοίχησης χωρίων χρειάζεται χρόνο $O(n \log n)$.
- Η προσπέλαση όλων των ακμών της κυρτής θήκης χρειάζεται χρόνο $O(n)$.
- Αφού κάθε τρίγωνο δεν μπορεί να προσπελαστεί περισσότερες από 3 φορές, όπως δείξαμε στην απόδειξη του Θεωρήματος 1, και οι δύο επαναλήψεις συνολικά δε θα έχουν περισσότερα από $O(n)$ βήματα.

- Η εύρεση των τριγώνων με άκρα σε 3 διαφορετικά πολύγωνα, για το σύνολο της επανάληψης χρειάζεται χρόνο $O(n)$.

- Οι αντικαταστάσεις ακμών, για το σύνολο της επανάληψης, χρειάζονται χρόνο $O(n)$ γιατί δεν ασχολούμαστε ποτέ με κάποια ακμή περισσότερες από 2 φορές στη χειρότερη περίπτωση (συμπεριλαμβανομένης της περίπτωσης αυτοτομής).

- Η σύγκριση της λίστας με την περιβάλλουσα καμπύλη θέλει κάθε φορά σταθερό χρόνο, αν κρατάμε ένα δείκτη ο οποίος θυμάται αν κατά τη διάρκεια των βημάτων 5-8 έγινε κάποια αλλαγή στην καμπύλη.

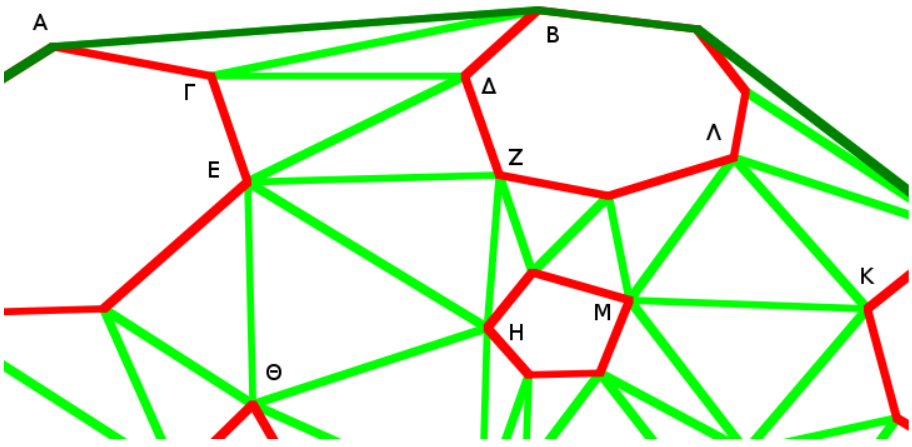
- Η εισαγωγή των ακμών στη λίστα, συνολικά θα γίνει σε χρόνο $O(n)$.

- Αν γίνει χρήση του κριτηρίου του τριγώνου, τότε ο έλεγχος αν η γωνία του τριγώνου θα γίνει αποδεκτή από το κριτήριο γίνεται σε σταθερό χρόνο. Άρα σε όλο το χρόνο εκτέλεσης του αλγορίθμου, ο χρόνος του κριτηρίου του τριγώνου είναι $O(n)$.

- Αν γίνει χρήση του κριτηρίου του κώνου, τότε κάθε φορά για να βρεθεί ο κώνος χρειάζεται χρόνος $O(n)$ για να ελεγχθούν όλα τα σημεία των δύο πολυγώνων στα οποία ακουμπάνε οι ακμές της γωνίας του κώνου. Άρα συνολικά ο έλεγχος του κριτηρίου του κώνου χρειάζεται χρόνο $O(n^2)$.

Άρα συνολικά ο αλγόριθμος χρειάζεται χρόνο $O(n \log n)$ αν χρησιμοποιείται το κριτήριο του τριγώνου και $O(n^2)$ αν χρησιμοποιείται το κριτήριο του κώνου. Αν χρησιμοποιηθεί για το κριτήριο του κώνου ο αλγόριθμος των Dobkin-Kirkpatrick που περιγράψαμε στο 3.2.2 τότε ο αλγόριθμός μας απαιτεί χρόνο $O(n \log n)$.

3.5.2 Αναδρομική κατασκευή τύπου DFS



Σχήμα 3.15:

Η δεύτερη μέθοδος κάνει αναζήτηση σε βάθος και όχι ομοιόμορφα όπως η προηγούμενη. Για ευκολότερη κατανόηση ας δούμε το παράδειγμα του Σχήματος 3.15. Αρχικά ο αλγόριθμος ξεκινάει από την ακμή AB η οποία βρίσκεται στην κυρτή θήκη, και χρησιμοποιώντας το κριτήριο βρίσκει το τρίγωνο EZH του οποίου οι κορυφές βρίσκονται σε τρία διαφορετικά πολύγωνα. Έστω επίσης ότι υπάρχει αποδοχή από το κριτήριο που χρησιμοποιείται, και αντικαθίσταται η AB με τις ακμές $A\Gamma$ - ΓE - EH - HZ - $Z\Delta$ - ΔB. Τώρα αρχίζει η αναδρομή.

Για κάθε μία από τις ακμές EH και HZ επαναλαμβάνουμε την ίδια διαδικασία που κάναμε και με την αρχική ακμή AB. Παίρνουμε δηλαδή την ακμή EH και αναζητάμε ένα τρίγωνο του οποίου οι κορυφές να ανήκουν σε τρία διαφορετικά πολύγωνα. Το τρίγωνο αυτό είναι το $EH\Theta$. Αν το κριτήριο δεχτεί τη γωνία $E\hat{\Theta}H$, τότε η EH αντικαθίσταται από τις ακμές $E\Theta$ - ΘH. Αντίστοιχα θα συνεχιστεί η αναδρομή για τις ακμές $E\Theta$ και ΘH.

Σε αυτό το σημείο είναι φανερό ότι το αποτέλεσμα της αναδρομής τύπου DFS εξαρτάται από την ακμή από την οποία ξεκίνησε ο αλγόριθμος. Όμως η επιλογή της αρχικής ακμής γίνεται τυχαία. Θα μπορούσε λοιπόν να αλλάξει ο τρόπος επιλογής και να επιλέγεται για κάθε είσοδο πολυγώνων, η ακμή εκείνη η οποία ανήκει σε ένα τρίγωνο τέτοιο ώστε η γωνία που σχηματίζεται απέναντί της να είναι η μεγαλύτερη μεταξύ όλων των αντίστοιχων γωνιών που είναι απέναντι από τις υπόλοιπες ακμές της κυρτής θήκης.

Ο αλγόριθμος της μεθόδου είναι ο παρακάτω.

1. Κατασκεύασε τριγωνοποίηση Delaunay των σημείων των πολυγώνων
2. Κατασκεύασε τη δομή αντιστοίχησης των σημείων σε ακεραίους έτσι ώστε όλα τα σημεία ενός πολυγώνου να αντιστοιχούν σε κοινό ακέραιο (με διαφοροποίηση του «απείρου σημείου»)
3. Κατασκεύασε τη δομή αντιστοίχησης των χωρίων σε λογικές μεταβλητές
4. Βρες τις ακμές που αντιστοιχούν στην κυρτή θήκη των πολυγώνων, και εισήγαγε τες σε:
5. μία στοίβα
6. μία λίστα L που θα περιέχει τις ακμές της περιβάλλουσας καμπύλης
7. Όσο η στοίβα δεν είναι άδεια:
8. Βγάλε από τη στοίβα μία ακμή e
9. Ξεκινώντας από την e βρες το τρίγωνο T που έχει τα άκρα του σε 3 διαφορετικά πολύγωνα (Έστω e_1 και e_2 οι ακμές του τριγώνου που ακουμπάνε στο εσωτερικό πολύγωνο)
10. Αν η γωνία που επιστρέφει το κριτήριο είναι μεγαλύτερη από το όριο του χρήστη, τότε:
11. Αντικατέστησε την e στην L με τη σειρά ακμών που ακολούθησε ο αλγόριθμος για να ανακαλύψει το T
12. Αν από αυτήν την αντικατάσταση δημιουργείται αυτοτομή της

καμπύλης, αντιμετώπισε την
13. Βάλε στη στοίβα τις ακμές e_1 και e_2
14. Επέστρεψε τη λίστα L

Αν ο χρήστης έχει καθορίσει μέχρι ποιο βάθος θα αναζητήσει ο αλγόριθμος, τότε η εισαγωγή, στην κορυφή της στοίβας, των ακμών e_1 και e_2 γίνεται τόσες φορές όσο είναι το προκαθορισμένο βάθος.

Ανάλυση πολυπλοκότητας:

Έστω ότι έχουμε ως είσοδο k πολύγωνα και το σύνολο των σημείων όλων των πολυγώνων είναι n.

- Η κατασκευή της τριγωνοποίησης Delaunay χρειάζεται χρόνο $O(n \log n)$ [15].
- Η κατασκευή της δομής αντιστοίχησης σημείων χρειάζεται χρόνο $O(n \log n)$.
- Η κατασκευή της δομής αντιστοίχησης χωρίων χρειάζεται χρόνο $O(n \log n)$.
- Η προσπέλαση όλων των ακμών της κυρτής θήκης χρειάζεται χρόνο $O(n)$.
- Αφού κάθε τρίγωνο δεν μπορεί να προσπελαστεί περισσότερες από 3 φορές, όπως δείξαμε στην απόδειξη του Θεωρήματος 1, τα βήματα της επανάληψης δε θα γίνουν περισσότερες από $O(n)$ φορές.
- Η εύρεση των τριγώνων με άκρα σε 3 διαφορετικά πολύγωνα, για το σύνολο της επανάληψης χρειάζεται χρόνο $O(n)$.
- Οι αντικαταστάσεις ακμών, για το σύνολο της επανάληψης, χρειάζονται χρόνο $O(n)$ γιατί δεν ασχολούμαστε ποτέ με κάποια ακμή περισσότερες από 2 φορές στη χειρότερη περίπτωση (συμπεριλαμβανομένης της περίπτωσης αυτοτομής).
- Αν γίνει χρήση του κριτηρίου του τριγώνου, τότε ο έλεγχος αν η γωνία του τριγώνου θα γίνει αποδεκτή από το κριτήριο γίνεται σε σταθερό χρόνο. Άρα σε όλο το χρόνο εκτέλεσης του αλγορίθμου, ο χρόνος του κριτηρίου του τριγώνου είναι $O(n)$.
- Αν γίνει χρήση του κριτηρίου του κώνου, τότε κάθε φορά για να βρεθεί ο κώνος χρειάζεται χρόνος $O(n)$ για να ελεγχθούν όλα τα σημεία των δύο πολυγώνων στα οποία ακουμπάνε οι ακμές της γωνίας του κώνου. Άρα συνολικά ο έλεγχος του κριτηρίου του κώνου χρειάζεται χρόνο $O(n^2)$.

