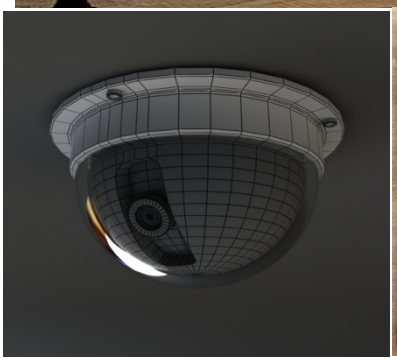
ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΜΑΚΕΔΟΝΙΑΣ
ΟΙΚΟΝΟΜΙΚΩΝ ΚΑΙ ΚΟΙΝΩΝΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΠΑΝΕΠΙΣΤΗΜΙΟ ΜΑΚΕΔΟΝΙΑΣ
ΜΕΤΑΠΤΥΧΙΑΚΟ ΤΜΗΜΑ
ΕΦΑΡΜΟΣΜΕΝΗΣ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΘΕΜΑ ΔΙΠΛΩΜΑΤΙΚΗΣ ΕΡΓΑΣΙΑΣ:

ΑΝΑΛΥΣΗ, ΕΦΑΡΜΟΓΗ ΚΑΙ ΣΥΓΚΡΙΣΗ ΑΛΓΟΡΙΘΜΩΝ ΓΙΑ ΤΗΝ ΦΥΛΑΞΗ ΓΚΑΛΕΡΙ ΤΕΧΝΩΝ ΜΕ ΧΡΗΣΗ ΜΕΘΟΔΟΥ ΤΡΙΓΩΝΟΠΟΙΗΣΗΣ ΠΟΛΥΓΩΝΩΝ ΣΤΟ ΕΠΙΠΕΔΟ

(**ART GALLERY PROBLEM**)



ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ :
κ. ΠΑΠΑΡΡΙΖΟΣ ΚΩΝΣΤΑΝΤΙΝΟΣ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ ΤΟΥ : ΦΙΛΙΟΥ ΝΙΚΟΛΑΟΥ
Α.Μ. : 19/07

ΜΑΪΟΣ 2010

Περιεχόμενα

Εισαγωγή

Τα έργα διάσημων ζωγράφων είναι πολύ δημοφιλή στους κύκλους λάτρεων της τέχνης , καλιτεχνών αλλά είναι επίσης πολύ δημοφιλή και σε εγληματίες. Τα έργα αυτά ειναι πολύτιμα, εύκολα στην μεταφορά και όπως φαίνεται όχι πολύ δύσκολο στο να πουληθούν. Οι διάφορες Γκαλερί Τεχνών θα πρέπει να φυλάττουν τα έργα τα οποία εκθέτουν με μεγάλη προσοχή για τους λόγους που αναφέραμε παραπάνω.

Κατά την διάρκεια της μέρας οι υπεύθυνοι ασφαλείας της Γκαλερί καθώς επίσης και οι ξεναγοί – συνοδοί της μπορούν να προστατεύουν και να φυλάττουν τα έργα τέχνης που εκθέτουν. Κατά την διάρκεια όμως της νύχτας αυτό μπορεί να γίνει με την χρήση καμερών.

Οι κάμερες αυτές είναι συνήθως τοποθετημένες στην οροφή αιθουσών και μπορούν να περιστρέφονται γύρω από ένα κάθετο άξονα. Οι εικόνες των καμερών στέλνονται σε οθόνες τηλεόρασης στο γραφείο του νυχτοφύλακα ο οποίος και είναι υπεύθυνος για την παρακολούθησή τους. Επειδή είναι πιο εύκολο να παρακολουθούνται λίγες οθόνες τηλεόρασης από ότι πολλές, ο αριθμός των καμερών που χρησιμοποιούνται για την φύλαξη της Γκαλερί θα πρέπει να έιναι ο μικρότερος δυνατός. Δύο ακόμα σημαντικοί λόγοι για την ύπαρξη όσο το δυνατόν λιγότερων καμέρων είναι καταρχήν ότι το συνολικό κόστος του συστήματος ασφαλείας θα είναι χαμηλότερο. Επίσης ας μην ξεχνάμε ότι ο χώρος τον οποίο φυλάσσουν οι κάμερες είναι χώρος έκθεσης έργων τέχνης και οι επισκέπτες, ή ακόμα και πελάτες, δεν πρέπει να αποσπώνται από την ύπαρξη πολλών καμερών, αντίθετως ο αριθμός τους καθώς και οι ίδιες θα πρέπει να είναι πολύ διακριτικές. Από την άλλη ο αριθμός τους θα πρέπει να επαρκεί για την κάλυψη ολόκληρης της Γκαλερί δηλαδή κάθε σημείο της θα πρέπει να είναι ορατό από τουλάχιστον μια κάμερα για λόγους ασφαλείας.

Σύμφωνα με όσα αναφέρθηκαν παραπάνω οι κάμερες θα πρέπει να τοποθετηθούν σε στρατηγικά σημεία ώστε κάθε μια από αυτές να καλύπτει ένα μεγάλο μέρος του χώρου. Γεννούνται λοιπόν ορισμένα ερωτήματα :

i) Που πρέπει να τοποθετούν οι κάμερες σε δεδομένο χώρο ώστε κάθε σημείο αυτού να είναι ορατό από τουλάχιστον μια κάμερα ;

ii) Ποιός είναι ο ελάχιστος αριθμός καμερών που χρειάζονται για την ασφάλεια του χώρου της Γκαλερί Τεχνών ;

Τα παραπάνω ερωτήματα αποτελούν το λεγόμενο **« Art Gallery Problem » (Πρόβλημα Γκαλερί Τεχνών).** Στα ερωτήματα αυτά θα δώσουμε απαντήσεις στα Κεφάλαια που ακολουθούν αναλύοντας διεξοδικά και τους αλγορίθμους υπολογιστικής γεωμετρίας που θα χρησιμοποιήσουμε.

Θα αναφερόμαστε, όπως έχει αντιληπτό και μέχρι τώρα, στην φύλαξη Γκαλερί Τεχνών. Ωστόσο τα ίδια ακριβώς ισχύουν και για την φύλαξη μουσείου αλλά και γενικότερα για την φύλαξη οποιουδήποτε χώρου.

Η υλοποίηση του αλγορίθμου, ο οποίος και θα αναλυθεί διεξοδικά μέσα από τα Κεφάλαια αυτής της διπλωματικής, θα γίνει με την χρήση του μαθηματικού προγραμματιστικού πακέτου **MATLAB**. Ως « είσοδος » θα δίνεται ένα πολύγωνο με γνωστές τις συντεταγμένες των κορυφών του. Ο τρόπος με τον οποίο ενώνονται οι κορυφές για τον σχηματισμό του πολυγώνου θα δίνεται, όπως και ο αλγόριθμος προστάζει, ανθορολογιακά (anticlockwise) . Η « έξοδος » του προγράμματος θα δίνει την τριγωνοποίηση του δεδομένου πολυγώνου όπως επίσης και την βέλτιστη δυνατή τοποθέτηση των καμερών σε συγκεκριμένες κορυφές του πολυγώνου για την πλήρη φύλαξή του.

Κεφάλαιο 1^{ο}

Φύλαξη και Τριγωνοποίηση

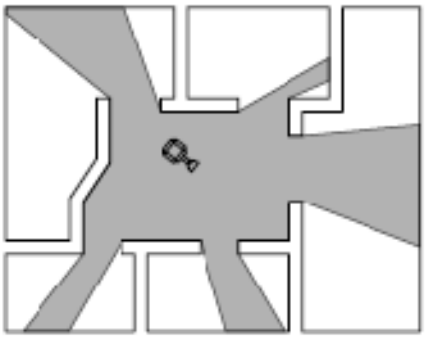
1.1 Εισαγωγικά στοιχεία για την φύλαξη πολυγώνου.

Για να ερμηνεύσουμε το « Πρόβλημα της Γκαλερί Τεχνών » (Art Gallery Problem) ακριβέστερα πρέπει αρχικά να τυποποιήσουμε την έννοια της Γκαλερί. Μια Γκαλερί είναι φυσικά τρισδιάστατη στον χώρο (3 – D) αλλά ένα σχέδιο του εδάφους (πατώματος) της μπορεί να μας δώσει τις απαραίτητες πληροφορίες για το που θα πρέπει να τοποθετηθούν οι κάμερες. Για τον λόγο αυτό μπορούμε να **μοντελοποιήσουμε την Γκαλερί σαν μια πολυγωνική περιοχή στο επίπεδο (2 –D).**

Περιοριζόμαστε στην συνέχεια σε σχήματα που είναι " Απλά Πολύγωνα " και περιοχές που περιβάλλονται από απλά κλειστά πολύγωνα που δεν τέμνονται μεταξύ τους. Επίσης προς το παρόν δεν θα ασχοληθούμε με πολύγωνα που περιέχουν οπές («τρύπες») στο εσωτερικό τους. Θα αναπτύξουμε τους αλγορίθμους της υπολογιστικής γεωμετρίας που θα χρησιμοποιήσουμε σε απλά πολύγωνα αρχικά και εφόσον γίνουν αυτοί πλήρως κατανοητοί θα αναφέρουμε και τον τρόπο όπου οι ίδιοι αλγόριθμοι μπορούν να επεκταθούν και σε πιο σύνθετες γεωμετρικές περιπτώσεις.

Η θέση της κάμερας στην Γκαλερί αντιστοιχεί σε ένα σημείο στο πολύγωνο. Η κάμερα έχει ορατά («βλέπει») εκείνα τα σημεία με τα οποία μπορεί να συνδεθεί με ένα ευθύγραμμο τμήμα στο εσωτερικό του πολυγώνου. Θεωρούμε οτι η οπτική θέα της κάθε κάμερας ειναι 360^{o} ή με άλλα λόγια ότι οι κάμερες μπορούν να περιστρέφονται γύρω από τον κατακόρυφο άξονά τους με άπειρη ταχύτητα. Ακόμη η κάθε κάμερα μπορεί να έχει οπτική ικανότητα σε άπειρη απόσταση ή μέχρι την απόσταση εκείνη όπου κάποιο εμπόδιο θα βρεθεί μπροστά της. Αγνοούμε το μέγεθος της κάμερας καθώς και την κατακόρυφη θέση της. Όλες οι παραπάνω παραδοχές είναι απαραίτητες για την μαθηματική, αλγοριθμική αλλά και προγραμματιστική προσέγγιση και επίλυση του προβλήματος χωρίς ωστόσο να επηρεάζουν τελικά το αποτέλεσμα, που δεν είναι άλλο από τον αριθμό των καμερών που απαιτούνται για την πλήρη κάλυψη του χώρου της Γκαλερί και των θέσεων όπου θα τοποθετηθούν αυτές. Ο τρόπος με τον οποίο λειτουργεί μια κάμερα τοποθετημένη σε ένα σημείο του

πολυγώνου φαίνεται στο σχήμα 1 που ακολουθεί. Το σκιαγραφημένο τμήμα είναι αυτό που καλύπτει η κάμερα.

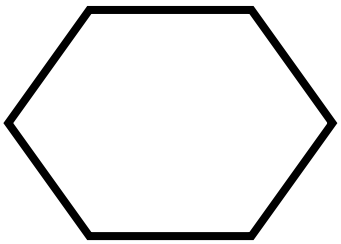


Σχήμα 1

Επαναδιατυπώνουμε λοιπόν το αρχικό μας ερώτημα ως εξής :

Πόσες κάμερες απαιτούνται για την φύλαξη ενός απλού πολυγώνου ;

Η απάντηση στο παραπάνω ερώτημα είναι ότι αυτό εξαρτάται αποκλειστικά από το πολύγωνο που έχουμε να διαχειριστούμε. Όσο πιο περίπλοκο είναι το πολύγωνο τόσο περισσότερες κάμερες θα απαιτούνται. Θα πρέπει επομένως να καθορίσουμε τον αριθμό των καμερών που χρειαζόμαστε ως συνάρτηση των κορυφών, έστω n, του πολυγώνου. Μπορεί ωστόσο δύο πολύγωνα να έχουν τον ίδιο αριθμό κορυφών αλλά κάποιο από αυτά να είναι πιο εύκολο στην φύλαξη του από το αλλό, δηλαδή να απαιτεί λιγότερες κάμερες για την πλήρη κάλυψη του. Για παράδειγμα ένα κυρτό πολύγωνο μπορεί να φυλαχθεί με μια μόνο κάμερα. Στο σχήμα 2 φαίνεται ένα κυρτό πολύγωνο η φύλαξη του οποίου μπορεί να γίνει με μια μόνο κάμερα, η οποία μπορεί να τοποθετηθεί σε οποιαδήποτε από τις κορυφές του.



Σχήμα 2

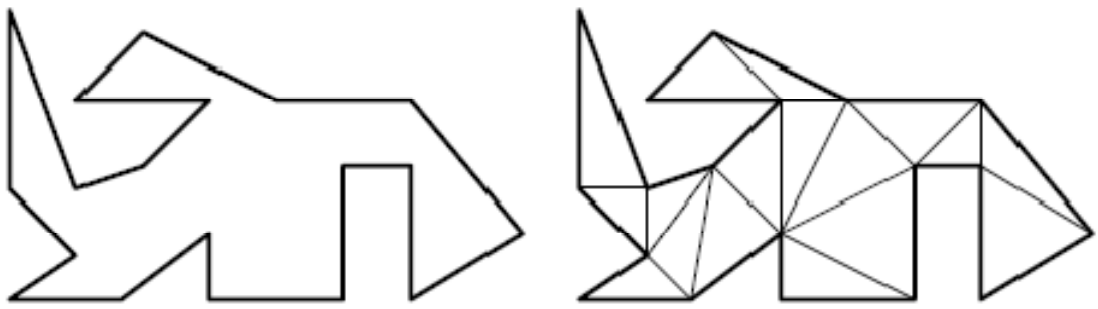
Θα πρέπει επομένως να βρούμε τον ελάχιστο δυνατό αριθμό καμερών που απαιτούνται για την πλήρη φύλαξη ενός οποιουδήποτε δεδομένου πολυγώνου όχι μόνο για τις απλές περιπτώσεις ενός κυρτού πολυγώνου αλλά λαμβάνοντας υπόψιν και την χειρότερη δυνατή περίπτωση (worst-case scenario).

1.2 Πολύγωνα και τριγωνοποίηση .

Στην συνέχεια αναφέρουμε ένα κανόνα ανάλυσης (αποσύνθεσης) ενός απλού πολυγώνου σε τρίγωνα . Ο κανόνας αυτός αποτελεί βασικό σημείο του αλγορίθμου, που θα αναφερθεί στη συνέχεια, για την φύλαξη και τριγωνοποίηση πολυγώνων.

Έστω ένα απλό πολύγωνο P με n κορυφές. Το P μπορεί να έχει οποιαδήποτε περίπλοκη μορφή που να είναι αρκετά δύσκολο αρχικά να πούμε οτιδήποτε για τον αριθμό, πόσο μάλλον για τη θέση τοποθέτησης, των καμερών. Οπότε από'δω και στο εξής αποσυνθέτουμε το P σε κομμάτια τα οποία είναι εύκολα στην φύλαξη, δηλαδή σε τρίγωνα. Το τρίγωνο αποτελεί το μικρότερο δυνατό « κομμάτι » στο οποίο μπορεί να υποδιαιρεθεί ένα πολύγωνο και μπορεί η φύλαξη του να γίνει με μια μόνο κάμερα τοποθετημένη σε μια από τις κορυφές του.

Την τριγωνοποίηση που αναφέραμε την πετυχαίνουμε σχεδιάζοντας διαγωνίους ανάμεσα σε ζευγάρια κορυφών του πολυγώνου P. Η διαγώνιος αυτή είναι ένα ευθύγραμμο τμήμα το οποίο ενώνει δύο κορυφές του P και βρίσκεται στο εσωτερικό του πολυγώνου. Επίσης οι διαγώνιοι ανάμεσα στις κορυφές του P δεν πρέπει να τέμνονται μεταξύ τους. Μια ανάλυση – αποσύνθεση ενός πολυγώνου σε τρίγωνα με μη-τεμνόμενες διαγωνίους μεταξύ των κορυφών του φαίνεται στο σχήμα 3 που ακολουθεί και ονομάζεται **τριγωνοποίηση του πολυγώνου**.



Σχήμα 3

Αριστερά φαίνεται το αρχικό πολύγωνο και δεξιά το τριγωνοποιημένο πλέον πολύγωνο. Στην τριγωνοποίηση αυτή απαιτούμε ο αριθμός των διαγωνίων ανάμεσα στις κορυφές του πολυγώνου να είναι ο μέγιστος δυνατός για να εξασφαλίσουμε ότι κανένα τρίγωνο δεν θα έχει κάποια κορυφή του πολυγώνου στο εσωτερικό των

πλευρών του (αυτό μπορεί να συμβεί αν το πολύγωνο έχει τρεις συνεχόμενες και συγγραμμικές κορυφές) . Οι τριγωνοποιήσεις συνήθως δεν είναι μοναδικές, για παράδειγμα το πολύγωνο στο σχήμα 3 θα μπορούσε να τριγωνοποιηθεί και με αλλούς τρόπους πέραν του τρόπου που φαίνεται στο σχήμα. Θα μπορούσαμε επομένως να τοποθετήσουμε μια κάμερα σε κάθε τρίγωνο του τριγωνοποιημένου πολυγώνου P για την φύλαξή του. Με την τριγωνοποίηση όμως του πολυγώνου δημιουργούνται δύο ερωτήματα :

i) Υπάρχει πάντα τριγωνοποίηση σε ένα πολύγωνο ;

ii) και πόσα τρίγωνα υπάρχουν με την τριγωνοποίηση;

Απάντηση στα παραπάνω ερωτήματα δίνεται με το εξής θεώρημα :

<u>Θεώρημα 1ο :</u> Κάθε απλό πολύγωνο μπορεί να τριγωνοποιηθεί και κάθε τριγωνοποίηση ενός πολυγώνου με n κορυφές αποτελείται από ακριβώς n-2 τρίγωνα.

<u>Απόδειξη:</u>

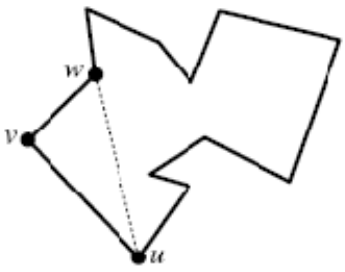
Αποδεικνύουμε το παραπάνω θεώρημα με επαγωγή του n.

Για n = 3 το πολύγωνο είναι τρίγωνο και το θεώρημα ισχύει.

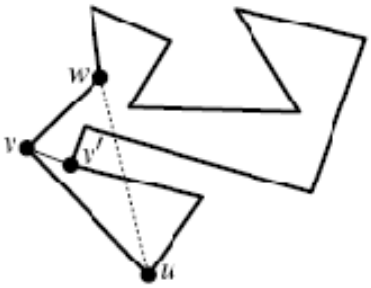
Έστω n > 3 και υποθέτουμε ότι το θεώρημα ισχύει για όλα τα m < n .

Έστω P ένα πολύγωνο με n κορυφές. Αρχικά θα αποδείξουμε την ύπαρξη μιας διαγωνίου στο P. Αν v η πιο αριστερή κορυφή του P και έστω u και w οι δύο γειτονικές κορυφές της v στην περιφέρεια (όριο) του P όπως φαίνονται στο σχήμα 4.

Αν το ευθύγραμμο τμήμα $\overline{uw}$ βρίσκεται στο εσωτερικό του P τότε έχουμε βρει μια διαγώνιο. Σε αντίθετη περίπτωση θα υπάρχουν μια ή και περισσότερες κορυφές μέσα στο τρίγωνο που σχηματίζεται από τις κορυφές v,w και u ή πάνω στο ευθύγραμμο τμήμα $\overline{uw}$. Έστω v΄ μια από τις κορυφές αυτές η πιο απομακρυσμένη από το $\overline{uw}$ όπως φαίνεται στο σχήμα 5.



Σχήμα 4



Σχήμα 5

Το ευθύγραμμο τμήμα $\overline{\nu\nu'}$ δεν μπορεί να τέμνει κάποια πλευρά του P, γιατί τότε θα έπρεπε ένα άκρο της πλευράς αυτής να βρίσκεται μέσα στο τρίγωνο που είναι πιο απομακρυσμένο από το $\overline{uw}$, που έρχεται σε αντίφαση με τον ορισμό του ν΄. Επομένως το $\overline{\nu\nu'}$ αποτελεί διαγώνιο. Άρα μια διαγώνιος υπάρχει. Κάθε διαγώνιος υποδιαιρεί το πολύγωνο P σε δύο άλλα υποπολύγωνα έστω P_1 και P_2 .Έστω m_1 ο αριθμός των κορυφών του P_1 και m_2 ο αριθμός κορυφών του P_2. Τότε τα m_1 και m_2 πρέπει να είναι μικρότερα από το n (m_1, m_2 < n). Επαγωγικά λοιπόν τα P_1 και P_2 μπορούν να τριγωνοποιηθούν. Συνεπώς και το P μπορεί επίσης να τριγωνοποιηθεί.

Στην συνέχεια απομένει να αποδείξουμε ότι κάθε τριγωνοποίηση αποτελείται από ακριβώς n-2 τρίγωνα. Θεωρούμε μια τριγωνοποίηση, έστω T_P, ενός πολυγώνου P και μια αυθαίρετη διαγώνιο του πολυγώνου. Τότε η διαγώνιος αυτή θα υποδιαιρεί το P σε δύο υποπολύγωνα P_1 και P_2 με m_1 και m_2 κορυφές αντίστοιχα. Κάθε κορυφή του P θα συναντάται ακριβώς σε ένα υποπολύγωνο εκτός από τις κορυφές που αποτελούν την διαγώνιο και οι οποίες θα συναντώνται και στα δύο υποπολύγωνα. Επομένως θα ισχύει η σχέση : $m_1 + m_2 = n + 2$. Αν T_{Pm} είναι ο αριθμός των τριγώνων της τριγωνοποίησης του πολυγώνου P_i τότε επαγωγικά μπορούμε να αποδείξουμε ότι $T_{Pm} = m - 2$, με m ο αριθμός κορυφών του P_m .

- Για m = 3 έχουμε : $T_{P3} = 1$, που ισχύει γιατί είναι τρίγωνο.
- Έστω ότι ισχύει για m = k , τότε θα έχουμε $T_{Pk} = k - 2$
- Θα αποδείξουμε ότι ισχύει για m = k+1

$T_{p(k+1)} = (k + 1) – 2 = (k-2) + 1 = T_{pk} + 1$, που ισχύει διότι για κάθε κορυφή που προστίθεται, προστίθεται και ένα τρίγωνο.

