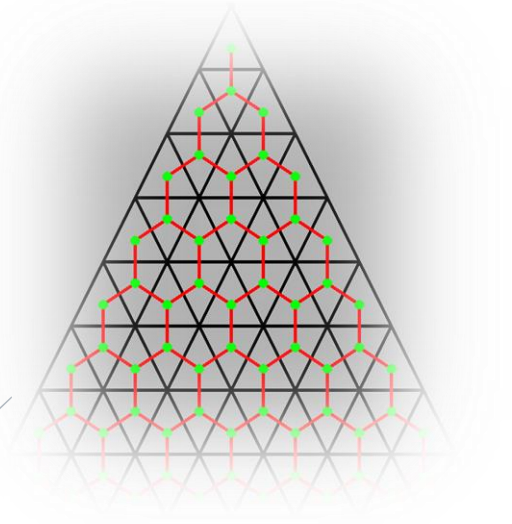
2025

3Δ ΥΠΟΛΟΓΙΣΤΙΚΗ ΓΕΩΜΕΤΡΙΑ ΚΑΙ ΟΡΑΣΗ

3η Εργαστηριακή Άσκηση



ΓΕΡΟΝΤΑΣ ΝΙΚΟΛΑΟΣ
1092813

Πίνακας περιεχομένων

Ασκήσεις 1 & 2

Υλοποιήστε τη συνάρτηση **findAdjacentTriangle**, η οποία θα ελέγχει εάν υπάρχει τρίγωνο με πλευρά τα σημεία p1, p2 που λαμβάνει ως παραμέτρους και, εάν υπάρχει, θα επιστρέφει το «κλειδί» του τριγώνου στο λεξικό στο οποίο είναι αποθηκευμένα όλα τα τρίγωνα (**tri_adj_key**) καθώς επίσης και το τρίτο, μη-κοινό σημείο (**opp_ver**).
Υπόδειξη: Τα p1 και p2 μπορεί να είναι οποιεσδήποτε από τις κορυφές v1, v2, v3 ενός τριγώνου.
Κάντε έλεγχο στα τελευταία τρία τρίγωνα που δημιουργούνται και, εάν παραβιάζουν τη συνθήκη Delaunay, τότε να χρωματίζονται με διαφορετικό χρώμα και να εμφανίζεται ο περιγεγραμμένος κύκλος του τριγώνου.

Απάντηση:

✓ Οι παραπάνω ασκήσεις υλοποιήθηκαν κατά τη διάρκεια του εργαστηρίου.

Αφέθηκε κενός χώρος για λόγους αισθητικής

Άσκηση 3η

> Υλοποιήστε το flip των τριγώνων που παραβιάζουν τη συνθήκη Delaunay. Εξετάστε **μόνο** τα 3 τρίγωνα του τρέχοντος βήματος.
> *Υπόδειξη: Για να γίνει το flip, πρέπει να διαγραφούν τα 2 παλιά τρίγωνα και να προστεθούν 2 νέα.*

Απάντηση:

Η αρχική υλοποίηση του Task 3 έγινε αυτούσια μέσα στο αντίστοιχο block της μεθόδου processPoint. Ωστόσο, κατά την υλοποίηση του Task 4, πάρθηκε η απόφαση να δημιουργηθεί η μέθοδος flip_violating_triangles, ώστε να είναι ευκολότερη η επαναχρησιμοποίηση του κώδικα.

Αναλυτικότερα, στο Task 3, για την επίλυση του προβλήματος, ελέγχουμε για κάθε τρίγωνο στη λίστα self.new_triangles {list[Triangle2D]} αν παραβιάζει τη συνθήκη Delaunay, χρησιμοποιώντας τη συνάρτηση findViolations. Στη συνέχεια, με τη βοήθεια της μεθόδου getTriangleNameByVertices, εντοπίζουμε το αντίστοιχο κλειδί του τριγώνου από τις κορυφές του.

Έτσι, όταν εισάγεται νέο σημείο, δημιουργούνται 3 νέα τρίγωνα. Αυτά εξετάζονται για πιθανή παραβίαση της συνθήκης Delaunay. Σε περίπτωση παραβίασης, εντοπίζεται το αντίστοιχο γειτονικό τρίγωνο, διαγράφονται και τα 2, και αντικαθίστανται με 2 νέα που προκύπτουν από την flip-αντικατάσταση της κοινής πλευράς!

