README

ΣΤΟΙΧΕΙΑ ΦΟΙΤΗΤΩΝ

ΟΝΟΜΑΤΕΠΩΝΥΜΟ: ΚΥΡΙΤΣΗΣ ΧΡΗΣΤΟΣ

ΑΡΙΘΜΟΣ ΜΗΤΡΩΟΥ: 1115202100075

ΟΝΟΜΑΤΕΠΩΝΥΜΟ: ΑΝΑΓΝΩΣΤΟΠΟΥΛΟΣ ΔΗΜΗΤΡΙΟΣ

ΑΡΙΘΜΟΣ ΜΗΤΡΩΟΥ: 1115202100002

ΚΑΤΑΛΟΓΟΣ ΑΡΧΕΙΩΝ ΚΑΙ Ο ΡΟΛΟΣ ΤΟΥΣ

Το project στο εξωτερικό του αποτελείται από τους φακέλους graphics, triangulation, utils και build καθώς και από τα αρχεία CMakeLIsts.txt, firstp.cpp, input.json.

Πιο συγκεκριμένα στον φάκελο graphics εμπεριέχονται τα αρχεία graphics.cpp και graphics.h στα οποία υπάρχουν η συνάρτηση visualizePoints, η οποία είναι υπεύθυνες για την οπτικοποίηση των αποτελεσμάτων από την τριγωνοποίηση.

Στον φάκελο triangulation βρίσκονται τα αρχεία triangulation.cpp και triangulation.h τα οποία έχουν συναρτήσεις οι οποίες βοηθάνε στην επίτευξη της μη-αμβλυγώνιας τριγωνοποίησης π.χ. μετρώντας καθε φορά το πλήθος των αμβλυγώνιων τριγώνων, είτε κοιτώντας αν η εισαγωγή ενός Steiner Point θα βελτίωνε την τριγωνοποίηση κτλ. Επίσης υπάρχει και η συνάρτηση processTriangulation η οποία είναι υπεύθυνη για την τριγωνοποίηση με την βοήθεια των άλλων συναρτησεων.

Στον φάκελο utils έχουμε τα αρχεία utils.cpp και utils.h, τα οποία ο ρόλος τους είναι να διαβάζουν και να επεξεργάζονται το input αρχείο κάθε φορά και να το τροποποιούν αναλόγως έτσι ώστε σε κάθε παράδειγμα να παίρνουμε την κατάλληλη οπτικοποίηση.

Ο φάκελος build περιέχει το Makefile το οποίο δημιουργείται από την εντολή cmake ..

Στο αρχείο CMakeLists.txt υπάρχει ο κώδικας ο οποίος είναι υπεύθυνος για την χρήση της εντολής cmake .. απο την οποία παίρνουμε και το αντίστοιχο Makefile.

Το αρχείο input.json υπάρχει για να μπαίνουν τα test cases, ένα κάθε φορά.

Στο αρχείο firstp.cpp έχουμε την main η οποία με την σειρά της καλεί αρχικά της συναρτήσεις που επεξεργάζονται τα αρχεία json κάνει μια αρχική οπτικοποίηση και στην συνέχεια καλεί τη συνάρτηση για την τριγωνοποίηση.

ΟΔΗΓΙΕΣ ΜΕΤΑΓΛΩΤΙΣΗΣ

Αρχικά μπαίνουμε μέσα στον φάκελο build με την εντολή “cd build” και κάνουμε “cmake ..” για την δημιουργία του Makefile. Έπειτα, στον ίδιο φάκελο εκτελούμε την εντολή “make” η οποία μεταγλωτίζει τα προγράμματα.

ΟΔΗΓΙΕΣ ΧΡΗΣΗΣ ΤΟΥ ΠΡΟΓΡΑΜΜΑΤΟΣ

Αφού γίνει η μεταγλώτιση, τότε μέσα στον ίδιο φάκελο που είμασταν και πριν (build) τρέχουμε στο τερματικό την εντολή “./test” η οποία θα τρέξει το πρόγραμμα με είσοδο το πρώτο test. Για να τρέξουμε και τα υπόλοιπα tests, απλά αλλάζουμε τον κώδικα του input.json με τον κώδικα του test του οποίου θέλουμε να τρέξουμε.

ΠΕΡΙΓΡΑΦΗ ΤΟΥ ΠΡΟΓΡΑΜΜΑΤΟΣ

Το πρόγραμμα όπως αναφέρθηκε και παραπάνω στον κατάλογο των αρχείων αποτελείται από τους φακέλους graphics, triangulation, utils και build καθώς και από τα αρχεία CMakeLIsts.txt, firstp.cpp, input.json. Εδώ θα υπάρξει μια πιο αναλυτική περιγραφή τους.