Άρα συνολικά ο αλγόριθμος χρειάζεται χρόνο $O(n \log n)$ αν χρησιμοποιείται το κριτήριο του τριγώνου και $O(n^2)$ αν χρησιμοποιείται το κριτήριο του κώνου. Αν χρησιμοποιηθεί για το κριτήριο του κώνου ο αλγόριθμος των Dobkin-Kirkpatrick που περιγράψαμε στο 3.2.2, τότε ο αλγόριθμός μας απαιτεί χρόνο $O(n \log n)$.

3.5.3 Κατασκευή με τη βοήθεια ουράς προτεραιότητας

Ο αλγόριθμος της μεθόδου είναι ο παρακάτω.

1. Κατασκεύασε τριγωνοποίηση Delaunay των σημείων των πολυγώνων
2. Κατασκεύασε τη δομή αντιστοίχησης των σημείων σε ακεραίους έτσι ώστε όλα τα σημεία ενός πολυγώνου να αντιστοιχούν σε κοινό ακέραιο (με διαφοροποίηση του «απείρου σημείου»)
3. Κατασκεύασε τη δομή αντιστοίχησης των χωρίων σε λογικές μεταβλητές
4. Βρες τις ακμές της κυρτής θήκης και εισήγαγε τες σε:
5. Μία max-ουρά προτεραιότητας Q με βάση τη γωνία που χρησιμοποιείται για το κριτήριο αποδοχής
6. Μία λίστα L που θα περιέχει τις ακμές της περιβάλλουσας καμπύλης
7. Όσο η ουρά προτεραιότητας δεν είναι άδεια:
8. Βγάλε την μέγιστη ακμή e από την ουρά προτεραιότητας
9. Αν η γωνία που αντιστοιχεί στην e είναι μεγαλύτερη από το όριο του χρήστη
10. Αφαίρεσε την e από τη λίστα L, και πρόσθεσε στη θέση της τη σειρά ακμών που ακολούθησε ο αλγόριθμος για να ανακαλύψει το τρίγωνο της e
11. Βάλε τις άλλες δύο ακμές, που ανήκουν στο ίδιο τρίγωνο με την e, στην ουρά προτεραιότητας
12. Επέστρεψε τη λίστα L

Αν ο χρήστης έχει καθορίσει το βάθος στο οποίο επιτρέπεται να φτάσει ο αλγόριθμος γίνεται το εξής: Κάθε ακμή που είναι στην ουρά προτεραιότητας κρατάει και το βάθος που έχει το τρίγωνο στο οποίο ανήκει. Κάθε φορά που εκτελείται το βήμα 11, οι ακμές που μπαίνουν στην ουρά προτεραιότητας θα κρατήσουν ως βάθος το βάθος της ακμής e συν 1. Άρα αν οι ακμές που πρόκειται να μπουν στη ουρά προτεραιότητας θα έχουν βάθος μεγαλύτερο από το επιτρεπτό, τότε δεν εισάγονται.

Ανάλυση πολυπλοκότητας:
Έστω ότι έχουμε ως είσοδο k πολύγωνα με n στο σύνολο σημεία.

- Η κατασκευή της τριγωνοποίησης Delaunay χρειάζεται χρόνο $O(n \log n)$ [15].
- Η κατασκευή της δομής αντιστοίχησης σημείων χρειάζεται χρόνο $O(n \log n)$.
- Η κατασκευή της δομής αντιστοίχησης χωρίων χρειάζεται χρόνο $O(n \log n)$.
- Η προσπέλαση όλων των ακμών της κυρτής θήκης και εισαγωγή τους σε max-ουρά προτεραιότητας, χρειάζεται χρόνο $O(n \log n)$.
- Η εισαγωγή των ακμών της κυρτής θήκης σε λίστα, χρειάζεται χρόνο $O(n)$.

- Αφού κάθε τρίγωνο δεν μπορεί να προσπελαστεί περισσότερες από 3 φορές, όπως δείξαμε στην απόδειξη του Θεωρήματος 1, τα βήματα της επανάληψης δε θα γίνουν περισσότερες από $O(n)$ φορές.

- Αφαίρεση όλων των ακμών από την ουρά προτεραιότητας σε χρόνο $O(n \log n)$.

- Η εύρεση των τριγώνων με άκρα σε 3 διαφορετικά πολύγωνα, για το σύνολο της επανάληψης χρειάζεται χρόνο $O(n)$.

- Οι αντικαταστάσεις ακμών, για το σύνολο της επανάληψης, χρειάζονται χρόνο $O(n)$ γιατί δεν ασχολούμαστε ποτέ με κάποια ακμή περισσότερες από 2 φορές στη χειρότερη περίπτωση (συμπεριλαμβανομένης της περίπτωσης αυτοτομής).

- Αν γίνει χρήση του κριτηρίου του τριγώνου, τότε ο έλεγχος αν η γωνία του τριγώνου θα γίνει αποδεκτή από το κριτήριο γίνεται σε σταθερό χρόνο. Άρα σε όλο το χρόνο εκτέλεσης του αλγορίθμου, ο χρόνος του κριτηρίου του τριγώνου είναι $O(n)$.

- Αν γίνει χρήση του κριτηρίου του κώνου, τότε κάθε φορά για να βρεθεί ο κώνος χρειάζεται χρόνος $O(n)$ για να ελεγχθούν όλα τα σημεία των δύο πολυγώνων στα οποία ακουμπάνε οι ακμές της γωνίας του κώνου. Άρα συνολικά ο έλεγχος του κριτηρίου του κώνου χρειάζεται χρόνο $O(n^2)$.

Άρα συνολικά ο αλγόριθμος χρειάζεται χρόνο $O(n \log n)$ αν χρησιμοποιείται το κριτήριο του τριγώνου και $O(n^2)$ αν χρησιμοποιείται το κριτήριο του κώνου. Αν χρησιμοποιηθεί για το κριτήριο του κώνου ο αλγόριθμος των Dobkin-Kirkpatrick που περιγράψαμε στο 3.2.2, τότε ο αλγόριθμός μας απαιτεί χρόνο $O(n \log n)$.

Κεφάλαιο 4

Παραδείγματα και αποτελέσματα

Στο κεφάλαιο αυτό γίνεται παράθεση κάποιων αποτελεσμάτων του προγράμματος που κατασκευάσαμε, από τα οποία προκύπτουν κάποια συμπεράσματα, για τις ιδιότητες που προσδίδουν στην καμπύλη οι διαφορετικές παράμετροι (αλγόριθμος αναζήτησης, κριτήριο επιλογής, γωνία χρήστη, βάθος αναζήτησης) που επιλέγονται.

Στις εικόνες όλων των σχημάτων που υπάρχουν σε αυτό το κεφάλαιο, με κόκκινο έχουν σχεδιαστεί τα κυρτά πολύγωνα, με σκούρο πράσινο η περιβάλλουσα καμπύλη και με ανοικτό πράσινο τα τρίγωνα της τριγωνοποίησης Delaunay που έχουν τα άκρα τους σε τρία διαφορετικά πολύγωνα.

4.1 Παράδειγμα 1

Αρχικά θα δώσουμε το αποτέλεσμα του παραδείγματος με το οποίο ασχοληθήκαμε στο Κεφάλαιο 3 (βλέπε Σχήμα 3.3) για τις δύο μεθόδους (BFS και DFS) έχοντας ως δοθείσα γωνία από το χρήστη, τη μηδενική γωνία και χωρίς περιορισμό βάθους. Αυτό το κάνουμε ώστε να δούμε καθαρά με ποιο τρόπο προσεγγίζει η κάθε μέθοδος τα πολύγωνα. Τα αποτελέσματα του συγκεκριμένου παραδείγματος, ενδεχομένως να μην είναι αυτά που θα επιλέγαμε για την περιβάλλουσα καμπύλη, αλλά τα παραθέτουμε για να γίνει ξεκάθαρος οπτικά, ο διαφορετικός τρόπος προσέγγισης των δύο αλγορίθμων.

Η είσοδος του αλγορίθμου για την παραγωγή αυτού του παραδείγματος είναι η εξής:

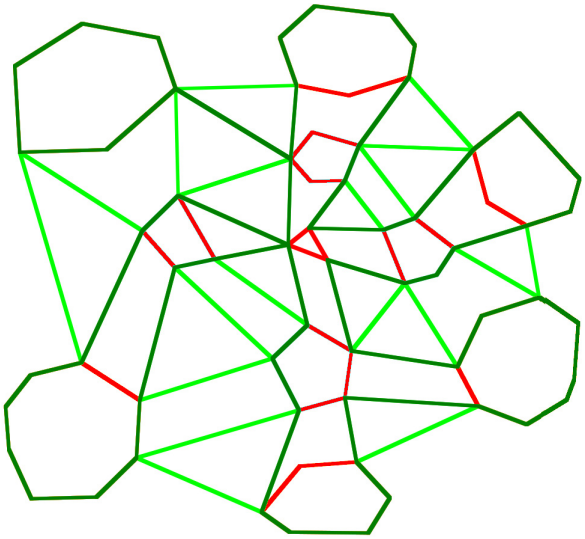
Δεδομένα παραδείγματος 1

```
( (15, 636), (106, 693), (208, 674), (231, 606), (138, 524), (22, 521) )
( (418, 717), (371, 674), (394, 611), (464, 597), (545, 622), (553, 664),
  (521, 705) )
( (693, 574), (631, 524), (650, 454), (703, 423), (762, 442),
  (774, 485) )
( (719, 326), (642, 302), (610, 233), (638, 181), (703, 156), (755, 187),
  (765, 214), (770, 290) )
( (474, 106), (398, 100), (346, 39), (385, 12), (490, 11), (519, 58) )
( (7, 123), (3, 180), (37, 222), (105, 239), (183, 189), (179, 111),
  (125, 59), (38, 56) )
( (235, 463), (187, 416), (230, 367), (284, 378) )
( (361, 245), (408, 289), (468, 255), (459, 192), (397, 175) )
( (539, 345), (510, 417), (553, 433), (605, 392), (582, 356) )
( (386, 512), (415, 548), (478, 530), (459, 483), (413, 482) )
( (382, 397), (410, 419), (435, 377) )
```

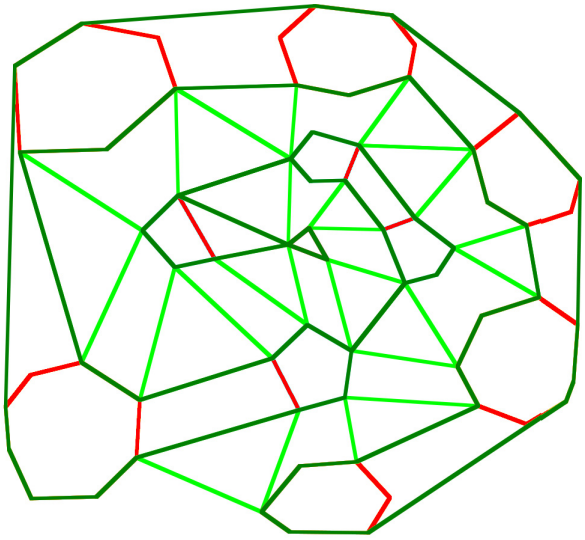
Δεδομένα παραδείγματος 1

Στο Σχήμα 4.1 φαίνεται το αποτέλεσμα της μεθόδου τύπου BFS και στο Σχήμα 4.2 το αποτέλεσμα της μεθόδου τύπου DFS.