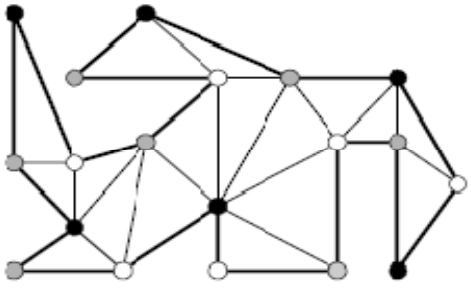
Το παραπάνω θεώρημα συνεπάγεται ότι κάθε απλό πολύγωνο με n κορυφές μπορεί να φυλαχθεί με (n – 2) κάμερες. Ωστόσο τοποθετώντας από μια κάμερα σε κάθε τρίγωνο φαίνεται υπερβολικό. Αν η κάμερα τοποθετηθεί για παράδειγμα σε μια διαγώνιο, τότε αυτή θα μπορεί να φυλάσει με μιας δύο τρίγωνα. Οπότε αν τοποθετήσουμε τις κάμερες σε σωστά επιλεγμένες διαγωνίους μπορούμε να μειώσουμε τον αριθμό των καμερών περίπου σε n/2 . **Τοποθετώντας τις κάμερες σε κορυφές** είναι μια ακόμα καλύτερη επιλογή γιατί μια κορυφή μπορεί να περιέχεται σε αρκετά τρίγωνα και μια κάμερα τοποθετημένη εκεί μπορεί να φυλάσσει όλα αυτά τα τρίγωνα.

1.3 Τριχρωματισμός Τριγωνοποιημένου Πολυγώνου (**3 – Coloring**)

Σύμφωνα με όσα αναφέρθηκαν παραπάνω καταλαβαίνουμε ότι είναι πολύ σημαντικό πλέον να τοποθετήσουμε τις κάμερες για την φύλαξη του πολυγώνου σε συγκεκριμένες θέσεις ώστε αυτές να είναι οι ελάχιστες δυνατές αλλά ωστόσο να μπορούν να καλύψουν όλο το εσωτερικό του πολυγώνου. Επίσης είδαμε ότι όταν μια κάμερα αποτελεί κορυφή ενός ή και παραπάνω τριγώνων τότε αυτή καλύπτει τον χώρο που καλύπτουν και τα συγκεκριμένα αυτά τρίγωνα. Στην συνέχεια αναλύεται μια πολύ αποτελεσματική μέθοδος τοποθέτησης των καμερών και ονομάζεται «Τριχρωματισμός του Τριγωνοποιημένου Πολυγώνου» (3 – Coloring of a Triangulated Polygon).

Έστω T_P μια τριγωνοποίηση του πολυγώνου P με n κορυφές. Αναζητάμε ένα υποσύνολο κορυφών του P τέτοιο ώστε κάθε τρίγωνο της T_P να έχει ως κορυφή του μια από τις κορυφές του παραπάνω υποσυνόλου και να τοποθετήσουμε με τον τρόπο αυτό τις κάμερες στις κορυφές του υποσυνόλου αυτού.

Για να βρούμε το υποσύνολο αυτό δίνουμε από ένα χρώμα σε κάθε κορυφή του P. Τα χρώματα πρέπει διαφορετικά μεταξύ τους και τρία σε αριθμό έστω μαύρο, γκρι και άσπρο. Ο χρωματισμός πρέπει να γίνει έτσι ώστε δύο κορυφές οι οποίες συνδέονται μεταξύ τους (¨γειτονικές¨) είτε σε κάποια πλευρά του πολυγώνου είτε σε κάποια διαγώνιο να έχουν διαφορετικό χρώμα μεταξύ τους. Ο τρόπος αυτός αποκαλείται «Τριχρωματισμός του Τριγωνοποιημένου Πολυγώνου». Σε κάθε τριχρωματισμένο τριγωνοποιημένο πολύγωνο κάθε τρίγωνο έχει μια άσπρη, μια γκρι και μια μαύρη κορυφή. Επομένως αν τοποθετήσουμε τις κάμερες σε όλες τις κορυφές ιδίου χρώματος, για παράδειγμα γκρι, θα μπορούμε φυσικά να φυλάσσουμε ολόκληρο το πολύγωνο. Ένα παράδειγμα τριχρωματισμού τριγωνοποιημένου πολυγώνου φαίνεται στο σχήμα 6.



Σχήμα 6

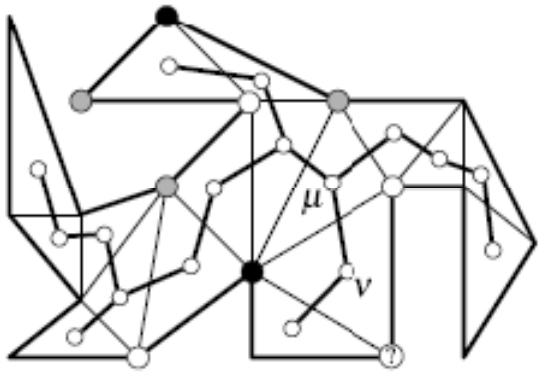
Στο σχήμα παρατηρούμε ότι οι κορυφές κάθε τριγώνου έχουν και τα τρία είδη χρώματος. Συγκεκριμένα οι συνολικά 19 κορυφές του πολυγώνου έχουν ως εξής:

- Άσπρες κορυφές : 6
- Γκρι κορυφές : 7
- Μαύρες κορυφές : 6

Επομένως οι ιδανικότερες θέσεις για την τοποθέτηση των καμερών είναι στις θέσεις των άσπρων ή των μαύρων κορυφών και όπως εύκολα μπορούμε να παρατηρήσουμε με την τοποθέτηση αυτών των 6 καμερών θα μπορούμε να έχουμε πλήρη κάλυψη του πολυγώνου.

Επιλέγοντας επομένως τον μικρότερο αριθμό χρωματισμένων κορυφών για να τοποθετήσουμε τις κάμερες μπορούμε να φυλάξουμε το πολύγωνο με την χρήση το πολύ $\lfloor n/3 \rfloor$ καμερών.

Δημιουργείται όμως το εξής ερώτημα : Ο τριχρωματισμός ενός πολυγώνου υπάρχει πάντα; Η απάντηση είναι πως ναι υπάρχει πάντα. Για να το δούμε αυτό πρέπει να αναλύσουμε το λεγόμενο « Διπλό Γράφημα της Τριγωνοποίησης T_P » (Dual Graph of T_P). Έστω λοιπόν το τριγωνοποιημένο πολύγωνο P ,με T_P την τριγωνοποίηση του, του σχήματος 6. Το Dual Graph του P δίνεται στο σχήμα 7 που ακολουθεί και το οποίο θα σχολιάσουμε.



Σχήμα 7

Το γράφημα αυτό, $G(T_P)$, έχει ένα κόμβο (node) για κάθε τρίγωνο της T_P. Χαρακτηρίζουμε κάθε τρίγωνο αντίστοιχα με τον κόμβο που βρίσκεται στο εσωτερικό του. Έστω ν ο κόμβος τότε t(ν) θα είναι το αντίστοιχο τρίγωνο. Υπάρχει

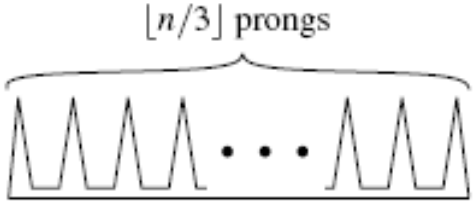
μια γραμμή μεταξύ δύο κόμβων, έστω ν και μ, αν τα τρίγωνα t(ν) και t(μ) μοιράζονται μια διαγώνιο. Οι γραμμές στο γράφημα $G(T_P)$ αντιστοιχούν σε διαγωνίους στην τριγωνοποίηση T_P. Επειδή όπως έχουμε ήδη αναφέρει μια διαγώνιος «κόβει» το πολύγωνο P σε δύο υποπολύγωνα, η μετακίνηση από μια πλευρά του γραφήματος $G(T_P)$ χωρίζει το γράφημα σε δύο άλλα. Επομένως το $G(T_P)$ αποτελεί δένδρο. (Παρατήρηση : Αυτό δεν ισχύει όταν το πολύγωνο έχει «τρύπες»). Αυτό σημαίνει ότι μπορούμε να βρούμε τον τριχρωματισμό ενός τριγωνοποιημένου πολυγώνου χρησιμοποιώντας την διαδρομή ενός απλού γραφήματος, όπως είναι η πρώτη αναζήτηση (depth first search). Στην συνέχεια περιγράφουμε πως γίνονται τα παραπάνω.

Καθώς διεξάγουμε την αναζήτηση στο δένδρο $G(T_P)$ διατηρούμε τον εξής κανόνα : όλες οι κορυφές των τριγώνων που έχουμε ήδη συναντήσει έχουν χρωματιστεί με άσπρο, γκρι ή μαύρο χωρίς δύο συνδεδεμένες κορυφές να έχουν το ίδιο χρώμα. Η μέθοδος αυτή απαιτεί ότι ένας τριχρωματισμός είναι σωστός εφόσον όλα τα τρίγωνα έχουν επεξεργαστεί. Η αναζήτηση μπορεί να ξεκινήσει από οποιονδήποτε κόμβο του $G(T_P)$ και οι κορυφές του τριγώνου που αντιστοιχεί στον κόμβο αυτό χρωματίζονται με άσπρο,γκρι και μαύρο. Ας υποθέσουμε ότι φθάνουμε στον κόμβο ν προερχόμενοι από τον κόμβο μ όπως φαίνεται στο σχήμα 7. Επομένως τα τρίγωνα t(ν) και t(μ) μοιράζονται μια διαγώνιο. Εφόσον οι κορυφές του τριγώνου t(μ) έχουν ήδη χρωματιστεί μόνο μια κορυφή του τριγώνου t(ν) απομένει για χρωματισμό (οι άλλες δύο κορυφές συνδεδεμένες αποτελούν την διαγώνιο που μοιράζονται τα δύο τρίγωνα). Υπάρχει επομένως ένα μόνο χρώμα το οποίο απομένει να δοθεί στην εναπομείνων κορυφή, δηλαδή αυτό που δεν χρησιμοποιούν οι κορυφές της διαγωνίου. Στην περίπτωση του σχήματος 7 η κορυφές της διαγωνίου έχουν μαύρο και άσπρο, συνεπώς στην κορυφή που χαρακτηριστεί με ? δίνουμε το γκρι. Επειδή το γράφημα $G(T_P)$ είναι δένδρο οι παρακείμενοι κόμβοι στον κόμβο ν δεν έχουν « επισκεφθεί » ακόμα μέσω της αναζήτησης οπότε έχουμε την ελευθερία να δώσουμε στην απομένουσα κορυφή χρώμα που απομένει.

1.4 Art Gallery Theorem

Ανακεφαλαιώνοντας λοιπόν τα παραπάνω είδαμε ότι ένα τριγωνοποιημένο απλό πολύγωνο μπορεί πάντα να τριχρωματιστεί. Αυτό έχει ως αποτέλεσμα ότι ένα απλό πολύγωνο μπορεί να φυλαχθεί με $\lfloor n/3 \rfloor$ κάμερες, όπου n ο αριθμός των κορυφών του. Πιθανότατα ο αριθμός αυτός να μπορεί να γίνει ακόμα μικρότερος γιατί μια κάμερα τοποθετημένη σε μια κορυφή μπορεί να φυλάσσει περισσότερα από τα προσκείμενα σε αυτή τρίγωνα. Δυστηχώς για κάθε n κορυφές υπάρχουν απλά πολύγωνα που απαιτούν ακριβώς $\lfloor n/3 \rfloor$ κάμερες. Ένα τέτοιο παράδειγμα είναι ένα πολύγωνο σε σχήμα «χτένας» (comb-shaped polygon) που φαίνεται στο σχήμα 8.

Στο διπλανό σχήμα τα δόντια (prongs) του σχήματος είναι συνδεδεμένα με ευθύγραμμα τμήματα. Η κατασκευή είναι τέτοια που καμία κάμερα δεν μπορεί να φυλάσσει ταυτόχρονα δύο «δόντια».



Σχήμα 8

Στην περίπτωση αυτή χρειαζόμαστε ακριβώς $\lfloor n/3 \rfloor$ κάμερες. Συνεπώς αυτή η μέθοδος τριχρωματισμού είναι βέλτιστη στην χειρότερη περίπτωση (worst-case scenario).

Μόλις αποδείξαμε λοιπόν το Art Gallery Theorem, ένα κλασσικό πρόβλημα της υπολογιστικής γεωμετρίας.

Θεώρημα 2 (Art Gallery Theorem) : Για ένα απλό πολύγωνο με n κορυφές, $\lfloor n/3 \rfloor$ κάμερες είναι περιστασιακά απαραίτητες και πάντα επαρκείς ώστε κάθε σημείο του πολυγώνου να είναι ορατό από μια τουλάχιστον κάμερα.

Γνωρίζουμε λοιπόν πλέον ότι $\lfloor n/3 \rfloor$ κάμερες είναι πάντα επαρκείς. Ωστόσο δεν έχουμε ακόμα ένα αποτελεσματικό αλγόριθμο για τον υπολογισμό των θέσεων τοποθέτησης των καμερών. Χρειαζόμαστε λοιπόν ένα γρήγορο αλγόριθμο που να τριγωνοποιεί ένα απλό πολύγωνο. Ο αλγόριθμος θα πρέπει να δίνει μια κατάλληλη αναπαράσταση της τριγωνοποίησης του πολυγώνου ώστε να μπορούμε να μεταβούμε από ένα τρίγωνο σε ένα άλλο γειτονικό του σε σταθερό χρόνο. Με μια τέτοια αναπαράσταση θα μπορούμε να υπολογίσουμε ένα σύνολο με το πολύ $\lfloor n/3 \rfloor$ θέσεων

κάμερας σε γραμμικό χρόνο σύμφωνα με την μέθοδο τριχρωματισμού που περιγράφηκε παραπάνω και να επιλέγουμε τις κορυφές εκείνες που είναι χρωματισμένες με τον μικρότερο αριθμό για να τοποθετήσουμε τις κάμερες. Στα επόμενα κεφάλαια περιγράφουμε πως μπορούμε να υπολογίσουμε την τριγωνοποίηση ενός πολυγώνου σε O(nlogn) χρόνο.

Προβλέποντας το παραπάνω αναφέρουμε το τελικό αποτέλεσμα σχετικά με την φύλαξη ενός πολυγώνου.

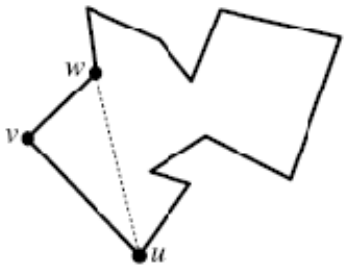
Θεώρημα 3 : Έστω P ένα απλό πολύγωνο με n κορυφές. Ένα σύνολο από $\lfloor n/3 \rfloor$ θέσεων κάμερας στο P ώστε κάθε σημείο εντός του πολυγώνου να είναι ορατό από τουλάχιστον μια κάμερα μπορεί να υπολογιστεί σε χρόνο O(nlogn).

Κεφάλαιο 2^{ο}

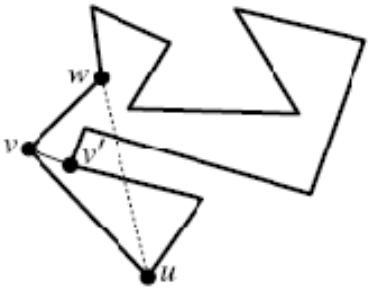
Διαίρεση Πολυγώνων σε Μονότονα Υποπολύγωνα.

2.1 y – Μονότονα Πολύγωνα.

Έστω P ένα απλό πολύγωνο με n κορυφές. Όπως είδαμε στο Θεώρημα 1 η τριγωνοποίηση ενός πολυγώνου υπάρχει πάντα. Η απόδειξη του Θεωρήματος αυτού είναι κατασκευαστική και οδηγεί σε ένα επαναληπτικό αλγόριθμο τριγωνοποίησης : Βρίσκουμε μια διαγώνιο και τριγωνοποιούμε τα δύο υποπολύγωνα που προκύπτουν επαναληπτικά. Ανατρέχουμε ξανά στο παράδειγμα των σχημάτων 4 και 5 που τα ξαναδίνουμε παρακάτω για καλύτερη εποπτεία.



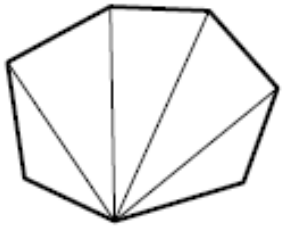
Σχήμα 4



Σχήμα 5

Για να βρούμε μια διαγώνιο παίρνουμε την πιο αριστερή κορυφή, έστω v, του P και προσπαθούμε να ενώσουμε τις δύο γειτονικές της, έστω w και u, αν η διαγώνιος αυτή δεν βρίσκεται στο εσωτερικό του P τότε συνδέουμε την v με την κορυφή που βρίσκεται μακρύτερα από το $\overline{uw}$ και μέσα στο τρίγωνο που σχηματίζεται από τις κορυφές u,v και w, όπως αποδείξαμε στο Θεώρημα 1 του 1^{ου} Κεφαλαίου. Η εύρεση μιας διαγωνίου μπορεί να υπολογιστεί σε γραμμικό χρόνο. Η διαγώνιος αυτή μπορεί να χωρίσει το P σε ένα τρίγωνο και ένα υποπολύγωνο με (n-1) κορυφές. Ως αποτέλεσμα ο αλγόριθμος θα εκτελείται σε τετραγωνικό χρόνο ($O(n^2)$) στην χειρότερη περίπτωση. Μπορούμε να επιτύχουμε καλύτερο χρόνο από αυτόν; Για συγκεκριμένα είδη πολυγώνων η απάντηση είναι πως ναι . Κυρτά πολύγωνα για παράδειγμα είναι πολύ εύκολα στην τριγωνοποίηση τους : Επιλέγουμε μια κορυφή τους και σχεδιάζουμε τις διαγώνιους από την κορυφή αυτή σε όλες τις υπόλοιπες εκτός από τις γειτονικές της, με τις οποίες είναι ήδη συνδεδεμένες και αποτελούν

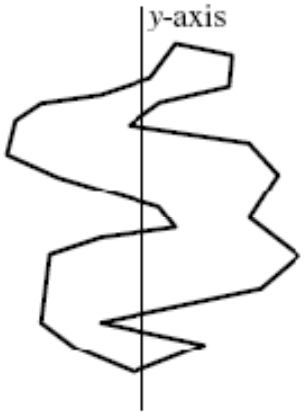
πλευρές του πολυγώνου. Ένα παράδειγμα τριγωνοποίησης κυρτού πολυγώνου δίνεται στο σχήμα 9 που ακολουθεί.



Σχήμα 9

Ο τρόπος αυτός απαιτεί γραμμικό χρόνο. Μια πιθανή προσέγγιση για την τριγωνοποίηση ενός μη-κυρτού πολυγώνου είναι ο χωρισμός του σε κυρτά υποπολύγωνα και στην συνέχεια την τριγωνοποίησή τους. Δυστηχώς ο χωρισμός αυτός είναι το ίδιο δύσκολος με την τριγωνοποίηση του. Για τον λόγο αυτό διαιρούμε το πολύγωνο P σε μονότονα υποπολύγωνα που είναι πιο απλή, εύκολη και γρήγορη από την αντίστοιχη διαίρεση σε κυρτά υποπολύγωνα.

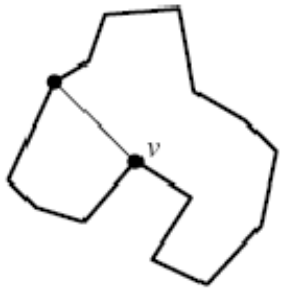
Ένα απλό πολύγωνο είναι μονότονο ως προς μια ευθεία λ αν για κάθε ευθεία λ΄ κάθετη στην λ η τομή του πολυγώνου με την λ΄ είναι συνδεδεμένη. Με άλλα λόγια, η τομή μπορεί να είναι ένα ευθύγραμμο τμήμα, ένα σημείο ή τίποτα. Ένα πολύγωνο το οποίο είναι μονότονο ως προς τον άξονα y ονομάζεται y-μονότονο. Ένα χαρακτηριστικό των y-μονότονων πολυγώνων είναι ότι αν «περπατήσουμε» από την ψηλότερη προς την χαμηλότερη κορυφή του, είτε από αριστερά είτε από δεξιά πάνω στην περιφέρεια του P, τότε θα κινούμαστε συνέχεια προς τα κάτω ή οριζόντια ποτέ προς τα πάνω. Ένα παράδειγμα y-μονότονου πολυγώνου δίνεται στο σχήμα 10.



Σχήμα 10

Η στρατηγική για την τριγωνοποίηση ενός πολυγώνου P είναι καταρχήν να χωρίσουμε το P σε y-μονότονα υποπολύγωνα και στην συνέχεια να τα τριγωνοποιήσουμε. Μπορούμε να διαιρέσουμε ένα πολύγωνο σε y-μονότονα υποπολύγωνα ως εξής : Ας φανταστούμε ότι κινούμαστε από την ψηλότερη προς την χαμηλότερη κορυφή του P είτε δεξιά είτε αριστερά στην περιφέρεια του P. Μια κορυφή όπου η κατεύθυνση αλλάζει από «προς τα κάτω» σε «προς τα πάνω» ή το αντίθετο ονομάζεται κορυφή «αλλαγής» (turn vertex). Ένα παράδειγμα μη-μονότονου πολυγώνου δίνεται στο σχήμα 11 όπου η κορυφή ν αποτελεί κορυφή αλλαγής.

Για να διαιρέσουμε το πολύγωνο P σε y-μονότονα υποπολύγωνα πρέπει να απαλλαγούμε από τις κορυφές αλλαγών. Αυτό μπορεί να γίνει με την προσθήκη διαγωνίων. Αν μια κορυφή του P, όπως η ν στο διπλανό σχήμα, έχει και τις δύο πλευρές της να κατευθύνονται προς τα κάτω και το εσωτερικό του πολυγώνου βρίσκεται πάνω από την κορυφή αυτή τότε πρέπει να επιλεγεί μια διαγώνιος που θα ξεκινάει από αυτή και θα κατευθύνεται προς τα πάνω. Η διαγώνιος αυτή χωρίζει πλέον το P σε δύο άλλα πολύγωνα που όπως φαίνεται και στο διπλανό παράδειγμα είναι y-μονότονα.