Η διαδικασία περιορίζεται αυστηρά στα 3 νέα τρίγωνα που δημιουργούνται κατά το τρέχον βήμα, καθώς η μέθοδος εφαρμόζεται μόνο στη λίστα self.new_triangles.

✍ **Παράδειγμα:**

Άσκηση 4η

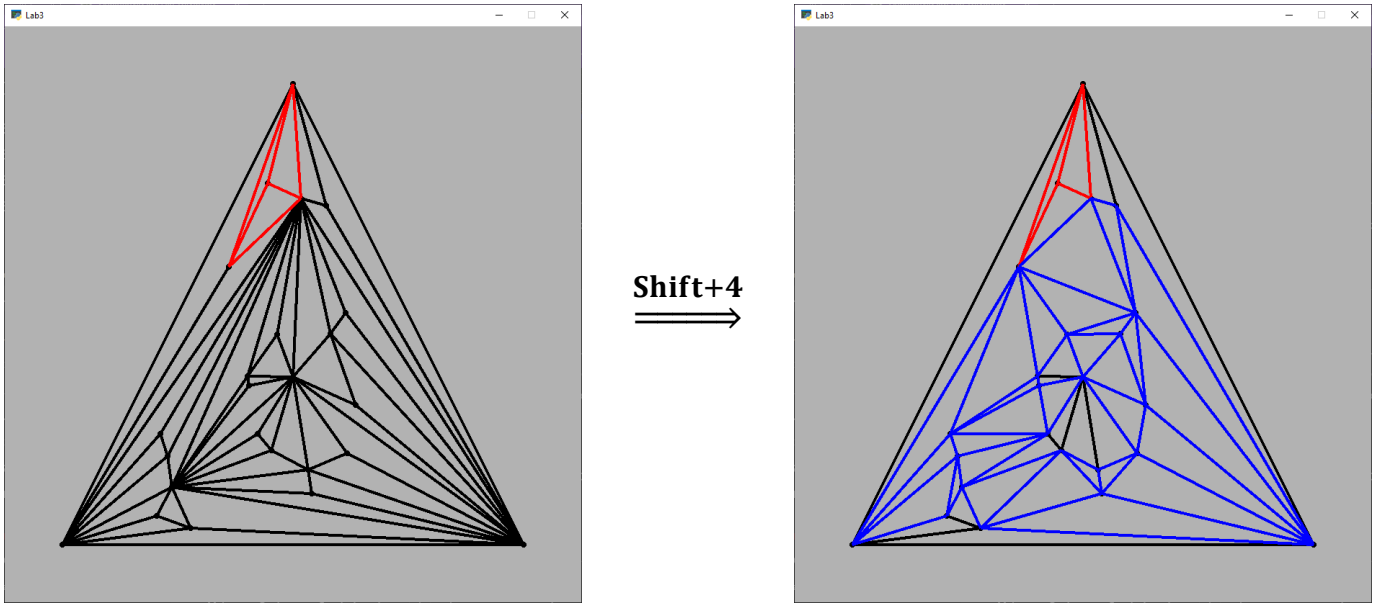
Διορθώστε τις παραβιάσεις της συνθήκης Delaunay σε ολόκληρο το επίπεδο. Εξετάστε πλέον **αναδρομικά όλα** τα τρίγωνα που υπάρχουν στο λεξικό **self.triangles**.
Υπόδειξη: Πρέπει να ελέγξετε για παραβιάσεις, να τις διορθώσετε, να ελέγξετε ξανά κ.ο.κ. Πρόκειται για μια αναδρομική διαδικασία η οποία, ωστόσο, δεν είναι ανάγκη να υλοποιηθεί και προγραμματιστικά αναδρομικά (δεν είναι απαραίτητο, δηλαδή, η συνάρτηση να καλεί τον εαυτό της).

