

Plague Spread

3Δ Υπολογιστική Γεωμετρία & Όραση

ΚΟΣΜΑΣ ΑΡΧΟΝΤΗΣ, 1084020

Πανεπιστήμιο Πατρών

14 Ιουλίου 2024

Περιεχόμενα

| | |
|--|-----------|
| 1 Εισαγωγή | 3 |
| 2 Περιγραφή του Προβλήματος | 4 |
| 3 Μεθοδολογία | 7 |
| 3.1 Μέρος Α | 7 |
| 3.1.1 Ερώτημα 1 | 7 |
| 3.1.2 Ερώτημα 2 | 11 |
| 3.1.3 Ερώτημα 3 | 14 |
| 3.2 Μέρος Β | 19 |
| 3.2.1 Ερώτημα 4 | 19 |
| 3.2.2 Ερώτημα 5 | 22 |
| 3.2.3 Ερώτημα 6 | 28 |
| 3.2.4 Ερώτημα 7 | 33 |
| 4 Θεωρία/Θεωρητικό Υπόβαθρο | 39 |
| 4.1 Διάγραμμα Voronoi | 39 |
| 4.1.1 Αλγόριθμος του Fortune | 39 |
| 4.2 Γεωδαισιακές Αποστάσεις | 41 |
| 4.3 Δυϊκός Γράφος | 41 |
| 4.4 Αλγόριθμος του Dijkstra - Αναζήτηση Βέλτιστου Μονοπατιού | 41 |
| 4.5 Χωρική αναζήτηση, k-d Trees | 41 |
| 4.6 Βαρυκεντρική Παρεμβολή | 42 |
| 4.7 Κυρτό περίβλημα | 42 |
| 5 Υλοποιήσεις Υπολογιστικής Γεωμετρίας στον Κώδικα | 43 |
| 5.1 Η κλάση Voronoi | 43 |
| 5.2 Η κλάση KdNode | 43 |
| 5.3 Η κλάση Dijkstra | 43 |
| 5.4 Το αρχείο <i>GeometryUtils</i> | 43 |
| 5.5 Η μέθοδος <i>createRandomWeighted</i> | 43 |

| | | |
|-----------|---|-----------|
| 5.6 | Η συνάρτηση <code>triangulate_grid</code> | 43 |
| 5.7 | Η συνάρτηση <code>is_inside_polygon_2d</code> | 44 |
| 6 | Αποτελέσματα & Συζήτηση | 45 |
| 6.1 | Αποδόσεις | 45 |
| 6.1.1 | Υπολογισμός Μητρώων | 45 |
| 6.1.2 | Σε χρόνο εκτέλεσης | 48 |
| 6.2 | Αδυναμίες - Πιθανές Βελτιώσεις | 50 |
| 7 | Συμπεράσματα | 51 |
| 8 | Οδηγός Κώδικα | 52 |
| 9 | Οδηγίες Χρήσης Προγράμματος | 54 |
| 9.1 | Για τη 2Δ σκηνή: | 54 |
| 9.2 | Για τη 3Δ σκηνή: | 55 |
| 9.3 | Για τη 3Δ σκηνή με terrain: | 55 |
| 10 | Βιβλιογραφία | 56 |

1 Εισαγωγή

Σκοπός αυτής της αναφοράς είναι η περιγραφή της διαδικασίας εκπόνησης και αποτελεσμάτων της απαλλακτικής εργασίας για το μάθημα 3Δ Υπολογιστική Γεωμετρία & Όραση του τμήματος Ηλεκτρολόγων Μηχανικών & Τεχνολογίας Υπολογιστών του Πανεπιστημίου Πατρών.

Η θεματολογία της εργασίας περιλαμβάνει διατάξεις του δισδιάστατου και τρισδιάστατου χώρου, διαχείριση μεγάλων νέφων σημείων, δυϊκους γράφους, αναζήτηση βέλτιστου μονοπατιού, παρεμβολή και άλλες έννοιες και προβλήματα που παρουσιάζονται στην υπολογιστική γεωμετρία, σε ένα γενικότερο (και απλοποιημένο) πλαίσιο επιδημιολογίας.

Το πρόγραμμα που έχει αναπτυχθεί είναι γραμμένο σε python, κάνοντας εκτενής χρήση της Numpy/Scipy, Open3D και του framework που έχει αναπτυχθεί από το VVR Group του τμήματος.

2 Περιγραφή του Προβλήματος

Η εργασία φέρει τον τίτλο “Plague Spread”, το οποίο παραπέμπει στο επιδημιολογικό στοιχείο της. Ζητείται αρχικά, η δειγματοληψία ενός πληθυσμού σημείων που θα αναπαριστούν ανθρώπους, και μερικών πηγαδιών στα οποία οι άνθρωποι θα ταξιδεύουν για το νερό τους. Μερικά από τα πηγάδια είναι μολυσμένα, και πρέπει να δειχθεί ο μολυσμένος πλυνθησμός συγκριτικά με τον μη-μολυσμένο.

Αυτή είναι η θεμελιώδης απαίτηση της εργασίας, η οποία επεκτείνεται και γενικεύεται αργότερα. Αρχικά ο πλυνθησμός απόλυτα διαλέγει το κοντινότερο στον ίδιο πηγάδι, και μετά ζητείται μοντελοποίηση για πιθανοτική επιλογή των 3 κοντινότερων πηγαδιών. Ξεκινάμε σε μια δισδιάσταση σκηνή και προχωράμε σε τρισδιάστατη, όπου λαμβάνουμε υπόψιν γεωδαισιακές αποστάσεις πια, ενώ ζητείται στο τέλος και μια εφαρμογή σε πραγματικό χάρτη.

Το θέμα μας παραπέμπει στην εργασία του John Snow το 1854, ο οποίος αξιοποίησε διαγράμματα Voronoi 4.1 για να δείξει ότι οι πλειονότητα ανθρώπων που πέθαναν από χολέρα στο Σόχο του Λονδίνου επίσης ζούσαν πιο κοντά στη μολυσμένη πηγή νερού, από ότι σε οποιαδήποτε άλλη.

Παρατίθεται αναλυτικότερα ολόκληρη η λίστα με τα ζητούμενα της εργασίας:

Μέρος Α.

1. Δημιουργήστε έναν υποθετικό χάρτη και δειγματοληπτήστε πολλά σημεία επάνω του, τα οποία θα αντιπροσωπεύουν την παρουσία ανθρώπων σε διάφορες περιοχές. Έπειτα δειγματοληπτήστε και ένα μικρότερο σύνολο σημείων που θα αντιπροσωπεύει πηγάδια.
2. Επιλέξτε ένα τυχαίο υποσύνολο των πηγαδιών που δημιουργήσατε, τα οποία θα είναι μολυσμένα με χολέρα. Αν υποθέσουμε πως κάθε άνθρωπος θα επιλέξει το κοντινότερο σε αυτόν πηγάδι για να ξεδιψάσει, δημιουργήστε αλγόριθμο που θα παρουσιάζει με χρωματική παλετα της επιλογής σας την κατανομή των μολυσμένων στις διάφορες περιοχές.
3. Υποθέστε πως με πιθανότητες p_0 , p_1 , p_2 ($p_0 >> p_1, p_2$) ο κάθε άνθρωπος επιλέγει το πρώτο, δεύτερο και τρίτο κοντινότερο πηγάδι αντίστοιχα. Δείξτε τον τρόπο με τον οποίο επαναπροσδιορίζονται τα όρια των μολυσμένων περιοχών δημιουργώντας ένα animation που θα εκτελεί το ερώτημα 2 για διάφορες τιμές p_0, p_1, p_2 .

Μέρος Β.

4. Δημιουργήστε ένα πυκνό επίπεδο πλέγμα, και ανυψώστε το κάθε σημείο του σύμφωνα με μια συνάρτηση θορύβου που θα επιλέξετε (προτιμήστε θορύβου τύπου perlin που είναι smooth), και επαναλάβετε την διαδικασία του ερωτήματος 1.
5. Επαναλάβετε τα ερωτήματα 2 και 3 επάνω στο πλέγμα χρησιμοποιώντας γεωδαισιακές αποστάσεις.
6. Αυτή την φορά, δημιουργήστε μια δική σας συνάρτηση απόστασης η οποία θα λαμβάνει υπό όψιν όχι μόνο την απόσταση μεταξύ ενός ανθρώπου και ενός πηγαδιού, αλλά και το elevation (η ανηφόρα θα αντιστοιχεί σε περισσότερη απόσταση και η κατηφόρα σε λιγότερη, θεωρώντας την διαδρομή μονόδρομη). Επαναλάβετε τα ερωτήματα 2 και 3 με τα νέα δεδομένα.
7. Δημιουργήστε τώρα με τον ίδιο τρόπο έναν χάρτη (fixed για δική σας ευκολία) και αναθέστε βάρη στις ακμές του γράφου δικαιολογώντας τα με βάση την προσβασιμότητα των διαφόρων περιοχών, π.χ. μια διαδρομή

ενδεχομένως να είναι γρηγορότερη επειδή υπάρχει τρένο, ενώ μια άλλη μπορεί να περνάει ανάμεσα από βουνά. Κάντε customizations κατά το δοκούν, προκειμένου να γίνει πιο ρεαλιστικό το αποτέλεσμα, π.χ.

- a. Μπορείτε να στηριχθείτε σε πραγματικούς χάρτες για να βελτιώσετε οπτικά το αποτέλεσμα.
- b. Μπορείτε το πλέγμα που θα δημιουργήσετε να το κάνετε να έχει ανομοιόμορφη τριγωνοποίηση, που θα εξαρτάται από το terrain.