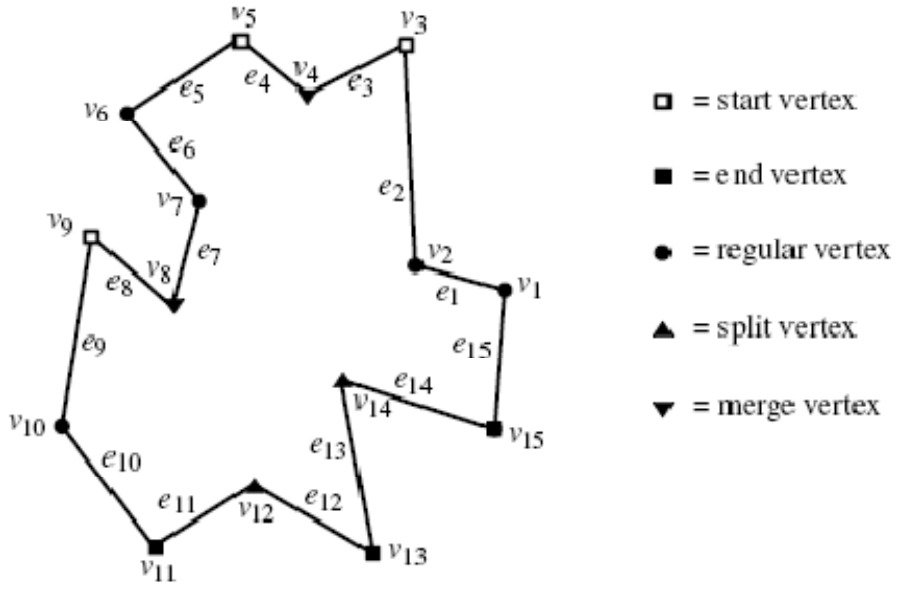
Σχήμα 11

Η κορυφή ν θα περιέχεται και στα δύο πλέον πολύγωνα όπου η μια πλευρά της θα κατευθύνεται προς τα πάνω και η άλλη προς τα κάτω. Επομένως η κορυφή ν παύει πλέον να αποτελεί κορυφή αλλαγής (turn vertex). Αντίστοιχα σε περίπτωση που οι πλευρές που πρόσκεινται σε μια κορυφή κατευθύνονται προς τα πάνω και το εσωτερικό του πολυγώνου βρίσκεται κάτω από την κορυφή αυτή τότε πρέπει να επιλεγεί μια διαγώνιος που να κατευθύνεται προς τα κάτω. Όπως γίνεται αντιληπτό υπάρχουν διάφοροι τύποι κορυφών αλλαγής (turn vertices) και οι οποίοι εξετάζονται στην επόμενη ενότητα.

2.2 Κατηγοροποίηση κορυφών ενός πολυγώνου.

Για την διαίρεση ενός πολυγώνου P σε y-μονότονα υποπολύγωνα θα πρέπει αρχικά να κατηγοροποιηθούν οι κορυφές του P ανάλογα με την συμπεριφορά των γειτονικών τους κορυφών (οι κορυφές με τις οποίες ενώνονται) αλλά και την συμπεριφορά των ακμών που συνδέουν τις κορυφές αυτές. Για να μπορέσουμε να διαχωρίσουμε τους τύπους των κορυφών πρέπει να δώσουμε ιδιαίτερη προσοχή στις κορυφές με ίδια τεταγμένη. Καθορίζουμε και εισάγουμε τους όρους «από κάτω» (below) και «από πάνω» (above) ως εξής: Ένα σημείο ή κορυφή p είναι «κάτω» από το σημείο ή κορυφή q αν $p_y < q_y$ ή αν $p_y = q_y$ και $p_x > q_x$ και είναι το p «πάνω» από το q αν $p_y > q_y$ ή αν $p_y = q_y$ και $p_x < q_x$.

Στην συνέχεια θα διακρίνουμε πέντε διαφορετικούς τύπους κορυφών με την βοήθεια ενός παραδείγματος όπως αυτό φαίνεται στο σχήμα 12.



Σχήμα 12

Οι τέσσερις από τους τύπους κορυφών αποτελούν κορυφές αλλαγής και είναι οι εξής : οι κορυφές αρχής (start vertices), οι κορυφές τέλους (end vertices), οι κορυφές διαίρεσης (split vertices) και οι κορυφές συγχώνευσης (merge vertices). Οι τύποι αυτοί καθορίζονται ως εξής : μια κορυφή v είναι κορυφή αρχής αν οι δύο γειτονικές της είναι «από κάτω» (below) της και η εσωτερική γωνία είναι μικρότερη από 180^o , αν η εσωτερική γωνία είναι μεγαλύτερη από 180^o τότε είναι κορυφή διαίρεσης. Στο παράδειγμα η κορυφή v_5 είναι κορυφή αρχής γιατί οι γειτονικές της v_4 και v_6 είναι από κάτω της (έχουν μικρότερες τεταγμένες) και η εσωτερική γωνία της

που σχηματίζεται από τις πλευρές e_4 και e_5 είναι μικρότερη από 180^o ενώ η κορυφή v_{14} είναι κορυφή διαίρεσης διότι η εσωτερική της γωνία είναι μεγαλύτερη από 180^o. Μια κορυφή αποτελεί κορυφή τέλους αν οι γειτονικές κορυφές της είναι «από πάνω» της (above) και η εσωτερική γωνία είναι μικρότερη από 180^o , αν η εσωτερική γωνία είναι μεγαλύτερη από 180^o τότε είναι κορυφή συγχώνευσης. Για παράδειγμα η v_{11} είναι κορυφή τέλους ενώ η v_8 είναι κορυφή συγχώνευσης. Θα πρέπει να σημειωθεί ότι η εσωτερική γωνία δεν μπορεί να είναι 180^o αν οι δύο γειτονικές πλευρές βρίσκονται και οι δύο είτε από πάνω είτε από κάτω. Οι κορυφές που δεν ανήκουν στους παραπάνω τύπους ονομάζονται « κανονικές » (regular vertices) κορυφές. Επομένως οι κανονικές κορυφές έχουν μια γειτονική κορυφή από πάνω και μια από κάτω. Οι συγκεκριμένες ονομασίες των τύπων κορυφών δεν είναι τυχαίες αλλά ονομάστηκαν με τον τρόπο αυτό διότι ο αλγόριθμος μια προς τα κάτω σάρωση στο επίπεδο (downward plane sweep) συγκρατώντας τις τομές της γραμμής σάρωσης με το πολύγωνο. Όταν η γραμμή σάρωσης φθάσει σε κορυφή διαίρεσης τότε ένα στοιχείο της τομής διαιρείται ενώ όταν φθάνει σε κορυφή συγχώνευσης τότε δύο στοιχεία του αλγορίθμου συγχωνεύονται. Όπως θα έχει γίνει πιθανόν αντιληπτό μέχρι τώρα οι κορυφές διαίρεσης και συγχώνευσης αποτελούν πηγές μη y – μονοτονίας. Στην συνέχεια αναφέρεται ένα Λήμμα χωρίς την απόδειξή του σχετικά με την y-μονοτονία ενός πολυγώνου.

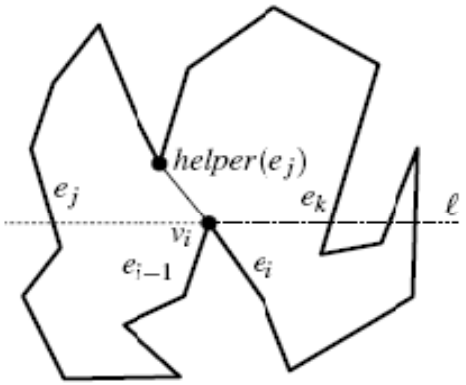
Λήμμα 1 : Ένα πολύγωνο είναι y-μονότονο αν δεν έχει κορυφές διαίρεσης (split vertices) ή κορυφές συγχώνευσης (merge vertices).

Το παραπάνω Λήμμα συνεπάγεται ότι το πολύγωνο P έχει πλέον χωριστεί σε y-μονότονα υποπολύγωνα εφόσον έχουμε απαλλαγεί από τις κορυφές διαίρεσης και συγχώνευσης. Αυτό μπορούμε να το επιτύχουμε με την προσθήκη διαγωνίων που να πηγαίνουν προς τα πάνω από κάθε κορυφή διαίρεσης και προς τα κάτω από κάθε κορυφή συγχώνευσης. Οι διαγώνιοι αυτοί δεν θα πρέπει να τέμνονται φυσικά μεταξύ τους. Όταν θα γίνει αυτό το πολύγωνο P θα έχει διαιρεθεί σε y-μονότονα υποπολύγωνα. Στην επόμενη υποενότητα δίνεται η ανάλυση του αλγορίθμου που επιτυγχάνει το επιθυμητό αυτό αποτέλεσμα.

2.3 Ο αλγόριθμος διαίρεσης Πολυγώνου P σε y-Μονότονα Υποπολύγωνα. (*Μετέτρεψε_Σε_Μονότονα*)

Στην συνέχεια αναλύεται διεξοδικά ο αλγόριθμος για την διαίρεση ενός πολυγώνου P σε y-μονότονα υποπολύγωνα. Αρχικά ας δούμε πως μπορούμε να προσθέσουμε διαγωνίους για τις κορυφές διαίρεσης (split vertices) του P. Χρησιμοποιούμε για τον σκοπό αυτό μια μέθοδο σάρωσης στο επίπεδο. Έστω v_1, v_2, ... , v_n μια **ανθορολογιακή** αρίθμηση των κορυφών του P και έστω e_1, e_2, …. , e_n το σύνολο των πλευρών του P, όπου $e_i = \overline{v_i v_{i+1}}$ για $1 \leq i < n$ και $e_n = \overline{v_{nv_1}}$. Ο αλγόριθμος σάρωσης χρησιμοποιεί μια φανταστική γραμμή σάρωσης προς τα κάτω στο επίπεδο. Η γραμμή σάρωσης σταματά σε συγκεκριμένα σημεία συμβάντος (event points) . Στην περίπτωσή μας τα σημεία συμβάντος θα είναι οι κορυφές του P και μόνον αυτές καθώς δεν δημιουργούνται άλλα σημεία συμβάντος κατά την διάρκεια του αλγορίθμου. Τα σημεία αυτά (event points) αποθηκεύονται σε μια σειρά συμβάντων Q (event queue). Η σειρά συμβάντων είναι μια σειρά προτεραιότητας, όπου προτεραιότητα μιας κορυφής είναι η y – συντεταγμένη της (τεταγμένη της). Άρα είναι μια σειρά προτεραιότητας ως προς τον άξονα y . Αν δύο κορυφές έχουν την ίδια τεταγμένη τότε αυτή με την μικρότερη x – συντεταγμένη έχει μεγαλύτερη προτεραιότητα. Με τον τρόπο αυτό το νέο σημείο συμβάντος που πρέπει να διαχειριστούμε μπορεί να βρεθεί σε χρόνο O(logn).

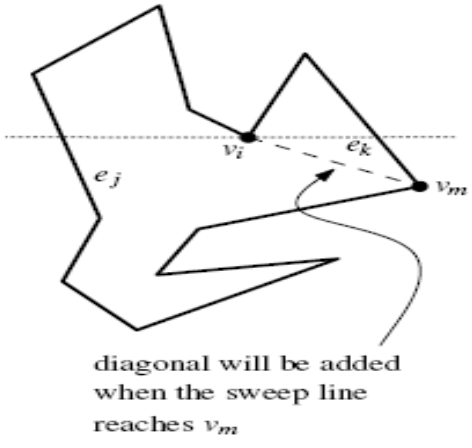
Στόχος της σάρωσης είναι η προσθήκη διαγωνίων από κάθε μια από τις κορυφές διαίρεσης του P προς μια κορυφή που θα βρίσκεται από πάνω τους. Έστω λοιπόν ότι η γραμμή σάρωσης φθάνει σε μια κορυφή διαίρεσης v_i όπως φαίνεται στο παρακάτω σχήμα 13.



Σχήμα 13

Με ποιά κορυφή θα πρέπει να ενώσουμε το v_i ; Μια ευνοϊκή υποψήφιος κορυφή θα ήταν μια κορυφή κοντά στο v_i, επειδή πιθανότατα θα μπορούσαμε να συνδέσουμε τις κορυφές αυτές χωρίς να τέμνεται η ευθεία σύνδεσης με κάποια από τις πλευρές του P. Πιο συγκεκριμένα έστω e_j η πλευρά αμέσως στα αριστερά της κορυφής v_i πάνω στην γραμμή σάρωσης και έστω e_k η πλευρά αμέσως στα δεξιά της v_i πάνω στην γραμμή σάρωσης. Στην συνέχεια θα μπορούσαμε να συνδέσουμε την v_i με την χαμηλότερη κορυφή ανάμεσα στις πλευρές e_j και e_k και πάνω από την v_i. Αν δεν υπάρχει μια τέτοια κορυφή τότε μπορούμε να συνδέσουμε την v_i με την πάνω κορυφή του e_j ή με την πάνω κορυφή του e_k . Ονομάζουμε την κορυφή αυτή helper (βοηθό) της πλευράς e_j και την συμβολίζουμε με helper(e_j). Γενικά ο helper(e_j) ορίζεται ως η χαμηλότερη κορυφή πάνω από την γραμμή σάρωσης ώστε η οριζόντια γραμμή που συνδέει την κορυφή αυτή με την πλευρά e_j του P να βρίσκεται εντός του P, όπως φαίνεται για παράδειγμα στο σχήμα 13. Ας σημειωθεί ότι ο helper(e_j) μπορεί να είναι η πάνω κορυφή της πλευράς e_j.

Γνωρίζουμε επομένως πως να απαλλαγούμε από τις κορυφές διαίρεσης, τις συνδέουμε με τον helper της πλευράς του P που βρίσκεται στα αριστερά τους. Τι γίνεται όμως με τις κορυφές συγχώνευσης; Με μια πρώτη ματιά φαίνεται ότι θα είναι πιο δύσκολο να απαλλαγούμε από αυτές γιατί θα πρέπει να προσθέσουμε μια διαγώνιο από την κορυφή συγχώνευσης σε μια κορυφή που θα βρίσκεται πιο κάτω. Αφού το τμήμα του πολυγώνου P κάτω από την γραμμή σάρωσης δεν έχει «εξερευνηθεί» ακόμα δεν μπορούμε να προσθέσουμε μια διαγώνιο όταν θα συναντήσουμε μια κορυφή συγχώνευσης. Ωστόσο η λύση του προβλήματος είναι πιο εύκολη από ότι φανταζόμαστε. Ας υποθέσουμε ότι η γραμμή σάρωσης φθάνει σε μια κορυφή συγχώνευσης έστω v_i όπως φαίνεται στο σχήμα 14.



Σχήμα 14

Έστω e_j και e_k οι πλευρές του P αμέσως στα αριστερά και στα δεξιά της κορυφής v_i πάνω στην γραμμή σάρωσης αντίστοιχα. Παρατηρούμε ότι η κορυφή v_i γίνεται ο νέος helper της πλευράς e_j όταν θα φθάσουμε σε αυτή. Θέλουμε να ενώσουμε την κορυφή αυτή με την κορυφή που βρίσκεται ψηλότερα κάτω από την γραμμή σάρωσης και ανάμεσα στις πλευρές e_j και e_k. Αυτό είναι ακριβώς το αντίθετο του τρόπου με τον οποίο χειριστήκαμε τις κορυφές διαίρεσης, τις οποίες και ενώναμε με την χαμηλότερη κορυφή πάνω από την γραμμή σάρωσης και ανάμεσα στις e_j και e_k. Αυτό συμβαίνει γιατί θα μπορούσαμε να θεωρήσουμε ότι οι κορυφές συγχώνευσης είναι κορυφές διαίρεσης γυρισμένες ανάποδα και το αντίθετο. Φυσικά δεν γνωρίζουμε ακόμα ποιά κορυφή είναι ψηλότερα κάτω από την γραμμή σάρωσης όταν φθάνουμε το v_i. Βέβαια μπορούμε να βρούμε την κορυφή αυτή πολύ εύκολα στην συνέχεια. Όταν φθάνουμε σε μια κορυφή έστω v_m που αντικαθιστά την κορυφή v_i ως helper της πλευράς e_j τότε αυτή είναι και η κορυφή που αναζητάμε. Οπότε κάθε φορά που αντικαθιστούμε τον helper μιας πλευράς πάντα θα ελέγχουμε αν ο παλιός helper είναι κορυφή συγχώνευσης και αν ναι προσθέτουμε την διαγώνιο ανάμεσα στον παλιό helper και τον νέο. Η διαγώνιος αυτή προστίθεται πάντα αν ο νέος helper είναι κορυφή διαίρεσης ώστε να απαλλαγούμε από αυτή. Αν ωστόσο ο προηγούμενος helper αποτελούσε κορυφή συγχώνευσης θα μπορούσαμε να απαλλαγούμε από μια κορυφή διαίρεσης και μια κορυφή συγχώνευσης με μια μόνο διαγώνιο. Θα μπορούσε ωστόσο να συμβεί ο helper της πλευράς e_j να μην αντικαθίσταται κάτω από την κορυφή v_i. Στην περίπτωση αυτή συνδέουμε την κορυφή συγχώνευσης v_i με την χαμηλότερη κορυφή της πλευράς e_j.

Στην παραπάνω προσέγγιση χρειάζεται να βρίσκουμε την πλευρά που βρίσκεται στα αριστερά μιας κορυφής. Για τον λόγο αυτό αποθηκεύουμε τις πλευρές του πολυγώνου P που τέμνουν την γραμμή σάρωσης στα φύλλα ενός δυναμικού δένδρου αναζήτησης, έστω T. Η από αριστερά στα δεξιά σειρά των φύλλων των δένδρων αντιστοιχεί στην από αριστερά προς τα δεξιά σειρά των πλευρών του P. Επειδή ενδιαφερόμαστε για πλευρές που βρίσκονται αριστερά από κορυφές διαίρεσης και συγχώνευσης χρειάζεται να αποθηκεύσουμε στο T μόνο τις πλευρές που έχουν στα δεξιά τους το εσωτερικό του P. Με κάθε πλευρά αποθηκεύουμε στο T και τον helper. Το δένδρο T και οι helpers μαζί με τις πλευρές αποτελούν το λεγόμενο status του αλγορίθμου σάρωσης. Το status αλλάζει καθώς η γραμμή σάρωσης κινείται προς τα κάτω όπου πλευρές του πολυγώνου αρχίζουν και σταματάνε να τέμνουν την γραμμή σάρωσης και ο helper κάποιας πλευράς μπορεί να αντικατασταθεί.

Στην συνέχεια παρουσιάζεται με την μορφή ψευδοκώδικα ο αλγόριθμος διαίρεσης ενός πολυγώνου P σε y-μονότονα υποπολύγωνα.

Αλγόριθμος *Μετέτρεψε_Σε_Μονότονο (P)*
Είσοδος. Ένα απλό πολύγωνο P.
Έξοδος. Μια διαίρεση του P σε y-μονότονα υποπολύγωνα.

1. **Δημιούργησε** μια σειρά προτεραιότητας Q από τις κορυφές του P, χρησιμοποιώντας τις y-συντεταγμένες ως προτεραιότητα. Αν δύο κορυφές έχουν την ίδια y-συντεταγμένη, η κορυφή με την μικρότερη x-συντεταγμένη έχει μεγαλύτερη προτεραιότητα.
2. **Δημιούργησε** ένα άδειο δέντρο αναζήτησης T.
3. **While** Q δεν είναι άδειο
4. **Do** Αφαίρεσε την κορυφή v_i με την μεγαλύτερη προτεραιότητα από το Q.
5. **Κάλεσε** την κατάλληλη συνάρτηση χειρισμού ανάλογα με τον τύπο της κορυφής.

Υπάρχουν δύο σημαντικές παράμετροι που πρέπει να έχουμε πάντα υπόψιν μας όταν χειριζόμαστε μια κορυφή. Καταρχήν πρέπει να ελέγχουμε πότε πρέπει να προσθέσουμε μια διαγώνιο. Πάντα προσθέτουμε μια σε περίπτωση που συναντάμε μια κορυφή διαίρεσης (split vertex) καθώς και όταν αντικαθιστούμε ένα helper μιας πλευράς ενώ ταυτόχρονα ο προηγούμενος helper ήταν μια κορυφή συγχώνευσης (merge vertex). Επίσης πρέπει να κρατάμε ενημερωμένο το status του δένδρου T. Στην συνέχεια δίνουμε τους αλγορίθμους σε μορφή ψευδοκώδικα για κάθε συγκεκριμένο τύπο σημείων - συμβάντος (κορυφές του P) καθώς και ένα παράδειγμα για να γίνει πιο ξεκάθαρος ο τρόπος με τον οποίο λειτουργεί ο αλγόριθμος.

Έστω το διάνυσμα, όπως έχουμε αναφέρει και παραπάνω, $e_i = \overline{v_i v_{i+1}}$ για $1 \leq i < n$ και $e_n = \overline{v_{nv_1}}$. Επίσης έστω ότι η κορυφή που επεξεργαζόμαστε κάθε φορά είναι η v_i .

Αλγόριθμος *Επεξεργάσου_Κορυφή_Αρχής* (v_i) *(HANDLE_START_VERTEX)*

1. **Εισήγαγε** το διάνυσμα e_i στο δέντρο T και βάλε ως helper (e_i) το v_i .

Αλγόριθμος *Επεξεργάσου_Κορυφή_Τέλους* (v_i) *(HANDLE_END_VERTEX)*

1. **Αν** ο helper(e_{i-1}) είναι κορυφή συγχώνευσης
2. **Τότε** πρόσθεσε την διαγώνιο που συνδέει την v_i με τον helper(e_{i-1}).
3. **Διέγραψε** από το δέντρο T το e_{i-1}.

Αλγόριθμος *Επεξεργάσου_Κορυφή_Διαίρεσης* (v_i) *(HANDLE_SPLIT_VERTEX)*

1. **Αναζήτησε** στο T να βρεις την πλευρά e_j του πολυγώνου P ακριβώς στα αριστερά της κορυφής v_i που επεξεργαζόμαστε.
2. **Πρόσθεσε** την διαγώνιο που συνδέει την v_i με τον helper(e_j).
3. **Θέσε** Helper (e_j)← v_i .
4. **Εισήγαγε** το e_i στο T και θέσε helper(e_i) στην κορυφή v_i .

Αλγόριθμος *Επεξεργάσου_Κορυφή_Συγχώνευσης* (v_i) *(HANDLE_MERGE_VERTEX)*

1. **Αν** ο helper(e_{i-1}) είναι κορυφή συγχώνευσης
2. **Τότε** πρόσθεσε την διαγώνιο που συνδέει την v_i με τον helper(e_{i-1}).
3. **Διέγραψε** από το δέντρο T το e_{i-1}.
4. **Αναζήτησε** στο T να βρεις την πλευρά e_j του πολυγώνου P ακριβώς στα αριστερά της κορυφής v_i που επεξεργαζόμαστε.
5. **Αν** ο helper(e_j) είναι κορυφή συγχώνευσης
6. **Τότε** πρόσθεσε την διαγώνιο που συνδέει την v_i με τον helper(e_j).
7. **Θέσε** Helper (e_j)← v_i .

Η μόνη περίπτωση που απομένει να εξετάσουμε τώρα είναι αν η κορυφή που έχουμε να επεξεργαστούμε είναι κανονική κορυφή (regular vertex). Οι ενέργειες στις οποίες πρέπει να προβούμε εξαρτώνται από το αν το εσωτερικό του πολυγώνου P βρίσκεται αριστερά ή δεξιά της κορυφής.