Απάντηση:

Η υλοποίηση του task 4 επεκτείνει τη λογική του task 3, εφαρμόζοντας την αναστροφή (flip) σε όλα τα τρίγωνα της «σκηνής» που παραβιάζουν τη συνθήκη Delaunay. Η διαδικασία γίνεται επαναληπτικά: σε κάθε επανάληψη ελέγχονται όλα τα τρίγωνα για παραβιάσεις μέσω της findViolations(), και εφόσον βρεθούν παραβιασμένα τρίγωνα, εφαρμόζεται η flip_violating_triangles() σε αυτά. **Η επανάληψη συνεχίζεται έως ότου δεν υπάρχουν πλέον παραβιάσεις.** Αυτό διασφαλίζει πως το τελικό αποτέλεσμα είναι πλήρως Delaunay.

Χρησιμοποιείται η ίδια λογική με αυτή του task 3, με τη διαφορά ότι εδώ η εφαρμογή γίνεται σε όλα τα τρίγωνα, και όχι μόνο στα 3 που δημιουργούνται κατά την εισαγωγή ενός νέου σημείου.

Παράδειγμα:



Άσκηση 5η

Χρησιμοποιήστε τη συνάρτηση **findAdjacentTriangle** για να βρείτε όλα τα γειτονικά τρίγωνα κάθε τριγώνου στο **self.triangles** και υπολογίστε τα ευθύγραμμα τμήματα που ενώνουν το βαρύκεντρο του τριγώνου με αυτά των γειτονικών του, υπολογίζοντας έτσι τον δυαδικό γράφο (γράφο Voronoi στην περίπτωση Delaunay τριγωνοποίησης). Προσθέστε τα σε ένα **LineSet2D** και απεικονίστε τα με χρώμα της επιλογής σας. Ένα επιτυχές παράδειγμα του παραπάνω φαίνεται στην εικόνα παρακάτω (μπορείτε να αγνοήσετε τα ευθύγραμμα τμήματα που διαπερνούν το κυρτό περίβλημα).

Απάντηση:

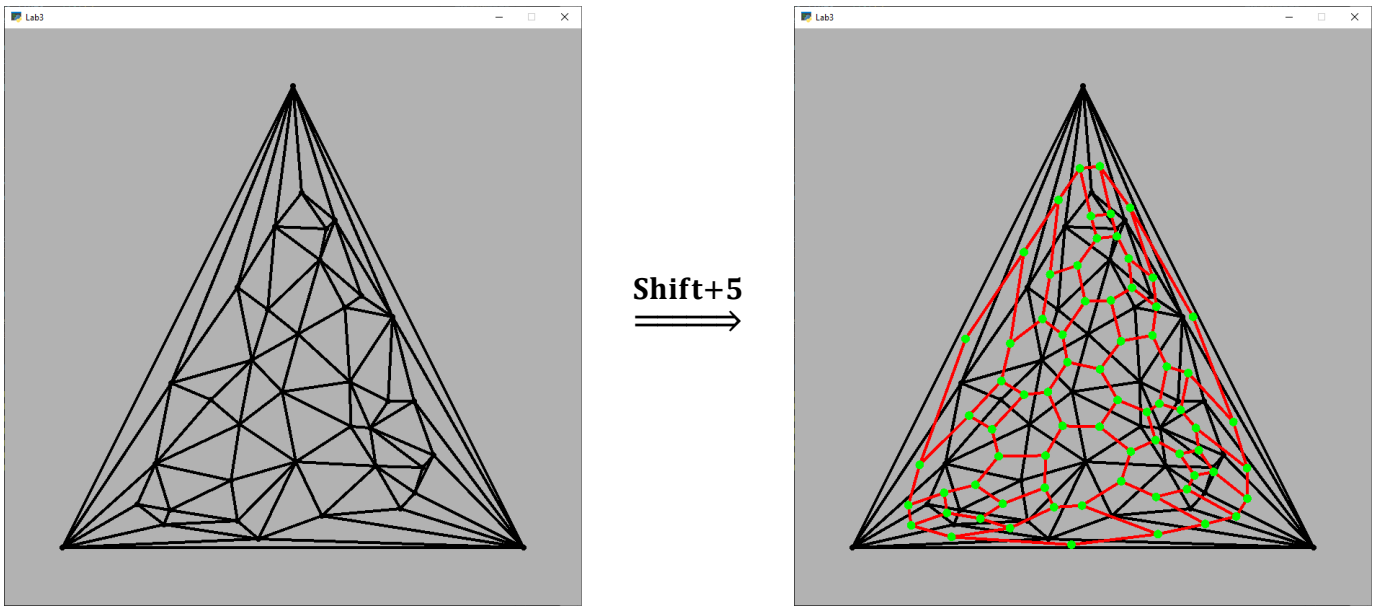
Για το task 5, υλοποιήθηκε η συνάρτηση create_voronoi_diagram(), η οποία υπολογίζει τον δυαδικό γράφο. Για κάθε τρίγωνο στη δομή self.triangles, υπολογίζεται το βαρύκεντρό του με τη συνάρτηση get_barycenter_tuple_wrapper(). Έπειτα, για κάθε πλευρά του, εντοπίζεται το γειτονικό τρίγωνο με χρήση της findAdjacentTriangle().