Στο αρχείο firstp.cpp υπάρχουν αρχικά οι βιβλιοθήκες όπως η QApplication η οποία είναι υπεύθυνη για τα γραφικά του προγράμματος, η boost::json η οποία επεξεργάζεται δεδομένα σε μορφή JSON, καθώς και βιβλιοθήκες όπως graphics.h, triangulation.h και utils.h οι οποίες δημιουργήθηκαν για γραφική απεικόνιση, τριγωνοποίηση και βοηθητικές λειτουργίες αντίστοιχα. Έπειτα, καλούνται συναρτήσεις για φόρτωση των δεδομένων από τα αρχεία json όπως η loadJsonFile και αναλύονται από την συνάρτηση parseJson της boost::json βιβλιοθήκης, η οποία τα μετατρέπει σε αντικείμενο Object. Στη συνέχεια μέσω της συνάρτηση retreiveFields ανακτώνται τα πεδία όπως instance_uid, num_points, points_x, points_y, num_constraints, additional_constraints και region_boundary. Εφόσον γίνουν οι παραπάνω διαδικασίες δημιουργούνται τα σημεία ως ζεύγη συντεταγμένων και απεικονίζονται πριν την τριγωνοποίηση με την κλήση της visualizePoints και τέλος πριν τον τερματισμό του προγράμματος καλείται η CallprocessTriangulation με σκοπό την τριγωνοποίηση.

Στον φάκελο graphics υπάρχει η υλοποίηση της συνάρτησης visualizePoints, της οποίας η λειτουργία είναι να εμφανίζει σε ένα παράθυρο τα σημεία με τους περιορισμούς. Συγκεκριμένα δημιουργεί ένα αντικείμενο QScatterSeries για την αποθήκευση των σημείων και ορίζει το μέγεθος των δεικτλων για να είναι ορατά τα σημεία. Έπειτα, παίρνει τα μέγιστα και τα ελάχιστα x και y αντίστοιχα με σκοπό τα αποτελέσματα να είναι ορατά και κατανοητά ανεξαρτήτως από το μέγεθος των τιμών. Δημιουργεί ακόμα ένα αντικείμενο QLineSeries για την απεικόνιση των περιορισμών και για κάθε περιορισμό σχεδιάζει γραμμές ανάμεσα στα σημεία ελέγχοντας αν είναι εντός ορίων. Τέλος δημιουργεί ένα γράφημα, θέτει τους άξονες x και y με βάση τα ελάχιστα και μέγιστα των συντεταγμένων των σημείων, συνδέει σημεία και περιορισμούς και δημιουργεί ένα QChartView για την απεικόνιση του γραφήματος.

Στον φάκελο utils περιέχονται διάφορες βοηθητικές συναρτήσεις για την επεξεργασία JSON αρχείων και την κλήση της τριγωνοποίησης.

Η συνάρτηση printtJsonFormatted δέχεται ως είσοδο ένα αντικείμενο τύπου boost::json::object και το εκτυπώνει έτσι ώστε να είναι να είναι πιο κατανοητό στον χρήστη.

Η συνάρτηση loadJsonFile φορτώνει και αποθηκέυει τα δεδομένα από το json αρχείο που δίνεται ως είσοδος και επιστρέφει false σε πιρίπτωση μη ανοίγματος του αρχείου, και true αν επιτύχει η διαδικασία φόρτωσης.

Η συνάρτηση parseJson αναλύει τα δεδομένα JSON από την είσοδο χρησιμοποιώντας τη boost::json::parse και επιστρέφει το αντικείμενο JSON ως boost::json::object.

Η συνάρτηση retreiveFields ανακτά ορισμένα πεδία από το JSON και τα αποθηκεύει σε έτοιμες δομές της C++, όπως std::vector<int> και std::vector<std::pair<int, int>>.

Η συνάρτηση createPointsVector δημιουργεί ένα vector από ζεύγη double συντεταγμένων.

Τέλος, η συνάρτηση CallprocessTriangulation καλεί την διαδικασία της τριγωνοποίησης δημιουργώντας ένα αντικείμενο Triangulation::CDTProcessor από τα σημεία, τους περιορισμούς τα όρια και το μοναδικό αναγνωριστικό και αυτό το αντικείμενο καλεί την processTriangulation, δηλαδή την συνάρτηση τριγωνοποίησης.

Στο triangulation υπάρχει αρχικά ο Constructor του CDTProcessor που δέχεται σημεία, περιορισμούς όρια και το unique id και εισάγει πρόσθετους περιορισμούς στα όρια.

Η συνάρτηση squareDistance υπολογίζει την τετραγωνική απόσταση 2 σημείων.

Η συνάρτηση isObtuseTriangle επιστρέφει true στην περίπτωση που το τρίγωνο που δίνετε ως όρισμα είναι αμβλυγώνιο, αλλιώς false.