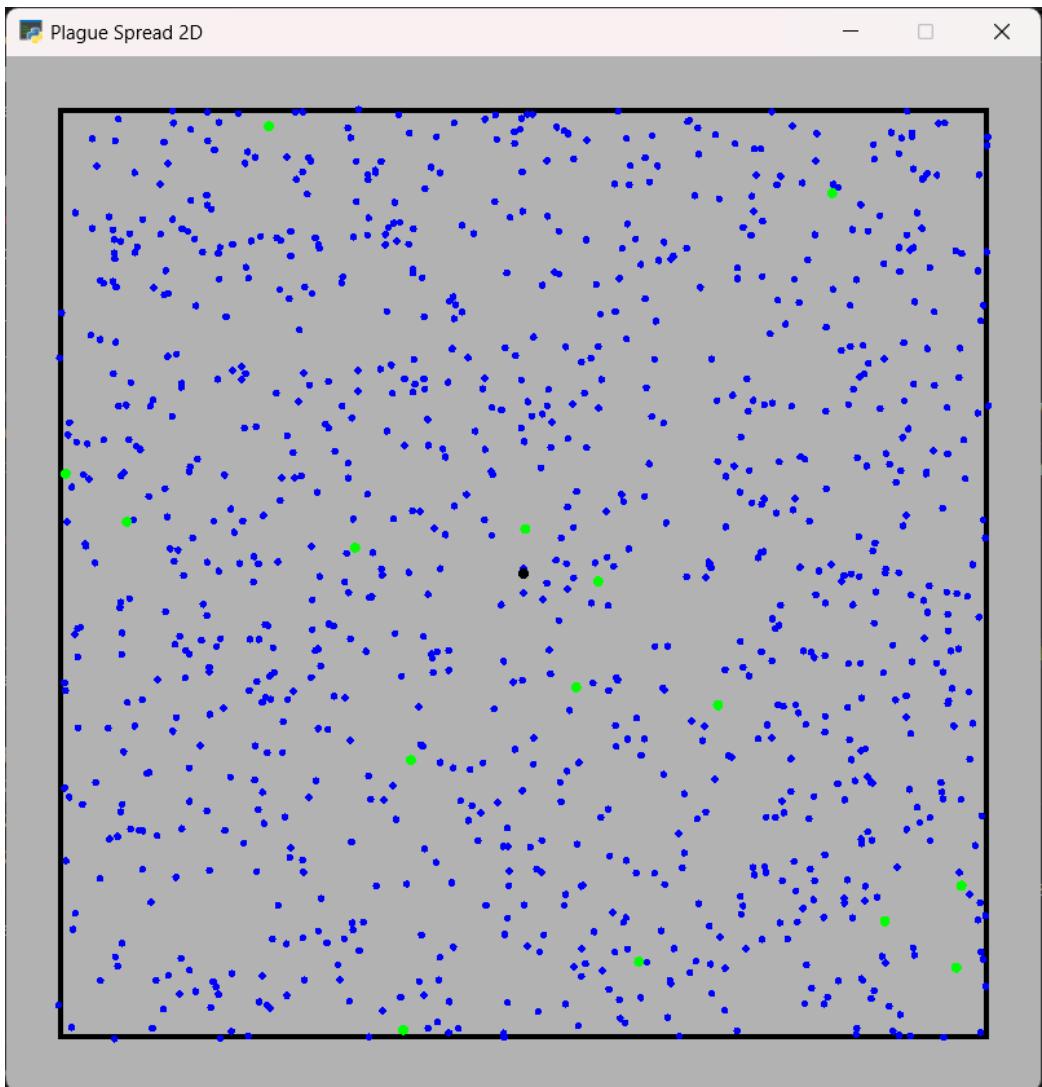
3 Μεθοδολογία

Σε αυτή την ενότητα θα περιγράψουμε τη διαδικασία επίλυσης των ερωτημάτων, τον τρόπο σκέψης και θα περιγράψουμε επίσης την ανάπτυξη του προγράμματος χρονολογικά. Θα αναφερθούν περιληπτικά αρχικές σκέψεις και αποτυχημένες προσπάθειες, καθώς και πόση δουλειά και πόσα ερωτήματα όλοκληρώθηκαν εν τέλει.

3.1 Μέρος Α

3.1.1 Ερώτημα 1

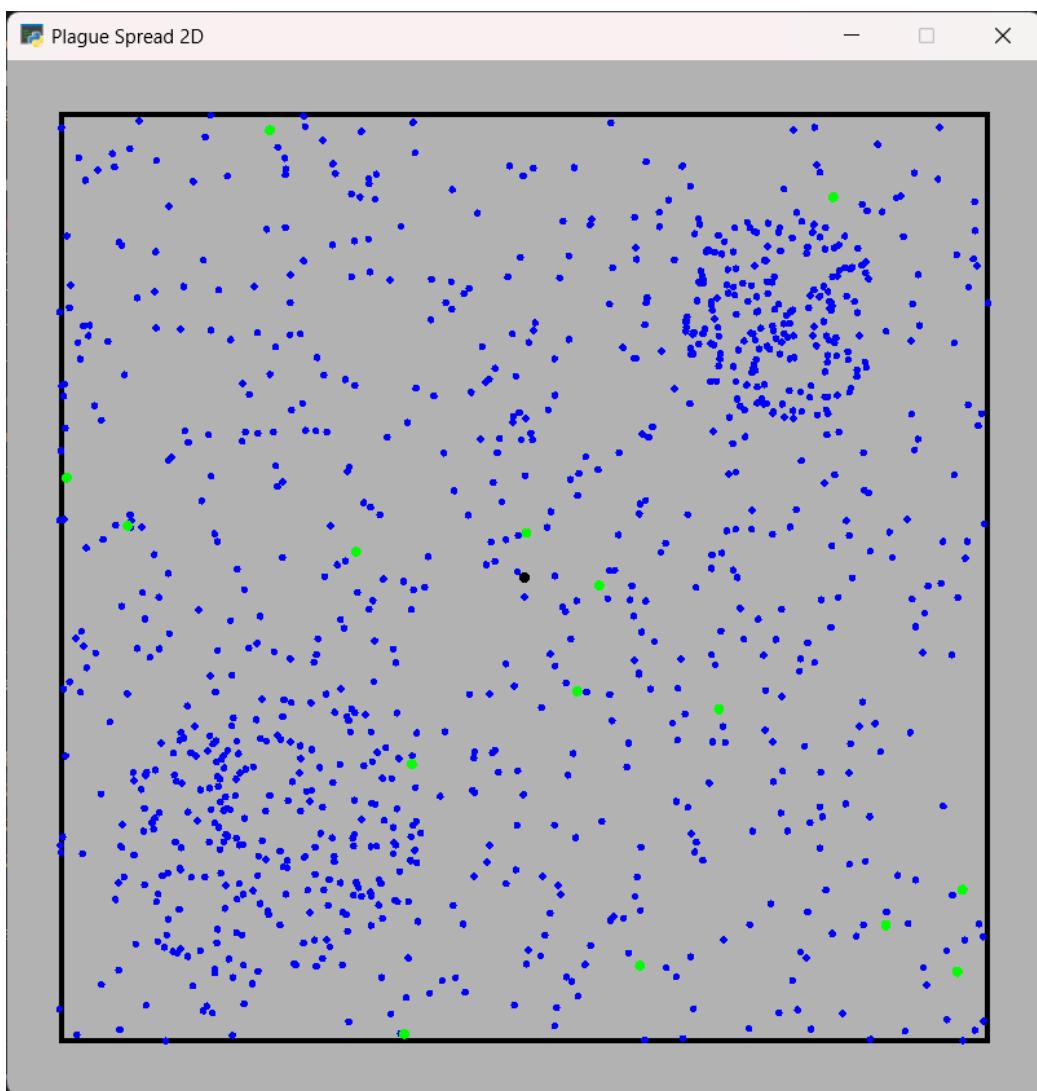
Για το ερώτημα αυτό, χρησιμοποιούμε τη κλάση `PointSet2D` που μας προσφέρεται από το `vvrywork` και ακολουθούμει τον σκελετό που προσφέρεται από τα εργαστήρια του μαθήματος για τη κατασκευή της 2Δ σκηνής. Η μέθοδος `createRandom` ενός `instance` του αντικειμένου παράγει έναν τυχαίο αριθμό σημείων, ομοιόμορφα κατανεμημένα σε εναν περιβάλλοντα χώρο (bounding box). Μέσω αυτής της μεθόδου, λαμβάνουμε τον πληθυσμό σημείων που αναπαριστούν ανθρώπους και, σε μικρότερο αριθμό, τα πηγάδια. Οι άνθρωποι φέρουν το μπλε χρώμα και τα πηγάδια είναι πράσινα.



Σχήμα 3.1: 2Δ σκηνή, ομοιόμορφη δειγματοληψία.

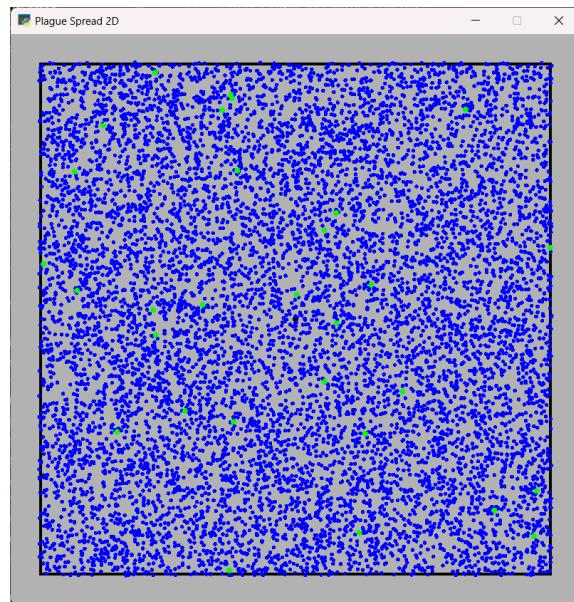
Αν δεν θέλουμε τα σημεία να είναι ομοιόμορφα κατανεμημένα, χρειαζόμαστε έναν τρόπο ώστε να ενθαρρύνουμε κάποιες περιοχές να δειγματοληπτούνται πιο συχνά από άλλες. Προσθέτουμε τη μέθοδο `createRandomWeighted` 5.5, στη κλάση `PointSet2D`, με την οποία δίνουμε όσα “regions of interest”, ως σημεία, θέλουμε, την ακτίνα της περιοχής ενδιαφέροντος και τη προτεραιότητα που θα ’χει (πόσο θα προτιμάται).

Έτσι, μπορούμε να δημιουργήσουμε πιο πυκνές περιοχές, που μπορούμε να θεωρήσουμε αστικά κέντρα ή άλλη γεωγραφική ερμηνεία που ταιριάζει.

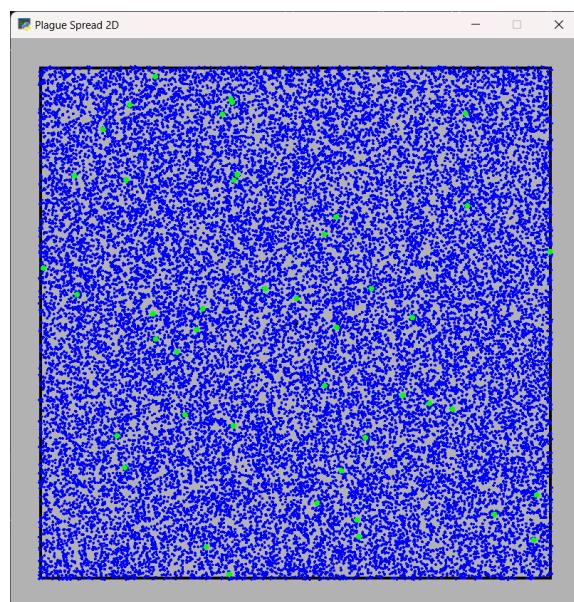


Σχήμα 3.2: 2Δ σκηνή, περιοχές ενδιαφέροντος.

