National and Kapodistrian University of Athens

Department of Informatics and Telecommunications

Point-set Polygonization (CGAL-C++)

Authors:

PSYCHARIS EVANGELOS 1115201800217

SKORDAS CHARISIS 1115201900342

Supervisor:

DR.EMIRIS

An Assignment submitted for the NKUoA:

(DI352) Software Development for Algorithmic Problems

November, 2022

1 ***Εισαγωγή***

Αρχικά υλοποιήσαμε και τους 3 αλγορίθμους (incremental, convex_hull, onion) και τον pick σε μια πρώτη όχι ιδιαίτερα bug-free εκδοχή, σε python, καθώς υπήρχε προηγούμενη εμπειρία και κώδικας από το μάθημα Υπολογιστική Πολυπλοκότητα.

Στη συνέχεια επιλέξαμε να υλοποιήσουμε σε C++/CGAL τους αλγορίθμους incremental και convex_hull. Ωστόσο υπάρχει και ένα dummy module για τον onion σε περίπτωση που μας φανεί χρήσιμο να τον υλοποιήσουμε και αυτόν, ίσως για κάποια επόμενη εργασία. Υλοποιήσαμε και τον Pick σε C++/CGAL με έναν απλοϊκό τρόπο αλλά επιλέξαμε να μην τον χρησιμοποιήσουμε (δείτε και το 2.7 pick).

Υλοποιήσαμε επίσης ένα πρόγραμμα σε python το οποίο από διάφορα αρχεία, που παράγουν κατά την εκτέλεση τους οι αλγόριθμοι, δημιουργούν οπτικοποιήσεις (περισσότερα στο 5.3 vis_app.py)

Σας στέλνουμε μαζί και ένα μικρό video με μια ενδεικτική χρήση του προγράμματος οπτικοποίησης.

Στις υλοποιήσεις των incremental και convex_hull, σε διάφορα σημεία, υπάρχει κώδικας σχετικός με οπτικοποίηση. Αυτός συνοδεύεται από σχόλια και if statements με την λέξη ή μεταβλητή vis. Ο κώδικας αυτός δεν εκτελείται εκτός και αν στην κλήση του ./to_polygon προστεθεί η παράμετρος -vis <1/2>, 1 για πλήρη οπτικοποίηση (κάθε βήμα), 2 για σημεία και τελική πολυγωνική γραμμή.

Τον κώδικα που σας στέλνουμε μπορείτε επίσης να τον βρείτε στο github στο repository:

https://github.com/Poympas/project-2022

Το παραδοτέο είναι ο φάκελος final_no_cmake:

https://github.com/Poympas/project-2022/tree/main/final_no_cmake

Οι υπόλοιποι φάκελοι περιέχουν παλαιότερο ή δοκιμαστικό κώδικα.

2 *Αρχεία κώδικα και περιγραφή*

Στο παραδοτέο θα βρείτε την main.cpp το CMakeLists, και το script call_cmake.sh. Θα βρείτε επίσης τον φάκελο includes που περιέχει υποφακέλους, έναν για κάθε module που δημιουργήσαμε. Κάθε υποφάκελος περιέχει μια κεφαλίδα (.hpp), ένα αρχείο πηγαίου κώδικα (.cpp) και το αντίστοιχο CMakeLists. Παρακάτω παραθέτουμε μια σύντομη περιγραφή για κάθε module και τις συναρτήσεις που περιέχει. Πιο αναλυτική περιγραφή μπορείτε να βρείτε στον σχολιασμό του κώδικα σε κάθε αρχείο.

2.1 cgal_config

Στο header γίνεται η επιλογή και το include του kernel της CGAL και περιέχονται typedefs για διάφορους τύπους της CGAL που χρησιμοποιήσαμε (π.χ. Point_2, Segment_2, ...). Επιλέξαμε τον kernel Exact_predicates_inexact_constructions_kernel διότι επαρκεί για τους αλγορίθμους που κατασκευάσαμε. Γίνεται επίσης typedef του long long int ως NUM. Ο τύπος NUM χρησιμοποιείται κυρίως για την αποθήκευση εμβαδών. Αρχικά δοκιμάσαμε int αλλά προέκυπταν μεγαλύτεροι αριθμοί. Το αρχείο .cpp δεν περιέχει κάποιον κώδικα, απλά χρειάζεται για το cmake.

2.2 io_manip

Το module περιέχει συναρτήσεις για τον χειρισμό του input και του output (input output manipulation). Συγκεκριμένα περιέχει τις παρακάτω συναρτήσεις.

io_manip::read_data

Διαβάζει από ένα αρχείο με ένα σύνολο σημείων τα σημεία και το εμβαδόν του κυρτού τους περιβλήματος. Το αρχείο υποθέτουμε ότι έχει την μορφή που έχουν αυτά που μας δόθηκαν στον φάκελο instances. Δηλαδή, πρώτη γραμμή τίτλος, δεύτερη γραμμή εμβαδόν κυρτού περιβλήματος και στις υπόλοιπες γραμμές τα σημεία, ένα ζευγάρι συντεταγμένων σε κάθε γραμμή.

io_manip::process input

Από τα argc και argv ελέγχει αν έχει γίνει σωστή κλήση του to_polygon και εξάγει το αρχείο εισόδου και εξόδου, τον αλγόριθμο και τις επιλογές που δόθηκαν για τον αλγόριθμο.

io_manip::create_output

Με είσοδο διάφορα στοιχεία όπως η πολυγωνική γραμμή, το εμβαδόν του πολυγώνου και τον χρόνο εκτέλεσης, δημιουργεί ένα string για να γραφτεί στο αρχείο εξόδου με την μορφή που ζητείται στην εκφώνηση.

ΠΑΡΑΤΗΡΗΣΗ : Στο output περιλαμβάνουμε επιπλέον το εμβαδόν που υπολογίζει η CGAL για την πολυγωνική γραμμή (cgal_area) για σύγκριση με το εμβαδόν που υπολογίζουμε κατά την εκτέλεση του αλγορίθμου. Οι δύο αριθμού πρέπει να είναι ίδιοι. Περιλαμβάνουμε και το pick_calculated_area με τιμή πάντα -1 γιατί όπως είπαμε δεν τον χρησιμοποιήσαμε. Επίσης εμφανίζουμε τα πεδία is simple (με συνάρτηση της CGAL), points και points on poly. Όλα αυτά τα επιπλέον πεδία βοηθήσανε στο debugging και είναι χρήσιμες ενδείξεις για την σωστή εκτέλεση του αλγορίθμου.

io_manip::save_points_to_file

Σώζει τα σημεία που δόθηκαν σε ένα αρχείο. Χρησιμοποιείται για οπτικοποίηση.

io_manip::save_points_and_vis_to_file

Σώζει τα σημεία και μια τιμή bool για κάθε σημείο στο αρχείο που δόθηκε. Χρησιμοποιείται για οπτικοποίηση.

io_manip::print_points

Εκτυπώνει τα δοσμένα σημεία. Χρησιμοποιήθηκε κυρίως για debugging.