Ο αλγόριθμος τύπου BFS έφτασε βάθος τριγώνου μόλις 3 και η καμπύλη εισχωρεί προς το πιο εσωτερικό πολύγωνο με παρόμοιο τρόπο από κάθε ακμή της κυρτής θήκης. Αντίθετα ο αλγόριθμος τύπου DFS έφτασε βάθος τριγώνου έως και 14 και το αποτέλεσμα της περιβάλλουσας καμπύλης θυμίζει έντονα έλικα. Διαισθητικά γίνεται φανερό ότι παρ΄ όλο που το αποτέλεσμα της καμπύλης εξαρτάται από το ποια είναι η πρώτη ακμή που εξετάζει ο αλγόριθμος, το ελικοειδές αποτέλεσμα θα συνέβαινε ανεξαρτήτως πρώτης ακμής.



Σχήμα 4.1: Περιβάλλουσα καμπύλη που σχηματίστηκε με τη μέθοδο BFS, όριο γωνίας 0 και μη φραγμένο βάθος. (Το κριτήριο αποδοχής για μηδενική γωνία δεν παίζει ρόλο, γιατί όλα τα κριτήρια αποδέχονται όλες τις γωνίες.)



Σχήμα 4.2: Περιβάλλουσα καμπύλη που σχηματίστηκε με τη μέθοδο DFS, όριο γωνίας 0 και μη φραγμένο βάθος.

4.2 Παράδειγμα 2

Η είσοδος που δημιουργεί το παρόν παράδειγμα είναι η παρακάτω:

Δεδομένα παραδείγματος 2

```
( (52, 680), (140, 694), (129, 602) )
( (222, 602), (273, 539), (195, 540) )
( (323, 490), (355, 397), (255, 460) )
( (418, 482), (511, 483), (509, 409) )
( (660, 553), (681, 473), (595, 516) )
( (705, 592), (765, 637), (777, 534) )
( (616, 165), (743, 198), (724, 82) )
( (112, 254), (204, 194), (196, 136), (63, 129) )
```

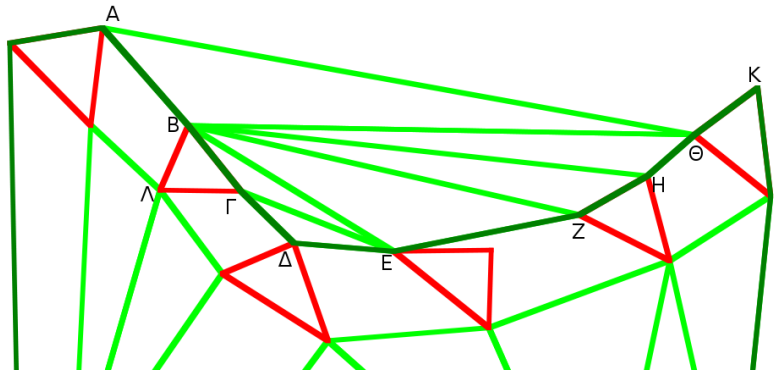
Δεδομένα παραδείγματος 2

Εδώ θα κάνουμε μία σύγκριση μεταξύ του κριτηρίου του τριγώνου και του κριτηρίου του κώνου. Στο Σχήμα 4.3 φαίνεται η μεγέθυνση της κοιλότητας που σχηματίζεται στο πάνω μέρος του Σχήματος 4.4. Θέλουμε να δούμε, καθένας από τους δύο αλγορίθμους που έχουμε υλοποιήσει, με κάθε κριτήριο, από ποια γωνία χρήστη και κάτω καταφέρνει να κάνει την καμπύλη να ακουμπάει όλα τα πολύγωνα της κοιλότητας αυτής (έχοντας δώσει στους αλγορίθμους το αναγκαίο βάθος ώστε να τα φτάσει).

Φυσικά το αποτέλεσμα που αναμένουμε είναι ότι το κριτήριο του κώνου θα είναι αυστηρότερο από το κριτήριο του τριγώνου, διότι εξ΄ ορισμού, η γωνία του κώνου είναι πάντα μικρότερη ή ίση της γωνίας του τριγώνου.

Τώρα θα παραθέσουμε τα μεγέθη κάποιων από τις γωνίες που σχηματίζονται στο παράδειγμά μας, ώστε να δοθεί αναλυτική εξήγηση της συμπεριφοράς της καμπύλης για την κάθε περίπτωση:

- $A\hat{B}\Theta = 132.9^\circ$
- $B\hat{H}\Theta = 132.7^\circ$
- $B\hat{E}Z = 137.64^\circ$
- $A\hat{B}K = 128.02^\circ$
- $B\hat{E}H = 132.17^\circ$
- $\Gamma\hat{\Delta}E = 140.4^\circ$
- $B\hat{\Delta}E = 136.86^\circ$



Σχήμα 4.3: Μεγέθυνση της κοιλότητας του Σχήματος 4.4.

Το τρίγωνο $AB\Theta$, το οποίο έχει βάθος 1, σχηματίζει γωνία μεγαλύτερη από τη γωνία του επόμενου σε βάθος τριγώνου $BH\Theta$. Άρα πρέπει να υπάρχει κάποιο όριο χρήστη μικρότερο από 132.9° και μεγαλύτερο από 132.7°, στο οποίο η καμπύλη θα ακουμπάει μόνο το πολύγωνο $B\Gamma\Lambda$ και κανένα άλλο από τα εσωτερικά πολύγωνα της κοιλότητας, ανεξαρτήτως του βάθους που επιτρέπουμε να φτάσει.

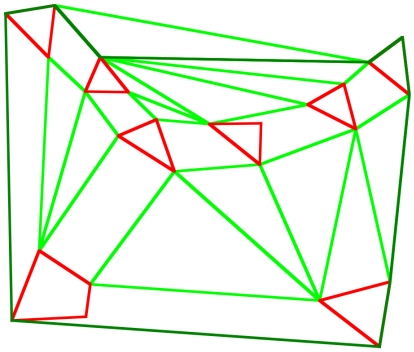
Η υπόθεση αυτή επιβεβαιώθηκε από το πρόγραμμα, το οποίο με κριτήριο του τριγώνου, και γωνία 132.8° και για τους δύο αλγορίθμους, (χωρίς περιορισμό βάθους) δίνει την καμπύλη του Σχήματος 4.4(α′).

Αν ο χρήστης ορίσει γωνία μικρότερη από 132.7°, τότε η καμπύλη θα περάσει σίγουρα από τα τρίγωνα $AB\Theta$ και $BH\Theta$. Συνεχίζοντας, επειδή οι γωνίες των τριγώνων BEZ και $\Gamma\Delta E$ είναι και οι 2 μεγαλύτερες από 132.7°, η καμπύλη θα περάσει και από αυτά τα τρίγωνα.

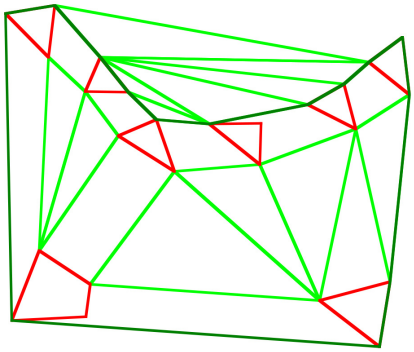
Πράγματι, το πρόγραμμά μας, με τον αλγόριθμο τύπου BFS, το κριτήριο του τριγώνου και με γωνία χρήστη 131^o καταφέρνει να ακουμπήσει όλα τα πολύγωνα της κοιλότητας, όπως δείχνει το Σχήμα 4.4(β′), και με τον αλγόριθμο τύπου DFS, το κριτήριο του τριγώνου και με γωνία χρήστη 132.6^o καταφέρνει και αυτό να ακουμπήσει όλα τα πολύγωνα της κοιλότητας (βλέπε Σχήμα 4.4(β′)).

Ας εξετάσουμε τώρα τη συμπεριφορά της καμπύλης για το κριτήριο του κώνου: Ελέγχοντας τις παραπάνω γωνίες βλέπουμε ότι στο τρίγωνο $AB\Theta$ η γωνία του κώνου θα είναι η $A\hat{B}K$, του BEZ θα είναι η $B\hat{E}H$ και του $\Gamma\Delta E$ θα είναι η $B\hat{\Delta}E$. Στο $BH\Theta$ η γωνία του κώνου ταυτίζεται με αυτή του τριγώνου. Από αυτές τις γωνίες, η γωνία του $AB\Theta$ (το οποίο έχει βάθος 1) είναι και η μικρότερη (ισούται με 128.02°), συνεπώς όταν το όριο του χρήστη γίνει μικρότερο από 128.02° αναμένουμε να ακουμπήσει η καμπύλη σε όλα τα πολύγωνα της κοιλότητας, διότι όλες οι επόμενες γωνίες που θα ελέγξει θα είναι μεγαλύτερες και θα γίνουν αποδεκτές (αν το βάθος δεν είναι φραγμένο).

Πράγματι, στο πρόγραμμα βλέπουμε ότι και στους δύο αλγορίθμους (BFS και DFS) το κριτήριο του κώνου με γωνία χρήστη 127° ακουμπάει όλα τα πολύγωνα της κοιλότητας,



(α′) Εδώ χρησιμοποιήσαμε κριτήριο τριγώνου, με γωνία 132° για τον αλγόριθμο τύπου BFS και με γωνία 132.8° για τον αλγόριθμο τύπου DFS. (Βάθος μεγαλύτερο ή ίσο με 1)



(β′) Η καμπύλη περνάει από την κοιλότητα που σχηματίζουν τα πολύγωνα. Το αποτέλεσμα αυτό παίρνουμε αν εφαρμόσουμε το κριτήριο του τριγώνου με γωνία χρήστη 131° τόσο στον αλγόριθμο τύπου BFS όσο και στον αλγόριθμο τύπου DFS, καθώς και με το κριτήριο του κώνου και γωνία χρήστη 127° και στους δύο αλγορίθμους. (Βάθος μεγαλύτερο ή ίσο με 4)

Σχήμα 4.4: Δύο στιγμιότυπα της περιβάλλουσας καμπύλης.

ενώ για μεγαλύτερη γωνία ταυτίζεται με την κυρτή θήκη, όπως δείχνει το Σχήμα 4.4(β′).