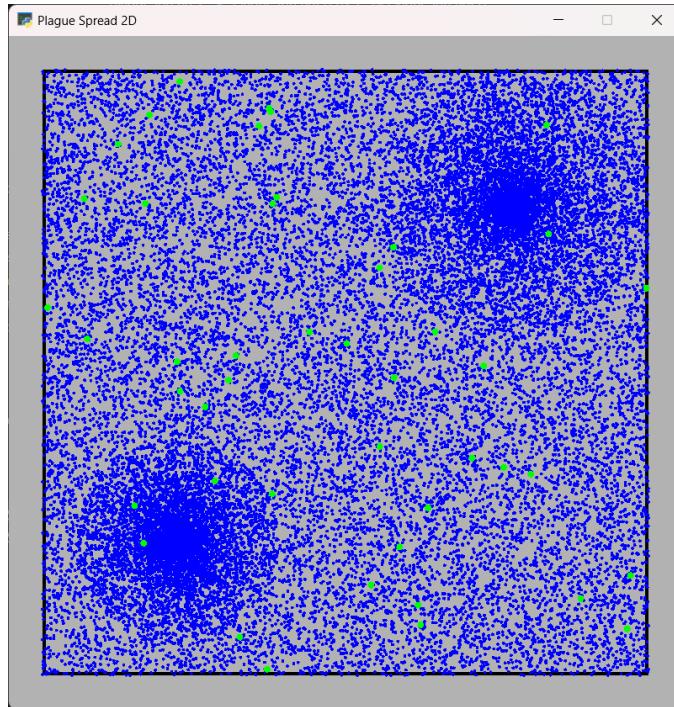
Για καλύτερη παρουσίαση του προγράμματος, έχουμε 3 σενάρια εκτέλεσης με αυξανόμενα νούμερα για τον πληθυσμό ανθρώπων και πηγαδιών, με 1000 ανθρώπους - 15 πηγάδια, 10,000 ανθρώπους - 30 πηγάδια και 30,000 ανθρώπους - 45 πηγάδια.



Σχήμα 3.3: 10,000 άνθρωποι με 30 πηγάδια.



Σχήμα 3.4: 30,000 άνθρωποι με 45 πηγάδια, ομοιόμορφη δειγματοληψία.



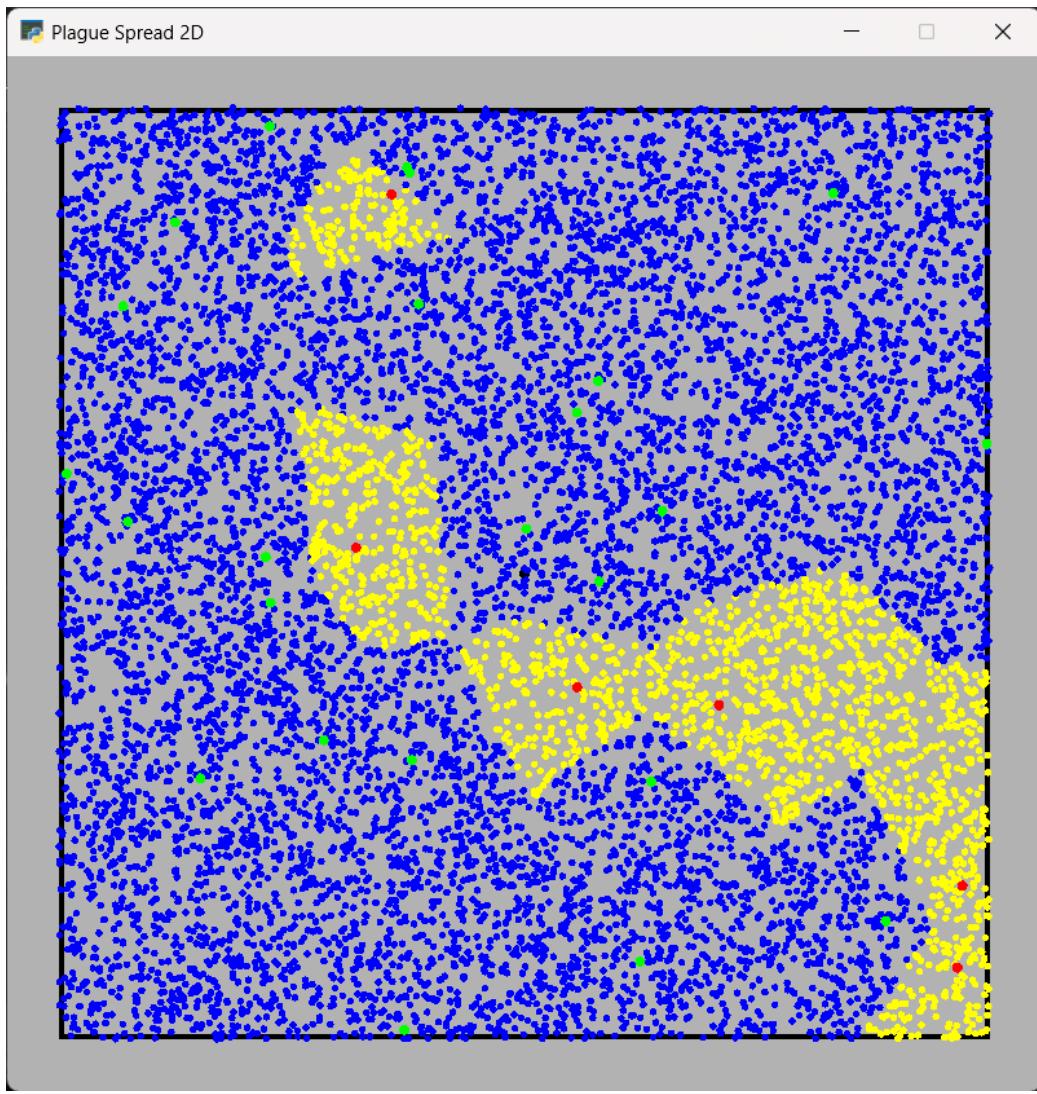
Σχήμα 3.5: 30,000 άνθρωποι με 45 πηγάδια, περιοχές ενδιαφέροντος.

3.1.2 Ερώτημα 2

Από τον πληθυσμό μας για πηγάδια, πρέπει να μολύνουμε ένα τυχαίο υποσύνολο τους και να δείξουμε την κατανομή των ανθρώπων που μολύνονται επίσης, αν πίνουν νερό από το κοντινότερο σε αυτούς πηγάδι. Το κόκκινο χρώμα αναπαριστά το μολυσμένο πηγάδι και το κίτρινο τον μολυσμένο πληθυσμό.

Για να το επιτύχουμε αυτό, κατασκευάζουμε 2 συναρτήσεις, `infect_wells` και `find_infected_people`. Βασιζόμαστε στην άκρως αποτελεσματική numpy για γρήγορους υπολογισμούς, και ιδιαίτερα στο vectorization της που επιτρέπει παράλληλη και αποδοτική επεξεργασία μεγάλων πινάκων. Πράγματι, οι πληθυσμοί μας και οι συγκρίσεις μεταξύ τους (κοντινότερη απόσταση), αποτελούν πράξεις πινάκων.

Για την εύρεση κοντινότερου πηγαδιού δοθέντος ενός ανθρώπου, θα μπορούσαμε να χρησιμοποιήσουμε ένα k-d Tree 4.5, για πιο αποτελεσματική και πιο δομημένη αναζήτηση του κοντινότερου γείτονα (nearest neighbor), όμως για τα νούμερα που εξετάζουμε (1,000 πληθυσμός, 10,000, 30,000) η numpy τα διαχειρίζεται επαρκώς. Όντως, θα κάνουμε χρήση ενός k-d Tree στη 3Δ σκηνή για σύγκριση ευκλείδιας απόστασης και γεωδαισιακής 3.2.2.



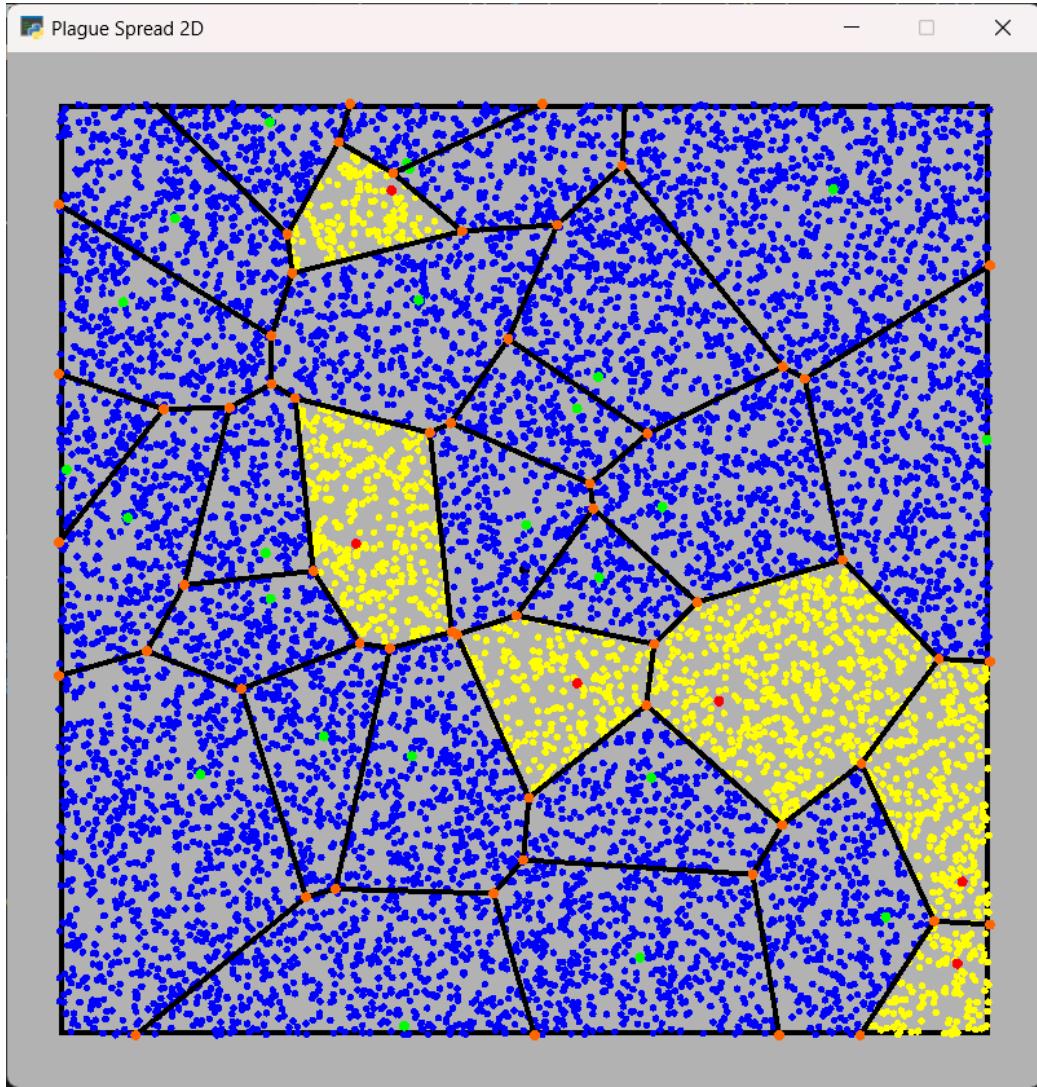
Σχήμα 3.6: Μολυσμένα πηγάδια και μολυσμένες περιοχές ανθρώπων.