2.3 visibility

Το module περιέχει συναρτήσεις σχετικές με την ορατότητα, όπως για τον έλεγχο ορατότητας ακμής από σημείο δεδομένης μιας πολυγωνικής γραμμής.

visibility::choose_index

Η συνάρτηση παίρνει ως είσοδο μια λίστα με εμβαδά τριγώνων και μια λίστα με bools ορατότητας που αντιστοιχούν σε ζευγάρια σημείων και ακμών. Παίρνει ακόμα έναν τρόπο επιλογής ακμής (τυχαία, ελάχιστο/μέγιστο εμβαδόν). Στη συνέχεια διαλέγει και επιστρέφει την θέση στη λίστα ενός ζευγαριού που η ακμή είναι ορατή από το σημείο με βάση την επιλεγμένη στρατηγική.

visibility::are_intersecting

Ελέγχει αν δύο ακμές τέμνονται.

visibility::is_visible_p_from_e

Ελέγχει αν μια ακμή είναι ορατή από ένα σημείο δεδομένης μιας πολυγωνικής γραμμής που μπορεί να εμποδίζει την ορατότητα. Ο έλεγχος γίνεται ελέγχοντας αν τρία ευθύγραμμα τμήματα τέμνονται με οποιαδήποτε από τις ακμές της πολυγωνικής γραμμής. Αν δεν υπάρχει καμία τομή, η ακμή είναι ορατή. Τα τρία ευθύγραμμα τμήματα είναι (σημείο, αρχή ακμής), (σημείο,μέσο ακμής), (σημείο, τέλος ακμής).

2.4 poly_from_ch

Το module περιέχει μια υλοποίηση του αλγορίθμου convex_hull για την εύρεση μιας πολυγωνικής γραμμής.

poly_from_ch::find_inner_points

Από δύο σύνολα σημείων βρίσκει ποια ανήκουν στο πρώτο και όχι στο δεύτερο. Την χρησιμοποιούμε με πρώτο σύνολο όλα τα σημεία και δεύτερο το κυρτό περίβλημα, οπότε ουσιαστικά υπολογίζονται τα εσωτερικά σημεια.

poly_from_ch::point_closest_to_edge

Από μια ακμή και ένα σύνολο σημείων βρίσκει την κοντινότερη ακμή στο σημείο.

poly_from_ch::remove_point

Διαγράφει ένα σημείο από ένα σύνολο σημείων.

poly_from_ch::run

Εκτελεί τον αλγόριθμο convex hull στα δοσμένα σημεία με την δοσμένη στρατηγική επιλογής ακμής. Πρώτα βρίσκουμε το κυρτό περίβλημα, αρχικοποιούμε με αυτό την πολυγωνική γραμμή και διακρίνουμε τα εσωτερικά σημεία. Σε κάθε βήμα βρίσκουμε τα κοντινότερα σε κάθε ακμή σημεία και επιλέγουμε να προσθέσουμε ένα με βάση το εμβαδόν του τριγώνου που δημιουργεί με την αντίστοιχη ακμή ή τυχαία (ανάλογα με την στρατηγική επιλογής ακμής).

ΠΑΡΑΤΗΡΗΣΗ : Σε κάθε βήμα δεν ξανα υπολογίζουμε όλα τα κοντινότερα σημεία. Αυτά είναι αποθηκευμένα σε μια λίστα και κάθε φορά υπολογίζουμε τα κοντινότερα σημεία μόνο για τις ακμές που είχαν ως κοντινότερο το σημείο που μπήκε στην πολυγωνική γραμμή στο προηγούμενο βήμα. Αρχικά κάναμε τον υπολογισμό για όλα τα σημεία αλλά με αυτή την αλλαγή ο αλγόριθμος έγινε πάρα πολύ πιο γρήγορος. Γενικά φαίνεται να λειτουργεί αλλά όχι πάντα όπως θα δούμε στον σχολιασμό των αποτελεσμάτων.

2.5 poly_incremental

Το module περιέχει μια υλοποίηση του αλγορίθμου incremental για την εύρεση μιας πολυγωνικής γραμμής.

poly_incremental::bb_inc_step

Εκτελεί ένα βήμα του αυξητικού αλγορίθμου. Από το τελευταίο κυρτό περίβλημα, πολυγωνική γραμμή και σημείο που μπήκε σε αυτή, υπολογίζει τα επόμενα και το νέο εμβαδόν της πολυγωνικής γραμμής. Το βήμα γίνεται διαλέγοντας μια κόκκινη ακμή. Δηλαδή μια ακμή πάνω στην πολυγωνική γραμμή που βρίσκεται πίσω από τις ακμές του προηγούμενου κυρτού περιβλήματος, οι οποιες είναι ορατές από το νέο σημείο.

poly_incremental::run

Εκτελεί τον αλγόριθμο incremental στα δοσμένα σημεία με την δοσμένη στρατηγική επιλογής ακμής και αρχικοποίησης. Ξεκινάει με το πιο ακραίο τρίγωνο βάση της αρχικοποίησης και προσθέτει σημεία καλώντας την bb_inc_step μέχρι να εξαντληθούν τα σημεία.

2.6 poly_onion

Δεν έχει γίνει υλοποίηση του αλγορίθμου. Πρόκειται για ένα dummy module σε περίπτωση μελλοντικής υλοποίησης.

2.7 pick

Το module περιέχει μια υλοποίηση του αλγορίθμου pick για την εύρεση εμβαδού πολυγώνου.

pick::run

Καλεί τον αλγόριθμο pick. Κάναμε από περιέργεια μια απλή υλοποίηση του αλγορίθμου, αν και είμασταν ομάδα δύο ατόμων και δεν ήταν ζητούμενο. Εν τέλει δεν τον χρησιμοποιήσαμε γιατί στην απλή παρούσα μορφή του - έλεγξε για όλα τα σημεία στο bounding box του πολυγώνου αν είναι εντός του πολυγώνου ή πάνω σε αυτό - είναι πολύ αργός.

2.8 poly_line_algorithms

Το header περιέχει συναρτήσεις για την κλήση των αλγορίθμων πολυγωνοποίησης και του αλγορίθμου pick. Το αρχείο .cpp δεν περιέχει κάποιον κώδικα, απλά χρειάζεται για το cmake.

ΠΑΡΑΤΗΡΗΣΗ : Αντιμετωπίσαμε προβλήματα στο linking, συγκεκριμένα με το visibility.hpp. Για αυτό τον λόγο κάνουμε εδώ include το visibility.cpp. Γνωρίζουμε ότι δεν είναι καλή πρακτική αλλά δεν βρήκαμε διαφορετικό τρόπο να γίνει σωστά το linking, παρότι κάνουμε include το visibility.hpp όπου αυτό χρειάζεται.

poly_line_algorithms::incremental

Καλεί τον αλγόριθμο incremental.

poly_line_algorithms::convex_hull

Καλεί τον αλγόριθμο convex_hull.

poly_line_algorithms::onion

Καλεί τον αλγόριθμο onion (σε περίπτωση μελλοντικής υλοποίησης).

poly_line_algorithms::pick

Καλεί τον αλγόριθμο pick.