Αν το βαρύκεντρο του τρέχοντος τριγώνου διαφέρει από αυτό του γειτονικού, τότε προστίθεται 1 ευθύγραμμο τμήμα που τα ενώνει (ακμή του Voronoi γράφου). Για την αποφυγή διπλών ακμών, γίνεται έλεγχος ώστε να μην προστεθούν αντίστροφες εγγραφές (π.χ.: (A, B) & (B, A))!

Τα σημεία και οι ακμές επιστρέφονται ως 2 λίστες, που στη συνέχεια απεικονίζονται στη «σκηνή» με χρήση των PointSet2D και LineSet2D. Έτσι, προκύπτει ο γράφος Voronoi που αντιστοιχεί στην τρέχουσα Delaunay τριγωνοποίηση (**υπό την προϋπόθεση ότι έχει προηγηθεί η εκτέλεση του task 4 με Shift+4**).

Ο έλεγχος if barycenter2 == barycenter1: continue χρησιμοποιείται για να αποφευχθεί η περίπτωση όπου η συνάρτηση findAdjacentTriangle επιστρέφει το ίδιο το τρίγωνο αντί για κάποιο πραγματικά γειτονικό!

Παράδειγμα:



Άσκηση 6η

(Bonus/Προαιρετικό) Πραγματοποιήστε αναδρομικά κατακερματισμό του μεγαλύτερου, σε εμβαδόν, τριγώνου της υπάρχουσας τριγωνοποίησης, βρίσκοντας το μέσο των 3 πλευρών του και ενώνοντάς τα δημιουργώντας 4 νέα τρίγωνα στη θέση του αρχικού. Πραγματοποιήστε κατάλληλη επεξεργασία στα γειτονικά τρίγωνα ώστε κάθε πλευρά των τριγώνων να έχει μόνο ένα γειτονικό τρίγωνο. Δείξτε τα αποτελέσματα για 5, 10 & 50 αναδρομές.

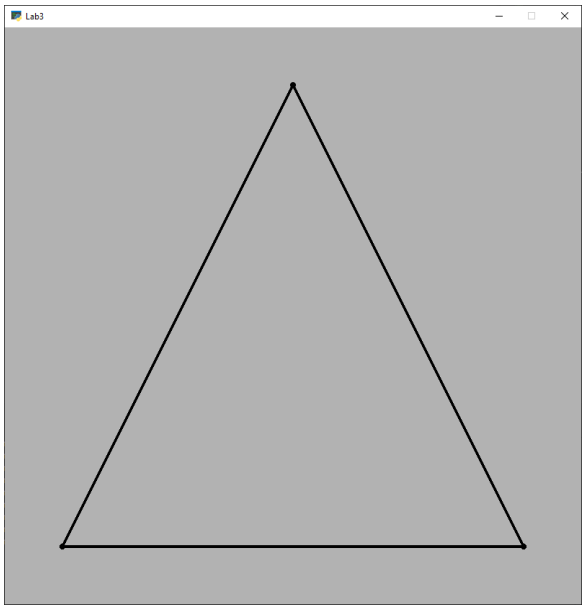
Απάντηση:

Στο πλαίσιο του task 6, πραγματοποιείται αναδρομικός κατακερματισμός του τριγώνου με το μεγαλύτερο εμβαδόν στη «σκηνή». Για τον υπολογισμό του εμβαδού κάθε τριγώνου, προστέθηκε η μέθοδος get_area() στην κλάση Triangle2D, η οποία βασίζεται στον τύπο Shoelace formula (Shoelace formula - Wikipedia) και επιστρέφει το εμβαδόν.