4.3 Παράδειγμα 3

Η είσοδος αυτού του παραδείγματος είναι η παρακάτω:

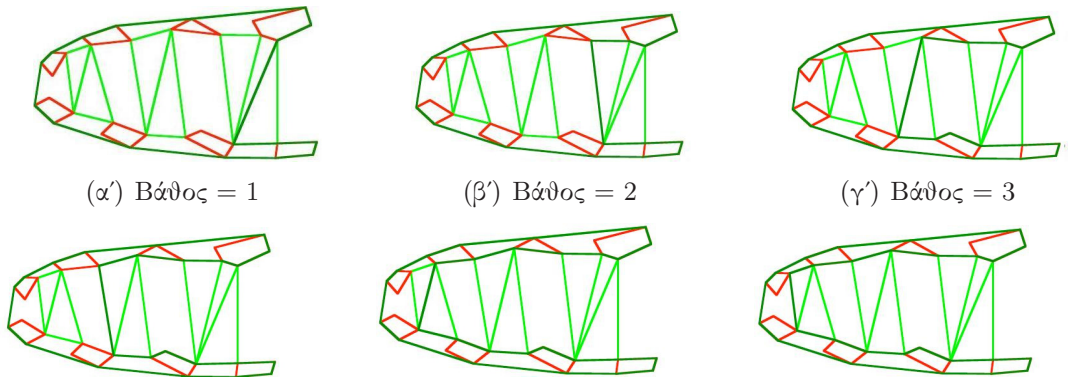
Δεδομένα παραδείγματος 3

```
( (79, 486), (104, 511), (138, 510), (106, 456) )
( (98, 403), (63, 386), (109, 343), (156, 368) )
( (251, 344), (223, 317), (292, 285), (330, 315) )
( (198, 530), (178, 548), (258, 575), (293, 540) )
( (390, 567), (438, 592), (507, 553) )
( (585, 586), (604, 552), (643, 540), (724, 577), (710, 618) )
( (424, 309), (461, 325), (538, 293), (518, 261) )
( (643, 295), (639, 260), (728, 268), (737, 298) )
```

Δεδομένα παραδείγματος 3

Εδώ θα δείξουμε πώς το βάθος που ορίζουμε, επηρεάζει όπως αναμένεται, την περιβάλλουσα καμπύλη.

Αν χρησιμοποιήσουμε οποιοδήποτε αλγόριθμο, με κριτήριο τριγώνου, γωνία χρήστη 65° και μη φραγμένο βάθος, ή κριτήριο κώνου, γωνία χρήστη 56° και μη φραγμένο βάθος, παίρνουμε την καμπύλη του Σχήματος 4.5(ϛ′). Αν όμως αντί για μη φραγμένο βάθος, περιορίσουμε το βάθος σε $1, 2, \ldots, 5$ τότε παίρνουμε τις καμπύλες που φαίνονται στα Σχήματα 4.5(α′)-4.5(ε′).



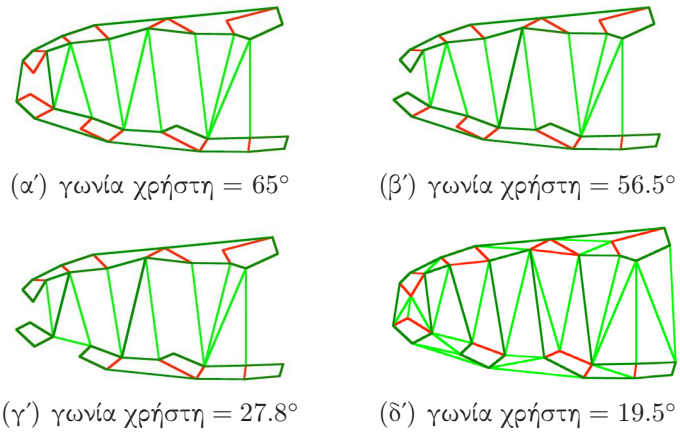
(α′) Βάθος $= 1$ (β′) Βάθος $= 2$ (γ′) Βάθος $= 3$

(δ′) Βάθος $= 4$ (ε′) Βάθος $= 5$ (ϛ′) Βάθος ≥ 6

Σχήμα 4.5: Με οποιονδήποτε αλγόριθμο και γωνία χρήστη 65° στο κριτήριο του τριγώνου και 56° στο κριτήριο του κώνου, παίρνουμε τις παραπάνω καμπύλες αν το βάθος κυμανθεί στις τιμές $\{1, 2, 3, 4, 5, 6\}$.

Στα Σχήματα 4.6, 4.7, 4.8 και 4.9 φαίνονται διάφορα στιγμιότυπα της καμπύλης για μη φραγμένο βάθος, διαφοροποιώντας το είδος του αλγορίθμου, τα κριτήρια και τη γωνία. Από αυτά τα στιγμιότυπα σίγουρα μπορεί κανείς να καταλάβει ότι από μία γωνία χρήστη και κάτω, (της οποίας το μέτρο εξαρτάται από το κριτήριο, στο παράδειγμά μας είναι 56.5° για

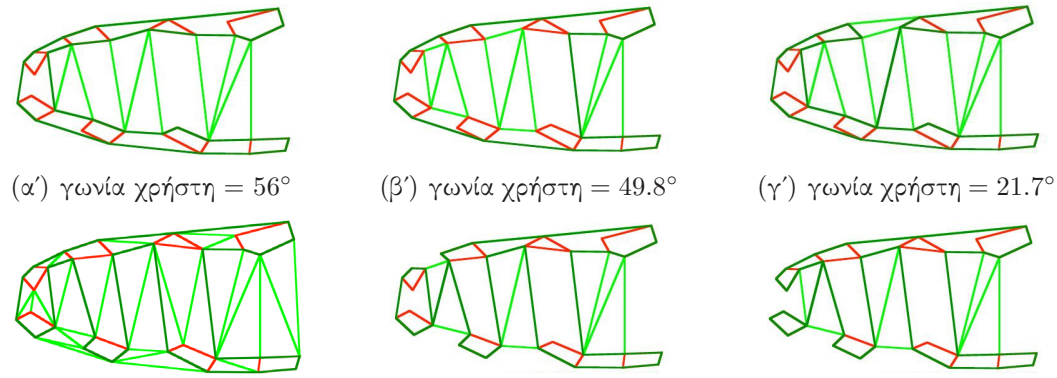
το κριτήριο τριγώνου και $49.8°$ για κώνου), το αποτέλεσμα της περιβάλλουσας καμπύλης και στους δύο αλγορίθμους δεν είναι το αναμενόμενο, σύμφωνα με τη γεωμετρία του σχήματος. Υπάρχει όμως μία περιοχή, την οποία μπορούμε να χαρακτηρίσουμε ως *περιοχή ισορροπίας* (στο παράδειγμά μας $65° - 56.5°$ για το κριτήριο του τριγώνου και $56° - 49.9°$ για του κώνου) στην οποία η καμπύλη σταθεροποιείται σε ένα σχήμα ανάλογο της γεωμετρίας των δεδομένων, όπως δείξαμε. Γενικά μπορεί να υπάρχουν περισσότερες από μία περιοχές ισορροπίας, αναλόγως της εισόδου.



(α΄) γωνία χρήστη $= 65°$ (β΄) γωνία χρήστη $= 56.5°$

(γ΄) γωνία χρήστη $= 27.8°$ (δ΄) γωνία χρήστη $= 19.5°$

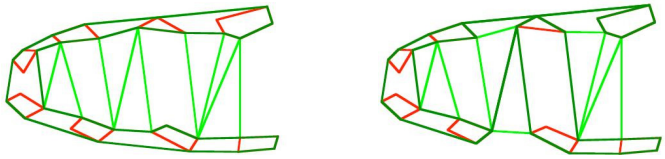
Σχήμα 4.6: Στιγμιότυπα της περιβάλλουσας καμπύλης με χρήση αλγορίθμου τύπου BFS, χωρίς όριο βάθους και κριτήριο τριγώνου, για διάφορες γωνίες χρήστη.



(α΄) γωνία χρήστη $= 56°$ (β΄) γωνία χρήστη $= 49.8°$ (γ΄) γωνία χρήστη $= 21.7°$

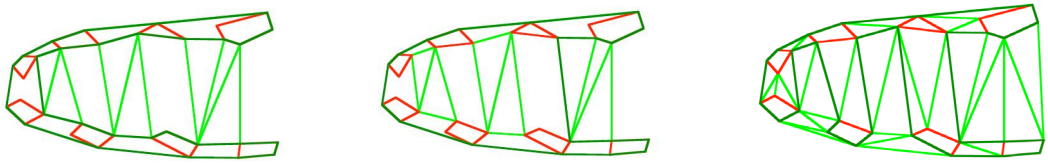
(δ΄) γωνία χρήστη $= 20.1°$ (ε΄) γωνία χρήστη $= 12.9°$ (ϛ΄) γωνία χρήστη $= 7.4°$

Σχήμα 4.7: Στιγμιότυπα της περιβάλλουσας καμπύλης με χρήση αλγορίθμου τύπου BFS, χωρίς όριο βάθους και κριτήριο κώνου, για διάφορες γωνίες χρήστη.



(α′) γωνία χρήστη = 65° (β′) γωνία χρήστη = 0°

Σχήμα 4.8: Στιγμιότυπα της περιβάλλουσας καμπύλης με χρήση αλγορίθμου τύπου DFS, χωρίς όριο βάθους και κριτήριο τριγώνου, για διάφορες γωνίες χρήστη.



(α′) γωνία χρήστη = 56° (β′) γωνία χρήστη = 50.7° (γ′) γωνία χρήστη = 18.1°

Σχήμα 4.9: Στιγμιότυπα της περιβάλλουσας καμπύλης με χρήση αλγορίθμου τύπου DFS, χωρίς όριο βάθους και κριτήριο κώνου, για διάφορες γωνίες χρήστη.

4.4 Παράδειγμα 4

Τα δεδομένα για το τελευταίο μας παράδειγμα είναι:

Δεδομένα παραδείγματος 3

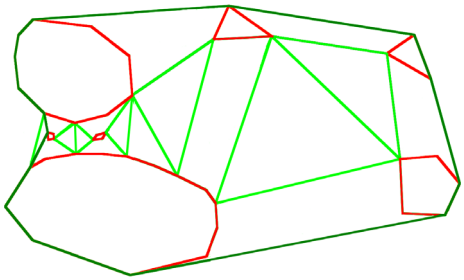
```
( (185, 637), (153, 604), (146, 558), (153, 491), (208, 437),
  (274, 417), (343, 434), (398, 475), (391, 559), (309, 622) )
( (178, 322), (211, 340), (277, 351), (331, 351), (386, 345),
  (447, 324), (494, 303), (555, 273), (576, 245), (579, 192),
  (556, 133), (393,  93), (184, 167), (125, 239) )
( (604, 665), (572, 594), (696, 600) )
( (1001, 604), (942, 564), (1036, 510) )
( (970, 339), (976, 222), (1066, 219), (1098, 287), (1049, 345) )
( (216, 396), (217, 381), (229, 383), (229, 393) )
( (321, 391), (313, 381), (333, 383), (340, 396) )
```

Δεδομένα παραδείγματος 3

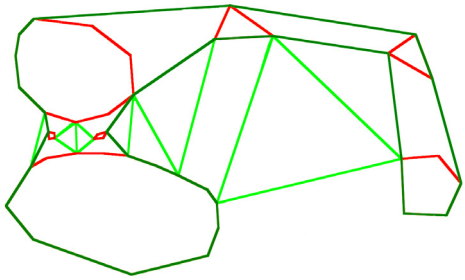
Εδώ μελετάμε άλλο ένα αναλυτικό παράδειγμα, στο οποίο εμφανίζονται πολλές παραλλαγές της περιβάλλουσας καμπύλης, στα ίδια δεδομένα, εφαρμόζοντας διαφορετικούς συνδυασμούς αλγορίθμων - κριτηρίων - γωνίας. Στόχος μας είναι να καταλάβουμε πώς συμπεριφέρεται το πρόγραμμα σε διαφορετικού είδους δεδομένα, ώστε να καταλήξουμε σε κάποια συμπεράσματα (βλέπε Παράγραφο 4.5).