Από τα αποτελέσματα της διαδικασίας μόλυνσης πηγαδιών και ανθρώπων, μπορούμε εύκολα να διαπιστώσουμε ότι οι περιοχές αυτές περιγράφονται πλήρως από το διάγραμμα Voronoi 4.1.

Για να παρατηρήσουμε τα κελιά Voronoi, τις περιοχές επιρροής των πηγαδιών, αναπτύσσουμε σε αυτό το σημείο κάθικα για την εύρεση του διαγράμματος 5.1 εφαρμόζοντας μία παραλλαγή του Αλγόριθμου του Fortune 4.1.1, όπως περιγράφεται στο βιβλίο των Mark de Berg et al, Υπολογιστική Γεωμετρία: Αλγόριθμοι και Εφαρμογές. Αυτή η απόπειρα δε κατέστει επιτυχής, αφού η κλάση δε παράγει σωστά αποτελέσματα για μια σκηνή.

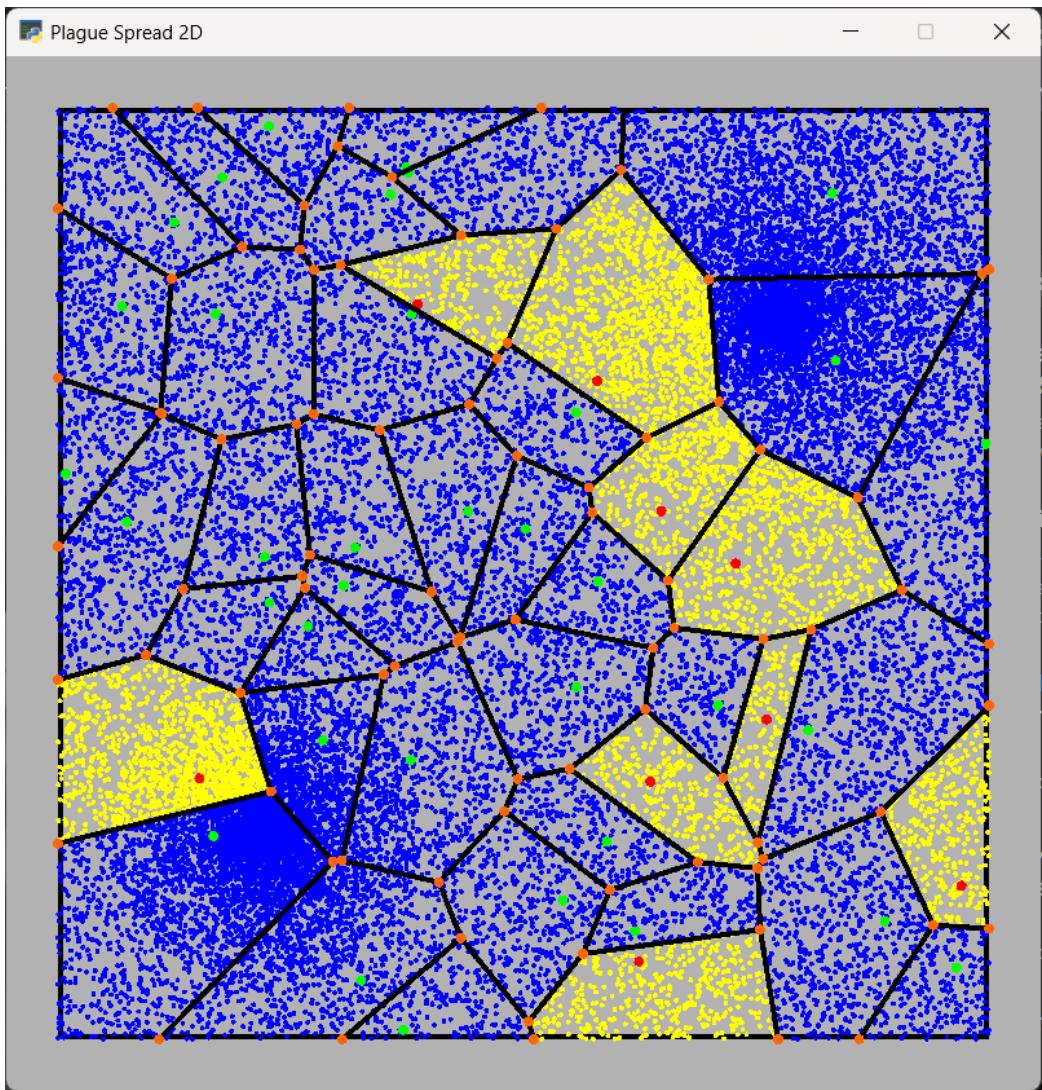
Ως εναλλακτική, μπορούμε να χρησιμοποιήσουμε την κλάση που προσφέρε-

ται από το πακέτο `scipy.spatial`. Αφού δημιουργήσουμε το διάγραμμα, πρέπει να το αποκόψουμε στο bounding box της σκηνής, έχουμε στη διαθεσή μας τις ακμές και τα σημεία του Voronoi, με εστίες τα πηγάδια.



Σχήμα 3.7: Το διάγραμμα Voronoi στη 2Δ σκηνή με την έκταση επιρροής των πηγαδιών.

Οπως υποθέσαμε, τα κελιά Voronoi περιγράφουν πλήρως τον μολυσμένο πληθυσμό, εφόσον οι άνθρωποι επιλέγουν να πιουν νερό από το κοντινότερο σε αυτούς πηγάδι.



Σχήμα 3.8: Το διάγραμμα Voronoi στη 2Δ σκηνή με 30,000 ανθρώπους και 45 εστίες - πηγάδια.

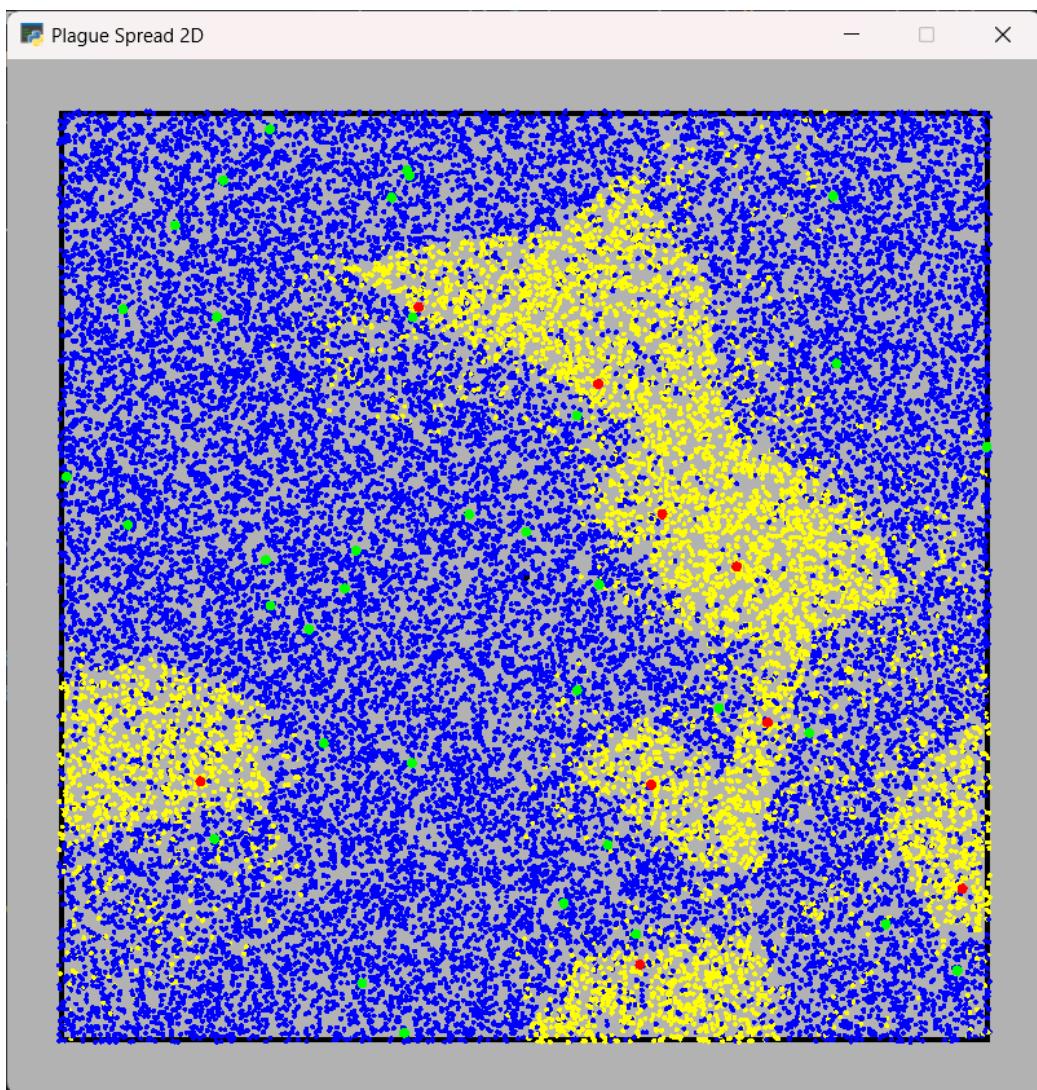
3.1.3 Ερώτημα 3

Για μια καλύτερη μοντελοποίηση του σεναρίου που παρουσιάζεται από την εργασία, απαιτείται να μετατρέψουμε την επιλογή πηγαδιού από τους ανθρώπους από απόλυτα το κοντινότερο πηγάδι σε πιθανοτικά μεταξύ τα 3 κοντινότερα πηγάδια.