Αφού εντοπιστεί το τρίγωνο με το μεγαλύτερο εμβαδόν, αυτό υποδιαιρείται σε 4 μικρότερα τρίγωνα χρησιμοποιώντας τα μέσα των πλευρών του, τα οποία υπολογίζονται μέσω της βοηθητικής συνάρτησης get_medial_triangle_vertices(). Η υποδιαίρεση υλοποιείται από τη μέθοδο subdivide_triangle_into_4(), η οποία αφαιρεί το αρχικό τρίγωνο από τη «σκηνή» και προσθέτει τα 4 νέα τρίγωνα, όλα με κόκκινο χρώμα.

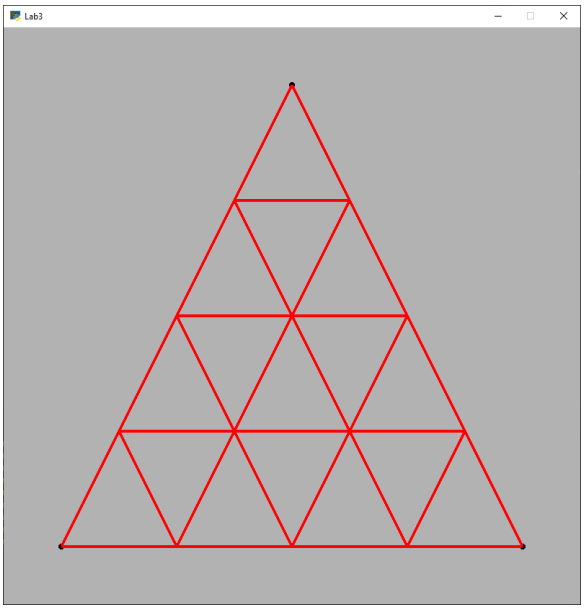
Για να επιτευχθεί το ζητούμενο της εκφώνησης {να έχει κάθε πλευρά μόνο 1 γειτονικό τρίγωνο} προστέθηκε μια **επιπλέον λογική (add-on), η οποία ενεργοποιείται με το πλήκτρο 7**. Η λογική αυτή ελέγχει επαναληπτικά όλα τα τρίγωνα και, αν εντοπίσει κάποιο που δεν πληροί την απαίτηση των γειτονικών πλευρών, το υποδιαιρεί περαιτέρω με την ίδια διαδικασία! Ο έλεγχος για μοναδικούς γείτονες βασίζεται στη συνάρτηση get_triangle_adjacency_count(), η οποία επαληθεύει ότι κάθε πλευρά ενός τριγώνου έχει το πολύ μία γειτονική.

Παραδείγματα:

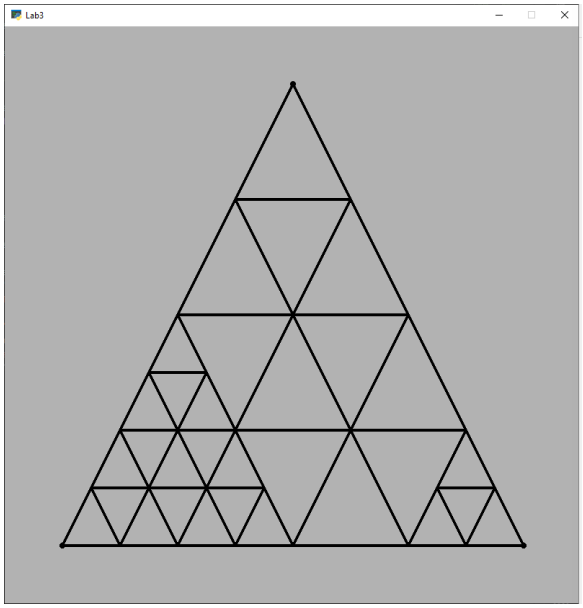


Shift+6 ⟹

για 5 αναδρομές

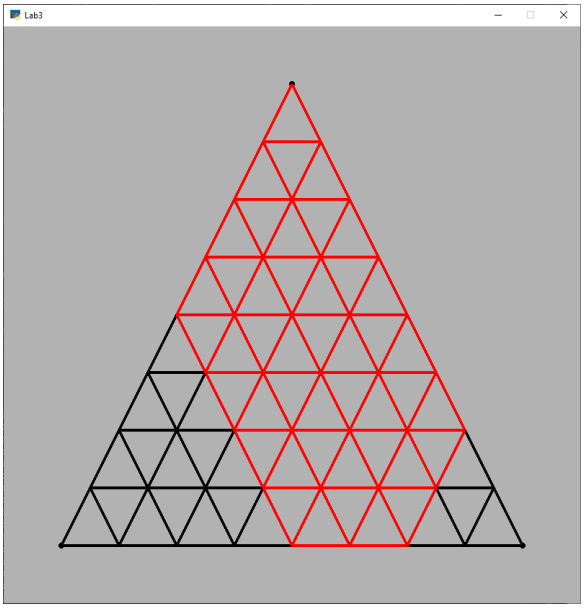


⌛ Για 10 αναδρομές:

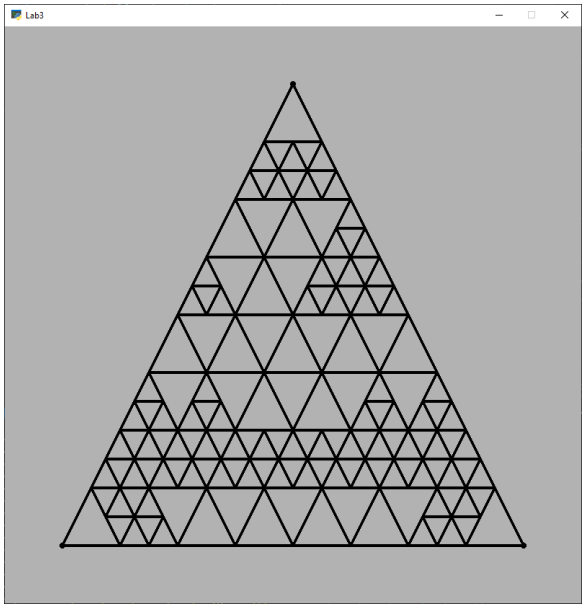


Shift+7 ⟹

χρήση του add-on

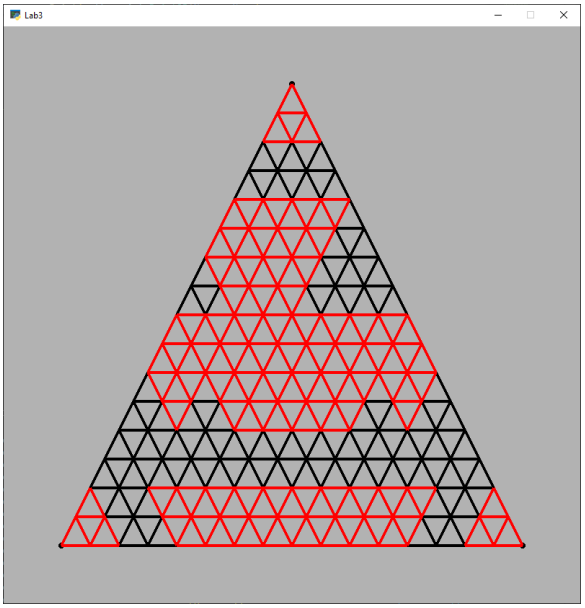


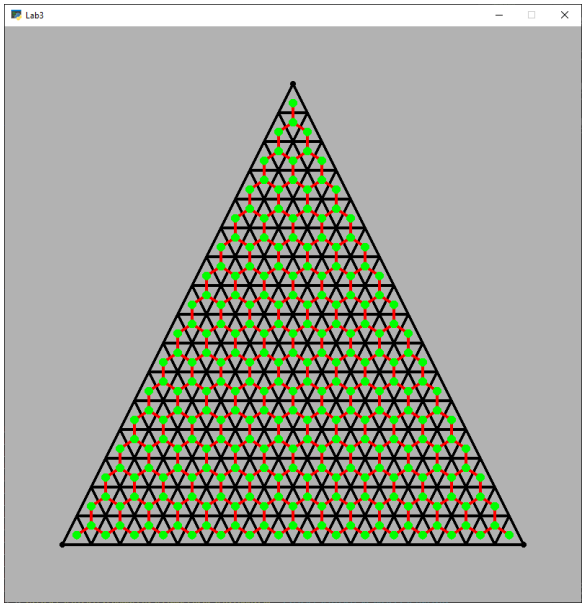
⌛ Για 50 αναδρομές:

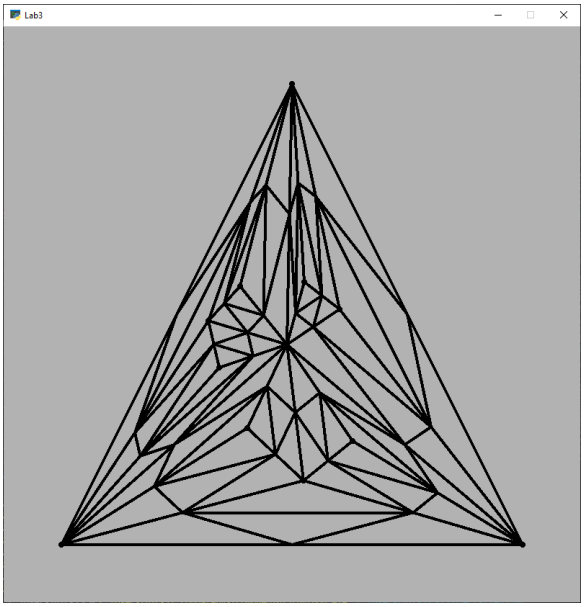


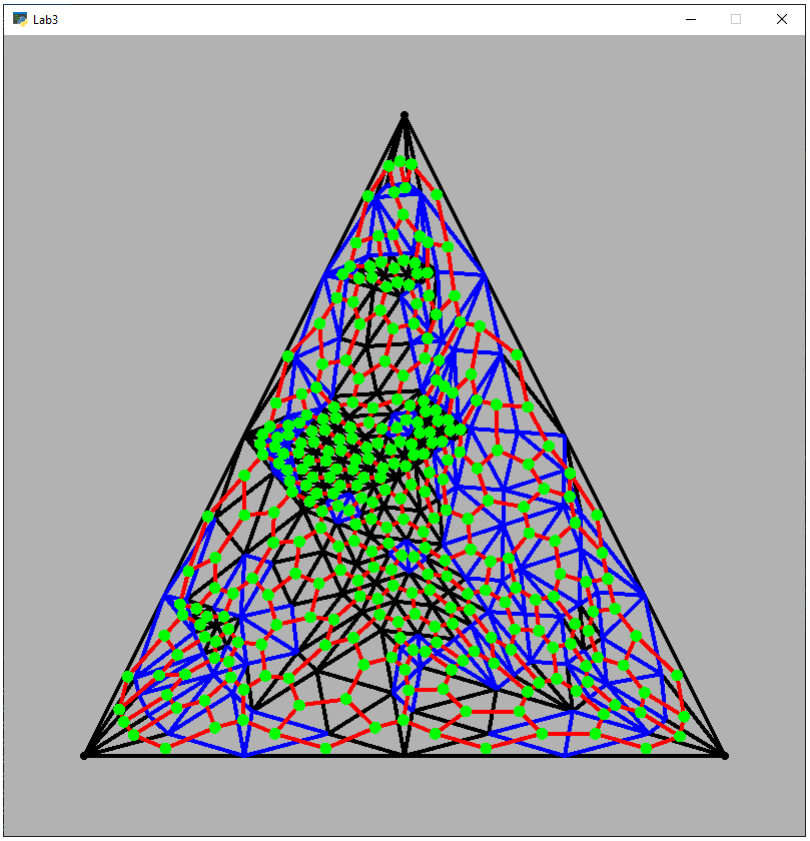
Shift+7 ⟹

χρήση του add-on




Lab3


Lab3


Lab3