Για πολύ μεγάλες γωνίες, η καμπύλη, αφού διαφοροποιηθεί από την κυρτή θήκη, μοιάζει όπως δείχνει το Σχήμα 4.10(α΄). Αυτήν την καμπύλη μπορούμε να τη βρούμε ως αποτέλεσμα και με τους δύο αλγορίθμους και τα κατάλληλα κριτήρια - γωνίες (βλέπε Σχήματα 4.11(α΄), 4.12(α΄), 4.13(α΄)).

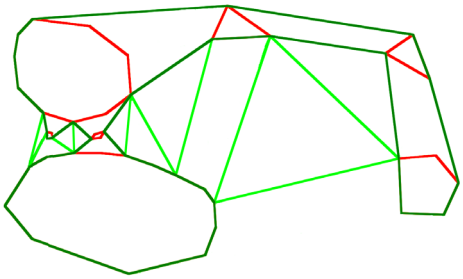
Καθώς μικραίνουν οι γωνίες, η καμπύλη αλλάζει και παίρνει διάφορες μορφές, οι οποίες φαίνονται στα Σχήματα 4.10, 4.11, 4.12 και 4.13.



(α΄) γωνία χρήστη = 140°

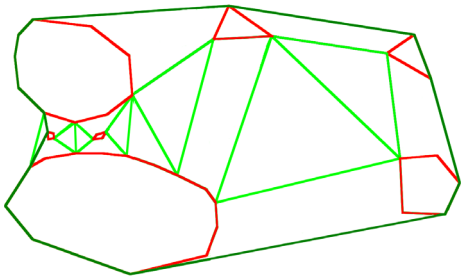


(β΄) γωνία χρήστη = 55°

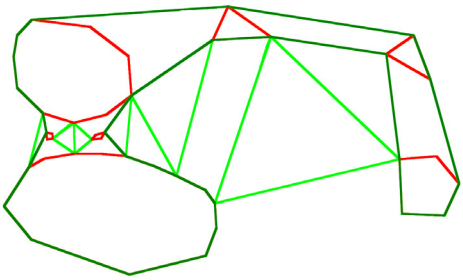


(γ΄) γωνία χρήστη ≤ 55°

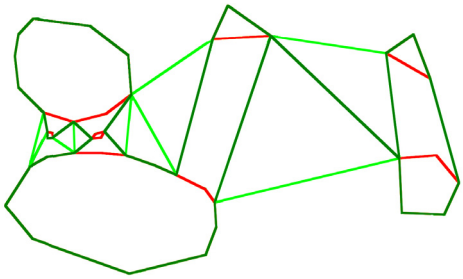
Σχήμα 4.10: Στιγμιότυπα της περιβάλλουσας καμπύλης με χρήση αλγορίθμου τύπου BFS, χωρίς όριο βάθους και κριτήριο τριγώνου, για διάφορες γωνίες χρήστη.



(α΄) γωνία χρήστη = 114°

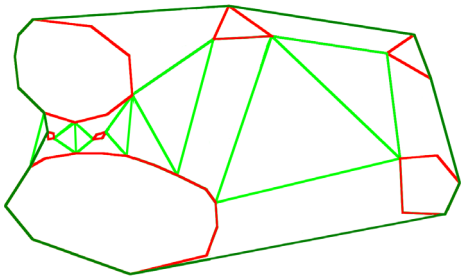


(β΄) γωνία χρήστη = 52°

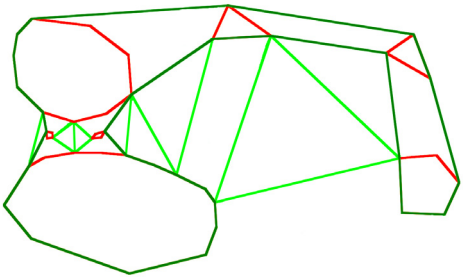


(γ΄) γωνία χρήστη ≤ 6°

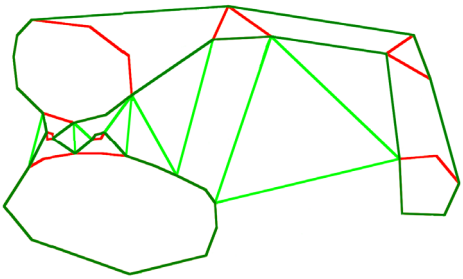
Σχήμα 4.11: Στιγμιότυπα της περιβάλλουσας καμπύλης με χρήση αλγορίθμου τύπου BFS, χωρίς όριο βάθους και κριτήριο κώνου, για διάφορες γωνίες χρήστη.



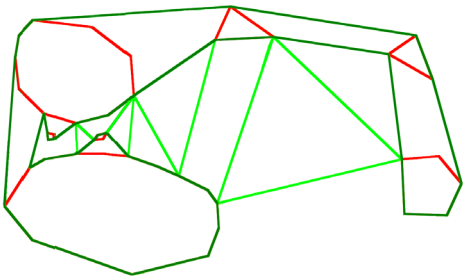
(α′) γωνία χρήστη = 140°



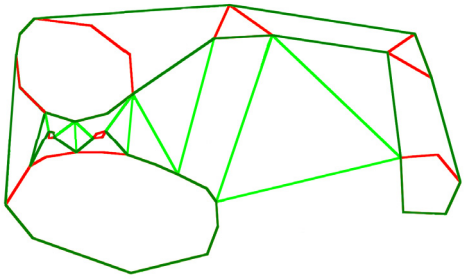
(β′) γωνία χρήστη = 55°



(γ′) γωνία χρήστη = 51°

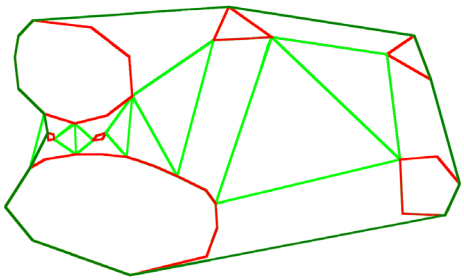


(δ′) γωνία χρήστη = 23°

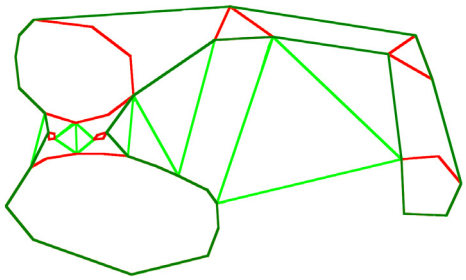


(ε′) γωνία χρήστη = 11°

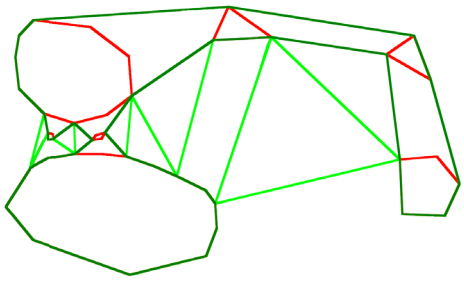
Σχήμα 4.12: Στιγμιότυπα της περιβάλλουσας καμπύλης με χρήση αλγορίθμου τύπου DFS, χωρίς όριο βάθους και κριτήριο τριγώνου, για διάφορες γωνίες χρήστη.



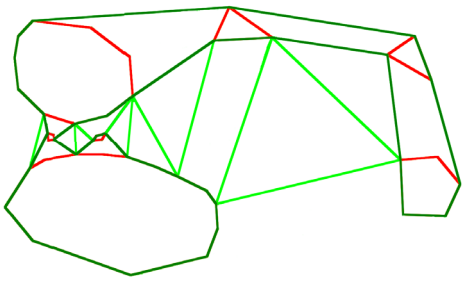
(α΄) γωνία χρήστη = 115°



(β΄) γωνία χρήστη = 52°



(γ΄) γωνία χρήστη = 7°



(δ΄) γωνία χρήστη = 2°

Σχήμα 4.13: Στιγμιότυπα της περιβάλλουσας καμπύλης με χρήση αλγορίθμου τύπου DFS, χωρίς όριο βάθους και κριτήριο κώνου, για διάφορες γωνίες χρήστη.

4.5 Συμπεράσματα και ανοικτά προβλήματα

Στην εργασία αυτή εξηγήσαμε πώς μπορεί να κατασκευαστεί μία καμπύλη η οποία να περιβάλλει κυρτά πολυγωνικά αντικείμενα, με τρόπο ώστε να ακουμπάει και αντικείμενα στο εσωτερικό της κυρτής θήκης των πολυγώνων. Στη συνέχεια υλοποιήσαμε την κατασκευή αυτή και τέλος, δοκιμάσαμε κάποια παραδείγματα για να δούμε τα αποτελέσματα που εξάγονται.

Από τα παραδείγματα αυτά, διαπιστώσαμε ότι για ίδια κριτήρια και ίδιο βάθος, όσο μεγαλύτερη είναι η γωνία του χρήστη, τόσο οι δύο αλγόριθμοι τείνουν να βγάζουν παρόμοια αποτελέσματα. Επιπλέον έγινε φανερό ότι για ιδιαίτερα μικρές γωνίες χρήστη, τα αποτελέσματα της περιβάλλουσας καμπύλης δεν ανταποκρίνονται σε αυτά που θα περίμενε κανείς παρατηρώντας τα πολύγωνα που δίνονταν στα παραδείγματα. Αυτό, σε συνδυασμό με το γεγονός ότι οι δύο αλγόριθμοι κάνουν αισθητές τις διαφορές τους κυρίως σε μικρές γωνίες, μας οδηγεί στο συμπέρασμα ότι τελικά δεν έχει τόση σημασία ποιον από τους δύο αλγορίθμους θα επιλέξει κανείς αλλά ποια γωνία.

Όσον αφορά τα κριτήρια αποδοχής, παρατηρείται το εξής: Έστω ότι η καμπύλη παίρνει τη μορφή A με χρήση του κριτηρίου του τριγώνου και γωνία χρήστη από a έως b. Τότε, είναι πιθανόν να υπάρχει ένα διάστημα γωνίας $[c, d]$ στο οποίο, με κριτήριο του κώνου, η καμπύλη να παίρνει και πάλι τη μορφή A. Από τα παραδείγματα 3 και 4 διαπιστώνουμε ότι αυτή η πιθανότητα δε μοιάζει να ελαττώνεται καθώς μειώνουμε τη γωνία του χρήστη.