2.9 main

Περιέχει την βασική εκτέλεση του προγράμματος:

1. Διαβάζει, ελέγχει και εξάγει το input από την κλήση ./to_polygon.
2. Διαβάζει τα σημεία και το εμβαδόν του κυρτού πολυγώνου από το αρχείο εισόδου.
3. Καλεί και χρονομετρά τον αλγόριθμο με τις δοσμένες παραμέτρους.
4. Γράφει στο αρχείο εξόδου τα αποτελέσματα της εκτέλεση.

3 ***Οδηγίες μεταγλώττισης και χρήσης***

Οδηγίες μεταγλώττισης:

1. chmod 777 cal_cmake.sh
2. ./call_cmake.sh (ή cmake -DCGAL_DIR=/usr/local/CGAL-5.5.1 .)
3. make

Για την εκτέλεση καλέστε την to_polygon όπως αναφέρεται στην εκφώνηση.

Για τον αλγόριθμο convex_hull:

./to_polygon -i <input_file> -o <output_file> -algorithm convex_hull -edge_selection <1/2/3>

Για τον αλγόριθμο incremental:

./to_polygon -i <input_file> -o <output_file> -algorithm incremental -edge_selection <1/2/3>
-initialization <1a/1b/2a/2b>

4 Σχολιασμός αποτελεσμάτων

algorithm	incremental																				convex_hull					
edge_selection	1								2								3								1		2		3	
initialization	1a		1b		2a		2b		1a		1b		2a		2b		1a		1b		2a		2b		-		-		-	
ratio, time	r	t	r	t	r	t	r	t	r	t	r	t	r	t	r	t	r	t	r	t	r	t	r	t	r	t	r	t	r	t
Uniform-100	0.44	88	0.57	213	-	-	0.47	209	0.32	93	0.33	146	-	-	0.26	321	0.71	168	0.68	132	0.68	134	0.70	173	0.54	135	0.80	124	0.31	140
Euro-night-100	0.56	148	0.53	177	0.60	166	0.58	207	0.22	99	0.25	106	0.31	270	0.31	123	0.76	276	0.77	171	0.81	201	0.77	219	0.58	167	0.90	168	0.28	111
Us-night-100	0.47	102	0.45	128	0.59	190	0.48	128	0.20	139	0.22	106	0.30	142	0.25	167	0.69	139	0.84	247	0.76	182	0.78	148	0.53	168	0.89	167	0.30	154
London-100	0.52	138	0.45	127	0.45	233	0.53	161	0.22	114	0.26	125	0.33	130	0.24	119	0.75	281	0.75	276	0.75	196	0.71	265	0.56	102	0.86	109	0.25	126
Stars-100	0.55	113	0.51	141	0.57	215	0.50	169	0.32	87	0.25	122	0.25	128	0.27	108	0.69	126	0.70	180	0.68	168	0.71	147	0.49	153	0.83	192	0.32	116
Euro-night-200	0.50	423	0.49	488	0.53	1407	0.53	934	0.31	461	0.24	349	0.23	493	0.24	686	0.76	1168	0.71	687	0.76	1536	0.75	1050	0.50	929	0.89	828	0.25	903
Uniform-200	0.46	339	0.54	511	0.54	463	0.50	801	0.26	587	0.26	497	0.29	599	0.32	657	0.69	485	0.70	484	0.72	1065	0.68	601	0.52	576	0.84	600	0.31	525
Us-night-200	0.53	785	0.47	603	0.47	712	0.51	1059	0.20	449	0.20	472	0.22	583	0.26	749	0.78	1102	0.71	1135	0.78	1115	0.80	969	0.63	794	0.90	1032	0.28	644
Paris-200	-	-	0.48	434	0.50	536	0.45	630	-	-	0.21	433	0.25	533	0.25	624	-	-	0.72	894	0.71	835	0.70	835	0.55	558	0.87	656	0.25	543
Stars-200	0.49	603	0.47	312	0.52	502	0.56	1135	0.26	461	0.27	397	0.29	714	0.28	659	0.68	752	0.69	801	0.77	1633	0.74	1031	0.52	793	0.86	635	0.25	598
Euro-night-500	0.45	6322	0.46	2283	0.48	6717	0.46	6842	0.19	4186	0.20	2985	0.17	6199	0.21	3898	0.75	15585	0.73	6144	0.73	13421	0.76	14633	0.46	5416	0.88	5179	0.23	5362
Us-night-500	0.41	5481	0.45	3480	0.46	5944	0.53	4961	0.13	4458	0.14	5443	0.18	10872	0.18	5061	-	-	0.75	17757	0.78	4604	0.75	9890	0.43	7150	0.91	4118	0.24	6192
Paris-500	0.45	4567	0.46	2190	0.47	4151	0.48	4641	0.24	3603	0.22	2544	0.23	4999	0.26	4555	0.72	3027	0.70	6531	0.71	4625	0.71	6161	0.50	4503	0.85	3865	0.27	3877
Uniform-500	0.51	1732	0.52	3266	-	-	0.51	4094	0.30	2785	0.29	2214	-	-	0.30	4108	0.70	3228	0.69	3416	-	-	0.71	3966	0.57	4702	0.82	3755	0.27	4548
Stars-500	0.45	4227	0.50	2171	0.46	2575	0.49	2933	0.22	4384	0.24	4949	0.25	4513	0.26	5181	0.70	5798	0.72	7638	0.70	6044	0.73	5872	0.55	4225	0.85	4079	0.27	3485
Euro-night-1000	0.49	24265	0.45	22907	0.41	27535	0.43	17414	-	-	-	-	-	-	-	-	0.70	29063	0.77	72763	0.71	37955	0.71	78460	0.58	30089	-	-	-	-
Us-night-1000	0.44	25324	0.41	17121	-	-	-	-	0.19	18069	0.13	20994	-	-	-	-	0.73	71860	0.72	89137	0.74	43458	0.80	51296	-	-	-	-	0.22	16671
Paris-1000	0.42	11811	0.46	15068	0.51	23226	0.47	14978	0.21	12165	0.21	11842	-	-	-	-	0.69	32305	0.70	19900	0.72	25029	0.70	29437	0.52	22489	0.87	25688	0.26	15943
Skylake-1000	0.48	12338	0.49	17600	0.50	13247	0.52	26996	0.27	12230	0.26	13377	0.29	14512	0.30	16868	0.71	19895	0.69	18291	0.70	19762	0.70	23440	0.53	26290	0.82	15394	0.31	22536
Uniform-100	0.53	14231	0.50	8483	0.49	21042	0.51	20044	0.27	12813	0.26	11867	0.27	17027	0.29	14970	0.69	14832	0.68	17821	0.69	21484	0.70	20605	0.54	24080	0.82	18545	0.30	16573
average area	0.48		0.48		0.50		0.50		0.24		0.23		0.26		0.26		0.72		0.72		0.73		0.73		0.53		0.86		0.27	