Αλγόριθμος *Επεξεργάσου_Κανονική_Κορυφή* (v_i)
(HANDLE_REGULAR_VERTEX)

1. **Αν** το εσωτερικό του P βρίσκεται δεξιά της v_i
2. **Τότε αν** ο helper(e_{i-1}) είναι κορυφή συγχώνευσης
3. **Τότε** εισήγαγε την διαγώνιο που συνδέει την v_i με τον helper(e_{i-1}) **.**
4. **Διέγραψε** από το δέντρο T το e_{i-1}.
5. **Εισήγαγε** το e_i στο T και θέσε helper(e_i) στην κορυφή v_i .
6. **Αλλιώς αναζήτησε** στο T να βρεις την πλευρά e_j του πολυγώνου P ακριβώς στα αριστερά της κορυφής v_i .
7. **Αν** ο helper(e_j) είναι κορυφή συγχώνευσης
8. **Τότε** πρόσθεσε την διαγώνιο που συνδέει την v_i με τον helper(e_j).
9. **Θέσε** Helper (e_j)← v_i .

Λήμμα 2 : Ο παραπάνω αλγόριθμος Μετέτρεψε_Σε_Μονότονο προσθέτει ένα σύνολο από μη τεμνόμενες διαγωνίους που διαιρούν το πολύγωνο P σε y-μονότονα υποπολύγωνα.

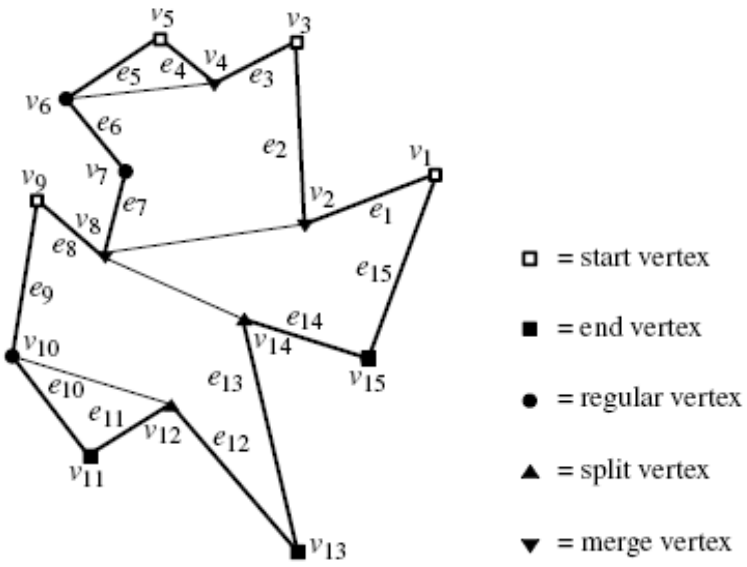
Πριν περιγράψουμε ένα παράδειγμα για να κατανοήσουμε πλήρως τον τρόπο με τον οποίο λειτουργεί ο αλγόριθμος ας αναλύσουμε τον χρόνο εκτέλεσης του. Η δημιουργία της σειράς προτεραιότητας Q γίνεται σε γραμμικό χρόνο και η δημιουργία του δέντρου T σε σταθερό χρόνο. Για να χειριστούμε ένα σημείο συμβάντος κατά την διάρκεια της σάρωσης εκτελούμε μια εργασία στο Q, το πολύ μια άντληση πληροφοριών, μια εισαγωγή και μια διαγραφή από το T και εισάγουμε το πολύ δύο διαγωνίους. Οι σειρές προτεραιότητας και τα δέντρα αναζήτησης επιτρέπουν την άντληση και την ενημέρωση πληροφοριών σε χρόνο

O(logn) και η εισαγωγή διαγωνίων σε χρόνο O(1). Επομένως για την επεξεργασία ενός σημείου συμβάντος (μιας κορυφής) χρειάζεται O(logn) χρόνος και ο συνολικός αλγόριθμος για n κορυφές θα εκτελείται σε O(nlogn) χρόνο. Προκύπτει επομένως το εξής θεώρημα:

Θεώρημα 4 : Ένα απλό πολύγωνο με n κορυφές μπορεί να διαιρεθεί σε y-μονότονα υποπολύγωνα σε O(nlogn) χρόνο.

2.4 Παράδειγμα εφαρμογής του αλγορίθμου *Μετέτρεψε_Σε_Μονότονα*.

Έστω το πολύγωνο του παρακάτω σχήματος :



Σχήμα 15

Το πολύγωνο αυτό αποτελείται από 15 κορυφές των οποίων οι τύποι δίνονται στα δεξιά. Όπως μπορούμε να διακρίνουμε το πολύγωνο αυτό δεν είναι y-μονότονο. Θα εφαρμόσουμε λοιπόν τον αλγόριθμο *Μετέτρεψε_Σε_Μονότονο* για τις κορυφές του πολυγώνου αυτού ώστε να προκύψουν τελικά τα 5 y-μονότονα υποπολύγωνα που φαίνονται στο σχήμα με την προσθήκη 4 διαγωνίων.

Αρχικά δημιουργούμε την σειρά προτεραιότητας Q η οποία περιέχει όλες τις κορυφές του πολυγώνου από αυτή με μεγαλύτερη τεταγμένη προς αυτή με μικρότερη. Σε περίπτωση ίδιων τεταγμένων η κορυφή με μικρότερη τετμημένη έχει προτεραιότητα (η αριστερότερη κορυφή έχει προτεραιότητα) .

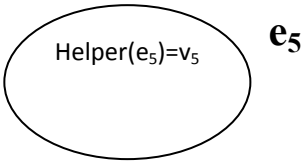
Έτσι λοιπόν η Q θα έχει ως εξής :

Q = [v_5 v_3 v_4 v_6 v_7 v_1 v_9 v_2 v_8 v_{14} v_{15} v_{10} v_{12} v_{11} v_{13}].

1. Επεξεργασία $1^{ου}$ σημείου συμβάντος (event point v_5).

Η κορυφή v_5 αποτελεί κορυφή αρχής. Ανατρέχοντας στον αλγόριθμο *Επεξεργάσου_Κορυφή_Αρχής* έχουμε :
Εισάγουμε το διάνυσμα e_5 στο T και θέτουμε v_5 = helper(e_5).

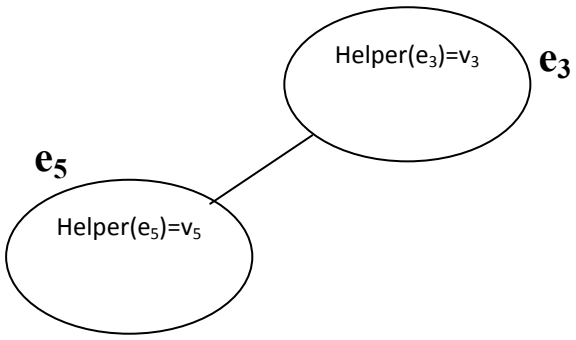
Το δέντρο T θα έχει ως εξής :



Ο μοναδικός κόμβος του δέντρου είναι το διάνυσμα e_5 και αποθηκεύουμε επίσης τον helper κάθε διανύσματος που είναι στο δέντρο.

2. Επεξεργασία $2^{ου}$ σημείου συμβάντος (event point v_3).

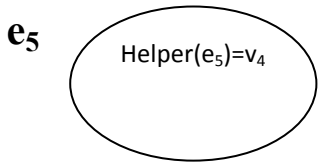
Η v_3 είναι κορυφή αρχής. Επομένως όπως και στην επεξεργασία του $1^{ου}$ σημείου συμβάντος θα έχουμε το εξής δέντρο T :



3. Επεξεργασία $3^{ου}$ σημείου συμβάντος (event point v_4).

Το v_4 αποτελεί κορυφή συγχώνευσης (merge vertex) επομένως μέσω του αλγορίθμου *Επεξεργάσου_Κορυφή_Συγχώνευσης* έχουμε:

helper(e_{i-1}) = helper(e_{4-1}) = helper(e_3) = v_3. Η v_3 δεν είναι κορυφή συγχώνευσης και επομένως δεν προσθέτουμε διαγώνιο. Διαγράφουμε από το T το διάνυσμα e_3. Στην συνέχεια αναζητάμε στο T την πλευρά που βρίσκεται ακριβώς στα αριστερά της v_4 που είναι το διάνυσμα e_5. Ο helper(e_5) = v_5 που δεν είναι κορυφή συγχώνευσης και επομένως δεν προσθέτουμε διαγώνιο. Θέτουμε helper(e_5) = v_4 και το T θα είναι :

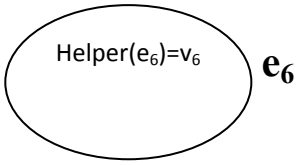


4. Επεξεργασία 4ου σημείου συμβάντος (event point v_6).

Το v_6 είναι κανονική κορυφή και ανατρέχοντας στον αλγόριθμο *Επεξεργάσου_Κανονική_Κορυφή* έχουμε :

Το εσωτερικό του πολυγώνου P βρίσκεται στα δεξιά της κορυφής v_6. Ο helper(e_5) = v_4 όπως βλέπουμε από το T όπου η v_4 αποτελεί κορυφή συγχώνευσης. Προσθέτουμε λοιπόν την διαγώνιο που συνδέει τις κορυφές v_6 και v_4 , $\overline{\boldsymbol{v_6 v_4}}$. Διαγράφουμε από το T το διάνυσμα e_5 και εισάγουμε το e_6 και θέτουμε helper(e_6)=v_6.

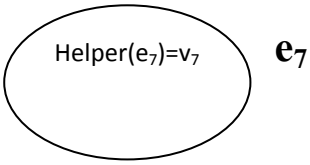
Το T θα έχει ως εξής:



5. Επεξεργασία 5ου σημείου συμβάντος (event point v_7).

Το v_7 είναι κανονική κορυφή και όπως και στο βήμα 4 έχουμε:

Το εσωτερικό του P βρίσκεται στα δεξιά του v_7 και ο helper(e_6) είναι κανονική κορυφή. Άρα δεν προσθέτουμε κάποια διαγώνιο και διαγράφουμε από το T το διάνυσμα e_6 και εισάγουμε το e_7 με helper(e_7) = v_7. Το T θα είναι :



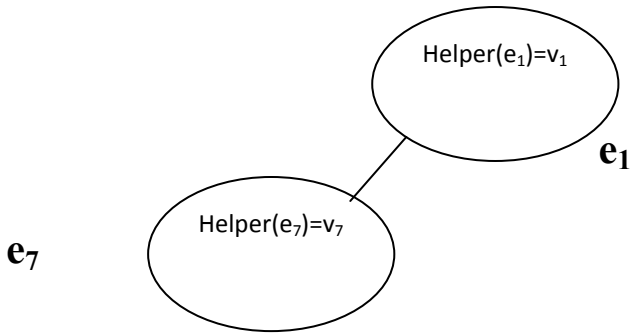
Ας δούμε πως θα είναι το Q μετά την επεξεργασία του v_7 καθώς και των σημείων συμβάντος που προηγήθηκαν :

Q = [v_1 v_9 v_2 v_8 v_{14} v_{15} v_{10} v_{12} v_{11} v_{13}]. Απομένει λοιπόν η επεξεργασία των σημείων αυτών.

6. Επεξεργασία 6ου σημείου συμβάντος (event point v_1).

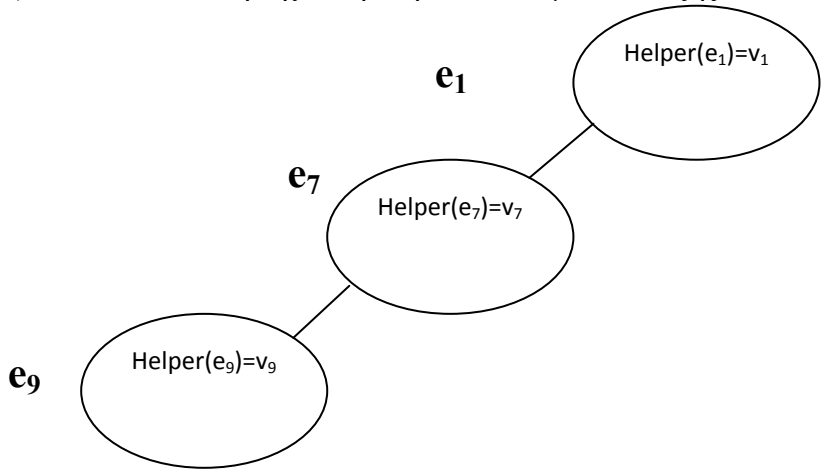
Αποτελεί κορυφή αρχής και όπως στα βήματα 1 και 2 θα έχουμε:

Εισάγουμε στο T το διάνυσμα e_1 και v_1 = helper(e_1) και το δέντρο θα έχει ως εξής :



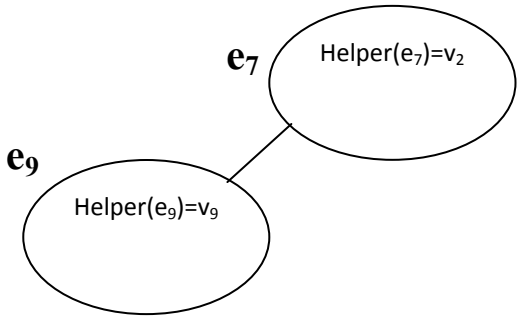
7. <u>Επεξεργασία 7ου σημείου συμβάντος (event point v_9).</u>

Αποτελεί και αυτό κορυφή αρχής και επομένως εισάγουμε το e_9 στο T και θέτουμε helper(e_9) = v_9. Το T θα περιέχει τώρα τρία διανύσματα τα εξής :



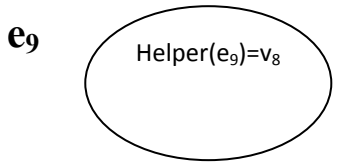
8. <u>Επεξεργασία 8ου σημείου συμβάντος (event point v_2).</u>

Αποτελεί κορυφή συγχώνευσης, helper(e_1) = v_1 που δεν είναι κορυφή συγχώνευσης και επομένως δεν προσθέτουμε κάποια διαγώνιο. Διαγράφουμε από το T το διάνυσμα e_1. Αναζητάμε στο T την πλευρά του P που βρίσκεται ακριβώς στα αριστερά του v_2 και είναι το e_7 με helper(e_7) = v_7 που δεν είναι κορυφή συγχώνευσης. Δεν προσθέτουμε διαγώνιο και θέτουμε v_2 = helper(e_7). Το δέντρο T θα είναι :



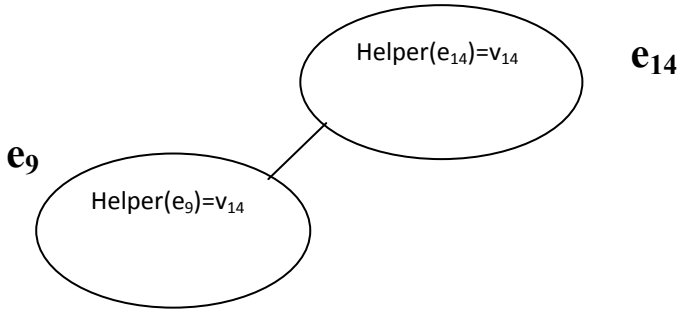
9. Επεξεργασία 9ου σημείου συμβάντος (event point v_8).

Το v_8 αποτελεί κορυφή συγχώνευσης. Ο helper(e_7) = v_2, είναι κορυφή συγχώνευσης και όπως προστάζει ο αλγόριθμος *Επεξεργάσου_Κορυφή_Συγχώνευσης* προσθέτουμε την διαγώνιο που συνδέει τις κορυφές v_8 και v_2, $\overline{\boldsymbol{v_8 v_2}}$. Διαγράφουμε το e_7 από το T και αναζητάμε την πλευρά που βρίσκεται ακριβώς στα αριστερά του v_8 που είναι το e_9. Ο helper(e_9) = v_9 που δεν είναι κορυφή συγχώνευσης. Δεν προσθέτουμε άλλη διαγώνιο και θέτουμε v_8 = helper(e_9). Το T πλέον θα έχει ένα διάνυσμα, το e_9.



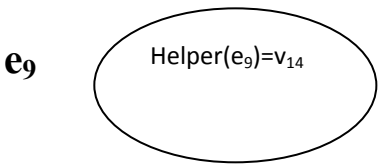
10. Επεξεργασία 10ου σημείου συμβάντος (event point v_{14}).

Η κορυφή v_{14} αποτελεί κορυφή διαίρεσης (split vertex). Ανατρέχοντας στον αλγόριθμο *Επεξεργάσου_Κορυφή_Διαίρεσης* έχουμε : Αναζητάμε στο T την πλευρά που είναι ακριβώς αριστερά της v_{14} και βρίσκουμε ότι είναι το e_9. Ο helper(e_9) = v_8 και προσθέτουμε την διαγώνιο που συνδέει τις κορυφές v_{14} και v_8, $\overline{\boldsymbol{v_{14} v_8}}$. Θέτουμε helper(e_9) = v_{14} και εισάγουμε το e_{14} στο T με helper(e_{14}) = v_{14}. Το T έχει ως εξής :



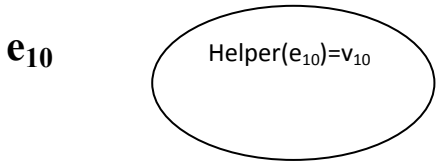
11. Επεξεργασία 11ου σημείου συμβάντος (event point v_{15}).

Η κορυφή v_{15} είναι κορυφή τέλους. Ο helper(e_{14}) = v_{14} δεν αποτελεί κορυφή συγχώνευσης, συνεπώς δεν προσθέτουμε κάποια διαγώνιο και διαγράφουμε από το T το διάνυσμα e_{14}. Το T τώρα είναι :



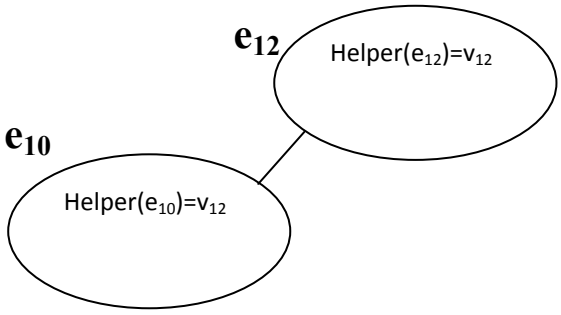
12. Επεξεργασία 12ου σημείου συμβάντος (event point v_{10}).

Η κορυφή v_{10} είναι κανονική κορυφή. Το εσωτερικό του P βρίσκεται στα δεξιά της. Ο helper(e_9) = v_{14} δεν αποτελεί κορυφή συγχώνευσης. Δεν προσθέτουμε διαγώνιο και διαγράφουμε το e_9 από το T. Εισάγουμε το e_{10} με v_{10} = helper(e_{10}). Το T θα είναι :



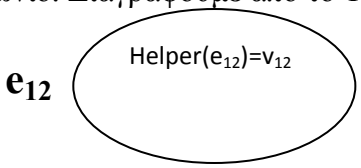
13. Επεξεργασία 13ου σημείου συμβάντος (event point v_{12}).

Η v_{12} αποτελεί κορυφή διαίρεσης και όπως έχουμε ήδη αναφέρει σε τέτοιου είδους κορυφές πάντα προσθέτουμε διαγωνίους. Αναζητάμε στο T την πλευρά που βρίσκεται ακριβώς στα αριστερά της. Είναι το μοναδικό διάνυσμα στο T, το e_{10} με helper(e_{10}) = v_{10} και προσθέτουμε επομένως την διαγώνιο που συνδέει τις κορυφές αυτές, $\overline{v_{12}v_{10}}$. Θέτουμε helper(e_{10}) = v_{12} και εισάγουμε στο T το e_{12}. Θα είναι επομένως :



14. Επεξεργασία 14ου σημείου συμβάντος (event point v_{11}).

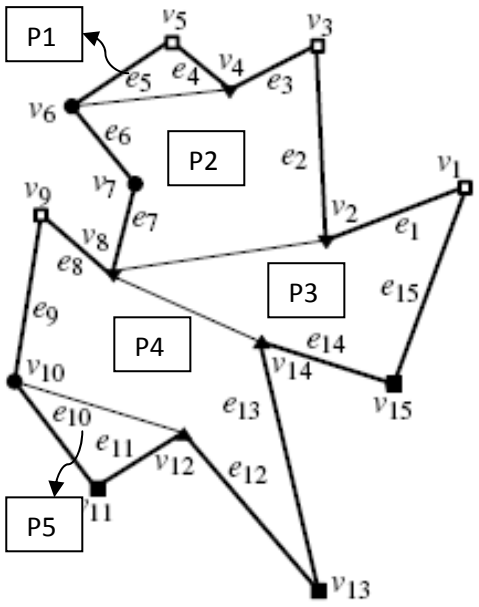
Το v_{11} είναι κορυφή τέλους. Ο helper(e_{10}) δεν είναι κορυφή συγχώνευσης και άρα δεν προσθέτουμε κάποια διαγώνιο. Διαγράφουμε από το T το e_{10} και θα είναι :



15. Επεξεργασία 15ου σημείου συμβάντος (event point v_{13}).

Η κορυφή αυτή, που είναι και η τελευταία στο Q, αποτελεί και αυτή κορυφή τέλους. Ο helper(e_{12}) δεν αποτελεί κορυφή συγχώνευσης και δεν προσθέτουμε διαγώνιο. Διαγράφουμε από το T το e_{12} και πλέον το T είναι άδειο.

Τελικά προσθέτουμε 4 διαγωνίους για την διαίρεσης του μη-μονότονου πολυγώνου P σε y-μονότονα υποπολύγωνα. Οι διαγώνιοι είναι οι εξής : $\overline{v_6v_4}$, $\overline{v_8v_2}$, $\overline{v_{14}v_8}$ και $\overline{v_{12}v_{10}}$. Προκύπτουν επομένως 5 υποπολύγωνα όπως φαίνεται και στο παρακάτω σχήμα :



Τα υποπολύγωνα P1 και P5 είναι τρίγωνα. Τα υπόλοιπα όμως πρέπει να τριγωνοποιηθούν για μπορέσουμε τελικά να βρούμε τις κατάλληλες θέσεις για την τοποθέτηση των καμέρων ώστε να επιτετύχουμε την πλήρη κάλυψη του P. Η τριγωνοποίηση αναλύεται στο επόμενο Κεφάλαιο.

Κεφάλαιο 3^{ο}

Τριγωνοποίηση Μονότονου Πολυγώνου.