Είναι ευνόητο ότι όταν κοιτάζει κανείς κάποια πολύγωνα και προσπαθήσει να φανταστεί πώς πρέπει να είναι η περιβάλλουσά τους καμπύλη, σίγουρα δε θα έχει στο μυαλό του όλα τα δυνατά αποτελέσματα που θα του δώσει το πρόγραμμά μας. Για την ακρίβεια, όσο μεγαλύτερο το εύρος γωνίας από το οποίο προκύπτει μία περιβάλλουσα καμπύλη, τόσο πιθανότερο είναι να βρίσκεται κοντά στο αναμενόμενο αποτέλεσμα[1]. Αυτή η διαπίστωση μπορεί να οδηγήσει και σε έναν αυτόματο τρόπο επιλογής γωνίας χρήστη. Να σημειώνονται τα διαστήματα τα οποία μας δίνουν διαφορετικά αποτελέσματα καμπύλης, και από αυτά να επιλέγεται μία γωνία η οποία θα ανήκει στο μεγαλύτερο σε εύρος διάστημα. Όμως αν για παράδειγμα τα πολύγωνα βρίσκονται αρκετά κοντά μεταξύ τους, τότε ίσως το μεγαλύτερο διάστημα γωνιών να είναι αυτό που δίνει ως αποτέλεσμα την κυρτή θήκη. Συνεπώς ο χρήστης σε αυτήν την περίπτωση θα προτιμούσε το επόμενο σε μέγεθος διάστημα. Για αυτό ίσως να ήταν προτιμότερος ένας ημιαυτόνομος τρόπος επιλογής, στον οποίο θα προτείνονται στο χρήστη οι διάφορες γωνίες με σειρά προτεραιότητας (με πρώτη αυτή που ανήκει στο μεγαλύτερο διάστημα), δείχνοντάς του δίπλα σε καθεμία το σχήμα το οποίο προκύπτει ως αποτέλεσμα, ώστε να μπορεί να επιλέξει την επιθυμητή.

Με αφορμή αυτήν την πιθανή βελτίωση του αλγορίθμου, αξίζει να συγκεντρωθούν σε αυτό το σημείο όλες οι βελτιώσεις που αναφέραμε μέσα στην εργασία, με στόχο την υλοποίησή τους σε μελλοντική δουλειά.

[1] Αναμενόμενο είναι το αποτέλεσμα το οποίο βρίσκεται πιο κοντά στη γεωμετρία της εισόδου μας.

1. Υλοποίηση της τρίτης μεθόδου κατασκευής της περιβάλλουσας καμπύλης, με τη βοήθεια της ουράς προτεραιότητας. Η προσέγγιση αυτή είναι ιδιαίτερα σημαντική προς την κατεύθυνση του αυτόματου υπολογισμού μιας «καλής» γωνίας, όπως αυτή περιγράφηκε παραπάνω, καθώς το εύρος (σε γωνία) ισχύος της κάθε περιβάλλουσας καμπύλης μπορεί να υπολογιστεί άμεσα από την ακολουθία ακμών που μας δίνει η ουρά προτεραιότητας.

2. Υλοποίηση του κριτηρίου του κώνου με ορατότητα προς την κυρτή θήκη.

3. Ένταξη της δυνατότητας να επιλέγεται από ποια ακμή θα ξεκινάει η κατασκευή της καμπύλης στη μέθοδο τύπου DFS. Απώτερος σκοπός θα ήταν ο αυτόματος τρόπος υπολογισμού της ακμής εκκίνησης. Μία επιλογή θα ήταν να χρησιμοποιούμε ως ακμή εκκίνησης του αλγορίθμου DFS την ακμή που αντιστοιχεί στη μεγαλύτερη γωνία (βάσει του κριτηρίου που χρησιμοποιούμε).

4. Αυτοματοποίηση (καθολική ή μερική) της επιλογής του ορίου γωνίας και κατά συνέπεια αυτόματος ή ημιαυτόματος υπολογισμός της περιβάλλουσας καμπύλης.

5. Εύρεση της γωνίας του κώνου με τη βοήθεια της ιεραρχίας Dobkin-Kirkpatrick, ή με κάποια άλλη μέθοδο, ώστε να μειωθεί η χρονική πολυπλοκότητα του αλγορίθμου εύρεσης της περιβάλλουσας καμπύλης, όταν γίνεται χρήση του κριτηρίου του κώνου, από $O(n^2)$ σε $O(n \log n)$.

6. Στην παρούσα υλοποίηση ο έλεγχος των γωνιών κάποιου κριτηρίου έναντι της γωνίας χρήστη γίνεται με άμεσο υπολογισμό των γωνιών που αφορούν το κάθε κριτήριο. Η υλοποίηση αυτή καθιστά αναγκαία τη χρήση υπερβατικών συναρτήσεων γεγονός που μπορεί να μας οδηγήσει σε λανθασμένα αποτελέσματα λόγω αριθμητικών λαθών. Προκειμένου να αποφύγουμε τα αριθμητικά αυτά λάθη είναι απαραίτητο να γίνεται η σύγκριση δύο γωνιών με ρητές πράξεις, ώστε να είναι δυνατός ο *ακριβής* υπολογισμός της περιβάλλουσας καμπύλης με αριθμητικούς τύπους που υποστηρίζουν με ακρίβεια ρητές πράξεις, και συνεπώς να μην εξαρτάται το τελικό αποτέλεσμα από τη σειρά των πράξεων και την αριθμητική ακρίβεια που χρησιμοποιείται. Ο τρόπος που μπορεί να γίνει η σύγκριση των γωνιών που χρησιμοποιεί ο αλγόριθμός μας με ρητές πράξεις περιγράφεται στο Παράρτημα Β′.

7. Κατασκευή της περιβάλλουσας καμπύλης χωρίς αυτοτομές στην κλειστότητά της. Θα θέλαμε δηλαδή να αποφεύγονται οι περιπτώσεις στις οποίες ακμές της περιβάλλουσας καμπύλης εφάπτονται μεταξύ τους ή κορυφές της περιβάλλουσας καμπύλης εμφανίζονται δύο φορές σε αυτήν.

8. Κατασκευή λείας (δηλαδή τουλάχιστον G^1-συνεχούς) αναπαράστασης της περιβάλλουσας καμπύλης, καθώς στην παρούσα εργασία επικεντρωθήκαμε στον συνδυαστικό υπολογισμό της περιβάλλουσας καμπύλης (υπολογίζουμε ποια αντικείμενα ακουμπάει) και δίνουμε μια πολυγωνική, C^0-συνεχή, αναπαράστασή της.

Βιβλιογραφία

[1] A. Aggarwal, L.J. Guibas, J.B. Saxe, and P.W. Shor. A linear-time algorithm for computing the Voronoi diagram of a convex polygon. *Discrete & Computational Geometry*, 4:591–604, 1989.

[2] O. Aichholzer, F. Aurenhammer, and B. Palop. Quickest paths, straight skeletons, and the city Voronoi diagram. In *Proceedinds of the 18th Annual ACM Symposium on Computational Geometry*, pages 151–159. ACM Press, 2002.

[3] F. Aurenhammer. Power diagrams: Properties, algorithms and applications. *SIAM Journal on Computing*, 16(1):78–96, 1987.

[4] F. Aurenhammer and R. Klein. *Voronoi diagrams*, chapter 5, pages 201–290. Handbook of Computational Geometry. North-Holland, Amsterdam, 2000.

[5] F. Aurenhammer and O. Schwarzkopf. A simple on-line randomized incremental algorithm for computing higher order Voronoi diagrams. *Symposium on Computational Geometry*, pages 142–151, 1991.

[6] C. Burnikel. *Exact Computation of Voronoi Diagrams and Line Segment Intersections*. PhD thesis, Universität des Saarlandes, March 1996.

[7] CGAL, Computational Geometry Algorithms Library. `http://www.cgal.org`.

[8] D.R. Chand and S.S. Kapur. An algorithm for convex polytopes. *Journal of the ACM*, 17(1):78–86, 1970.

[9] L.P. Chew. Constrained Delaunay triangulations. *Algorithmica*, 4(1):97–108, 1989.

[10] F.Y.L. Chin, , and C.A. Wang. Finding the constrained Delaunay triangulation and constrained Voronoi diagram of a simple polygon in linear time. *SIAM Journal on Computing*, 28(2):471–486, 1998.

[11] K.L. Clarkson. Applications of Random Sampling in Computational Geometry, II. In *Proceedinds of the 4th Annual ACM Symposium on Computational Geometry*, pages 1–11, 1988.

[12] K.L. Clarkson and P.W. Shor. Applications of Random Sampling in Computational Geometry, II. *Discrete & Computational Geometry*, 4(5):387–421, 1989.

[13] Computed tomography. `http://en.wikipedia.org/wiki/Computed_tomography`.

[14] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 2nd edition, 2000. ISBN 3-540-65620-0.

[15] O. Devillers. Improved incremental randomized Delaunay triangulation. In *Proceedings of the 14th Annual ACM Symposium on Computational Geometry*, pages 106–115. ACM Press, 1998.

[16] O. Devillers. The Delaunay hierarchy. *International Journal of Foundations of Computer Science*, 13(2):163–180, 2002.

[17] L. Devroye, C. Lemaire, and J.M. Moreau. Expected time analysis for Delaunay point location. *Computational Geometry: Theory and Applications*, 29(2):61–89, 2004.

[18] L. Devroye, E. Peter Mücke, and B. Zhu. A note on point location in Delaunay triangulations of random points. *Algorithmica*, 22:477–482, 1998.

[19] D.P. Dobkin and D.G. Kirkpatrick. Determining the separation of preprocessed polyhedra - a unified approach. In *Proceedings of the International Colloquium, ICALP*, pages 400–413, 1990.

[20] Delaunay triangulation. `http://en.wikipedia.org/wiki/Delaunay_triangulation`.

[21] R.A. Dwyer. Higher-dimensional Voronoi diagrams in linear expected time. *Discrete & Computational Geometry*, 6:343–367, 1991.

[22] H. Edelsbrunner and R. Seidel. Voronoi diagrams and arrangements. *Discrete & Computational Geometry*, 1:25–44, 1986.

[23] A.B. Ekoule, F.C. Peyrin, and C.L. Odet. A triangulation algorithm from arbitrary shaped multiple planar contours. *ACM Transactions on Graphics*, 10(2):182–199, 1991.

[24] S. Fortune. A sweepline algorithm for Voronoi diagrams. *Algorithmica*, 2:153–174, 1987.

[25] N. C. Gabrielides, A. I. Ginnis, P.D. Kaklis, and M. I. Karavelas. G^1-smooth branching surface construction from cross sections. *Computer-Aided Design*, 39(8):639–651, August 2007.

[26] C.M. Gold, P.M. Remmele, and T. Roos. Voronoi diagrams of line segments made easy. In *Proceedings of the 7th Canadian Conference on Computational Geometry*, pages 223–228, 1995.

[27] P.J. Green and R. Sibson. Computing Dirichlet tessellations in the plane. *The Computer Journal*, 21(2):168–173, 1978.

[28] L.J. Guibas and J. Stolfi. Primitives for the manipulation of general subdivisions and computation of Voronoi diagrams. *ACM Transactions on Graphics*, 4(2):74–123, 1985.

[29] M. I. Karavelas and M. Yvinec. Dynamic additively weighted Voronoi diagrams in 2D. In *Proceedings of the 10th European Symposium on Algorithms (ESA)*, pages 586–598, 2002.