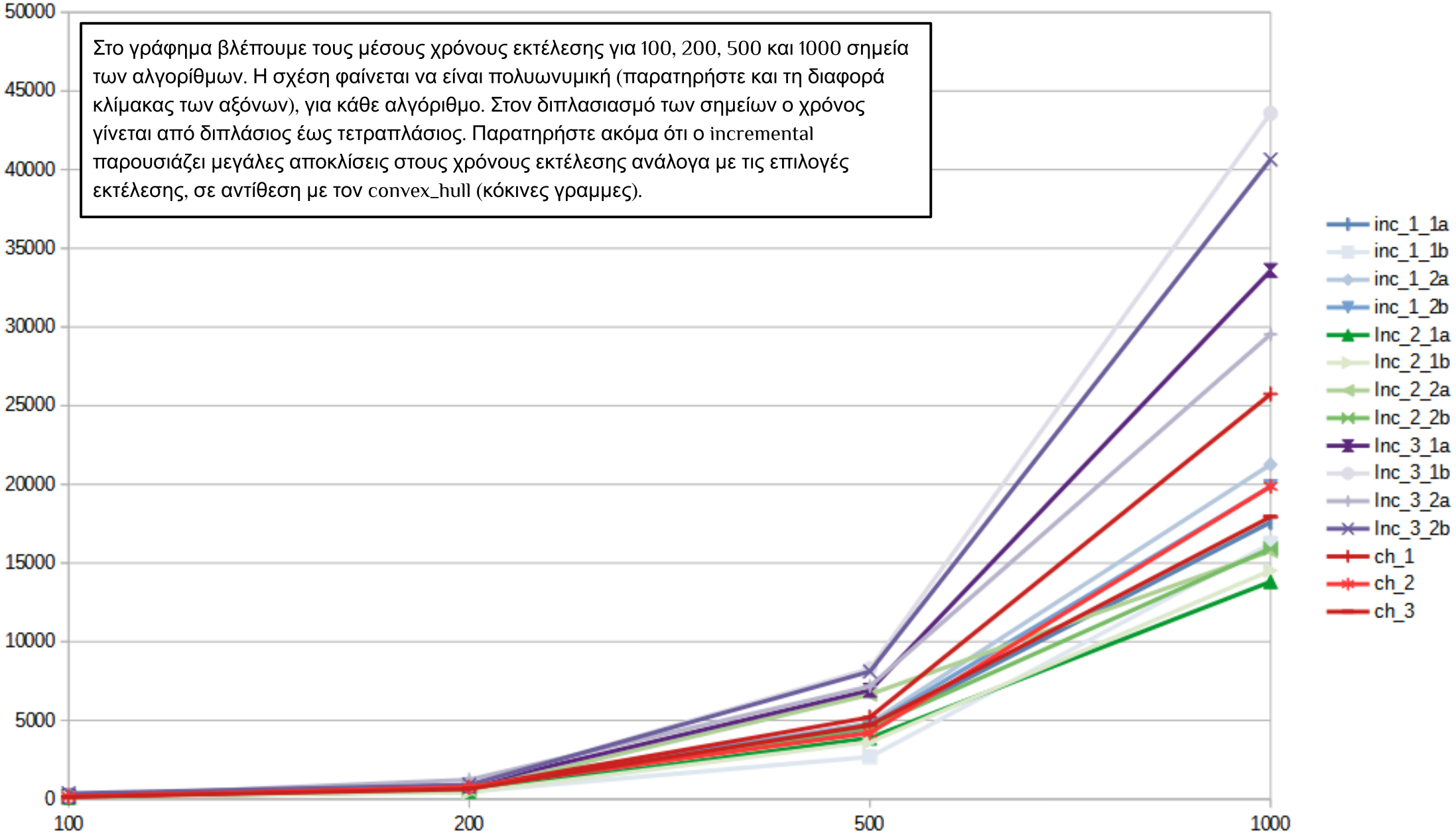
Παραπάνω βλέπουμε στοιχεία για όλα τις εκτελέσεις που δοκιμάσαμε στους αλγορίθμους. Για κάθε μια εμφανίζουμε τον λόγο εμβαδόν πολυγωνικής γραμμής / εμβαδόν κυρτού περιβλήματος και τον χρόνο εκτέλεσης σε ms.

Κόκκινα κελιά:
Σε κάποιες περιπτώσεις ο αλγόριθμος δεν τερματίζει διότι δεν υπάρχει ορατό σημείο να προστεθεί στην πολυωνική γραμμή. Αυτές οι περιπτώσεις εμφανίζονται στα κόκκινα κελιά. Ενδεχομένως όταν έχουμε τυχαία επιλογή ακμής, με αρκετές εκτελέσεις να μπορέσουμε να καταλήξουμε σε αποτέλεσμα, αλλά επιλέξαμε να τρέξουμε μία φορα κάθε αλγόριθμο και για τυχαία επιλογή. Αν δεν έχουμε τυχαία επιλογή ακμής το μη ορατό σημείο θα προκύψει ντετερμινιστικά και δε μπορούμε να κάνουμε κάτι. Το ποσοστό αποτυχίας είναι 8%.