Για τον σκοπό αυτό, αναπτύσσουμε τη συνάρτηση `find_infected_people_stochastic`, η οποία καλείται έναντι της `find_infected_people` αν το flag `RANDOM_SELECTION` είναι ενεργοποιημένο (μπορεί να γίνει μέσω του πατήματος του πλήκτρου R). Τα-

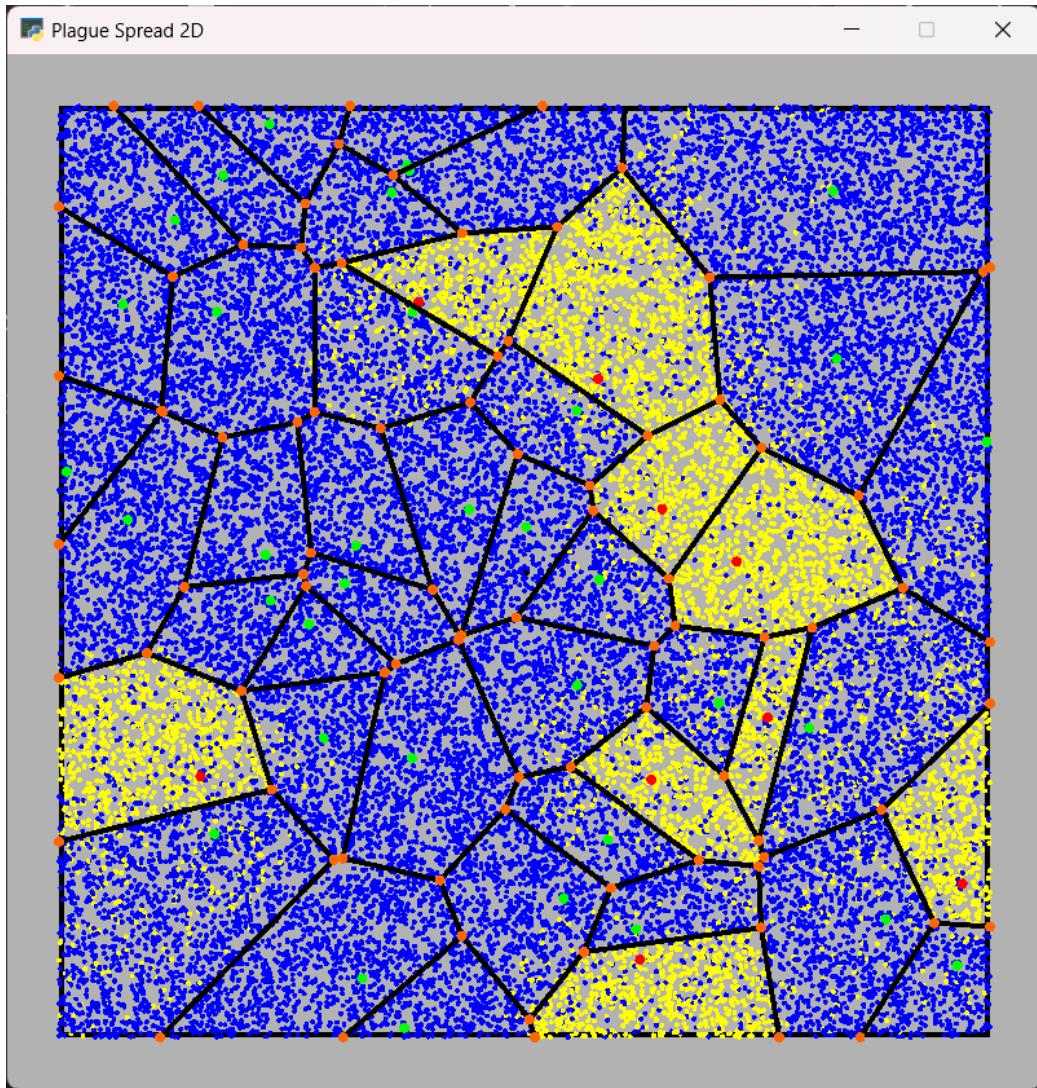
ξινομούμε τα πηγάδια με βάση την απόστασή τους από τον εξεταζόμενο άνθρωπο, κρατάμε τα 3 πρώτα, και διαλέγουμε μεταξύ τους σύμφωνα με δοθέντες πιθανότητες. Αν το πηγάδι που επιλεγεί είναι μολυσμένο, μολύνεται και ο άνθρωπος. Όπως αναφέραμε και στο Ερώτημα 2 3.1.2, θα μπορούσε να χρησιμοποιηθεί δομή k-d Tree 4.5 για γρήγορη αναζήτηση k nearest neighbors, αλλά η numpy δε προκαλεί ιδιαίτερες καθυστερήσεις.

Οι προεπιλεγμένες πιθανότητες είναι, από κοντινότερο πηγάδι στο πιο μακρινό, 0.8, 0.15, 0.05 και μπορούμε να τις αυξήσουμε ή μειώσουμε καθώς επιθυμούμε (ώστε να αθροίζει πάντα στο 1).



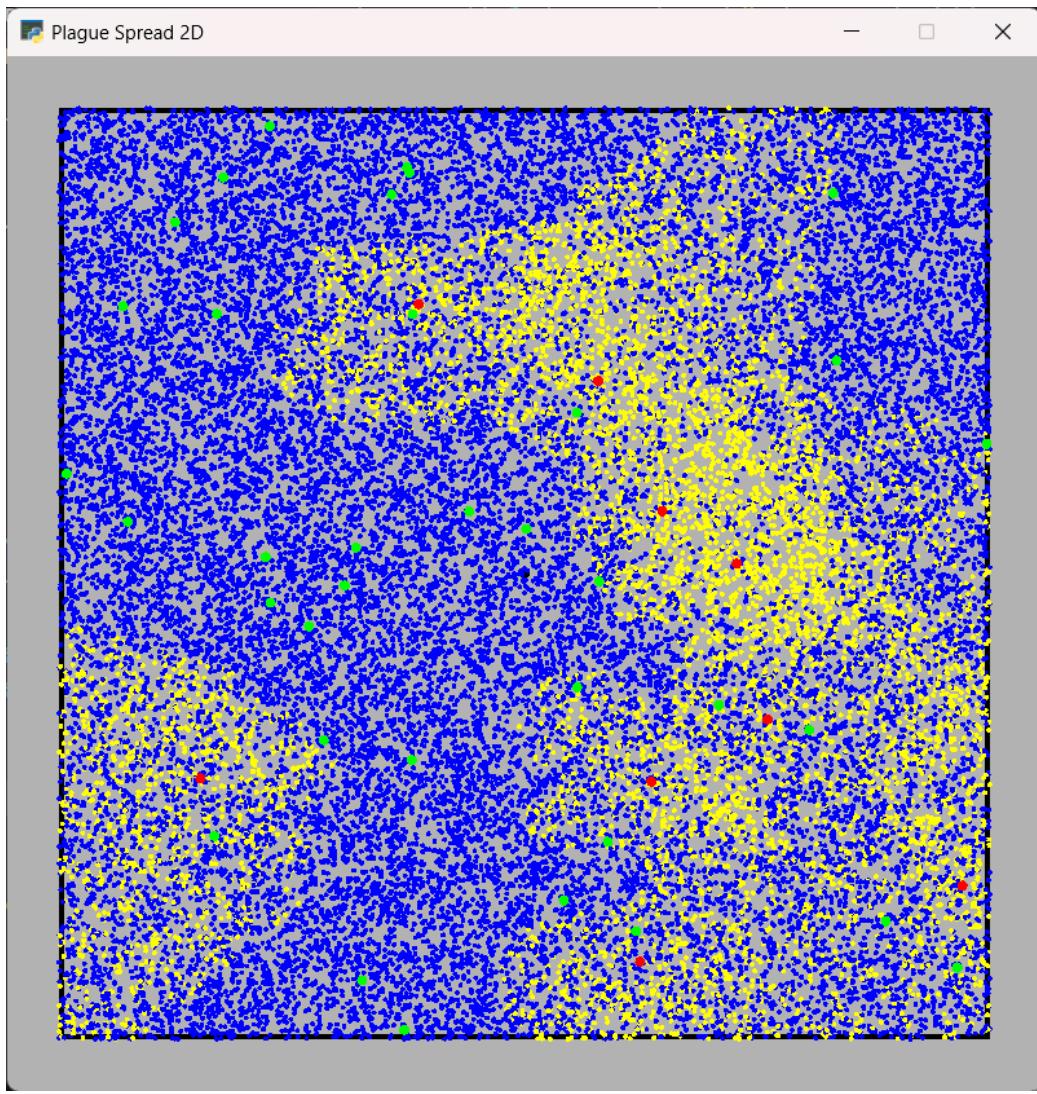
Σχήμα 3.9: 2Δ σκηνή, με 30,000 ανθρώπους και υψηλές πιθανότητες να διαλεχθεί το κοντινότερο πηγάδι.

Μπορούμε να δούμε τα αυστηρά όρια του διαγράμματος Voronoi να σπάνε, όμως είναι ακόμα εύκολα αναγνωρήσιμα με τις προεπιλεγμένες πιθανότητες.



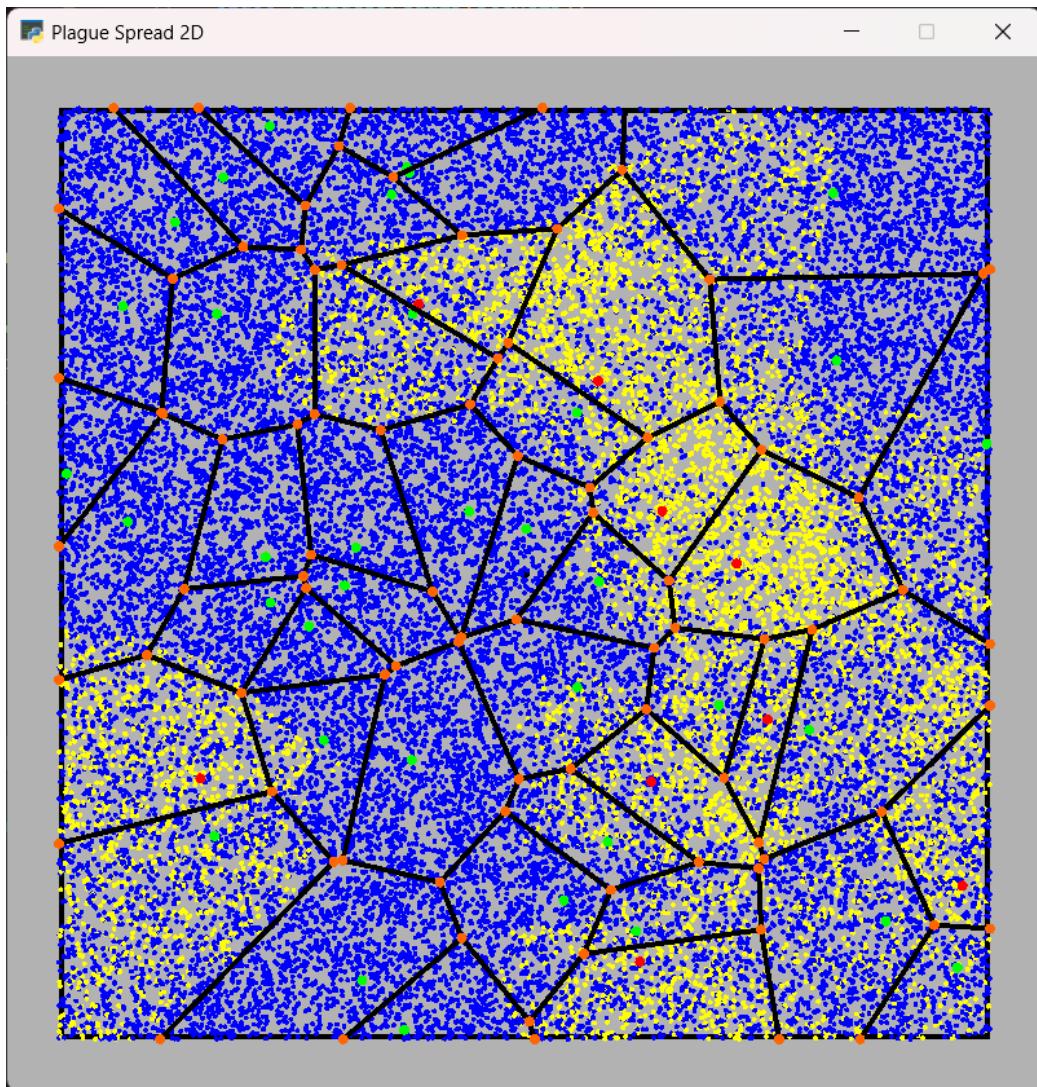
Σχήμα 3.10: 2Δ σκηνή, με 30,000 ανθρώπους και υψηλές πιθανότητες να διαλεχθεί το κοντινότερο πηγάδι, με το διάγραμμα Voronoi.