[30] M. I. Karavelas and M. Yvinec. 2D Apollonius graphs (Delaunay graphs of disks). In CGAL Editorial Board, editor, *CGAL User and Reference Manual*. 3.4 edition, 2008.

[31] M.I. Karavelas. A robust and efficient implementation for the segment Voronoi diagram. In *Proceedings of the International Symposium on Voronoi Diagrams in Science and Engineering*, pages 51–62, 2004.

[32] R. Klein, K. Mehlhorn, and S. Meiser. Randomized incremental construction of abstract Voronoi diagrams. *Computational Geometry: Theory and Applications*, 3:157–184, 1993.

[33] D.T Lee. Two-dimensional Voronoi diagrams in the L_p-metric. *Journal of the ACM*, 27(4):604–618, 1980.

[34] M. McAllister, D. Kirkpatrick, and J. Snoeyink. A compact piecewise-linear Voronoi diagram for convex sites in the plane. *Discrete & Computational Geometry*, 15(1):73–105, 1996.

[35] D. Meyers, S. Skinner, and K. Sloan. Surfaces from contours. *ACM Transactions on Graphics*, 11(3):228–258, 1992.

[36] K. Mulmuley. *Randomized algorithms in computational geometry*, chapter 16, pages 703–724. Handbook of Computational Geometry. North-Holland, Amsterdam, 2000.

[37] A. Odgaard and B. K. Nielsen. Applet υπολογισμού του διαγράμματος Voronoi βάσει του αλγορίθμου sweepline του Fortune. `http://www.diku.dk/hjemmesider/studerende/duff/Fortune/`.

[38] Klein R. *Concrete and Abstract Voronoi Diagrams*. Springer-Verlag, 1989.

[39] R. Seidel. Constrained Delaunay triangulations and Voronoi diagrams with obstacles (extended abstract). In *Rep. 260*, pages 178–191. IIG TU, Graz, 1988.

[40] M.I. Shamos and D. Hoey. Closest-point problems. In *Proceedings of the 16th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 151–162, 1975.

[41] M. Sharir. Intersection and closest-pair problems for a set of planar discs. *SIAM Journal on Computing*, 14(2):448–468, 1985.

[42] Y. Shinagawa and T.L. Kunii. The homotopy model: a generalized model for smooth surface generation from cross sectional data. *The Visual Computer*, 7(2):72–86, 1991.

[43] K. Sugihara and M. Iri. Construction of the Voronoi diagram for one million generators in single-precision arithmetic. *Proceedings of the IEEE*, 80(9):1471–1484, September 1992.

[44] Ν. Γαβριηλίδης. *Σχεδίαση Επιφανειών που Παρεμβάλλουν Παράλληλες - Μη Συνεκτικές Τομές Αντικειμένων*. PhD thesis, Εθνικό Μετσόβιο Πολυτεχνείο, Αθήνα, 2005.

[45] `http://www.prairienet.org/~bbrown/schools/gschools.html`.

[46] E. Welzl, P. Su, and R.L.S. Drysdale III. A comparison of sequential Delaunay triangulation algorithms. *Computational Geometry: Theory and Applications*, 7:361–385, 1997.

[47] M. Yvinec. 2D Triangulations. In CGAL Editorial Board, editor, *CGAL User and Reference Manual*. 3.4 edition, 2008.

Κεφάλαιο Α΄

Επιλεγμένα κομμάτια πηγαίου κώδικα

Το πρόγραμμα που αναπτύξαμε έχει γραφτεί σε C++, και κάνει χρήση της βιβλιοθήκης CGAL [7]. Στο παράρτημα αυτό παραθέτουμε το public interface των κλάσεων που έχουμε υλοποιήσει ως μία πρώτη επαφή του ενδιαφερόμενου χρήστη με τον κώδικά μας.

Έχουν υλοποιηθεί συνολικά 15 κλάσεις:

- Στην κλάση *Delaunay_triangulation_for_polygons_2* είναι υλοποιημένη η τριγωνοποίηση Delaunay για πολύγωνα, η οποία είναι υλοποιημένη με τη βοήθεια της τριγωνοποίηση σημείων, όπως περιγράψαμε στην αρχή του κεφαλαίου 3.

- Η κλάση *Edge_acceptor_base* είναι πρότυπη κλάση και μητρική των κλάσεων *Triangle_edge_acceptor_for_polygons* και *Cone_edge_acceptor_for_polygons* στις οποίες υλοποιούνται τα κριτήρια του τριγώνου και του κώνου αντίστοιχα.

- Στην κλάση *Convex_hull_edges* υπολογίζονται οι ακμές της κυρτής θήκης.

- Στην *Angle_cosine_2* υλοποιείται ο υπολογισμών των συνημιτόνων γωνιών, και χρησιμοποιείται από τις προαναφερθέντες κλάσεις των κριτηρίων. Στο παράρτημα περιγράφεται πώς μπορεί να αλλάξει αυτή η συνάρτηση, και στη θέση της να μπει μία η οποία να κάνει σύγκριση δύο γωνιών με ρητές πράξεις, ώστε να μην εξαρτάται το τελικό αποτέλεσμα, από την αριθμητική ακρίβεια που χρησιμοποιείται.

- Στο αρχείο *Search_policy_tags.h* υπάρχουν τρεις βοηθητικές δομές (ένα απαριθμητικές τύπος και δύο κενές δομές που χρησιμποιούνται ως tags), χάρις στις οποίες μπορεί να επιλέγεται στην *Surrounding_curve* ποιος από τους δύο αλγορίθμους (τύπου BFS ή DFS) θα χρησιμοποιηθεί. Αυτό γίνεται καλώντας την αντίστοιχη συνάρτηση *Surrounding_curve_BFS* ή *Surrounding_curve_DFS* οι οποίες είναι θυγατρικές της πρότυπης κλάσης *Surrounding_curve_base*.

- Οι υπόλοιπες κλάσεις: *Get_vertex*, και *Symmetric_edge* περιέχουν η κάθε μία από μία συνάρτηση-operator η οποία καλείται από διαφορετικά σημεία του προγράμματος, και γι΄ αυτό ήταν αναγκαίο να υπάρχει σε ξεχωριστή κλάση.

Α΄.1 Αρχείο Delaunay_triangulation_for_polygons_2.h

```
template<class DT2_, class Polygon_>
class Delaunay_triangulation_for_polygons_2 : public DT2_
{
  typedef DT2_        Base;

public:
  typedef DT2_                                    Delaunay_triangulation_2;
  typedef Polygon_                                Polygon_2;

  typedef typename Base::Vertex_handle            Vertex_handle;
  typedef typename Base::Face_handle              Face_handle;
  typedef typename Base::Edge                     Edge;
  typedef typename Base::Geom_traits              Geom_traits;

  typedef std::map<Vertex_handle,int>             Vertex_map;

  typedef typename Geom_traits::Point_2           Point_2;

protected:
  typedef Delaunay_triangulation_for_polygons_2<Base,Polygon_2> Self;

  typedef std::vector<Polygon_2>                Polygon_container;

  // Rest of "protected" code is omitted.

private:
  // Private data, types and methods. Code is omitted.

public:
  typedef typename Polygon_container::const_iterator  Polygons_iterator;

  // Constructor
  Delaunay_triangulation_for_polygons_2();

  // Destructor
  ~Delaunay_triangulation_for_polygons_2();
```

```
// Clears the triangulation contents
void clear();

// Copy constructor
Delaunay_triangulation_for_polygons_2(const Self& other);

// Assignment operator
Self& operator=(const Self& other);

// Returns the vertex map.
const Vertex_map& vertex_map() const;
Vertex_map& vertex_map();

// Inserts the polygon vertices into the Delaunay triangulation.
// All points of the polygon are associated with the same unique
// id the vertex map.
// Returns a Vertex_handle of one of the vertices of the input polygon.
Vertex_handle insert(const Polygon_2& poly);

// Inserts the polygons in the input iterator range [first, beyond)
// in the Delaunay triangulation.
// Returns the number of polygons inserted.
template<typename InputIterator>
int insert(InputIterator first, InputIterator beyond)
{
  // Code is omitted.
}

// Checks whether e is an edge of a input polygon
bool is_polygon_edge(Edge e) const;

// Starts from edge e and in the direction of the face of e, and
// returns an edge the adjacent face of which corresponds to 3
// distinct ids.
//
// It is possible that the adjacent face is an infinite face in
// which case the search has in fact failed, provided of course that
// we search for a finite face (this case can happen if we start
// from an edge inside a tunnel formed by two polygons.
```

```
  //
  // As we move from e to the triangle t with 3 distince ids, we form
  // a tunnel of triangles. The boundary edges (edge chains) of this
  // tunnel are stored in the output iterators left_edges_oit and
  // right_edges_oit; we assume here that the original triangle lies
  // to the north, the sought-for triangle south, thus the left chain
  // lies to the west and the right chain to the east. The edges are
  // stored so that the triangle of the edge is a triangle in the tunnel.
  template<typename OutputIterator>
  Edge find_face_with_3ids(Edge e, OutputIterator left_edges_oit,
                           OutputIterator right_edges_oit) const
  {
    // Code is omitted.
  }

  // Returns the number of polygons in the Delaunay triangulation
  unsigned int number_of_polygons() const;

  // Iterator to an arbitrary input polygon
  Polygons_iterator polygons_begin() const;

  // Past-the-end iterator
  Polygons_iterator polygons_end() const;

  // Returns the i-th polygon inserted
  const Polygon_2& operator[](unsigned int i) const;

  // Returns the i-th polygon inserted
  Polygon_2& operator[](unsigned int i);
};
```

Α΄.2 Αρχείο Get_vertex.h

```
// Returns the (i+e.second)-th vertex of the face adjacent to e.
template<class Vertex_handle, class Edge>
struct Get_vertex
{
  Vertex_handle operator()(const Edge& e, int i) const;
};
```

Α΄.3 Αρχείο Symmetric_edge.h

```
template<class DT2_>
class Symmetric_edge
{
  typedef DT2_  DT2;

protected:
  // Protected data, types and methods. Code is omitted.

public:
  typedef DT2_                          Delaunay_triangulation_2;
  typedef typename DT2_::Edge           Edge;
  typedef typename DT2_::Face_handle    Face_handle;

  // Constructor
  Symmetric_edge(const Delaunay_triangulation_2& dt);

  // Returns the symmetric edge of (f,i) in the triangulation
  // data structure of dt.
  Edge operator()(const Face_handle& f, int i) const;
};
```

Α΄.4 Αρχείο Convex_hull_edges.h

```
template<class DT_>
class Convex_hull_edges
{
public:
  typedef DT_                                  Delaunay_triangulation_2;

protected:
  // Protected data, types and methods. Code is omitted.

public:
  // Returns the edges of the convex hull.
  //
  // The orientation of the edges in the convex hull is such that the
  // points lie in the halfspace containing the face corresponding to
  // the edge.
  template<typename EdgeOutputIterator>
  EdgeOutputIterator operator()(const Delaunay_triangulation_2& dt,
                                EdgeOutputIterator eoit) const;
};
```