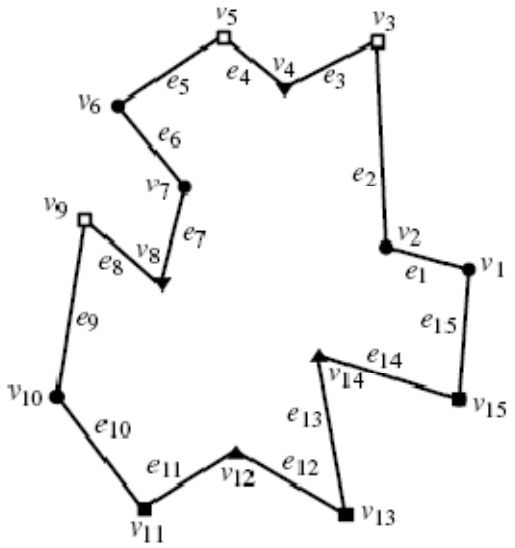
3.1 Ανάλυση του αλγορίθμου τριγωνοποίησης.

Στο προηγούμενο κεφάλαιο είδαμε πως μπορούμε να διαιρέσουμε ένα μη-μονότονο πολύγωνο σε μονότονα υποπολύγωνα με τον αλγόριθμο (*Μετέτρεψε_Σε_Μονότονα*) σε O(nlogn) χρόνο. Στο παρόν Κεφάλαιο θα δείξουμε ότι ένα μονότονο πολύγωνο μπορεί να τριγωνοποιηθεί σε γραμμικό χρόνο. Επομένως συνδέοντας τα δύο αυτά αποτελέσματα μπορούμε να τριγωνοποιήσουμε ένα οποιοδήποτε απλό πολύγωνο σε χρόνο O(nlogn) που είναι σαφώς γρηγορότερος από τον αλγόριθμο που αναφέραμε στο Κεφάλαιο 1 και ο οποίος χρειαζόταν O(n^2) χρόνο.

Έστω ένα y-μονότονο πολύγωνο P με n κορυφές. Το ότι είναι μονότονο στον άξονα y το πολύγωνο που εξετάζουμε σημαίνει ότι όταν «περπατάμε» από την ψηλότερη προς την χαμηλότερη κορυφή του, είτε από την δεξιά είτε από την αριστερή αλυσίδα, θα πηγαίνουμε συνέχεια «προς τα κάτω» δηλαδή προς κορυφές με μικρότερες τεταγμένες από αυτές που έχουμε ήδη συναντήσει. Αυτή είναι και η ιδιότητα των y-μονότονων πολυγώνων που τα κάνει σχετικά εύκολα στην τριγωνοποίηση. Θα μπορούμε λοιπόν κατά την διαδρομή από την κορυφή προς την βάση του πολυγώνου και από τις δύο αλυσίδες του, να προσθέτουμε διαγωνίους όπου είναι αυτό δυνατό. Στην συνέχεια περιγράφουμε τις λεπτομέρειες αυτού του αλγορίθμου.

Ο αλγόριθμος επεξεργάζεται τις κορυφές του πολύγωνου από αυτή με μεγαλύτερο προς αυτή με μικρότερη y-συντεταγμένη (order of decreasing-coordinate). Αν δύο κορυφές έχουν την ίδια y-συντεταγμένη τότε η πιο αριστερή (αυτή με μικρότερη x-συντεταγμένη) θα επεξεργαστεί πρώτη. Στο σημείο αυτό ας αναφέρουμε τι θα εννοούμε με τους όρους αριστερή και δεξιά αλυσίδα του πολυγώνου P. Έστω ότι θέλουμε να φτάσουμε από την ψηλότερη κορυφή του πολυγώνου προς την χαμηλότερη «περπατώντας» στην περιφέρεια του. Υπάρχουν δύο διαδρομές που μπορούμε να ακολούθησουμε για να το επιτύχουμε αυτό. Αυτή αριστερά και αυτή δεξιά της ψηλότερης κορυφής. Οι κορυφές που συναντάμε κατά την αριστερή διαδρομή ανήκουν στην αριστερή αλυσίδα και αυτές που συναντάμε

κατά την δεξιά διαδρομή ανήκουν στην δεξιά αλυσίδα. Στο παράδειγμα του πολυγώνου του σχήματος 16 που παρακάτω θα έχουμε :



Ψηλότερη Κορυφή η v_5

Χαμηλότερη Κορυφή η v_{13}

Αριστερή Αλυσίδα (Left Chain) :

v_6 , v_7 , v_8 , v_9 , v_{10} , v_{11} και v_{12}

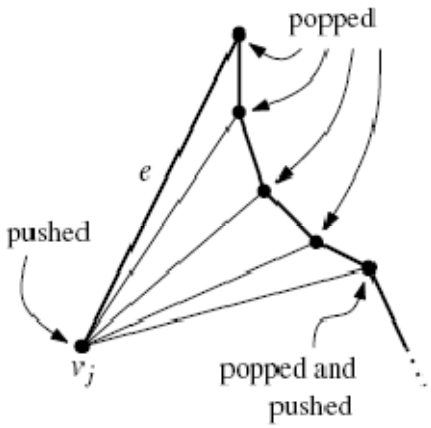
Δεξιά Αλυσίδα (Right Chain) :

v_4 , v_3 , v_2 , v_1 , v_{15} και v_{14}

Σχήμα 16

Ο αλγόριθμος απαιτεί μια σωρό (στοίβα)S ως βοηθητική δομή δεδομένων. Αρχικά η σωρός είναι άδεια, στην συνέχεια περιέχει τις κορυφές του P που έχουμε συναντήσει αλλά μπορεί να χρειάζεται να προσθέσουμε περισσότερες διαγωνίους. Όταν επεξεργαζόμαστε μια κορυφή προσθέτουμε όσο το δυνατόν περισσότερες διαγωνίους από την συγκεκριμένη κορυφή προς όλες τις κορυφές τις σωρού εφόσον αυτό βέβαια είναι εφικτό. Οι διαγώνιοι αυτοί διαχωρίζουν το P σε τρίγωνα. Οι κορυφές που έχουν επεξεργαστεί αλλά δεν έχουν διαχωριστεί , που είναι οι κορυφές στην σωρό S, βρίσκονται στο όριο του P που δεν έχει τριγωνοποιηθεί ακόμα. Η χαμηλότερη κορυφή, την οποία συναντήσαμε τελευταία, είναι στην κορυφή της σωρού, η δεύτερη χαμηλότερη είναι δεύτερη στην σωρό και ούτω καθ'εξής.

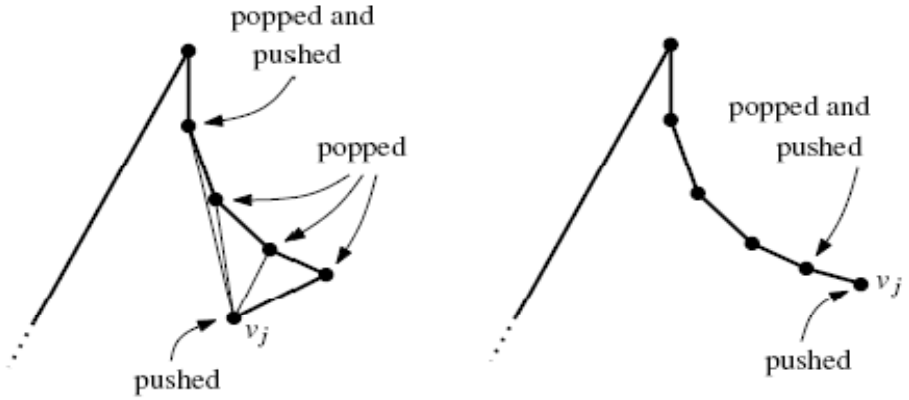
Στην συνέχεια θα δούμε ποιές διαγωνίους μπορούμε να προσθέσουμε όταν επεξεργαζόμαστε μια κορυφή, έστω v_j. Διακρίνουμε δύο περιπτώσεις : όταν η κορυφή v_j που έχουμε να επεξεργαστούμε βρίσκεται στην ίδια αλυσίδα (δεξιά ή αριστερή) με την κορυφή που βρίσκεται στην κορυφή της σωρού ή στην αντίθετη αλυσίδα. Αν η v_j βρίσκεται στην αντίθετη αλυσίδα τότε πρέπει αυτή να είναι το χαμηλότερο σημείο τέλους μιας πλευράς e όπως φαίνεται και στο σχήμα 17 που ακολουθεί :



Σχήμα 17

Αναλόγως βέβαια και με τη μορφή που έχει το σχήμα μπορούμε να προσθέσουμε διαγωνίους από την κορυφή v_j προς όλες τις κορυφές που βρίσκονται στην σωρό εκτός από την τελευταία με την οποία και είναι ήδη συνδεδεμένη αφού αποτελούν σημεία της πλευράς e. Όλες οι κορυφές αυτές βγαίνουν (pop) από την σωρό. Το μέρος του πολυγώνου P που δεν έχει τριγωνοποιηθεί ακόμα πάνω από την κορυφή v_j οριοθετείται από την διαγώνιο που συνδέει την v_j με την κορυφή που βρισκόταν προηγουμένως στην σωρό. Επομένως οι κορυφές αυτές εισάγονται (push) στην σωρό S καθώς αποτελούν μέρος του πολυγώνου που δεν έχει τριγωνοποιηθεί ακόμα.

Η άλλη περίπτωση είναι όταν η κορυφή v_j είναι στην ίδια αλυσίδα με την κορυφή που βρίσκεται στην κορυφή της σωρού. Στην περίπτωση αυτή ίσως να μην είναι δυνατόν να προσθέτουμε διαγωνίους από την v_j προς όλες τις κορυφές που βρίσκονται στην σωρό. Για να γίνει καλύτερα αντιληπτό έστω το σχήμα 18.



Σχήμα 18

Εφόσον η v_j είναι στην ίδια αλυσίδα οι κορυφές με τις οποίες μπορούμε τελικά να συνδέσουμε με την v_j είναι συνεχόμενες και βρίσκονται στην κορυφή της σωρού, οπότε μπορούμε να συνεχίσουμε ως εξής. Αρχικά βγάζουμε μια κορυφή από την σωρό, η κορυφή αυτή είναι ήδη συνδεδεμένη με την v_j με μια πλευρά του πολυγώνου P. Στην συνέχεια βγάζουμε κορυφές από την σωρό και τις συνδέουμε με την v_j μεχρί να βρούμε μια κορυφή με την οποία να μπορεί να γίνει η σύνδεσή τους, όπως φαίνεται και στο σχήμα 18. Ο έλεγχος για το πότε μια διαγώνιος μπορεί να προστεθεί από μια κορυφή v_j προς μια άλλη v_k της σωρού μπορεί να γίνει παρατηρώντας τις κορυφές v_j,v_k και την προηγούμενη κορυφή που έχουμε βγάλει από την σωρό. Όταν βρούμε μια κορυφή με την οποία δεν μπορούμε να συνδέσουμε την v_j εισάγουμε την τελευταία κορυφή που έχουμε βγάλει από την σωρό ξανά μέσα. Η κορυφή αυτή μπορεί να είναι η τελευταία κορυφή με την οποία προστέθηκε διαγώνιος με την v_j ή σε περίπτωση που δεν έχει προστεθεί κάποια διαγώνιος τότε η κορυφή αυτή θα είναι η γειτονική της v_j στην περιφέρεια του P όπως βλεπουμε και στο παραπάνω σχήμα. Στην συνέχεια εισάγουμε και την v_j στη σωρό.

Στις επόμενες δύο ενότητες δίνονται ο αλγόριθμος σε μορφή ψευδοκώδικα και η πολυπλοκότητά του, καθώς και ένα παράδειγμα για την καλύτερη κατανόηση των παραπάνω αντίστοιχα.

3.2 Παρουσίαση του αλγορίθμου και πολυπλοκότητα.

Στην συνέχεια παρουσιάζεται ο ψευδοκώδικας του αλγορίθμου τριγωνοποίησης ενός y-μονότονου πολυγώνου.

Αλγόριθμος *Τριγωνοποίησε_Μονότονο_Πολύγωνο (P)*

Είσοδος . Ένα y-μονότονο πολύγωνο P.
Έξοδος . Ένα τριγωνοποιημένο πολύγωνο P.

1. **Συγχωνεύουμε** τις κορυφές της αριστερής και της δεξιάς αλυσίδας του P σε μια ακολουθία, ταξινομημένες με μειώμενη τεταγμένη (ψηλότερη προς χαμηλότερη κορυφή). Αν δύο κορυφές έχουν την ίδια τεταγμένη η πιο αριστερή έρχεται πρώτη. Έστω u_1, u_2, …., u_n οι ταξινομημένες κορυφές της ακολουθίας.
2. **Εισάγουμε** τις u_1 και u_2 (που είναι και οι δύο μεγαλύτερες κορυφές) στην σωρό S.
3. **Για** j←3 μέχρι (n-1)
4. **Κάνε αν** η u_j και η κορυφή που βρίσκεται στην κορυφή της σωρού S ανήκουν σε διαφορετικές αλυσίδες
5. **Τότε** **Βγάλε** όλες τις κορυφές από την S
6. **Πρόσθεσε** τις διαγωνίους από την u_j σε κάθε μια κορυφή που έχουμε βγάλει από την S εκτός από την τελευταία.
7. **Εισήγαγε** τις u_{j-1} και u_j στην S.
8. **Αλλιώς** **Βγάλε** μια κορυφή από την S.
9. **Βγάλε** από την S τις άλλες κορυφές όσο οι διαγώνιοι από την u_j σε αυτές είναι εντός του πολυγώνου P. Εισήγαγε την τελευταία κορυφή που έχεις βγάλει από την S πίσω στην σωρό.
10. **Εισήγαγε** την u_j στην S.
11. **Πρόσθεσε** διαγωνίους από την u_n σε όλες τις κορυφές από τις σωρού εκτός από την πρώτη και την τελευταία.

Ποιός είναι ο χρόνος εκτέλεσης του αλγορίθμου; Το πρώτο βήμα χρειάζεται γραμμικό χρόνο και το δεύτερο σταθερό χρόνο. Ο βρόχος **Για** εκτελείται (n-3) φορές και η μια εκτέλεση μπορεί να χρειαστεί γραμμικό χρόνο. Αλλά σε κάθε εκτέλεση στον βρόχο αυτό το πολύ δύο κορυφές εισάγονται. Επομένως ο συνολικός αριθμός των εισαγωγών, συμπεριλαμβανομένων και των δύο στο δεύτερο βήμα είναι 2n-4. Επειδή ο αριθμός των εξαγωγών απο την σωρό δεν μπορεί να υπερβαίνει τον αριθμό των εισαγωγών, ο συνολικός χρόνος όλων των εκτελέσεων του βρόχου είναι O(n). Το τελευταίο βήμα του αλγορίθμου χρειάζεται το πολύ γραμμικό χρόνο, οπότε ο συνολικός χρόνος εκτέλεσης του αλγορίθμου είναι O(n). Προκύπτει επομένως το εξής θεώρημα:

Θεώρημα 3.1 : Ένα y-μονότονο πολύγωνο με n κορυφές μπορεί να τριγωνοποιηθεί σε γραμμικό χρόνο.

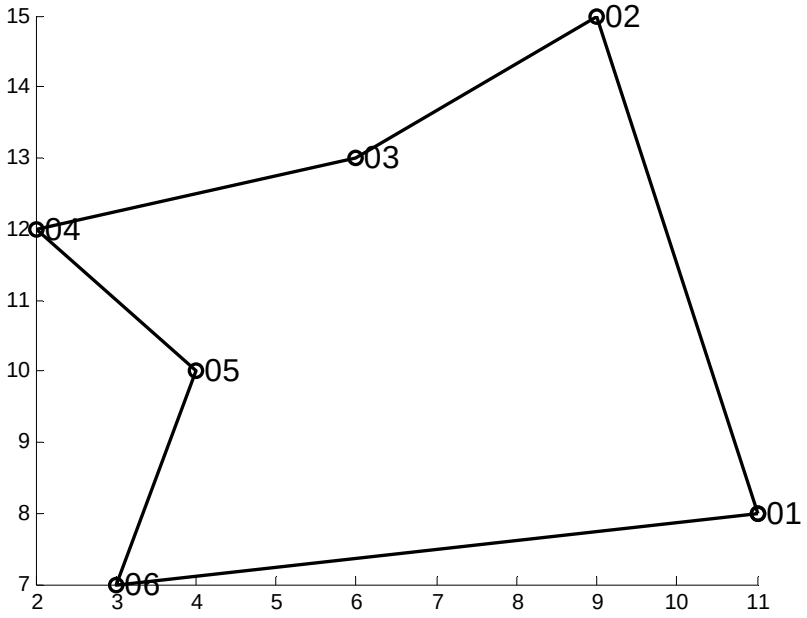
Επιθυμούμε ένα αλγόριθμο τριγωνοποίησης για μονότονα πολύγωνα ως υπορουτίνα για την τριγωνοποίηση αυθαίρετων απλών πολυγώνων. Η ιδέα είναι να διαιρέσουμε αρχικά το δεδομένο πολύγωνο σε μονότονα υποπολύγωνα (Θεώρημα 4) και στην συνέχεια να τα τριγωνοποιήσουμε. Μπορούμε επομένως να συνδυάσουμε τους δύο αλγορίθμους για να επιτύχουμε έναν αποτελεσματικό αλγόριθμο τριγωνοποίησης που να μπορεί να επεξεργαστεί ένα οποιοδήποτε απλό πολύγωνο. Προκύπτει επομένως το εξής θεώρημα:

Θεώρημα 3.2: Ένα απλό πολύγωνο με n κορυφές μπορεί να τριγωνοποιηθεί σε χρόνο O(nlogn)

Στην συνέχεια παρατίθεται ένα παράδειγμα τριγωνοποίησης y-μονότονου πολυγώνου.

3.3 Παράδειγμα τριγωνοποίησης πολυγώνου

Στην συνέχεια αναλύεται ένα παράδειγμα για την καλύτερη κατανόηση του αλγορίθμου τριγωνοποίησης μονότονου πολυγώνου. Έστω λοιπόν το πολύγωνο P του παρακάτω σχήματος :



Το πολύγωνο αυτό είναι, όπως εύκολα μπορούμε να παρατηρήσουμε, y-μονότονο και αποτελείται από 6 κορυφές. Στόχος μας είναι η τριγωνοποίηση του πολυγώνου αυτού. Εφαρμόζουμε τον αλγόριθμο *Τριγωνοποίησε_Μονότονο_Πολύγωνο* που αναλύθηκε στην προηγούμενη ενότητα και ακολουθώντας τα βήματα ένα προς ένα έχουμε :

Η ψηλότερη κορυφή του πολυγώνου είναι η κορυφή 02 ενώ η χαμηλότερη η κορυφή 06. Αρχικά πρέπει να βρούμε την αριστερή και την δεξιά αλυσίδα του πολυγώνου. Η αριστερή διαδρομή που ακολουθούμε για να φθάσουμε από την κορυφή 02 στην 06 συναντά τις κορυφές 03,04 και 05. Επομένως :

Αριστερή Αλυσίδα (Left Chain) = { 03, 04, 05 }. Ομοίως :

Δεξιά Αλυσίδα (Right Chain) = { 01 }.

Στο πρώτο βήμα του αλγορίθμου συγχωνεύουμε τις κορυφές των δύο αλυσίδων ταξινομημένα ως προς την τεταγμένη τους από την μεγαλύτερη προς την μικρότερη. Η ακολουθία των ταξινομημένων κορυφών θα είναι :

02 , 03 , 04 , 05 , 01 και 06 .

Με το δεύτερο βήμα εισάγουμε τις δύο μεγαλύτερες κορυφές του πολυγώνου στην σωρό S και έχουμε :

S = | 03 |
    | 02 |

Στο βρόχο **ΓΙΑ** που ακολουθεί έχουμε :

Για την κορυφή 04 ότι βρίσκεται στην ίδια αλυσίδα με την κορυφή που βρίσκεται στην κορυφή της σωρού (03). Βγάζουμε μια κορυφή από το S και συνεχίζουμε να βγάζουμε κορυφές από την σωρό εφόσον οι διαγώνιοι από το 04 προς αυτές βρίσκονται εντός του πολυγώνου. Με την εξαγωγή της 03 από το S απομένει μόνο η 02 εντός της S. Η διαγώνιος που συνδέει την 04 με την 02 βρίσκεται εκτός του πολυγώνου. Οπότε ξαναεισάγουμε την 03 στην S καθώς και την 04. Οπότε η σωρός θα είναι :

S = | 04 |
    | 03 |
    | 02 |

Για την κορυφή 05 έχουμε ότι είναι στην ίδια αλυσίδα με την 04. Βγάζουμε την 04 από την σωρό. Επίσης βγάζουμε τις κορυφές με τις οποίες η 05 συνδέεται με διαγωνίους εντός του πολυγώνου. Οι διαγώνιοι $\overline{05\ 03}$ και $\overline{05\ 02}$ είναι εντός του P. Εισάγουμε ξανά την 02 στην S καθώς και την 05. Η σωρός θα είναι τώρα ως εξής :

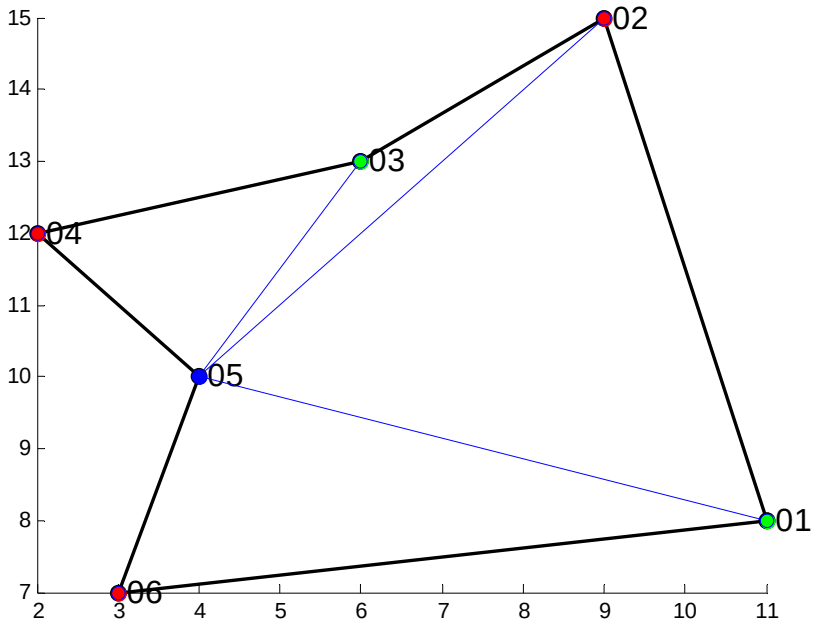
S = | 05 |
    | 02 |

Για την κορυφή 01 παρατηρούμε είναι σε διαφορετική αλυσίδα με την 05. Βγάζουμε από την σωρό όλες τις κορυφές και εισάγουμε διαγωνίους προς όλες τις κορυφές που μόλις εξάγαμε εκτός από την τελευταία. Άρα προσθέτουμε την διαγώνιο $\overline{01\ 05}$ και εισάγουμε τις κορυφές 01 και 05 στην σωρό η οποία θα είναι :

S = | 01 |
    | 05 |

Για την τελευταία κορυφή 06 προσθέτουμε διαγωνίους προς όλες τις κορυφές του πολυγώνου που βρίσκονται στην S εκτός από την πρώτη και την τελευταία. Εφόσον έχουμε δύο μόνο κορυφές αποθηκευμένες στην S δεν προσθέτουμε κάποια διαγώνιο.