Όσο μειώνεται η πιθανότητα να επιλεχθεί το κοντινότερο πηγάδι, τα όρια των μολυσμένων περιοχών γίνονται όλο και πιο χαώδη και, μετά από το σημείο που σταματάει να ισχύει $p_1 > p_2$ μεταβιβαζόμαστε στο σενάριο όπου οι άνθρωποι προτιμάνε το δεύτερο κοντινότερό τους πηγάδι. Το ίδιο συμβαίνει και για τη p_3 .



Σχήμα 3.11: 2Δ σκηνή, με 30,000 ανθρώπους και χαμηλές πιθανότητες να διαλεχθεί το κοντινότερο πηγάδι.

Είναι ακόμα ξεκάθαρο ότι υπάρχουν εστίες μολύνσεως, αλλά τα όρια δεν είναι τόσο ευδιάκριτα ποια. Η συγκεκριμένη είκονα έχει πιθανότητες $p_1 = 0.45$, $p_2 = 0.325$, $p_3 = 0.225$. Επίσης παρατηρούμε το φαινόμενο της υψηλής εξάπλωσης της μόλυνσης σε γειτονικές περιοχές, ακόμα κι αν έχουν μη-μολυσμένο πηγάδι ως εστία. Θα μπορούσαμε να θεωρήσουμε ότι αν άνω του 50% του πληθυσμού που περικλείεται σε ένα κελί είναι μολυσμένο, τότε είναι και η ίδια περιοχή μολυσμένη, ανεξαρτήτως της κατάστασης του πηγαδιού.



Σχήμα 3.12: 2Δ σκηνή, με 30,000 ανθρώπους και χαμηλές πιθανότητες να διαλεχθεί το κοντινότερο πηγάδι, με διάγραμμα Voronoi.

Δημιουργούμε χειροκίνητα τα ζητούμενα GIFs, με πιθανότητες που κυμαίνονται μεταξύ στα δύο άκρα που παρουσιάσαμε εδώ.

Έχοντας ολοκληρώσει το Α Μέρος της εργασίας, μπορούμε να προχωρήσουμε στην επέκταση του προγράμματος στις τρεις διαστάσεις, και στο Β Μέρος.

3.2 Μέρος Β

3.2.1 Ερώτημα 4

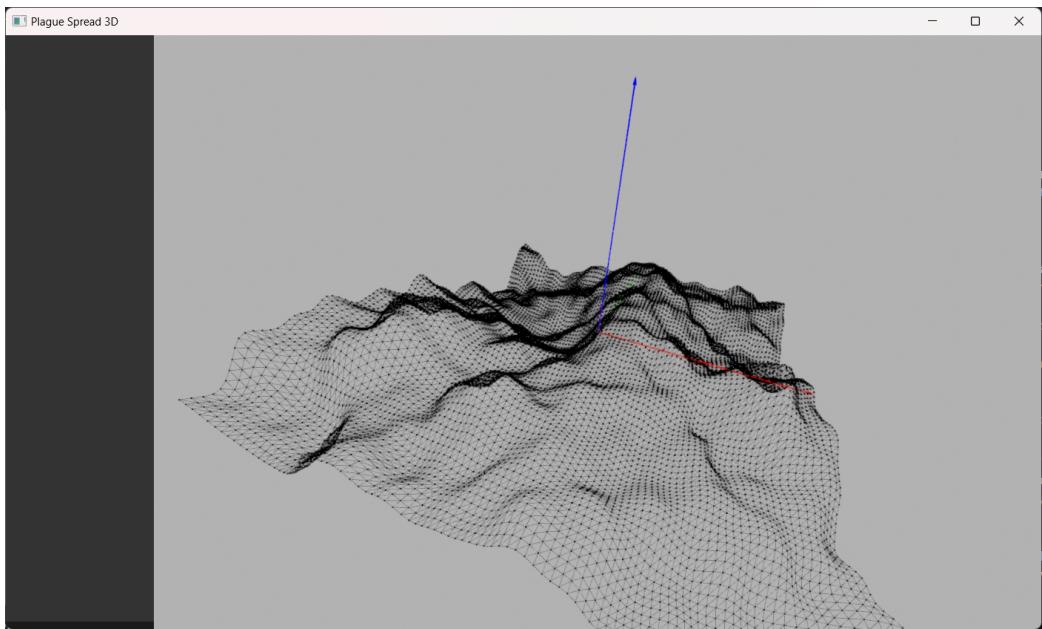
Τη θεμελίωση για την 3Δ σκηνή αποτέλει το επίπεδο πλέγμα που χρειάζεται να κατασκευάσουμε, και ύστερα να ανυψώσουμε ώστε να δώσουμε το στοιχείο του ύψους στη σκηνή μας.

Για να το πετύχουμε αυτό, χρησιμοποιούμε συναρτήσεις της `numpy`, συμπεριλαμβανόμενη και η `linspace`, στην οποία δίνουμε ένα `GRID_SIZE` και μοιράζει τη μία διάσταση σε τόσα σημεία. Στη περίπτωσή μας με ένα τετράγωνο πλέγμα (`grid`) με ισαπέχουνσα (τετράγωνα) σημεία μεταξύ τους, αποτελεί τη διαμέριση και στα `x` και `y`.

Επιλέγουμε θόρυβο τύπου Perlin για την ανύψωση, μέσω της βιβλιοθήκης `noise` της `python`. Πειράζουμε τις παραμέτρους τις συνάρτησης `noise2`, αυξάνοντας τα `octaves` για περισσότερα layers θορύβου, `persistance` που ελέγχει το πλάτος κάθε επόμενου octave, `lacunarity` που ελέγχει τη συχνότητα των διαδοχικών στρωμάτων και το `base` που μπορούμε να αλλάξουμε για να λάβουμε διαφορετικά αποτελέσματα για ίδια είσοδο.

Έτσι, λαμβάνουμε το ύψος των σημείων του πλέγματος.

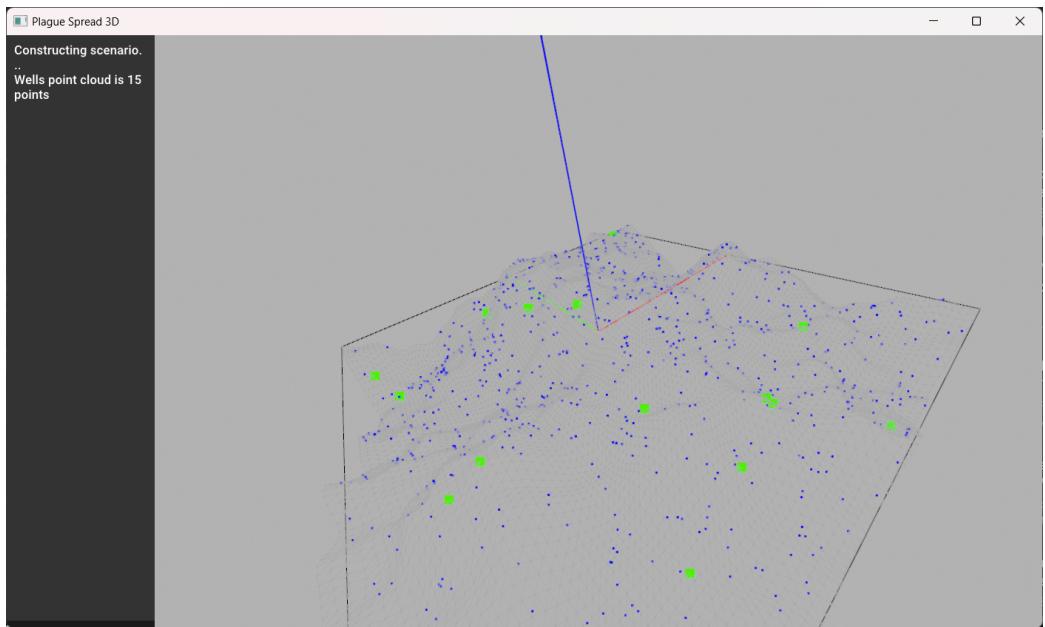
Για να κατασκευάσουμε ένα ολοκληρωμένο πλέγμα (`mesh`), πρέπει να τριγωνοποιήσουμε τα σημεία του `grid`, με έναν απλό αλγόριθμο όπως περιγράφεται στην επεξήγηση της συνάρτησης `triangulate_grid` 5.6, και επομένως έχουμε έτοιμη τη “βάση” της σκηνής μας.



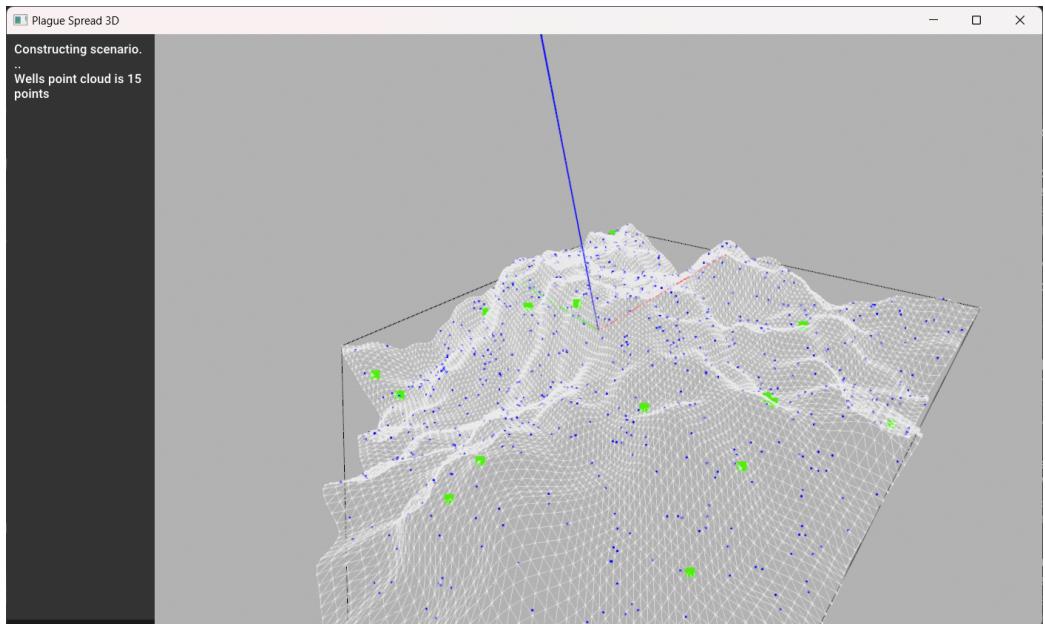
Σχήμα 3.13: 3Δ σκηνή, με τετραγωνικό πλέγμα διαστάσεων 100×100 και $octaves = 4, persistence = 0.5, lacunarity = 2.0, base = 0$.