Α΄.5 Αρχείο Angle_cosine_2.h

```
template<class DTP2_>
class Angle_cosine_2
{
public:
  typedef DTP2_                         Delaunay_triangulation_for_polygons_2;
  typedef typename DTP2_::Point_2       Point_2;
  typedef typename DTP2_::Edge          Edge;

private:
  typedef typename CGAL::Kernel_traits<Point_2>::Kernel      Kernel;

public:
  typedef typename Kernel::FT           FT;

  // Returns the cosine of the angle p2p1p3
  FT operator()(const Point_2& p1, const Point_2& p2,
                const Point_2& p3) const;

  // Returns the cosine of the angle opposite to the edge e.
  FT operator()(const Edge& e) const;
};
```

Α΄.6 Αρχείο Edge_acceptor_base.h

```
template<class DT2_>
class Edge_acceptor_base
{
public:
  typedef DT2_                                Delaunay_triangulation_2;
  typedef typename DT2_::Edge                 Edge;
  typedef typename DT2_::Geom_traits::FT      FT;

protected:
  // Protected data, types and methods. Code is omitted.

public:
  // Constructor
  Edge_acceptor_base(const FT& threshold);

  // Set the threshold angle
  void set_threshold(const FT& threshold);

  // Get the threshold angle
  const FT& threshold() const;

  // Get the cosine of the threshold angle
  FT threshold_cosine() const;
};
```

Α΄.7 Αρχείο Triangle_edge_acceptor_for_polygons.h

```
template<class DTP2_>
class Triangle_edge_acceptor_for_polygons
  : public Edge_acceptor_base<DTP2_>
{
public:
  typedef DTP2_                              Delaunay_triangulation_for_polygons_2;
  typedef typename DTP2_::Edge               Edge;
  typedef typename DTP2_::Polygon_2::FT      FT;

private:
  // Private data, types and methods. Code is omitted.

public:
  // Constructor
  Triangle_edge_acceptor_for_polygons(const DTP2_& dtp2,
                                      const FT& deg_threshold = FT(90));

  // Returns true, if the angle opposite to e in the triangle adjacent
  // to e is greater than the user's threshold, false otherwise.
  bool operator()(const Edge& e) const;
};
```

Α'.8 Αρχείο Cone_edge_acceptor_for_polygons.h

```
template<class DTP2_>
class Cone_edge_acceptor_for_polygons
  : public Edge_acceptor_base<DTP2_>
{
public:
  typedef DTP2_                              Delaunay_triangulation_for_polygons_2;
  typedef typename DTP2_::Edge               Edge;
  typedef typename DTP2_::Polygon_2::FT      FT;

private:
  // Private data, types and methods. Code is omitted.

protected:
  // Protected data, types and methods. Code is omitted.

public:
  // Constructor
  Cone_edge_acceptor_for_polygons(const DTP2_& dtp2,
                                  const FT& deg_threshold = FT(90));

  // Returns true, if the angle of cone with apex the vertex opposite
  // to e is greater than the user's threshold, false otherwise.
  bool operator()( const Edge e ) const;
};
```

Α΄.9 Αρχείο Search_policy_tags.h

```
// Enumeration type for the two search policies.
typedef enum {
  DFS = 1,
  BFS = 2
} Search_policy;

// Tag for the DFS search policy
struct DFS_search_policy_tag {};

// Tag for the BFS search policy
struct BFS_search_policy_tag {};
```

Α′.10 Αρχείο Surrounding_curve_base.h

```
template<class Acceptor_, class DTP_>
class Surrounding_curve_base
{
public:
  typedef DTP_                          Delaunay_triangulation_for_polygons_2;
  typedef Acceptor_                     Acceptor;

protected:
  // Protected types and methods. Code is omitted.

public:
  // Default (and dummy) implementation for the operator()
  template<typename EdgeOutputIterator>
  EdgeOutputIterator
  operator()(const Delaunay_triangulation_for_polygons_2&,
              const Acceptor&, EdgeOutputIterator oit, int) const
  {
    return oit;
  }
};
```

Α΄.11 Αρχείο Surrounding_curve_BFS.h

```
template<class Acceptor_, class DTP_>
class Surrounding_curve_BFS
  : public Surrounding_curve_base<Acceptor_, DTP_>
{
public:
  typedef DTP_          Delaunay_triangulation_for_polygons_2;
  typedef Acceptor_     Acceptor;

protected:
  // Protected types and methods. Code is omitted.

public:
  // Computes the surrounding curve using the BFS policy.
  template<typename EdgeOutputIterator>
  EdgeOutputIterator
  operator()(const Delaunay_triangulation_for_polygons_2& dtp,
             const Acceptor& accept, EdgeOutputIterator eoit,
             int depth) const
  {
    // Code is omitted.
  }
};
```

Α΄.12 Αρχείο Surrounding_curve_DFS.h

```
template<class Acceptor_, class DTP_>
class Surrounding_curve_DFS
  : public Surrounding_curve_base<Acceptor_, DTP_>
{
public:
  typedef DTP_          Delaunay_triangulation_for_polygons_2;
  typedef Acceptor_     Acceptor;

protected:
  // Protected types and methods. Code is omitted.

public:
  // Computes the surrounding curve using the DFS policy.
  template<typename EdgeOutputIterator>
  EdgeOutputIterator
  operator()(const Delaunay_triangulation_for_polygons_2& dtp,
             const Acceptor& accept, EdgeOutputIterator eoit,
             int depth) const
  {
    // Code is omitted.
  }
};
```

Α′.13 Αρχείο Surrounding_curve.h

```
template<class DTP_>
class Surrounding_curve
{
public:
  typedef DTP_      Delaunay_triangulation_for_polygons_2;

protected:
  // Protected types and methods. Code is omitted.

public:
  // Computes the surrounding curve using the BFS policy.
  // This method simply calls operator() of the Surrounding_curve_BFS
  // class.
  template<typename EdgeOutputIterator, typename Acceptor>
  EdgeOutputIterator
  operator()(const Delaunay_triangulation_for_polygons_2& dtp,
             const Acceptor& accept, EdgeOutputIterator eoit,
             int depth, const BFS_search_policy_tag&) const
  {
    // Code is omitted.
  }

  // Computes the surrounding curve using the DFS policy.
  // This method simply calls operator() of the Surrounding_curve_DFS
  // class.
  template<typename EdgeOutputIterator, typename Acceptor>
  EdgeOutputIterator
  operator()(const Delaunay_triangulation_for_polygons_2& dtp,
             const Acceptor& accept, EdgeOutputIterator eoit,
             int depth, const DFS_search_policy_tag&) const
  {
    // Code is omitted.
  }
};
```

Κεφάλαιο Β΄

Σύγκριση γωνιών με ρητές πράξεις

Η σύγκριση των γωνιών[1] στον αλγόριθμο μπορεί να γίνει με τέτοιο τρόπο, ώστε να χρησιμοποιούνται μόνο ρητές πράξεις. Με τον τρόπο αυτό είναι δυνατόν να αποφύγουμε τη χρήση υπερβατικών συναρτήσεων, οι οποίες, λόγω αριθμητικών λαθών, μπορεί να μας οδηγήσουν σε γεωμετρικά λανθασμένα αποτελέσματα. Αντίθετα, οι ρητές εκφράσεις είναι δυνατόν (με τη βοήθεια κατάλληλων αριθμητικών τύπων) να υπολογιστούν με *άπειρη ακρίβεια*, με αποτέλεσμα ο υπολογισμός της περιβάλλουσας καμπύλης να γίνει γεωμετρικά ακριβώς, χωρίς να εξαρτάται το αποτέλεσμα από τη σειρά των πράξεων και την αριθμητική ακρίβεια που χρησιμοποιείται. Πριν αναλύσουμε πώς θα γίνει αυτό, υπενθυμίζουμε ότι οι γωνίες που συγκρίνουμε είναι πάντα μικρότερες ή ίσες των 180^o, αφού ασχολούμαστε μόνο με κυρτά πολύγωνα.

Στον αλγόριθμό μας μπορεί να εμφανιστεί σύγκριση γωνιών που να περιλαμβάνει δύο τύπους ορισμάτων εισόδου:

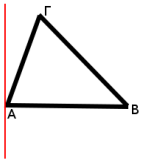
1. 3 σημεία που καθορίζουν τη μία γωνία και 3 σημεία που καθορίζουν τη δεύτερη.

2. 3 σημεία που καθορίζουν τη μία γωνία, έναν αριθμό που αντιπροσωπεύει το τετράγωνο του συνημιτόνου της δεύτερης γωνίας και ένα χαρακτήρα που αντιπροσωπεύει το πρόσημο του συνημιτόνου της.

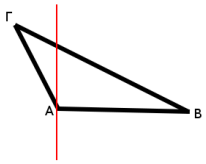
Και στις δύο περιπτώσεις πρέπει να ελέγξουμε αν η πρώτη γωνία είναι οξεία ή αμβλεία και στη συνέχεια να βρούμε το τετράγωνο του συνημιτόνου της.

Έστω ότι το τρίγωνο είναι το $AB\Gamma$ και έστω ότι θέλουμε να βρούμε το τετράγωνο του συνημιτόνου της γωνίας $B\hat{A}\Gamma$. Θεωρούμε την ευθεία που περνάει από το σημείο A και είναι κάθετη στην AB (αν διαλέγαμε να είναι κάθετη στην $A\Gamma$ δε θα άλλαζε τίποτα) και ελέγχουμε αν το σημείο Γ είναι στο ίδιο ημιεπίπεδο που ορίζει η ευθεία με αυτό που βρίσκεται το σημείο B. Αν ναι, τότε η γωνία είναι οξεία, αλλιώς είναι αμβλεία.

[1] Γίνεται στο αρχείο Angle_cosine_2.h.



(α΄) Οξεία γωνία.



(β΄) Αμβλεία γωνία.

Σχήμα Β΄.1: Εύρεση αν η γωνία $B\hat{A}\Gamma$ είναι αμβλεία ή οξεία.

Αν βρισκόμαστε στην περίπτωση (i) τότε κάνουμε τους ίδιους υπολογισμούς και για τη δεύτερη γωνία, ώστε να καθορίσουμε αν είναι αμβλεία ή οξεία. Αντίστοιχα στην περίπτωση (ii) από το πρόσημο του συνημιτόνου που παίρνουμε ως είσοδο μπορούμε και πάλι να συμπεράνουμε αν η δεύτερη γωνία είναι αμβλεία ή οξεία. Συνεπώς και στις δύο περιπτώσεις, αν η μία είναι αμβλεία και η άλλη οξεία, μπορούμε αμέσως να επιστρέψουμε την απάντηση.

Αν είναι και οι δύο γωνίες οξείες, ή και οι δύο αμβλείες, τότε θα αρκεί να υπολογίσουμε και να συγκρίνουμε τα τετράγωνα των συνημιτόνων τους, τα οποία μπορούν να υπολογιστούν ως ρητές εκφράσεις των καρτεσιανών συντεταγμένων των σημείων που ορίζουν τα τρίγωνα.