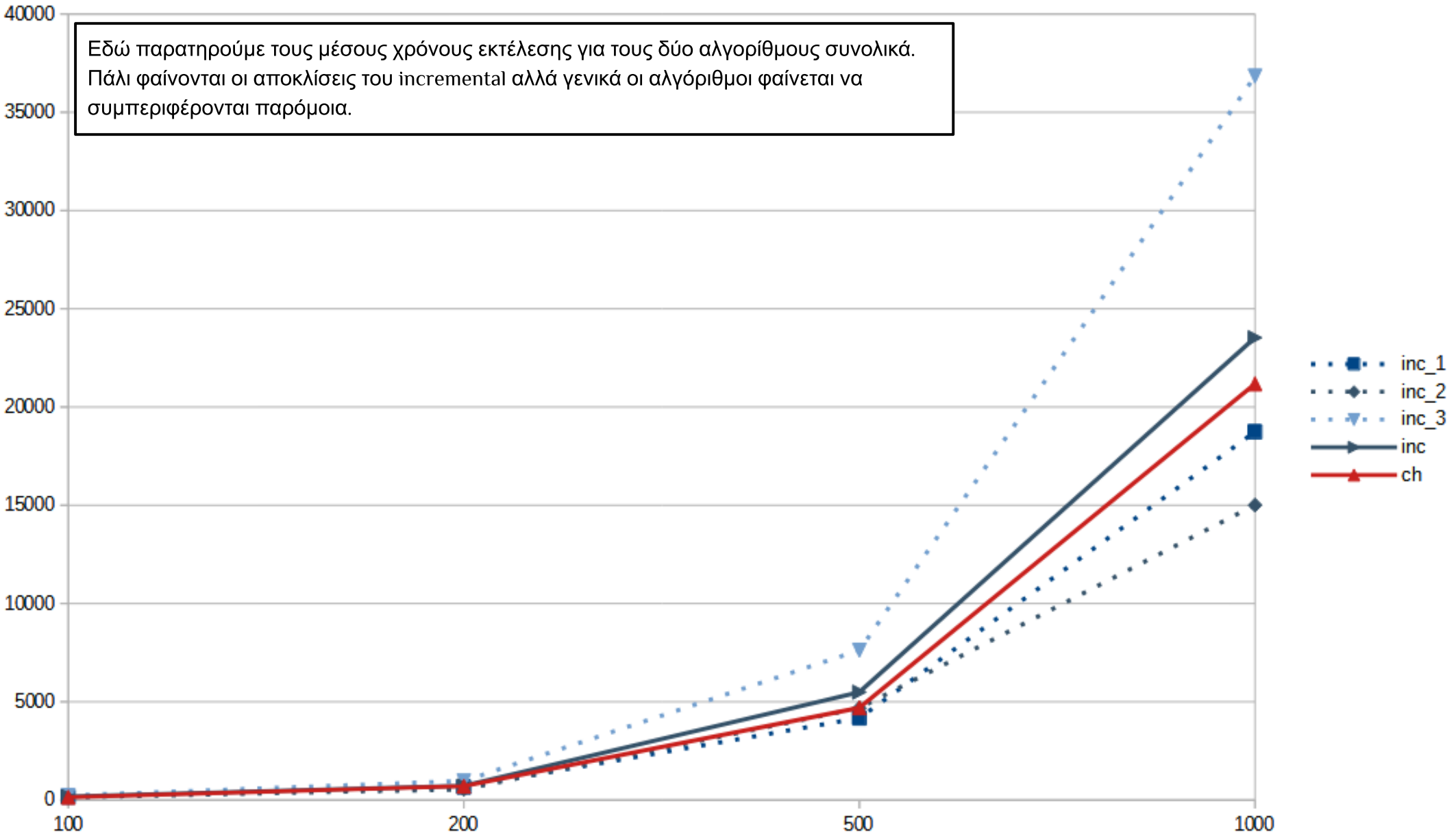
Μέσος λόγος εμβαδών:
Στην τελευταία γραμμή παραθέτουμε τους μέσους λόγους εμβαδών απο τις εκτελέσεις ενός αλγορίθμου για όλα τα δοκιμαστικά αρχεία. Παρατηρήστε ότι για τυχαία επιλογή ακμής ο λόγος είναι περίπου 0.5. Για τον incremental με ελάχιστου εμβαδού προκύπτει μικρός λόγος ενώ με μέγιστη επιλογή μεγάλος λόγος. Για τον convex_hull ισχύει το αντίστροφο. Αυτό είναι αναμενόμενο καθώς στον incremental σε κάθε βήμα προσθέτουμε μέγιστο ή ελάχιστο τρίγωνο ενώ στον convex_hull αφαιρούμε. Παρατηρήστε ακόμα ότι και οι δύο αλγόριθμοι βρίσκουν παρόμοια ελαχιστικό πολύγωνο (λόγος περίπου 0.25) ενώ ο convex_hull βρίσκει μεγαλύτερο μέγιστικο πολύγωνο (0.86 έναντι 0.73).

Χρόνοι εκτέλεσης: Παρατηρήστε ότι για ίδιο αριθμό σημείων υπάρχουν διαφορές στους χρόνους εκτέλεσης (ο μεγαλύτερος μπορεί να είναι μέχρι και 4 φορές μεγαλύτερος από τον μικρότερο).

Στο γράφημα βλέπουμε τους μέσους χρόνους εκτέλεσης για 100, 200, 500 και 1000 σημεία των αλγορίθμων. Η σχέση φαίνεται να είναι πολυωνυμική (παρατηρήστε και τη διαφορά κλίμακας των αξόνων), για κάθε αλγόριθμο. Στον διπλασιασμό των σημείων ο χρόνος γίνεται από διπλάσιος έως τετραπλάσιος. Παρατηρήστε ακόμα ότι ο incremental παρουσιάζει μεγάλες αποκλίσεις στους χρόνους εκτέλεσης ανάλογα με τις επιλογές εκτέλεσης, σε αντίθεση με τον convex_hull (κόκινες γραμμες).



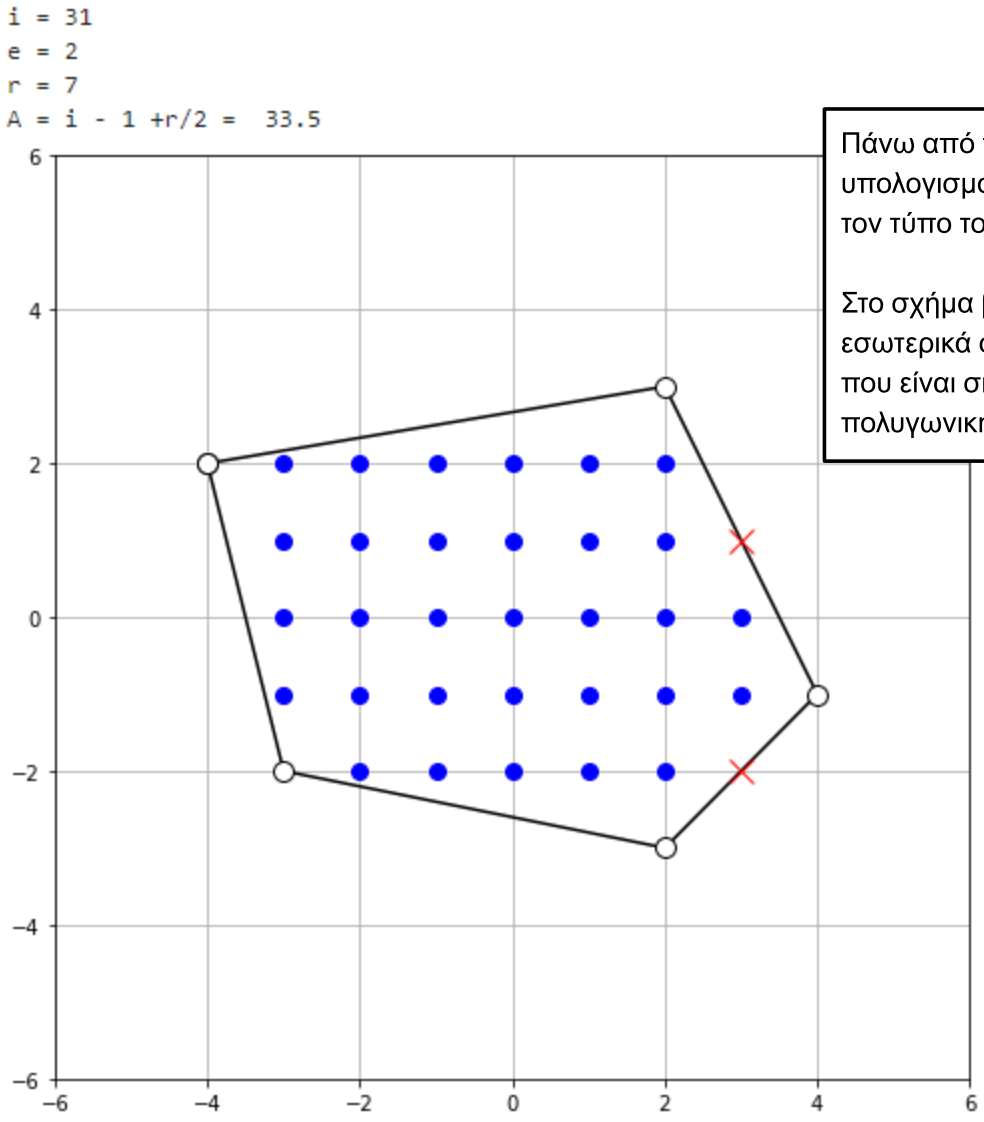
Εδώ παρατηρούμε τους μέσους χρόνους εκτέλεσης για τους δύο αλγορίθμους συνολικά. Πάλι φαίνονται οι αποκλίσεις του incremental αλλά γενικά οι αλγόριθμοι φαίνεται να συμπεριφέρονται παρόμοια.