Τελικά οι διαγώνιοι που προσθέτουμε είναι τρεις οι εξής $\overline{05\,03}$, $\overline{05\,02}$ και $\overline{01\,05}$. Προκύπτει επομένως το τριγωνοποιημένο πολύγωνο του παρακάτω σχήματος:



Η φύλαξη του παραπάνω πολυγώνου με την χρήση της μεθόδου τριχρωμίας, που περιγράψαμε στην ενότητα 1.3, μπορεί να γίνει με μια μόνο κάμερα τοποθετημένη στην μπλε χρωματισμένη κορυφή (05).

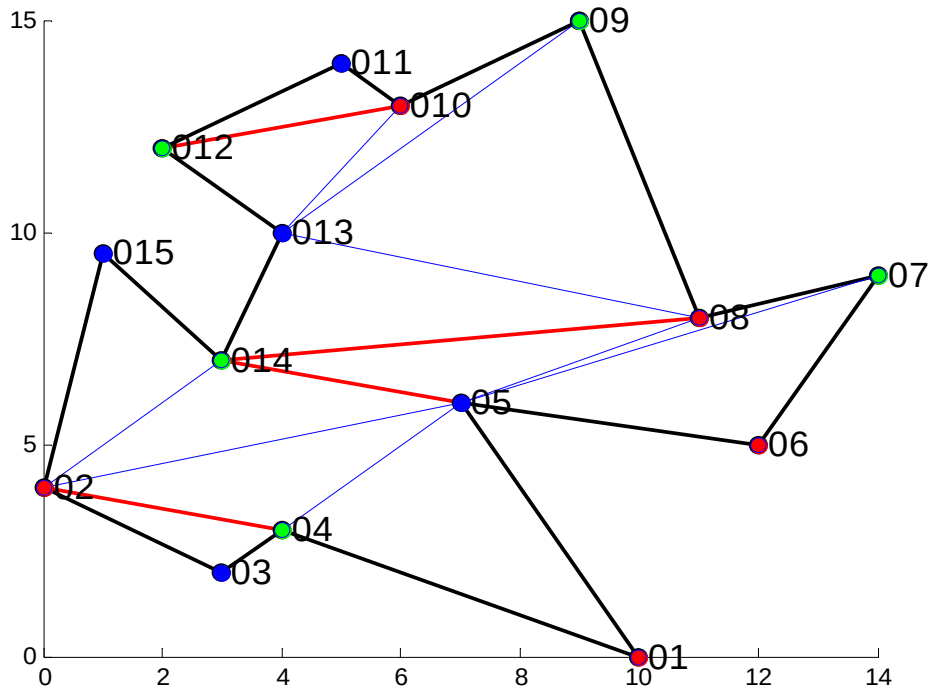
Συνοψίζοντας λοιπόν όλα τα παραπάνω συνδυάζοντας τους αλγορίθμους που αναλύθηκαν στα Κεφάλαια 2 και 3 της παρούσης διπλωματικής μπορούμε να διαιρέσουμε οποιοδήποτε απλό πολύγωνο σε y-μονότονα υποπολύγωνα και στην συνέχεια να τριγωνοποιήσουμε κάθε ένα από αυτά ξεχωριστά και να συνθέσουμε στην συνέχεια το τελικό αποτέλεσμα που δεν είναι άλλο από το τριγωνοποιημένο αρχικό πολύγωνο. Στο επόμενο Κεφάλαιο δίνονται κάποια χαρακτηριστικά παραδείγματα τριγωνοποίησης πολυγώνων που πραγματοποιήθηκαν με το πρόγραμμα σε MATLAB που αναπτύξαμε, ο κώδικας του οποίου παρατίθεται στο Παράρτημα.

Κεφάλαιο 4ο
Αποτελέσματα εφαρμογών στο MATLAB.

Στην συνέχεια παρατίθενται ορισμένα χαρακτηριστικά παραδείγματα τριγωνοποίησης πολυγώνων με την χρήση του μαθηματικού προγραμματιστικού πακέτου MATLAB. Ως είσοδο δίνουμε δύο .txt αρχεία ένα με τις συντεταγμένες των κορυφών του πολυγώνου και ένα με τον τρόπο σύνδεσης των κορυφών αυτών ανθορολωγιακά, όπως προστάζει και ο αλγόριθμος. Τα .txt αρχεία καθώς και ο κώδικας παρατίθενται στο Παράρτημα.

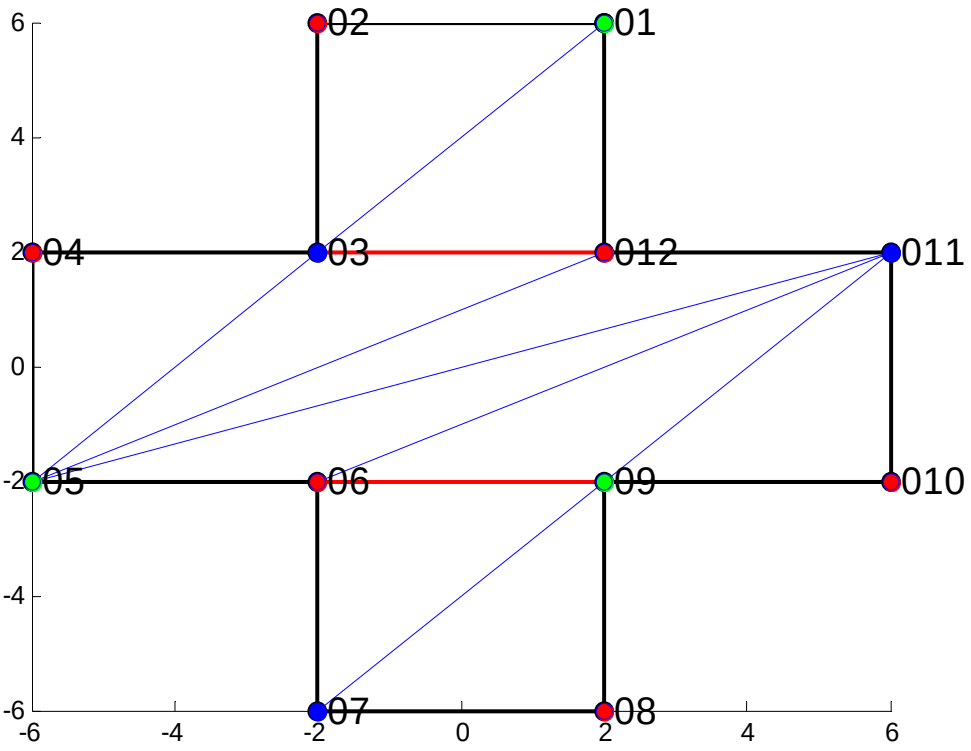
Εφαρμογή 1η .

Ως πρώτη εφαρμογή κατασκευάσαμε ένα μη y-μονότονο τυχαίο πολύγωνο. Οι κορυφές του πολυγώνου είναι αριθμημένες με βάση την σειρά που δόθηκαν. Με μαύρο είναι οι πλευρές που αποτελούν το πολύγωνο, με κόκκινο δίνονται οι διαγώνιοι που διαιρούν το πολύγωνο σε y-μονότονα υποπολύγωνα. Με μπλε είναι οι διαγώνιοι που τριγωνοποιούν το κάθε y – μονότονο υποπολύγωνο. Μπλε – Πράσινο και Κόκκινο είναι τα χρώματα που επιλέξαμε για τον τριχρωματισμό (3-Colored) του πολυγώνου. Οι κάμερες για την πλήρη φύλαξη του πολυγώνου θα τοποθετηθούν, όπως έχουμε αναφέρει και στο Κεφάλαιο 1, στις ενός χρώματος κορυφές με τον μικρότερο αριθμό.



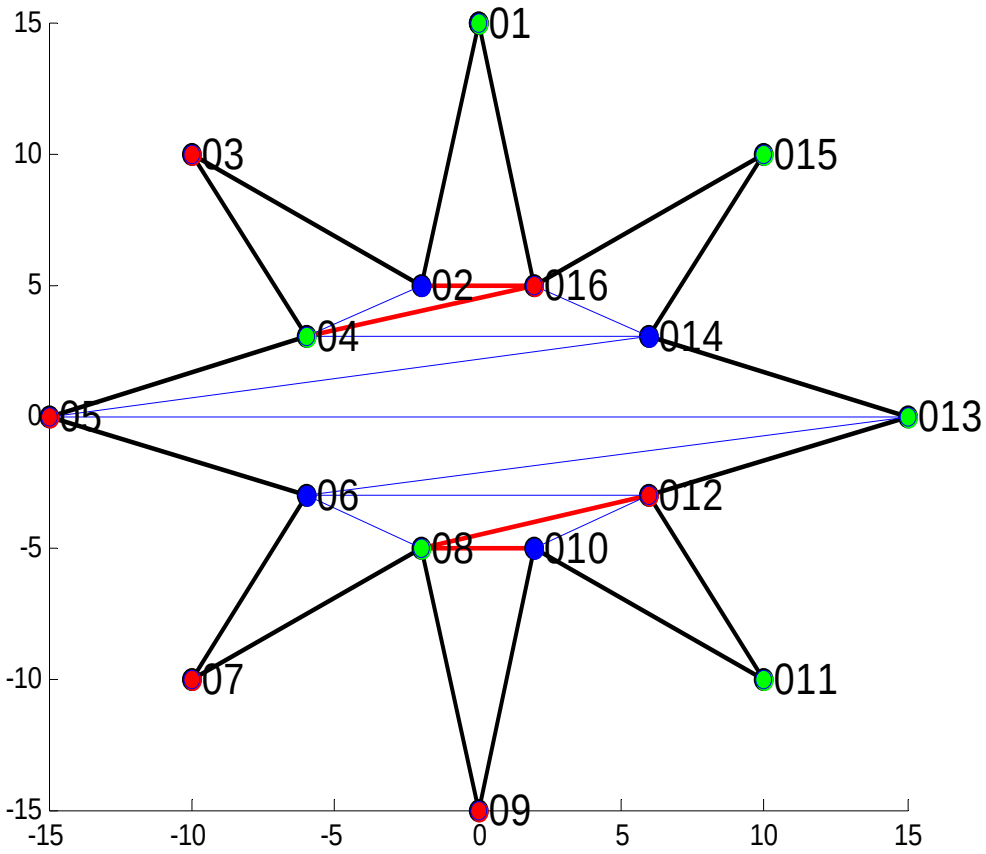
Αριθμός Μπλε Κορυφών	Αριθμός Κόκκινων Κορυφών	Αριθμός Πράσινων Κορυφών
5	**5**	**5**
Εφόσον ο αριθμός και των τριών διαφορετικών χρωμάτων είναι ο ίδιος μπορούμε να τοποθετήσουμε τις κάμερες σε οποιουδήποτε χρώματος κορυφές για να επιτύχουμε την πλήρη κάλυψή του.		

Εφαρμογή 2η .



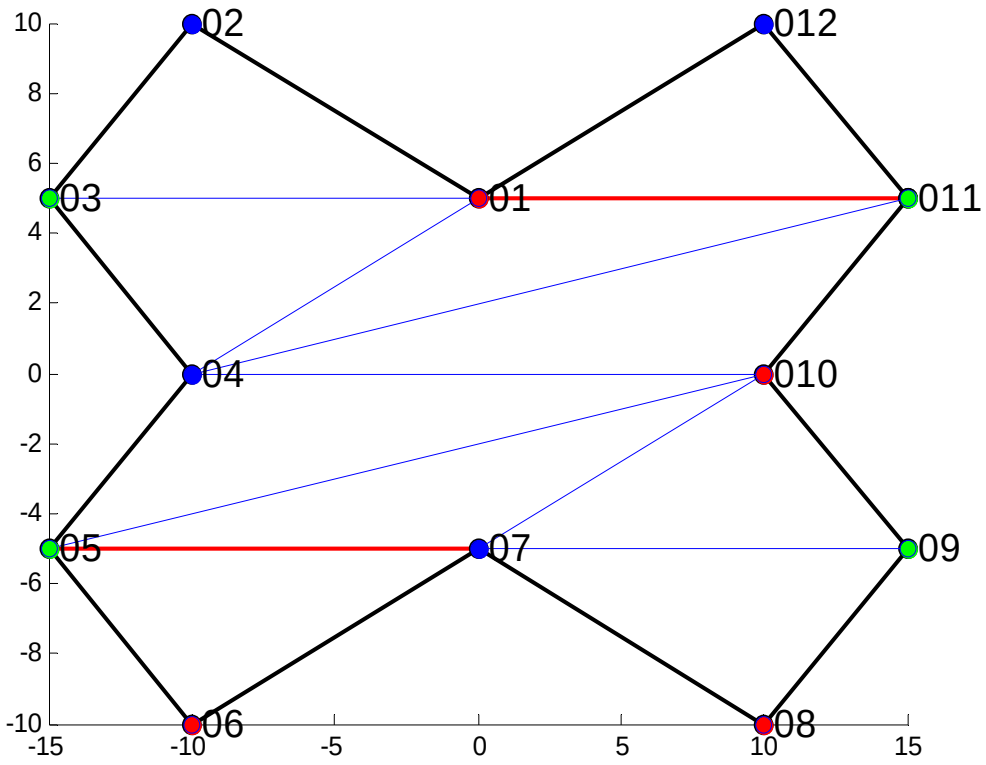
Αριθμός Μπλε Κορυφών	Αριθμός Κόκκινων Κορυφών	Αριθμός Πράσινων Κορυφών
3	**6**	**3**
Οι κάμερες μπορούν να τοποθετηθούν στις θέσεις είτε των Μπλε είτε των Πράσινων κορυφών του πολυγώνου.		

Εφαρμογή 3η .



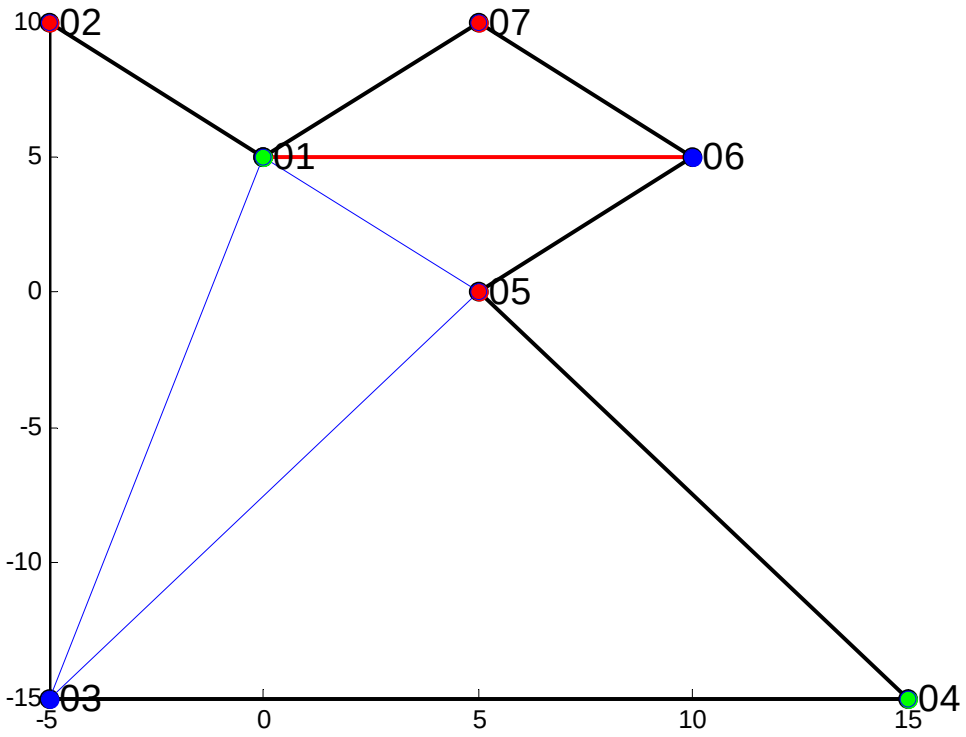
Αριθμός Μπλε Κορυφών	Αριθμός Κόκκινων Κορυφών	Αριθμός Πράσινων Κορυφών
4	**6**	**6**
Οι κάμερες θα τοποθετηθούν στις θέσεις των Μπλε κορυφών του πολυγώνου εφόσον αυτές έχουν τον μικρότερο αριθμό.		

Εφαρμογή 4η .



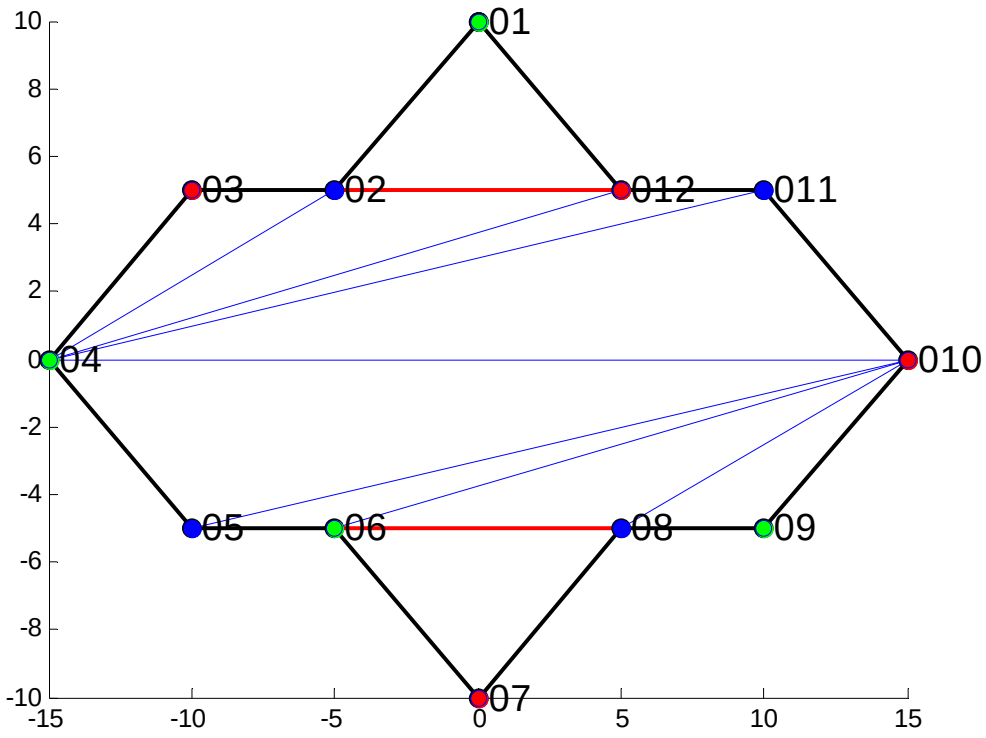
Αριθμός Μπλε Κορυφών	Αριθμός Κόκκινων Κορυφών	Αριθμός Πράσινων Κορυφών
4	**4**	**4**
Εφόσον ο αριθμός και των τριών διαφορετικών χρωμάτων είναι ο ίδιος μπορούμε να τοποθετήσουμε τις κάμερες σε οποιουδήποτε χρώματος κορυφές για να επιτύχουμε την πλήρη κάλυψή του.		

Εφαρμογή 5η .



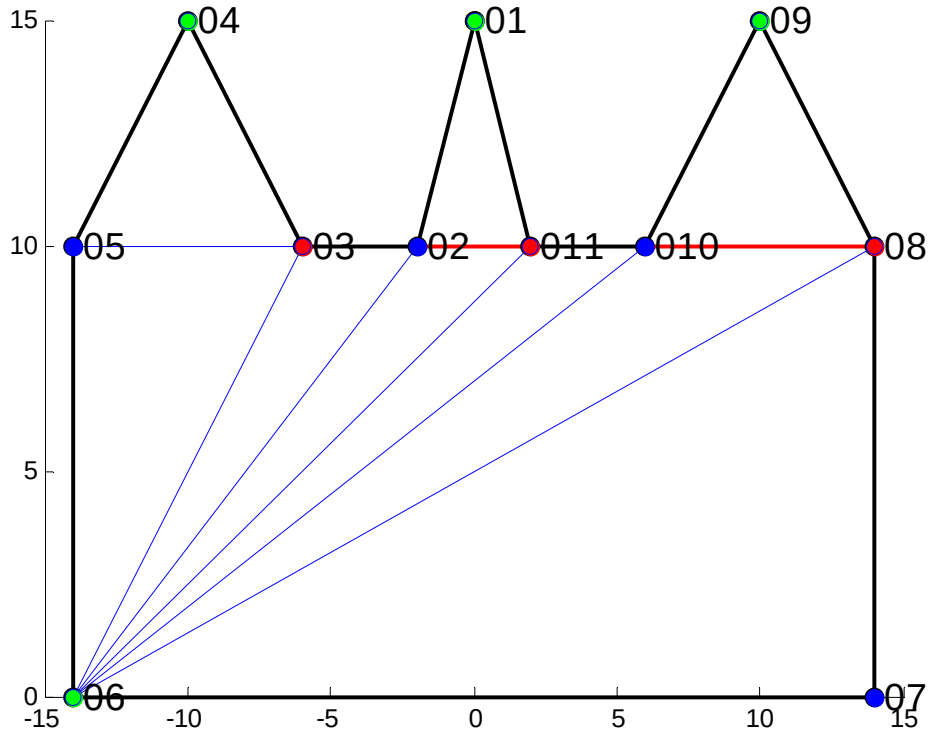
Αριθμός Μπλε Κορυφών	Αριθμός Κόκκινων Κορυφών	Αριθμός Πράσινων Κορυφών
2	**3**	**2**
Για να είναι όλα τα σημεία του πολυγώνου ορατά από τις κάμερες με την χρήση των ελάχιστων δυνατών καμερών θα πρέπει αυτές να τοποθετηθούν στις θέσεις είτε των μπλε είτε των πράσινων κορυφών.		

Εφαρμογή 6η .