Χρειάζεται να επαναλάβουμε το Ερώτημα 1 3.1.1, αλλά αυτή τη φορά για τη καινούργια, 3Δ σκηνή μας. Η προσέγγισή μας προσθέτει ένα επιπλέον βήμα σε σχέση με τη 2Δ σκηνή. Δειγματοληπτούμε δισδιάστατα τους ανθρώπους και τα πηγάδια, και κατασκευάζουμε μια συνάρτηση που θα εφαρμόζει τα σημεία στο κατάλληλο ύψος.

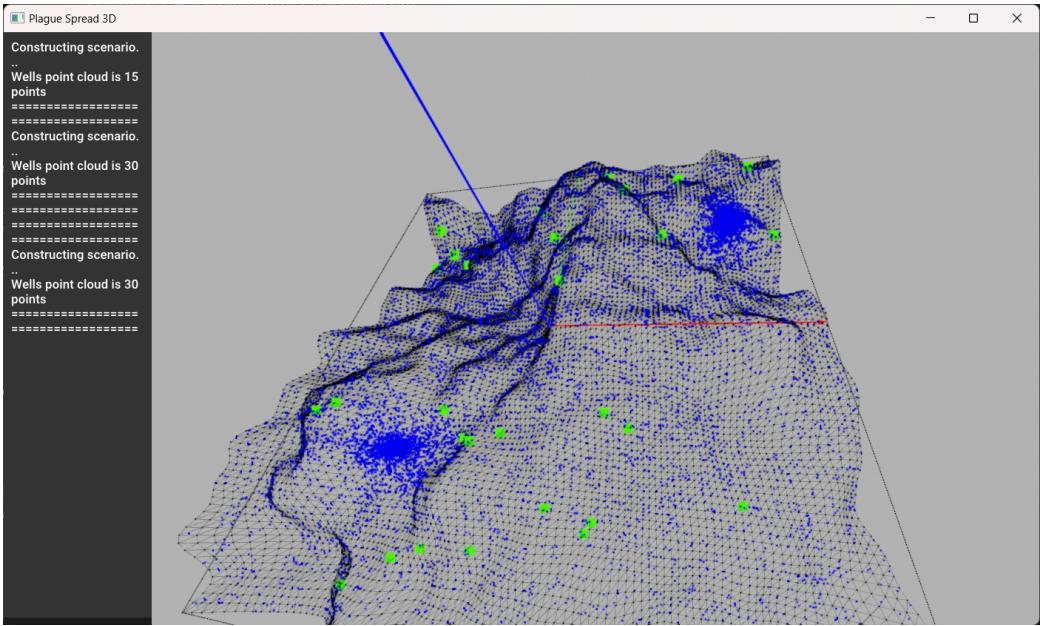
Μια προσέγγιση θα ήταν να τα έχουμε “fixed” σε ένα από τα σημεία του πλέγματος, το οποίο γίνεται περισσότερο αποδεκτό όσο αυξάνεται η ανάλυση του grid. Στη συνάρτησή μας, `adjust_height_of_points`, χρησιμοποιούμε βαρυκεντρική παρεμβολή 4.6 για να βρούμε το ύψος των σημείων για τα οποία ενδιαφερόμαστε. Οπότε, όπως και τη δισδιάστατη σκηνή, καταλήγουμε με πλυθησμό ανθρώπων και μερικά πηγαδιών.



Σχήμα 3.14: 3Δ σκηνή, με πληθυσμούς στο σωστό ύψος.



Σχήμα 3.15: 3Δ σκηνή, με δειγματοληψία και τονισμένο πλέγμα.



Σχήμα 3.16: 3Δ σκηνή, με δειγματοληψία προτεραιότητας και τονισμένο πλέγμα.

3.2.2 Ερώτημα 5

Αφού έχουμε ετοιμάσει το mesh μας, για να επαναλάβουμε τα Ερωτήματα 2 3.1.2 και 3 3.1.3 και να βρούμε το πλησιέστερο πηγάδι για κάθε άνθρωπο, χρειαζόμαστε μια διαφορετική συνάρτηση απόστασης από την ευκλείδια. Αυτό συμβάινει γιατί η ευκλείδια απόσταση περιλαμβάνει διαδρομές που είναι απίθανες διαμέσου του manifold. Για αυτό, ενδιαφερόμαστε για γεωδαισιακές αποστάσεις 4.2, που το μονοπάτι για την ελάχιστη απόσταση μεταξύ δύο σημείων θα χαράζεται επάνω στο mesh μας.

Δεν αποτελεί απλή διαδικασία η εύρεση γεωδαισιακής απόστασης μεταξύ δύο σημείων δοθέντος ενός mesh.

Αν φανταστούμε ένα απλό σενάριο, όπου έχουμε ένα τυχαίο σημείο-αρχή και ένα τυχαίο σημείο-προορισμός πάνω στο πλέγμα, πρέπει να κάνουμε traverse το ίδιο το πλέγμα βήμα-βήμα, προσθέτοντας θεμελιώδεις αποστάσεις έως ότου φτάσουμε στον στόχο. Η θεμελιώδης μονάδα του πλέγματος είναι τα τρίγωνα που το αποτελούνε, οπότε για να προσπαθήσουμε να προσεγγίσουμε τις γεωδαισιακές αποστάσεις πρέπει να μεταπηδήσουμε από τρίγωνο σε τρίγωνο, και να αποθηκεύουμε τις ευκλείδιες αποστάσεις μεταξύ τους.

Για λόγους που περιγράφονται στο κεφάλαιο Θεωρίας 4, το πρόβλημα που αντιμετωπίζουμε εδώ λύνεται από τον Δυϊκό Γράφο 4.3 του πλέγματος, οπότε χρειάζεται υπολογισμός των βάρων κέντρων των τριγώνων και η αποθήκευσή τους.

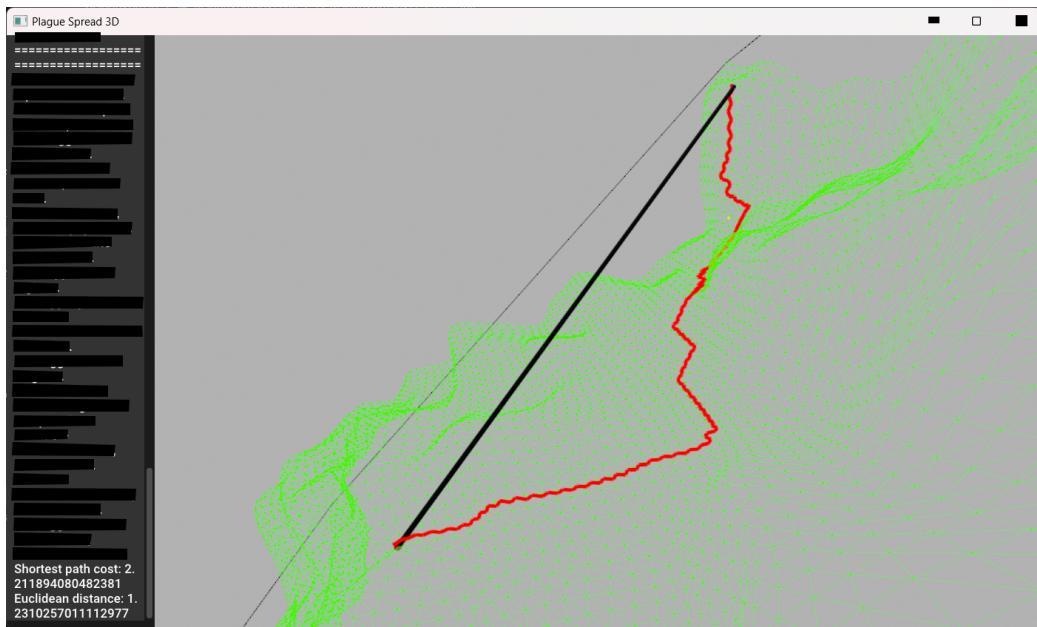
Το ερώτημα παραμένει όμως, ότι υπάρχει πάνω από ένας τρόπος να φτάσεις από το ένα σημείο σε ένα άλλο στο επίπεδο/manifold, και η γεωδαισιακή απόσταση (όπως και η ευκλείδια) είναι η συντομότερη διαδρομή μεταξύ τους. Το πρόβλημα είναι επί ουσίας η αναζήτηση βέλτιστης διαδρομής - ελάχιστοποίηση κόστους, για το οποίο θα βασιστούμε στον αλγόριθμο του Dijkstra 4.4 για να λύσουμε.

Η εφαρμογή του απαιτεί τον γράφο σε μια αναπαράσταση κατανοητή από τη κλάση που κατασκευάζουμε 5.3, οπότε βρίσκουμε το adjacency matrix των τριγώνων, για κάθε γειτνίαση υπολογίζουμε την (ευκλείδια) απόσταση των αντίστοιχων κεντροειδών και καταλήγουμε με έναν πίνακα διαστάσεων $N^2 \times N^2$, αν N το μέγεθος του grid στη μία διάσταση.

Είναι εύκολα κατανοητό ότι, ο αριθμός γειτόνων που μπορεί να έχει ένα κεντροειδές/τρίγωνο είναι πολύ, πολύ μικρότερος από N^2 . Κάνουμε χρήση των δομές των αραιών πινάκων (sparse matrices) που προσφέρονται από τη scipy, και συγκεκριμένα το format LIL (List of Lists) για τη κατασκευή, και CSR (Compressed Sparse Row) για χρήση. Όπου απαιτείται (για οπτικοποίηση ή για πράξεις στο σύνολο των πινάκων), μπορούμε να μετατρέψουμε το αραιό μητρώο σε πλήρη πίνακα.