5 *Επιπλέον σχόλια και περιεχόμενα*

5.1 pick

Όπως έχουμε ήδη αναφέρει, υλοποιήσαμε και τον pick (σε C++/CGAL και python). Η υλοποίηση του pick υπάρχει σε αντίστοιχο module αλλά όπως αναφέραμε ήδη δεν τον χρησιμοποιήσαμε γιατί η υλοποίηση ήταν αρκετά αργή. Ωστόσο παραθέτουμε μια ενδιαφέρουσα οπτικοποίηση που έγινε σε python.

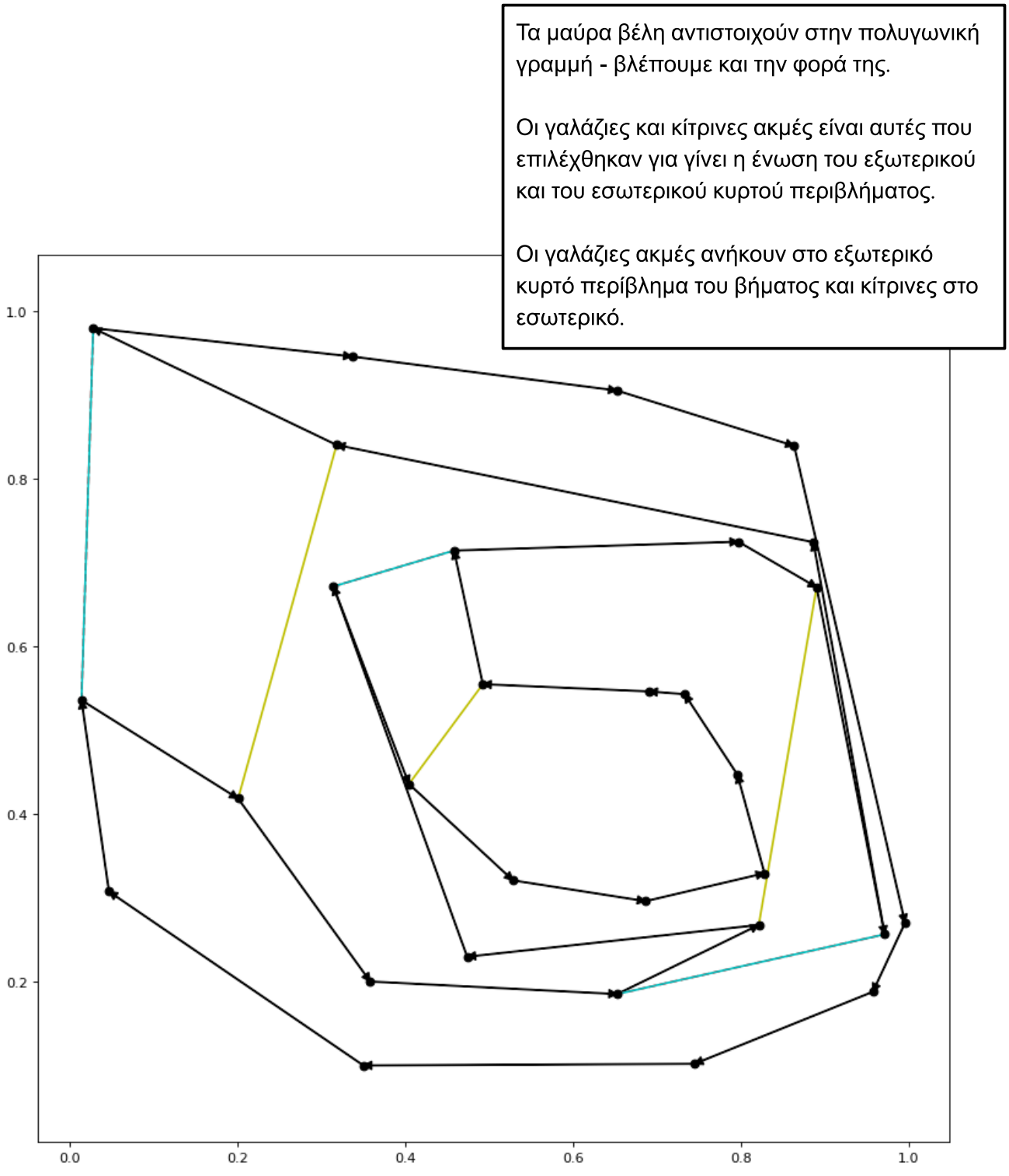


Πάνω από το σχήμα βλέπουμε τους υπολογισμούς για το εμβαδόν με τον τύπο του Pick.

Στο σχήμα βλέπουμε τα μπλε εσωτερικά σημεία και τα κόκκινα X που είναι σημεία που ανήκουν στην πολυγωνική γραμμή.

5.2 onion

Όπως έχουμε ήδη αναφέρει υλοποιήσαμε και τον onion (σε python). Δεν μεταφέραμε την υλοποίηση σε C++/CGAL διότι είμασταν δύο και δεν ήταν ζητούμενο οπότε προτιμήσαμε να εστιάσουμε στα υπόλοιπα υποχρεωτικά σημεία της εργασίας. Ωστόσο παραθέτουμε μια ενδιαφέρουσα οπτικοποίηση που έγινε σε python.



5.3 vis_app.py

Όπως αναφέραμε στην εισαγωγή, δημιουργήσαμε και ένα πρόγραμμα οπτικοποίησης των αλγορίθμων, το οποίο από διάφορα αρχεία που δημιουργούν οι αλγόριθμοι (αν επιλεχθεί η παράμετρος -vis <1/2>) δημιουργεί οπτικοποιήσεις.

Το πρόγραμμα λέγεται vis_app.py και το περιέχουμε και αυτό στο αρχείο που σας στέλνουμε. Γενικά ο κώδικας είναι σε δοκιμαστική φάση, δεν είναι καλά οργανωμένος και τερματίζει απρόοπτα σχετικά συχνά, ωστόσο μπορείτε να δοκιμάσετε να τον τρέξετε αν θέλετε.