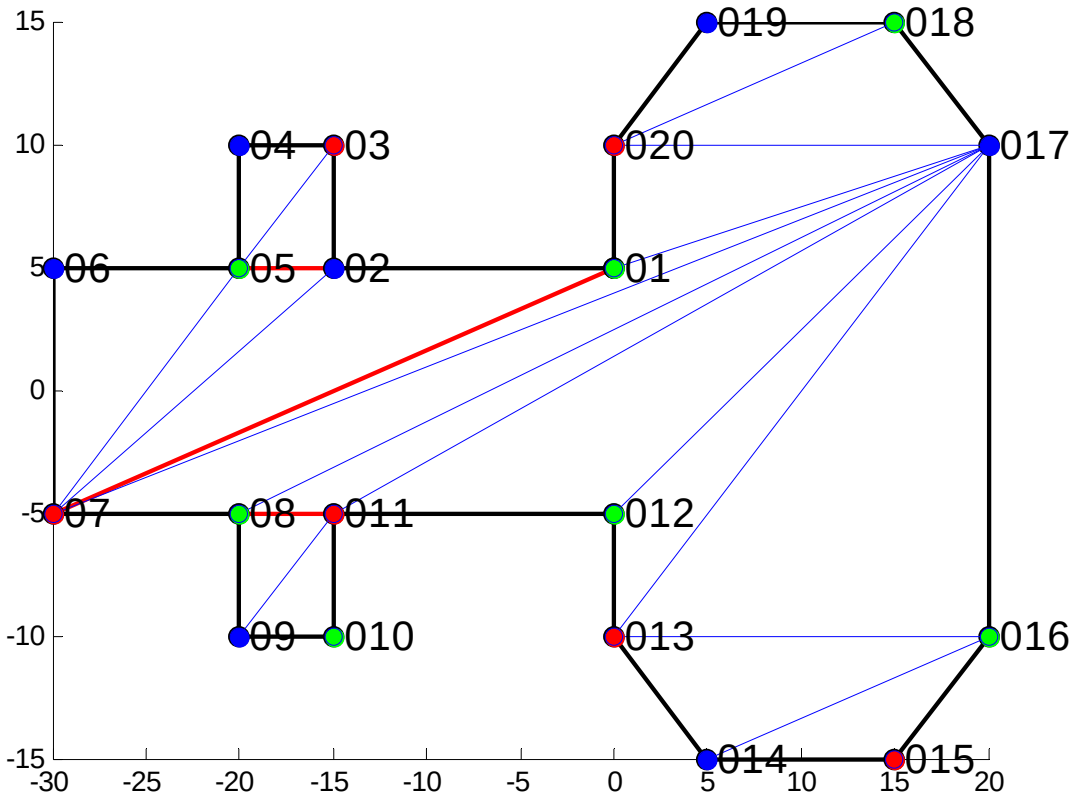
Αριθμός Μπλε Κορυφών	Αριθμός Κόκκινων Κορυφών	Αριθμός Πράσινων Κορυφών
4	**4**	**4**
Έχουμε την ελευθερία να επιλέξουμε τις θέσεις των καμερών σε οποιουδήποτε χρώματος κορυφές.		

Εφαρμογή 7η .



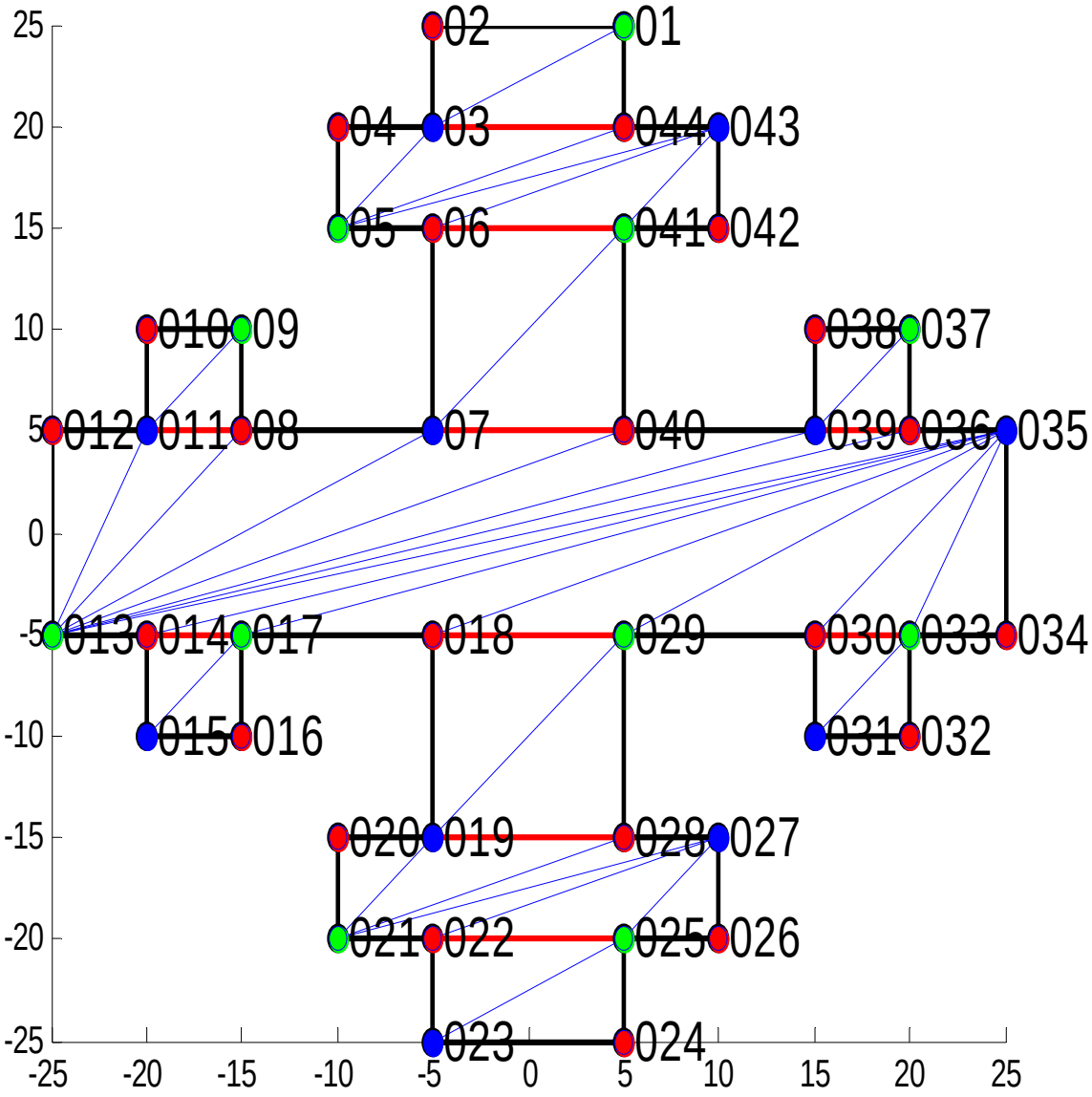
Αριθμός Μπλε Κορυφών	Αριθμός Κόκκινων Κορυφών	Αριθμός Πράσινων Κορυφών
4	**3**	**4**
Η βέλτιστη τοποθέτηση των καμερών είναι στις θέσεις των Κόκκινων κορυφών.		

Εφαρμογή 8η .



Αριθμός Μπλε Κορυφών	Αριθμός Κόκκινων Κορυφών	Αριθμός Πράσινων Κορυφών
7	**6**	**7**
Η βέλτιστη τοποθέτηση των καμερών είναι στις θέσεις των Κόκκινων κορυφών.		

Εφαρμογή 9η .



Αριθμός Μπλε Κορυφών	Αριθμός Κόκκινων Κορυφών	Αριθμός Πράσινων Κορυφών
11	**22**	**11**
Οι κάμερες φύλαξης του πολυγώνου μπορούν να τοποθετηθούν στις θέσεις των Μπλε ή των Πράσινων κορυφών.		

ΠΑΡΑΡΤΗΜΑ

Στην συνέχεια παρατίθεται ο κώδικας που αναπτύχθηκε σε MATLAB για την τριγωνοποίηση οποιουδήποτε απλού πολυγώνου. Για είσοδο που δίνουμε στην εκτελέσιμη συνάρτηση TriangulatePolygon είναι 2 .txt αρχεία. Το ένα πρέπει να περιέχει τις συντεταγμένες των κορυφών του πολυγώνου και το άλλο τον τρόπο που συνδέονται οι κορυφές ανθορολογιακά όπως προστάζει και ο αλγόριθμος. Ένα παράδειγμα των αρχείων αυτών δίνεται στην συνέχεια.

10 0
0 4
3 2
4 3
7 6
12 5
14 9
11 8
9 15
6 13
5 14
2 12
4 10
3 7
1 9.5

Στην $1^{η}$ στήλη δίνονται οι τετμημένες και με κενό οι τεταγμένες. Συνολικά στο πολύγωνο υπάρχουν 15 κορυφές. Η αρίθμησή τους είναι με βάση την σειρά που δόθηκαν στο αρχείο αυτό. Για παράδειγμα η κορυγή 10 0 είναι η $1^{η}$ και η κορυφή 1 9.5 η $15^{η}$.

(`sintetagmenes simeiwn.txt`)

Το δεύτερο αρχείο .txt περιέχει τις απαραίτητες πληροφορίες σύνδεσης την κορυφών αυτών ανθοπολογιακά (αντίθετα με την φορά του ρολογιού).

9 10 11 12 13 14 15 2 3 4 1 5 6 7 8 9 (`segments.txt`) , που σημαίνει ότι η $9^{η}$ κορυφή που δόθηκε στο πρώτο αρχείο συνδέεται με την $10^{η}$, η $10^{η}$ με την $11^{η}$ και ούτω καθ'εξής.

Η εκτελέσιμη συνάρτηση TriangulatePolygon :

```
function TriangulatePolygon

    D = ReadSubdivision( { 'sintetagmenes simeiwn.txt' 'segments.txt'
} );

    [output] = Calculate_Vertexes_Type(D);
    A = output{1};
    B = output{2};
    C = output{3};
    new_segments = output{4};

    CArray = Calculate_new_SubPolygons({C},new_segments);

    for i=1:(size(C,2)-1)
        x = A(C(i));
        y = B(C(i));
        text(x,y,strcat(' 0',num2str(C(i))),'FontSize',14);
    end

    [triangulation_array ] = Triangulate(A,B,CArray);

    point_marker_array = Insert_markers_to_triangles(
A,B,triangulation_array );
    point_marker_array = point_marker_array';

    for i=1:size(A,1)
        if point_marker_array(i) == 1
            plot(A(i),B(i),'o','MarkerFaceColor','g');
        elseif point_marker_array(i) == 2
            plot(A(i),B(i),'o','MarkerFaceColor','b');
        elseif point_marker_array(i) == 4
            plot(A(i),B(i),'o','MarkerFaceColor','r');
        else
            plot(A(i),B(i),'o','MarkerFaceColor','k');
        end
    end

    number_of_type1 = 0;
    number_of_type2 = 0;
    number_of_type3 = 0;

    for i=1:size(point_marker_array,1)
        if point_marker_array(i) == 1
            number_of_type1 = number_of_type1 + 1;
        end
        if point_marker_array(i) == 2
            number_of_type2 = number_of_type2 + 1;
        end
        if point_marker_array(i) == 4
            number_of_type3 = number_of_type3 + 1;
        end
    end

    number_of_green_vertices = number_of_type1
    number_of_blue_vertices = number_of_type2
```

```
    number_of_red_vertices = number_of_type3

end
```

Η συνάρτηση αυτή εκτυπώνει το τριγωνοποιημένο πολύγωνο, όπως φαίνεται και στο Κεφάλαιο 4, καθώς και τον αριθμό των χρωματισμών κορυφών για να επιλέξουμε εύκολα τις θέσεις όπου και θα τοποθετηθούν οι κάμερες.

Στην συνέχεια δίνουμε τον κώδικα των συναρτήσεων που δημιουργήσαμε όπου κάθε μια εκτελεί μια συγκεκριμένη λειτουργία.

Η `Calculate_new_SubPolygons` υπολογίζει τα νέα y- μονότονα υποπολύγωνα που δημιουργούνται με τις διαγωνίους που προστέθηκαν.

```
function [C_array] = Calculate_new_SubPolygons(C_array,new_segments)

    rows = size(new_segments,1);

    %Search all new segments
    for i=1:rows

        for jj=1:size(C_array,2)

            %Scan all sub C(jj) in C_array for interesecting with
specific
            %segmenet (i)
            C = C_array{jj};
            points_of_C = size(C,2);

            %Break polygon stored in array C into new smaller ones C1
and C2

            %points of specific segment (i)
            point1 = new_segments(i,1);
            point2 = new_segments(i,2);


            %
            %check if this segment's points are included in this
C%%%%%%%%%
            marker = 0;

            for j=1:(points_of_C-1)
                if C(j) == point1 || C(j) == point2
                    marker = marker + 1;
                end
            end

            %if 2 points of this segment is in C then the segment is
            %included in C and C has to be divided in sub polygons C1
and
```

```
            %C2
            if marker == 2
                do_divide = 1;
            else
                do_divide = 0;
            end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            %

            if do_divide == 0
                continue;
            end

            C1 = zeros(1);
            pointsofC1 = 0;
            C2 = zeros(1);
            pointsofC2 = 0;

            rows_of_C = size(C,2);
            marker = 0;

            %Breaking array C
            for j=1:(rows_of_C-1)

                marker_before = marker;

                if C(j) == point1 || C(j) == point2
                    marker = xor(marker,1);
                end

                if marker_before ~= marker
                    pointsofC1 = pointsofC1 + 1;
                    pointsofC2 = pointsofC2 + 1;
                    C1(pointsofC1) = C(j);
                    C2(pointsofC2) = C(j);
                    continue;
                end

                if (marker == 0)
                    pointsofC1 = pointsofC1 + 1;
                    C1(pointsofC1) = C(j);
                end
                if (marker == 1)
                    pointsofC2 = pointsofC2 + 1;
                    C2(pointsofC2) = C(j);
                end

            end

            %Insert large connecting points
            pointsofC1 = pointsofC1 + 1;
            C1(pointsofC1) = C1(1);
            pointsofC2 = pointsofC2 + 1;
            C2(pointsofC2) = C2(1);

            if pointsofC1 == 3 || pointsofC2 == 3
                continue;
```

```
            end

            %jj is which sub-C we are checking now....
            temp = size(C_array,2);
            for j=(temp+1):-1:(jj+2)
                C_array{j} = C_array{j-1};
            end

            C_array{jj} = C1;
            C_array{jj+1} = C2;

        end

    end

end
```

Η συνάρτηση `Calculate_Vertexes_Type` επεξεργάζεται τις κορυφές, βρίσκει τον τύπο κάθε μιας και αναπτύσει τον αλγόριθμο *Μετέτρεψε_Σε_Μονότονα* που παρουσιάστηκε στο Κεφάλαιο 2.

```
function [ output_args ] = Calculate_Vertexes_Type( input_args )

    A = input_args{1};
    B = input_args{2};
    C = input_args{3};

    D = size(A);
    number_of_vertexes = D(1);

    angle_array = zeros(1,number_of_vertexes);
    %angle_array_point = 0;

    new_segments = [];

    for i=1:number_of_vertexes

%        angle_array_point = angle_array_point+1;
        angle_array_point = C(i+1);

        point = C(i);
        first_point = [A(point) B(point)];

        point = C(i+1);
        second_point = [A(point) B(point)];

        if i~= number_of_vertexes
            point = C(i+2);
        else
            point = C(2);
        end
        third_point = [A(point) B(point)];

        %Transfer Coordinates
        second_point_new = [second_point(1)-first_point(1)
second_point(2)-first_point(2)];
```

```
        third_point_new = [third_point(1)-second_point(1)
third_point(2)-second_point(2)];

        [THETA1,RHO1] =
cart2pol(second_point_new(1),second_point_new(2));
        [THETA2,RHO2] =
cart2pol(third_point_new(1),third_point_new(2));

        theta1 = THETA1*180/pi;
        theta2 = THETA2*180/pi;

        while (theta1 < 0)
            theta1 = theta1 + 360;
        end

        while (theta2 < 0)
            theta2 = theta2 + 360;
        end

        theta = 180 + theta1 - theta2;

        while theta > 360
            theta = theta - 360;
        end

        while theta < 0
            theta = theta + 360;
        end

        if (theta < 180)
            angle_array(angle_array_point) = 1;
        end
        if (theta > 180)
            angle_array(angle_array_point) = -1;
        end
        if (theta == 180 || theta == 0)
            angle_array(angle_array_point) = 0;
        end

    end

    vertex_type_array = zeros(1,number_of_vertexes);

    for i=1:number_of_vertexes

        previous_point_position = i-1;
        next_point_position = i+1;

        if i==number_of_vertexes
            next_point_position = 1;
        end
        if i==1
            previous_point_position = number_of_vertexes;
        end

        current_point = C(i);
        previous_point = C(previous_point_position);
        next_point = C(next_point_position);
```

```
        y1 = B(previous_point);
        y2 = B(next_point);
        y0 = B(current_point);

        x1 = A(previous_point);
        x2 = A(next_point);
        x0 = A(current_point);

        if ((y0 < y1 || (y0 == y1 && x1 < x0)) && (y0 < y2 || (y0 ==
y2 && x2 < x0)) && angle_array(current_point) == +1)
            vertex_type_array(current_point) = 2;
        elseif ((y0 < y1 || (y0 == y1 && x1 < x0)) && (y0 < y2 || (y0
== y2 && x2 < x0)) && angle_array(current_point) == -1)
            vertex_type_array(current_point) = 5;
        elseif ((y0 > y1 || (y0 == y1 && x1 > x0)) && (y0 > y2 || (y0
== y2 && x2 > x0)) && angle_array(current_point) == +1)
            vertex_type_array(current_point) = 1;
        elseif ((y0 > y1 || (y0 == y1 && x1 > x0)) && (y0 > y2 || (y0
== y2 && x2 > x0)) && angle_array(current_point) == -1)
            vertex_type_array(current_point) = 4;
        else
            vertex_type_array(current_point) = 3;
        end

    end

    y_array = B;
    repeat = 1;
    Q = zeros(1,number_of_vertexes);
    for i=1:(number_of_vertexes)
        Q(i) = i;
    end

    while repeat == 1

        repeat = 0;

        for i=1:(number_of_vertexes-1)

            if y_array(i) < y_array(i+1)

                repeat = 1;

                temp = y_array(i+1);
                temp_point = Q(i+1);
                y_array(i+1) = y_array(i);
                y_array(i) = temp;
                Q(i+1) = Q(i);
                Q(i) = temp_point;

            end

        end

    end

    repeat = 1;
```

```
while repeat == 1

    repeat = 0;

    for i=1:(number_of_vertexes-1)

        if (y_array(i) == y_array(i+1) && A(Q(i)) > A(Q(i+1)))

            repeat = 1;

            temp = y_array(i+1);
            temp_point = Q(i+1);
            y_array(i+1) = y_array(i);
            y_array(i) = temp;
            Q(i+1) = Q(i);
            Q(i) = temp_point;

        end

    end

end

chain_array = zeros(number_of_vertexes,2);

for i=1:number_of_vertexes

    previous_point = i-1;
    next_point = i+1;

    if i==1
        previous_point = number_of_vertexes;
    end

    chain_array(C(i),1) =  C(previous_point);
    chain_array(C(i),2) =  C(next_point);
end

MAKE_MONOTONE_ARRAY = zeros(1,number_of_vertexes);
MAKE_MONOTONE_ARRAY_Helper = zeros(number_of_vertexes,2);
MAKE_MONOTONE_ARRAY_points = 0;
new_segments_number = 0;

for i=1:number_of_vertexes

    current_point = Q(i);


    if vertex_type_array(current_point) == 1

        %Insert ei in T and set helper(ei) to vi

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        for j=1:MAKE_MONOTONE_ARRAY_points
            x0 = A(current_point);
            x1 = A(MAKE_MONOTONE_ARRAY(j));
```

```
                if (x0<x1)
                    if (j == MAKE_MONOTONE_ARRAY_points)
                        j = j + 1;
                        break;
                    else
                        continue;
                    end
                end
                break;
            end

            if isempty(j)
                j=1;
            end

            for k=(MAKE_MONOTONE_ARRAY_points+1):-1:(j+1)
                MAKE_MONOTONE_ARRAY(k) = MAKE_MONOTONE_ARRAY(k-1);
                MAKE_MONOTONE_ARRAY_Helper(MAKE_MONOTONE_ARRAY(k),2)
= MAKE_MONOTONE_ARRAY_Helper(MAKE_MONOTONE_ARRAY(k),2) + 1;
            end
            MAKE_MONOTONE_ARRAY(j) = current_point;

            MAKE_MONOTONE_ARRAY_Helper(current_point,1) =
current_point;
            MAKE_MONOTONE_ARRAY_Helper(current_point,2) = j;

            MAKE_MONOTONE_ARRAY_points = MAKE_MONOTONE_ARRAY_points +
1;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        end

        if vertex_type_array(current_point) == 2
            helper =
MAKE_MONOTONE_ARRAY_Helper(chain_array(current_point,1),1);
            if vertex_type_array(helper) == 5
                new_segments_number = new_segments_number+1;
                new_segments(new_segments_number,1) = current_point;
                new_segments(new_segments_number,2) = helper;
            end


            %Delete ei-1 from
T%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            position_of_helper_in_MONOTONE_ARRAY =
MAKE_MONOTONE_ARRAY_Helper(chain_array(current_point,1),2);
            for
j=position_of_helper_in_MONOTONE_ARRAY:(MAKE_MONOTONE_ARRAY_points-1)
                MAKE_MONOTONE_ARRAY(j) = MAKE_MONOTONE_ARRAY(j+1);
                MAKE_MONOTONE_ARRAY_Helper(MAKE_MONOTONE_ARRAY(j),2)
= MAKE_MONOTONE_ARRAY_Helper(MAKE_MONOTONE_ARRAY(j),2) - 1;
            end
            MAKE_MONOTONE_ARRAY_points = MAKE_MONOTONE_ARRAY_points -
1;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


        end
```

```
        if vertex_type_array(current_point) == 3

            y0 = B(current_point);
            y1 = B(chain_array(current_point,1));
            y2 = B(chain_array(current_point,2));
            x0 = A(current_point);
            x1 = A(chain_array(current_point,1));
            x2 = A(chain_array(current_point,2));

            if ((y1 > y0) || (y1 == y0 && x1<x0) ) && ((y0 > y2) ||
(y0 == y2 && x0 < x2)) %interior of P lies to the right of vi


                helper =
MAKE_MONOTONE_ARRAY_Helper(chain_array(current_point,1),1);
                if vertex_type_array(helper) == 5
                    new_segments_number = new_segments_number+1;
                    new_segments(new_segments_number,1) =
current_point;
                    new_segments(new_segments_number,2) = helper;
                end


            %Delete ei-1 from
T%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            position_of_helper_in_MONOTONE_ARRAY =
MAKE_MONOTONE_ARRAY_Helper(chain_array(current_point,1),2);
            for
j=position_of_helper_in_MONOTONE_ARRAY:(MAKE_MONOTONE_ARRAY_points-1)
                MAKE_MONOTONE_ARRAY(j) = MAKE_MONOTONE_ARRAY(j+1);
                MAKE_MONOTONE_ARRAY_Helper(MAKE_MONOTONE_ARRAY(j),2)
= MAKE_MONOTONE_ARRAY_Helper(MAKE_MONOTONE_ARRAY(j),2) - 1;
            end
            MAKE_MONOTONE_ARRAY_points = MAKE_MONOTONE_ARRAY_points -
1;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%



            %Insert ei in T and set helper(ei) to vi

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            for j=1:MAKE_MONOTONE_ARRAY_points
                x0 = A(current_point);
                x1 = A(MAKE_MONOTONE_ARRAY(j));
                if (x0<x1)
                    if (j == MAKE_MONOTONE_ARRAY_points)
                        j = j + 1;
                        break;
                    else
                        continue;
                    end
                end
                break;
            end

            if isempty(j)
```

```
                j=1;
            end

            for k=(MAKE_MONOTONE_ARRAY_points+1):-1:(j+1)
                MAKE_MONOTONE_ARRAY(k) = MAKE_MONOTONE_ARRAY(k-1);
                MAKE_MONOTONE_ARRAY_Helper(MAKE_MONOTONE_ARRAY(k),2)
= MAKE_MONOTONE_ARRAY_Helper(MAKE_MONOTONE_ARRAY(k),2) + 1;
            end
            MAKE_MONOTONE_ARRAY(j) = current_point;

            MAKE_MONOTONE_ARRAY_Helper(current_point,1) =
current_point;
            MAKE_MONOTONE_ARRAY_Helper(current_point,2) = j;

            MAKE_MONOTONE_ARRAY_points = MAKE_MONOTONE_ARRAY_points +
1;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

            else
                for j=1:MAKE_MONOTONE_ARRAY_points
                    x0 = A(current_point);
                    x1 = A(MAKE_MONOTONE_ARRAY(j));
                    if (x0>x1)
                        break;
                    end
                end

                if isempty(j)
                    j=1;
                end


                helper =
MAKE_MONOTONE_ARRAY_Helper(MAKE_MONOTONE_ARRAY(j),1);

                if vertex_type_array(helper) == 5
                    new_segments_number = new_segments_number+1;
                    new_segments(new_segments_number,1) =
current_point;
                    new_segments(new_segments_number,2) = helper;
                end
                MAKE_MONOTONE_ARRAY_Helper(MAKE_MONOTONE_ARRAY(j),1)
= current_point;
            end

        end

        if vertex_type_array(current_point) == 4

            for j=1:MAKE_MONOTONE_ARRAY_points
                x0 = A(current_point);
                x1 = A(MAKE_MONOTONE_ARRAY(j));
                if (x0>x1)
                    break;
                end
            end
```

```
            helper =
MAKE_MONOTONE_ARRAY_Helper(MAKE_MONOTONE_ARRAY(j),1);

            new_segments_number = new_segments_number+1;
            new_segments(new_segments_number,1) = current_point;
            new_segments(new_segments_number,2) = helper;

            MAKE_MONOTONE_ARRAY_Helper(MAKE_MONOTONE_ARRAY(j),1) =
current_point;




            %Insert ei in T and set helper(ei) to vi

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            for j=1:MAKE_MONOTONE_ARRAY_points
                x0 = A(current_point);
                x1 = A(MAKE_MONOTONE_ARRAY(j));
                if (x0<x1)
                    if (j == MAKE_MONOTONE_ARRAY_points)
                        j = j + 1;
                        break;
                    else
                        continue;
                    end
                end
                break;
            end

            if isempty(j)
                j=1;
            end

            for k=(MAKE_MONOTONE_ARRAY_points+1):-1:(j+1)
                MAKE_MONOTONE_ARRAY(k) = MAKE_MONOTONE_ARRAY(k-1);
                MAKE_MONOTONE_ARRAY_Helper(MAKE_MONOTONE_ARRAY(k),2)
= MAKE_MONOTONE_ARRAY_Helper(MAKE_MONOTONE_ARRAY(k),2) + 1;
            end
            MAKE_MONOTONE_ARRAY(j) = current_point;

            MAKE_MONOTONE_ARRAY_Helper(current_point,1) =
current_point;
            MAKE_MONOTONE_ARRAY_Helper(current_point,2) = j;

            MAKE_MONOTONE_ARRAY_points = MAKE_MONOTONE_ARRAY_points +
1;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%



        end


        if vertex_type_array(current_point) == 5
```

```
            helper =
MAKE_MONOTONE_ARRAY_Helper(chain_array(current_point,1));
            if vertex_type_array(helper) == 5
                new_segments_number = new_segments_number+1;
                new_segments(new_segments_number,1) = current_point;
                new_segments(new_segments_number,2) = helper;
            end

            %Delete ei-1 from
T%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            position_of_helper_in_MONOTONE_ARRAY =
MAKE_MONOTONE_ARRAY_Helper(chain_array(current_point,1),2);
            for
j=position_of_helper_in_MONOTONE_ARRAY:(MAKE_MONOTONE_ARRAY_points-1)
                MAKE_MONOTONE_ARRAY(j) = MAKE_MONOTONE_ARRAY(j+1);
                MAKE_MONOTONE_ARRAY_Helper(MAKE_MONOTONE_ARRAY(j),2)
= MAKE_MONOTONE_ARRAY_Helper(MAKE_MONOTONE_ARRAY(j),2) - 1;
            end
            MAKE_MONOTONE_ARRAY_points = MAKE_MONOTONE_ARRAY_points -
1;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

            for j=1:MAKE_MONOTONE_ARRAY_points
                x0 = A(current_point);
                x1 = A(MAKE_MONOTONE_ARRAY(j));
                if (x0>x1)
                    break;
                end
            end

            helper =
MAKE_MONOTONE_ARRAY_Helper(MAKE_MONOTONE_ARRAY(j),1);

            if vertex_type_array(helper) == 5
                new_segments_number = new_segments_number+1;
                new_segments(new_segments_number,1) = current_point;
                new_segments(new_segments_number,2) = helper;
            end
            MAKE_MONOTONE_ARRAY_Helper(MAKE_MONOTONE_ARRAY(j),1) =
current_point;

        end



    end

    figure;
    clf;
    hold on;

    for i=1:(new_segments_number)
        figure_x = [A(new_segments(i,1)) A(new_segments(i,2))];
        figure_y = [B(new_segments(i,1)) B(new_segments(i,2))];
    plot( figure_x,figure_y,'r','LineWidth',2)
    end

    Anew = zeros(number_of_vertexes+1,1);
```

```
    Bnew = zeros(number_of_vertexes+1,1);

    for i=1:(number_of_vertexes+1)
        Anew(i) = A(C(i));
        Bnew(i) = B(C(i));
    end
    plot(Anew,Bnew,'-ko','LineWidth',2);

   output_args = {A B C new_segments};


end
```