Η συνάρτηση countObtuseTriangles επιστρέφει το πλήθος των αμβλυγώνιων τριγώνων στην συγκεκριμένη τριγωνοποίηση.

Η συνάρτηση processTriangulation είναι αυτή που εφαρμόζει τριγωνοποίηση σε μία δοσμένη περιοχή με σκοπό την μείωση των αμβλυγώνιων τριγώνων.

Αρχικά εισάγει όλα τα απαραίτητα δεδομένα τα οποία δόθηκαν από την είσοδο και δημιουργεί μία αρχική τριγωνοποίηση Delaunay. Έπειτα μετράει το πλήθος των αμβυγώνιων τριγώνων με σκοπό την σύγκριση και αξιολόγηση της μεθόδου τριγωνοποίησης. Στη συνέχεια, κάθε αμβλυγώνιο τρίγωνο που εντοπίζεται, γίνεται έλεγχος για την αντριστροφή κάποιας ακμής με την μέθοδο edge flipping. Αν αυτό δεν γίνει, τότε εξετάζονται διάφορες στρατηγικές για την προσθήκη Steiner Point. Οι στρατηγικές αυτές είναι
εισαγωγή Steiner Point:
- Στο περίκεντρο (circumcenter)
- Στο εσώκεντρο του τριγώνου (incenter)
- Στο μέσο της μεγαλύτερης ακμής
- Στο βαρύκεντρο (centroid)
- Σε σημείο πάνω στη διχοτόμο του τριγώνου
- Στην προβολή ενός εξωτερικού σημείου στο τρίγωνο

Για κάθε μία από αυτές τις στρατηγικές χρησιμοποιείται η συνάρτηση simulateSteinerEffect η οποία κάνει μία προσωμοίωση στο τι θα συμβεί χωρίς να επηρεάζει πραγματικά την τριγωνοποίηση και η καλύτερη μέθοδος είναι αυτή που επιλέγεται για να μπει το Steiner Point, εφόσον αυτή μειώνει τα αμβλυγώνια τρίγωνα. Αυτός ο αλγόριθμος επαναλαμβάνεται για 100 επαναλήψεις ή μέχρι να μην υπάρχουν πλέον αμβλυγώνια τρίγωνα. Τέλος, αφού ολοκληρωθεί αυτή η διαδικασία, αποθηκεύονται τα αποτελέσματα και γίνεται οπτικοποίηση των αποτελεσμάτων.

Η συνάρτηση isPointInsideBoundary προσδιορίζει αν κάποιο σημείο βρίσκεται εντός ορίων του πολυγώνου και λειτουργεί ως εξής. Εκτελείται μια νοητή ευθεία από το σημείο προς τα δεξιά και μετράμε πόσες φορές αυτή η ευθεία τέμνει τις ακμές του πολυγώνου που καθορίζουν τα όρια. Αν ο αριθμός τομών είναι περιττός τότε το σημείο βρίσκεται εντός ορίων του πολυγώνου,

αλλιώς είναι εκτός. Επίσης χρησιμοποιείται και η συνάρτηση is_on_segment, δηλαδή στην περίπτωση που το σημείο βρίσκεται πάνω σε κάποια ακμή του πολυγώνου να θεωρηθεί εντός ορίων.

Η συνάρτηση isPointOnSegment κοιτάει αν κάποιο σημείο βρίσκεται πάνω στη γραμμή που ενώνει 2 σημεία.

Η συνάρτηση isPointInPolygon ελέγχει αν κάποιο σημείο βρίσκεται πάνω σε ακμή του πολυγώνου.

Η συνάρτηση isTriangleWithinBoundary ελέγχει αν ένα σημείο βρίσκεται εντός των ορίων μιας περιοχής.

Η συνάρτηση simulateSteinerEffect προσομοιώνει την επίδραση προσθήκης ενός σημείου Steiner για να υπολογίσει αν μειώνονται τα αμβλυγώνια τρίγωνα.

Η συνάρτηση getMidpointOfLongestEdge υπολογίζει τα τετράγωνα των αποστάσεων για τις ακμές p2 –p3, p1 - p3, και p1-p2 και επιστρέφει το μέσο της μεγαλύτερης από τις τρεις ακμές χρησιμοποιώντας τη συνάρτηση CGAL::midpoint

Η συνάρτηση tryEdgeFlipping εφαρμόζει την τεχνική edge flipping σε ενα τρίγωνο εντός της τριγωνοποίησης. Συγκεκριμένα ελέγχει κάθε ακμή του τριγώνου και αν αυτή δεν είναι περιορισμένη ελέγχει αν μπορεί να γίνει flip, δηλαδή αν η ακμή μπορεί να αντικατασταθεί με την εσωτερική διαγώνιο του συνδυασμένου τετραπλεύρου. Αν το τρίγωνο είναι αμβλυγώνιο και το flip είναι εφικτό, προσπαθεί να κάνει flip, καταγράφοντας την επιτυχία ή αποτυχία.