Χρησιμοποιεί διάφορα πακέτα (όπως matplotlib και PySimpleGUI) που ίσως χρειαστεί να εγκαταστήσετε. Κάποιες εγκαταστάσεις που μπορεί να σας φανούν χρήσιμες:

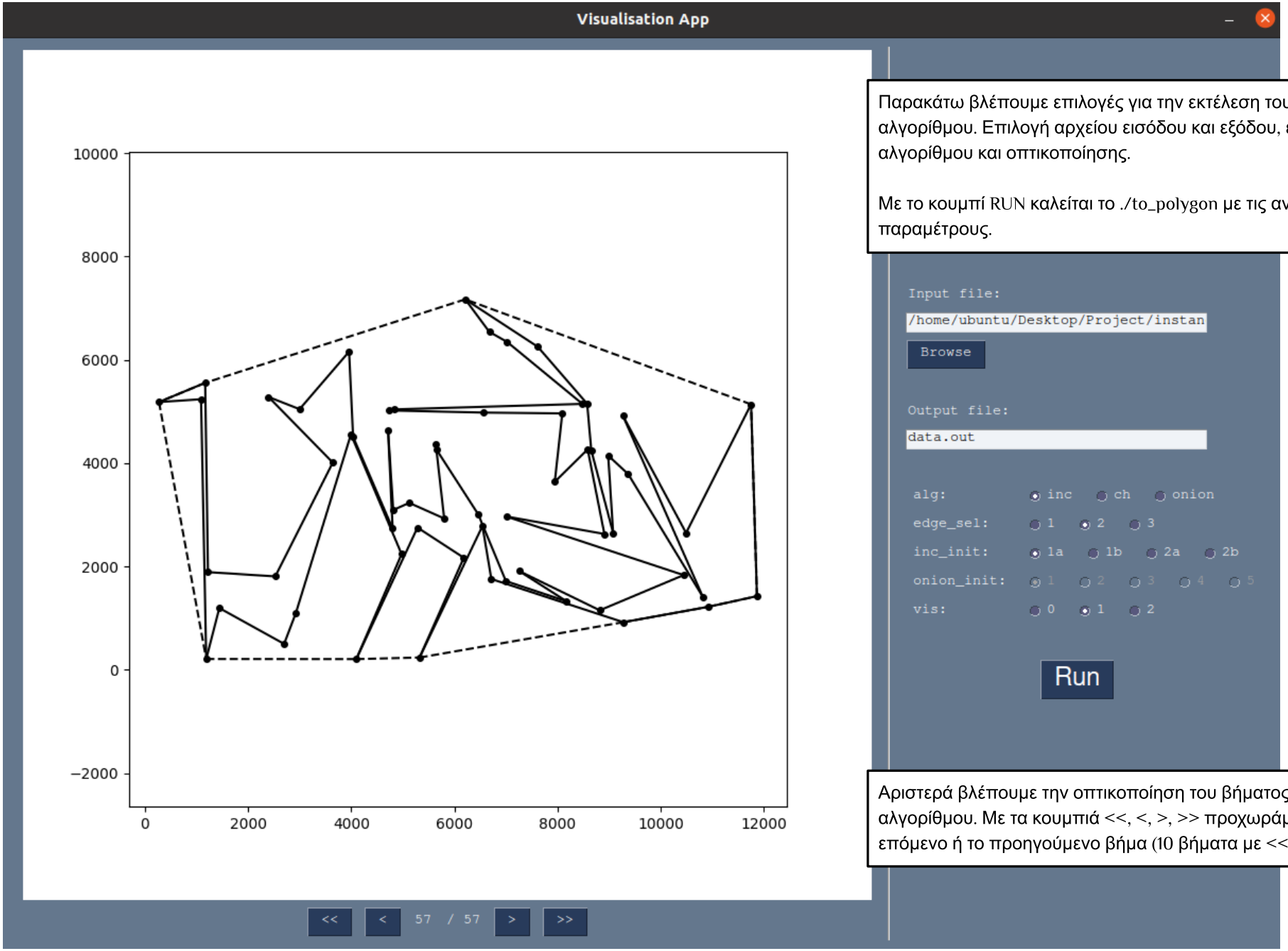
tkinter: sudo apt-get install python3-tk
pil: sudo apt-get install python3-pil
imagetk: sudo apt-get install python3-pil.imagetk

Για την εκτέλεση του προγράμματος αρκεί το vis_app.py να βρίσκεται στον ίδιο φάκελο με το to_polygon και η κλήση γίνεται με την εντολή:

```
python3 vis_app.py
```

Στη συνέχεια παραθέτουμε μερικές ενδεικτικές εικόνες από την εκτέλεση της εφαρμογής vis_app.py.

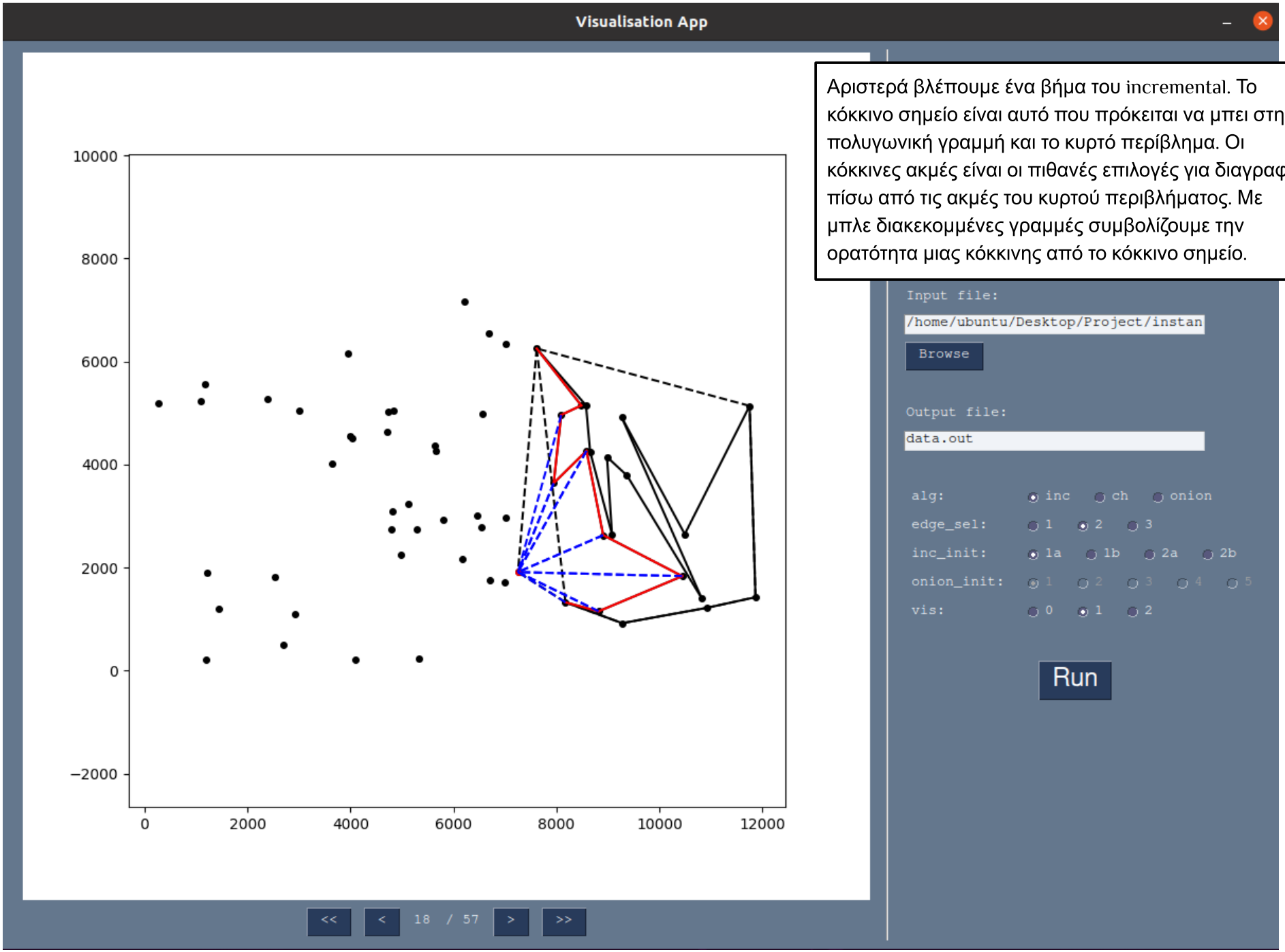
Επίσης όπως αναφέραμε σας στέλνουμε και ένα σχετικό βίντεο με μια ενδεικτική εκτέλεση της εφαρμογής.