Η συνάρτηση `FindIntersection` βρίσκει τυχόν τομές ευθυγράμμων τμημάτων που χρειάζεται κατά την διάρκεια του αλγορίθμου .

```
function [ intersection_array, intersection_array_points] =
FindIntersection(x1,y1,x2,y2,x3,y3,x4,y4, intersection_array,
intersection_array_points)

    slope12 = (y2-y1)/(x2-x1);
    slope34 = (y4-y3)/(x4-x3);

    x_intersect = (x1*slope12-x3*slope34-y1+y3)/(slope12-slope34);
    y_intersect = x_intersect*slope12 + (x1*y2-x2*y1)/(x1-x2);

    exists_already = 0;

    for i=1:intersection_array_points

        if intersection_array(i,1) == x_intersect &&
intersection_array(i,2) == y_intersect
            exists_already = 1;
            break;
        end

    end

    if x_intersect < min(x1,x2) || x_intersect > max(x1,x2)
        return;
    end

    if x_intersect < min(x3,x4) || x_intersect > max(x3,x4)
        return;
    end

    if (exists_already == 0)
        intersection_array_points = intersection_array_points+1;
        intersection_array(intersection_array_points,1) =
x_intersect;
        intersection_array(intersection_array_points,2) =
y_intersect;
```

```
    end

end
```

Η συνάρτηση `Insert_markers_to_triangles` επεξεργάζεται τα τρίγωνα που προκύπτουν από την τριγωνοποίηση του πολυγώνου.

```
function [ point_marker_array ] = Insert_markers_to_triangles(
A,B,triangulation_array )

    number_of_points = size(A,1);
    point_marker_array = zeros(1,number_of_points);
    number_of_triangles = size(triangulation_array,1);

    for i=1:number_of_triangles

        size_of_triangle = size(triangulation_array{i},2);
        triangle = triangulation_array{i};

        triangles_significant_vertexes = [];
        for j=1:(size_of_triangle-1)
            next_point = j+1;
            previous_point = j-1;
            if (j == 1)
                previous_point = size_of_triangle-1;
            end
            if (j == (size_of_triangle-1))
                next_point = 1;
            end
            if B(triangle(j)) == B(triangle(previous_point)) &&
B(triangle(j)) == B(triangle(next_point))
                continue;
            end
            if A(triangle(j)) == A(triangle(previous_point)) &&
A(triangle(j)) == A(triangle(next_point))
                continue;
            end

triangles_significant_vertexes(size(triangles_significant_vertexes,2)
+1) = j;
        end

        temp_sum = 0;

        temp_sum = temp_sum +
point_marker_array(triangle(triangles_significant_vertexes(1)));
        temp_sum = temp_sum +
point_marker_array(triangle(triangles_significant_vertexes(2)));
        temp_sum = temp_sum +
point_marker_array(triangle(triangles_significant_vertexes(3)));

        if temp_sum == 0

point_marker_array(triangle(triangles_significant_vertexes(1))) = 1;

point_marker_array(triangle(triangles_significant_vertexes(2))) = 2;

point_marker_array(triangle(triangles_significant_vertexes(3))) = 4;
```

```
        end

        if temp_sum == 1
            for j=1:3
                if
point_marker_array(triangle(triangles_significant_vertexes(j))) == 0

point_marker_array(triangle(triangles_significant_vertexes(j))) = 2;
                    break;
                end
            end
            for j=1:3
                if
point_marker_array(triangle(triangles_significant_vertexes(j))) == 0

point_marker_array(triangle(triangles_significant_vertexes(j))) = 4;
                    break;
                end
            end
        end

        if temp_sum == 2
            for j=1:3
                if
point_marker_array(triangle(triangles_significant_vertexes(j))) == 0

point_marker_array(triangle(triangles_significant_vertexes(j))) = 1;
                    break;
                end
            end
            for j=1:3
                if
point_marker_array(triangle(triangles_significant_vertexes(j))) == 0

point_marker_array(triangle(triangles_significant_vertexes(j))) = 4;
                    break;
                end
            end
        end

        if temp_sum == 4
            for j=1:3
                if
point_marker_array(triangle(triangles_significant_vertexes(j))) == 0

point_marker_array(triangle(triangles_significant_vertexes(j))) = 1;
                    break;
                end
            end
            for j=1:3
                if
point_marker_array(triangle(triangles_significant_vertexes(j))) == 0

point_marker_array(triangle(triangles_significant_vertexes(j))) = 2;
                    break;
                end
            end
        end

        if temp_sum == 3
```

```
            for j=1:3
                if
point_marker_array(triangle(triangles_significant_vertexes(j))) == 0

point_marker_array(triangle(triangles_significant_vertexes(j))) = 4;
                    break;
                end
            end
        end

        if temp_sum == 5
            for j=1:3
                if
point_marker_array(triangle(triangles_significant_vertexes(j))) == 0

point_marker_array(triangle(triangles_significant_vertexes(j))) = 2;
                    break;
                end
            end
        end

        if temp_sum == 6
            for j=1:3
                if
point_marker_array(triangle(triangles_significant_vertexes(j))) == 0

point_marker_array(triangle(triangles_significant_vertexes(j))) = 1;
                    break;
                end
            end
        end

    end

end
```

Η συνάρτηση `Triangulate` τριγωνοποιεί τα υποπολύνα που προκύπτουν από τη διαίρεση του πολυγώνου.

```
function [ Triangles ] = Triangulate( A,B,CArray)

    rows = size(CArray,2);

    Triangles = cell(1,1);
    number_of_triangles = 0;

    %Scan all the monotones subpolygons
    for i=1:rows
        %Specific subpolygom
        C = CArray{i};
        rows_of_C = size(C,2);

        %If polygon is triangle
        if size(C,2) == 4
            number_of_triangles = number_of_triangles + 1;
            for j=1:4
                Triangles{number_of_triangles,1}(j) = C(j);
            end
```

```
            continue;
        end


        u = C(1:(size(C,2)-1));
        number_of_vertexes = size(u,2);


        %Sort vertices in subpolygon from higher y-coordinates to
lower ones
        repeat = 1;

        while repeat == 1

            repeat = 0;

            for j=1:(number_of_vertexes-1)

                if B(u(j)) < B(u(j+1))

                    repeat = 1;

                    temp = u(j+1);
                    u(j+1) = u(j);
                    u(j) = temp;

                end

            end

        end

        repeat = 1;

        while repeat == 1

            repeat = 0;

            for j=1:(number_of_vertexes-1)

                if B(u(j)) == B(u(j+1)) && A(u(j)) > A(u(j+1))

                    repeat = 1;

                    temp = u(j+1);
                    u(j+1) = u(j);
                    u(j) = temp;

                end

            end

        end
        %End
Sorting%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
higher_point = u(1);
lower_point  = u(size(u,2));

chain1 =  zeros(1);
chain2 = zeros(1);
chain1_points  = 0;
chain2_points  = 0;

marker = 0;

%Dividing array C in two chains
for j=1:(rows_of_C-1)

    marker_before = marker;

    if C(j) == higher_point
        marker = xor(marker,1);
        if marker == 0
            chain1_marker = 1;
        elseif marker == 1
            chain1_marker = 2;
        end
    end
    if C(j) == lower_point
        marker = xor(marker,1);
    end

    if marker_before ~= marker
        continue;
    end

    if (marker == 0)
        chain1_points = chain1_points + 1;
        chain1(chain1_points) = C(j);
    end
    if (marker == 1)
        chain2_points = chain2_points + 1;
        chain2(chain2_points) = C(j);
    end

end

if chain1_marker == 1
    left_chain = chain1;
    right_chain = chain2;
end
if chain1_marker == 2
    left_chain = chain2;
    right_chain = chain1;
end

S = [u(1) u(2)];

clear('new_segments');
new_segments_points = 0;
new_segments = zeros(1,2);

for j=3:(number_of_vertexes-1)
```

```
            vertex_on_top_of_S = S(size(S,2));
            uj_was_found_in_chain = 0;
            vertex_on_top_of_S_was_found_in_chain = 0;

            for k=1:size(left_chain,2)
                if left_chain(k) == u(j)
                    uj_was_found_in_chain = 1;
                end
                if left_chain(k) == vertex_on_top_of_S
                    vertex_on_top_of_S_was_found_in_chain = 1;
                end
            end


            if uj_was_found_in_chain ~=
vertex_on_top_of_S_was_found_in_chain
                for k=1:(size(S,2) - 1)
                    new_segments(new_segments_points+k,1) = u(j);
                    new_segments(new_segments_points+k,2) = S(k+1);
                end
                new_segments_points = new_segments_points + size(S,2)
- 1;
                S = zeros(1,2);
                S(1) = u(j-1);
                S(2) = u(j);
            else
                last_popped_vertex = S(size(S,2));
                S = S(1:(size(S,2)-1));
                %Check Diagonals.....
                for k=size(S,2):-1:1
                    %(x1,y1) - (x2,y2) is the new diagonal to be
checked
                    x1 = A(u(j));
                    y1 = B(u(j));
                    x2 = A(S(k));
                    y2 = B(S(k));
                    IfSegmentIsInPolygon =
CheckIfSegmentIsInPolygon(A,B,C,x1,y1,x2,y2);
                    if (IfSegmentIsInPolygon == 1)
                        last_popped_vertex = S(size(S,2));
                        S = S(1:(size(S,2)-1));
                        new_segments_points = new_segments_points +
1;
                        new_segments(new_segments_points,1) = u(j);
                        new_segments(new_segments_points,2) =
last_popped_vertex;
                    end
                end
                S(size(S,2)+1) = last_popped_vertex;
                S(size(S,2)+1) = u(j);
            end


        end

        %Add diagonals from un to all stack vertices except the first
and
        %the last one

        for j=2:(size(S,2)-1)
```

```
            do_not_add_diagonal = 0;
            for jj=1:size(u,2)
                if B(u(jj)) == B(S(j)) && B(u(jj)) ==
B(u(number_of_vertexes)) && A(u(jj)) >=
min(A(S(j)),A(u(number_of_vertexes))) && A(u(jj)) <=
max(A(S(j)),A(u(number_of_vertexes)))
                    do_not_add_diagonal = 1;
                end
                if A(u(jj)) == A(S(j)) && A(u(jj)) ==
A(u(number_of_vertexes)) && B(u(jj)) >=
min(B(S(j)),B(u(number_of_vertexes))) && B(u(jj)) <=
max(B(S(j)),B(u(number_of_vertexes)))
                    do_not_add_diagonal = 1;
                end
            end
            if (do_not_add_diagonal == 1)
                continue;
            end
            new_segments_points = new_segments_points + 1;
            new_segments(new_segments_points,1) =
u(number_of_vertexes);
            new_segments(new_segments_points,2) = S(j);

        end

        for ii=1:new_segments_points
            Anew = zeros(2,1);
            Bnew = zeros(2,1);
            Anew(1) = A(new_segments(ii,1));
            Anew(2) = A(new_segments(ii,2));
            Bnew(1) = B(new_segments(ii,1));
            Bnew(2) = B(new_segments(ii,2));
            plot(Anew,Bnew,'b');
        end

        %Calculate new triangles from new segments and add them to
the
        %whole triangle array
        temp_cell = Calculate_new_SubPolygons({C},new_segments);
        temp_cell_number = size(temp_cell,2);
        for j=1:temp_cell_number
            Triangles{number_of_triangles+j,1} = temp_cell{j};
        end

        number_of_triangles = number_of_triangles + temp_cell_number;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    end


end
```

Οι συναρτήσεις `FindIfLinesIntersect` και `FindPlainIntersection` βρίσκουν αν ευθύγραμμα τμήματα τέμνονται και βρίσκουν τα σημεία τομών τους.

```
function [InteresectionFound] =
FindIfLinesIntersect(x0,y0,x3,y3,x4,y4)

    InteresectionFound = 0;

    if x3 == x4
        if y0 > min(y3,y4) && y0 < max(y3,y4) && x3 >= x0
            InteresectionFound = 1;
            return;
        else
            InteresectionFound = 0;
            return;
        end
    end

    slope = (y4-y3)/(x4-x3);

    if slope == 0
        if y0 == y3
            InteresectionFound = 1;
            return;
        else
            InteresectionFound = 0;
            return;
        end
    end

    x_intersect = x3 + (y0-y3)/slope;

    if x_intersect < min(x3,x4) || x_intersect > max(x3,x4) ||
max(x3,x4) < x0
        return;
    end
    if x_intersect < x0
        return;
    end

    InteresectionFound = 1;

end




function [intersection_found] =
FindPlainIntersection(x1,y1,x2,y2,x3,y3,x4,y4)

    intersection_found = 0;

    if (x1 == x3 && y1 == y3) || (x2 == x3 && y2 == y3) || (x1 == x4
&& y1 == y4) || (x2 == x4 && y2 == y4)
        return;
    end

    if x1 == x2
```

```
        slope = (y4-y3)/(x4-x3);
        y_intersect = x1*slope + (x3*y4-x4*y3)/(x3-x4);
        if y_intersect >= max(y1,y2) || y_intersect <= min(y1,y2)
            return;
        else
            intersection_found = 1;
            return;
        end
    end
    if x3 == x4
        slope = (y2-y1)/(x2-x1);
        y_intersect = x3*slope + (x1*y2-x2*y1)/(x1-x2);
        if y_intersect >= max(y3,y4) || y_intersect <= min(y3,y4)
            return;
        else
            intersection_found = 1;
            return;
        end
    end


    slope12 = (y2-y1)/(x2-x1);
    slope34 = (y4-y3)/(x4-x3);

    x_intersect = (x1*slope12-x3*slope34-y1+y3)/(slope12-slope34);
    y_intersect = x_intersect*slope12 + (x1*y2-x2*y1)/(x1-x2);


    if x_intersect < min(x1,x2) || x_intersect > max(x1,x2)
        return;
    end

    if x_intersect < min(x3,x4) || x_intersect > max(x3,x4)
        return;
    end

    if y_intersect < min(y1,y2) || y_intersect > max(y1,y2)
        return;
    end

    if y_intersect < min(y3,y4) || y_intersect > max(y3,y4)
        return;
    end

    intersection_found = 1;

end
```

Τέλος η συνάρτηση `CheckIfSegmentIsInPolygon` καλεί τις δύο παραπάνω συναρτήσεις και ελέγχει αν οι διαγώνιοι που προσθέτουμε βρίσκονται εντός του πολυγώνου όπως απαιτεί ο αλγόριθμος τριγωνοποίησης

```
function [ SegmentIsInPolygon ] = CheckIfSegmentIsInPolygon( A, B, C,
x1, y1, x2, y2)

    middle_point_x = (x1+x2)/2;
    middle_point_y = (y1+y2)/2;
```

```
    check_if_middle_point_is_in_polygon = 0;

    segments_number = size(C,2);

    HorizontalInteresectionFound = 0;

    for i=1:(segments_number-1)

        x3 = A(C(i));
        y3 = B(C(i));

        x4 = A(C(i+1));
        y4 = B(C(i+1));

        [temp] =
FindIfLinesIntersect(middle_point_x,middle_point_y,x3,y3,x4,y4);
        HorizontalInteresectionFound =
HorizontalInteresectionFound+temp;

    end

    if HorizontalInteresectionFound == 1
        check_if_middle_point_is_in_polygon = 1;
    end

    if check_if_middle_point_is_in_polygon == 1

        for i=1:(segments_number-1)

            x3 = A(C(i));
            y3 = B(C(i));

            x4 = A(C(i+1));
            y4 = B(C(i+1));

            InteresectionFound =
FindPlainIntersection(x1,y1,x2,y2,x3,y3,x4,y4);

            if InteresectionFound == 1
                SegmentIsInPolygon = 0;
                return;
            end

        end

    else
        SegmentIsInPolygon = 0;
        return;
    end

    SegmentIsInPolygon = 1;

end
```

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] M.de Berg , M. van Kreveld, M. Overmars, O. Schwarzkopf. Computational Geometry , Algorithms and Applications.

[2] A. Efrat and S.Har-peled. Guarding galleries and terrains. In R.A. Baeza-Yates, U.Montanari and N.Santoro,editors, IFIP TCS,volume 223 of IFIP Conference Proceedings. 2002.

[3] P.Valtr. Guarding galleries where no point sees a small area. Israel Journal of Mathematics . 1998.

[4] J.Urrutia. Art gallery and illumination problems.In J.-R. Sack and J.Urrutia, editors, Handbook of Computational Geometry. North-Holland 2000.

[5] D.T.Lee and A.K.Lin. Computational complexity of art gallery problems. 1990.

[6] S.K.Ghosh. Approximation algorithms for art gallery problems. In Proc. Canadian Inform. Process Soc.Congress.

[7] Jules Bloomenthal. An implicit surface polygonizer. In Paul Heckbert, editor , Graphics Gems IV. Academic Press,Boston.

[8] A.Fournier and D.Y.Montuno. Triangulating simple polygons and equivalent problems. ACM Trans. On Graphics.