Η συνάρτηση visualizeTriangulation δημιουργεί έναν πίνακα σημείων από τα τρίγωνα της τριγωνοποίησης και τον στέλνει στη συνάρτηση visualizePoints για να τον απεικονίσει.

Η συνάρτηση calculate_incenter υπολογίζει τις αποστάσεις των πλευρών και επιστρέφει τις συντεταγμένες του σημείου με τη βοήθεια ενός αθροίσματος των συντεταγμένων των κορυφών.

Η συνάρτηση calculate_perpendicular_bisector_point υπολογίζει ένα σημείο στην κάθετη διχοτόμο της πλευράς a – b. Στην περίπτωση που η πλευρά είναι κατακόρυφη (dx == 0), η κάθετη διχοτόμος είναι οριζόντια αλλιώς υπολογίζεται η κλίση της διχοτόμου και επιστρέφεται ένα σημείο μετατοπισμένο κατά 1 μονάδα πάνω στη διχοτόμο.

Η συνάρτηση projectPointOntoTriangle προβάλλει ένα σημείο πάνω στις πλευρές ενός τριγώνου, υπολογίζει τις κάθετες προβολές του σημείου πάνω στις τρεις πλευρές του τριγώνου και επιστρέφει την πλησιέστερη προβολή.

Οι συναρτήσεις createOutputJson και printOutputJson δημιουργούν και τυπώνουν το JSON αρχείο που περιέχει τα αποτελέσματα της τριγωνοποίησης. Το JSON περιλαμβάνει τον τύπο του περιεχομένου, το uid, τις συντεταγμένες των Steiner points, και τις ακμές της τριγωνοποίησης.

ΑΛΛΕΣ ΜΕΘΟΔΟΙ ΠΟΥ ΧΡΗΣΙΜΟΠΟΙΗΘΗΚΑΝ ΑΛΛΑ ΔΕΝ ΕΙΝΑΙ ΣΤΗΝ ΤΕΛΙΚΗ ΜΟΡΦΗ

Για την εισαγωγή Steiner Points κατά τη διάρκεια υλοποίησης χρησιμοποιήθηκαν διάφορες μέθοδοι, όπως επιλογή μεθόδου Steiner Point ανάλογα με το είδος του τριγώνου (αμβλυγώνιο με πολυ μεγάλη υποτείνουσα, ισοσκελές κλπ.) Επίσης δοκιμάστηκε και η χρήση πολλαπλών Steiner Points χωρίς ιδιαίτερη επιτυχία.

Τέλος δοκιμάστηκαν και παραλλαγές της τελικής έκδοσης, όπως να κάνει 1000 επαναλήψεις στην πρώτη στρατηγική και έπειτα 1000 στην δεύτερη κ.ο.κ. Ακόμη, δοκιμάστηκαν οι στρατηγικές με διαφορετική σειρά αλλά χωρίς ιδιαίτερη επιτυχία, καθώς στο πρώτο παράδειγμα τα αμβλυγώνια παρέμεναν περισσότερα από 11 (11 = βέλτιστη λύση που καταφέραμε). Τέλος δοκιμάσαμε να αυξήσουμε και στην τωρινή προσέγγιση των αριθμό επαναλήψεων για τυχόν

καλύτερα αποτελέσματα, αλλά ο αριθμός των τριγώνων παρέμενε 11 και έτσι καταλήξαμε σε αυτή τη μέθοδο, δηλαδή έναν greedy αλγόριθμο, ο οποίος επιλέγει κάθε φορά την στρατηγική που μειώνει τα αμβλωγώνια περισσότερο στο επόμενο βήμα.

ΑΠΟΤΕΛΕΣΜΑΤΑ ΕΚΤΕΛΕΣΕΩΝ

Τεστ της εκφώνησης:

21 -> 11

Πρώτη σειρά τεστ που ανέβηκε:

instance_test_1: 10 -> 7

instance_test_2: 10 -> 7

instance_test_3: 10 ->7

instance_test_4: 2 -> 0

instance_test_5: 3 -> 1

instance_test_6: 4 -> 4

instance_test_7: 4 -> 4

instance_test_8: 5 -> 5

instance_test_9: 7 -> 5

instance_test_10: 7 -> 7

instance_test_11: 10 -> 6

instance_test_12: 5 -> 5

instance_test_13: 9 -> 9

instance_test_14: 8 -> 7

instance_test_15: 8 -> 7

instance_test_16: 6 -> 5

Δεύτερη σειρά τεστ που ανέβηκε:

Instance_1: 0 -> 0

Instance_2: 2 -> 2

Instance_3: 4 -> 0

Instance_4: 10 -> 10

Instance_5: 10 -> 4

Instance_6: 8 -> 2

Instance_7: 6 -> 2