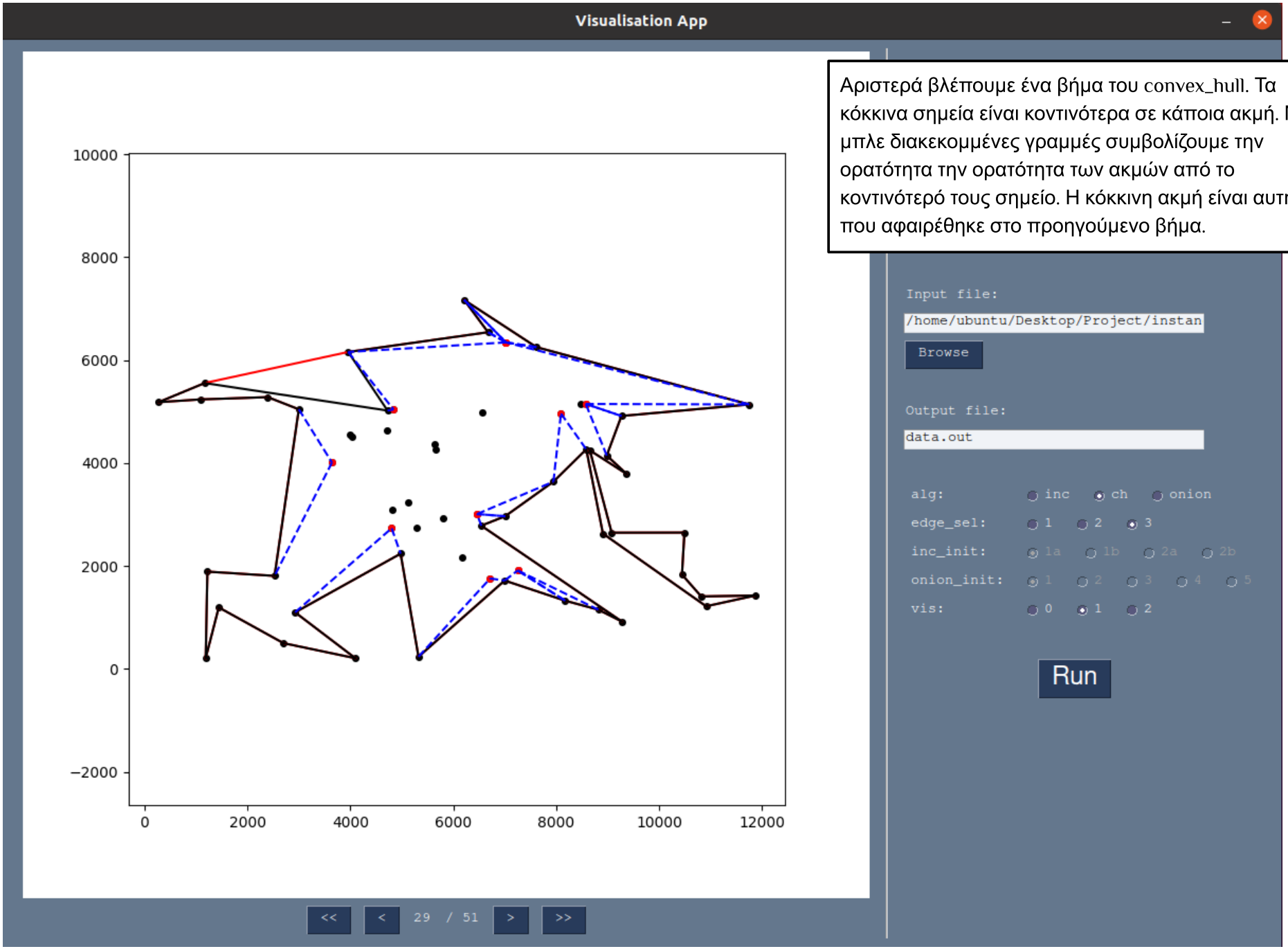
Παρακάτω βλέπουμε επιλογές για την εκτέλεση του αλγορίθμου. Επιλογή αρχείου εισόδου και εξόδου, επιλογές αλγορίθμου και οπτικοποίησης.

Με το κουμπί RUN καλείται το ./to_polygon με τις αντίστοιχες παραμέτρους.

Αριστερά βλέπουμε την οπτικοποίηση του βήματος του αλγορίθμου. Με τα κουμπιά <<, <, >, >> προχωράμε στο επόμενο ή το προηγούμενο βήμα (10 βήματα με << και >>).



Αριστερά βλέπουμε ένα βήμα του incremental. Το κόκκινο σημείο είναι αυτό που πρόκειται να μπει στην πολυγωνική γραμμή και το κυρτό περίβλημα. Οι κόκκινες ακμές είναι οι πιθανές επιλογές για διαγραφή, πίσω από τις ακμές του κυρτού περιβλήματος. Με μπλε διακεκομμένες γραμμές συμβολίζουμε την ορατότητα μιας κόκκινης από το κόκκινο σημείο.



Αριστερά βλέπουμε ένα βήμα του convex_hull. Τα κόκκινα σημεία είναι κοντινότερα σε κάποια ακμή. Με μπλε διακεκομμένες γραμμές συμβολίζουμε την ορατότητα την ορατότητα των ακμών από το κοντινότερό τους σημείο. Η κόκκινη ακμή είναι αυτή που αφαιρέθηκε στο προηγούμενο βήμα.