Οσο μεγενθύνονται οι διαστάσεις της σκηνής μας (του πλέγματος), τίθονται όλοι και περισσότερα ζητήματα αποδοτικότητας και αποτελεσματικότητας 6.1. Ενώ ο αλγόριθμος του Dijkstra βρίσκει με βεβαιότητα το ελάχιστο κόστος, είναι εξαιρετικά ακριβός να τρέξει σε runtime περιβάλλον, πόσο μάλλον για μεγάλα πλέγματα. Ως επίλυση αυτού, αναπτύσσουμε μια λογική υπό-ενότητα στο πρόγραμμά μας που ελέγχει για την υπάρξη και την αποθήκευση των πινάκων που υπολογίζουμε, ώστε το κόστος στη χρονική πολυπλοκότητα να μεταφερθεί στη χωρική. Μέσα στους πίνακες που αποθηκεύουμε είναι και ο πίνακας ελαχίστων κόστων που παράγεται από τη κλάση Dijkstra 5.3, ο οποίος είναι συμμετρικός, και με απλό indexing στη γραμμή i και στήλη j , μπορούμε να βρούμε την γεωδαισιακή απόσταση μεταξύ του κεντροειδούς i και j .

Για $N = 100$, το μέγεθος του πίνακα βέλτιστων αποστάσεων φτάνει τα 2.86 GB, δεν πρόκειται για αραιό πίνακα, και είναι στη πρώτη εκτέλεση χρονικά ακριβός να υπολογιστεί (κάτι που θα συζητήσουμε αναλυτικότερα παρακάτω 6.1), από εκεί και πέρα όμως, είναι στη διάθεσή μας συγκριτικά γρήγορα (κόστος φόρτωσης/loading) και έως ότου αλλάξουμε το μέγεθος του πλέγματος (GRID_SIZE).

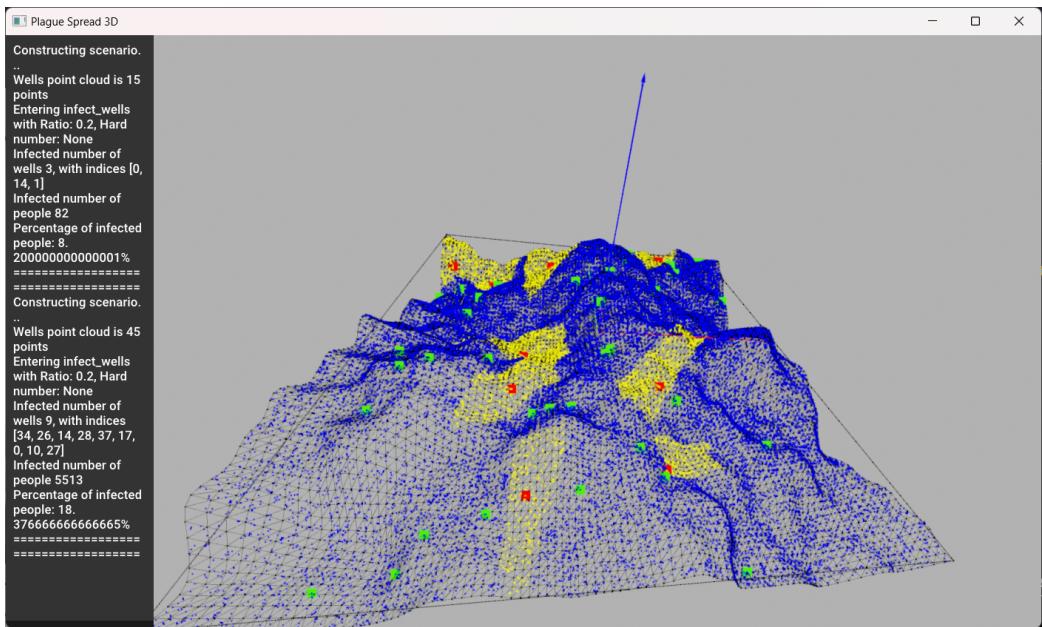


Σχήμα 3.17: 3Δ σκηνή, ευκλείδια απόσταση (μαύρο) μεταξύ αρχής και τέλους σε σύγκριση με την γεωδαισιακή απόσταση (κόκκινο) δια του πλέγματος.

Μπορούμε εύκολα να διαπιστώσουμε ότι οι άνθρωποι θα ταξιδεύουν σύμφωνα με τη γεωδαισιακή απόσταση στο κοντινότερό τους πηγάδι, έναντι της ευκλείδιας.

Είμαστε πλέον σε θέση να επαναλάβουμε το δεύτερο ερώτημα της εργασίας με γεωδαισιακές αποστάσεις. Για να βρούμε το κοντινότερο γεωδαισιακά πηγάδι σε έναν άνθρωπο, κατασκευάζουμε τη συνάρτηση

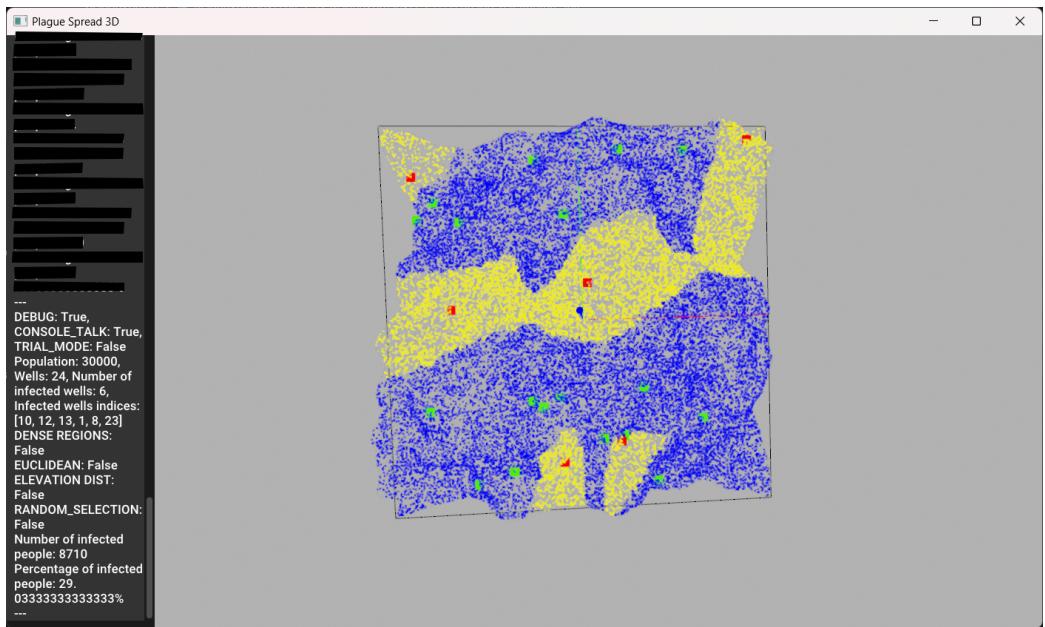
`find_closest_well_index_geodesic`, η οποία έχει 3 δουλειές: Να βρει το τρίγωνο στο οποίο ανήκει ο άνθρωπος και να υπολογίσει την ευκλείδια απόσταση μεταξύ του σημείου και του κεντροειδούς, να κάνει το ίδιο για όλα τα πηγάδια, και τέλος να αθρόσει τις ευκλείδιες αποστάσεις με την προ-αποθηκευμένη απόσταση των 2 βάρων κέντρων και να επιλέξει την χαμηλότερη τιμή. Για να βρίσκουμε γρήγορα και αποτελεσματικά τα βάρη κέντρων, κατασκευάζουμε ένα k-d Tree 4.5, και κάνουμε τη παραδοχή ότι για ένα τετραγωνικό grid με ισαπέχοντα σημεία, το τρίγωνο στο οποίο ανήκει ένα τυχαίο σημείο, και συνεπώς το κεντροειδές που του αντιστοιχεί, είναι το κεντροειδές εκείνο το οποίο βρίσκεται πιο κοντά του. Αξιοποιείται το broadcasting και το vectorization της numpy για άλλη μια φορά, για να αποφευχθούν εμφωλευμένα for loops και για ταχύτατη ταξινόμηση (argsort).



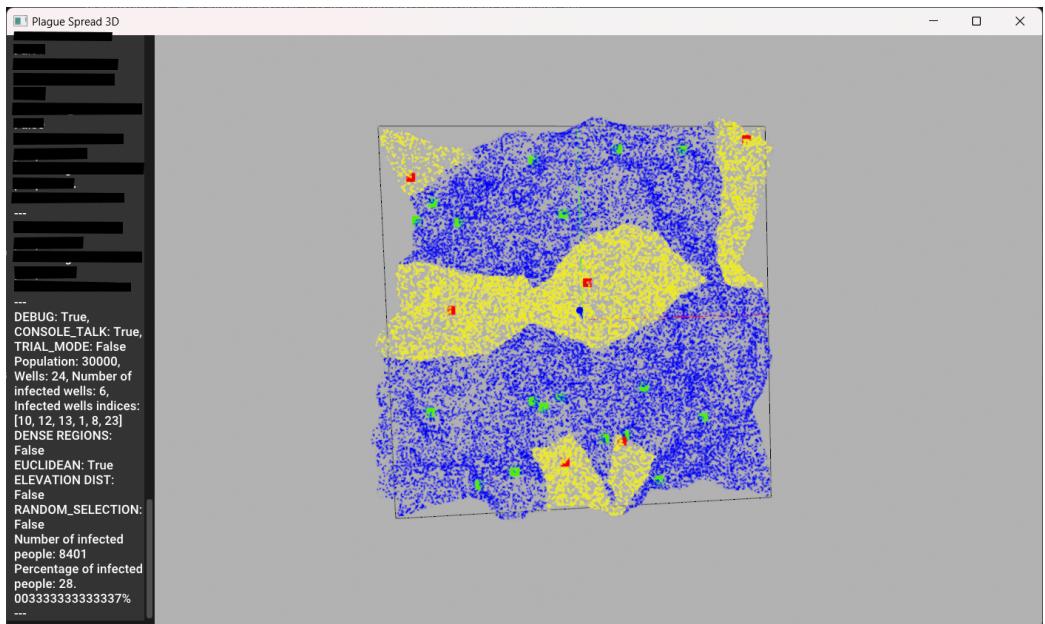
Σχήμα 3.18: 3Δ σκηνή, με γεωδαισιακές αποστάσεις για μόλυνση του πληθυσμού.

Μπορούμε πάλι να παρατηρήσουμε ξεκάθαρες περιοχές επιρροής των πηγαδιών/εστιών. Αυτά αποτελούνται τα αντίστοιχα κελιά Voronoi του διαγράμματος που θα υπολογιζόταν υπό γεωδαισιακές αποστάσεις.

Για καλύτερη σύγκριση, δύνονται τη δυνανότητα στο πρόγραμμα να υπολογίσει και ευκλείδια τις αποστάσεις, μέσω της συνάρτησης `find_closest_well_index_kd_tree`. Όπως υποδηλώνει το όνομα, κατασκευάζουμε ένα k-d Tree 4.5 των πηγαδιών για γρήγορη αναζήτηση κοντινότερου γείτονα, και πράγματι επιταχύνει πολύ την εκτέλεση του προγράμματος για απαιτητικά νούμερα, όμως είναι κατάλληλο μόνο για ευκλείδιες αποστάσεις.



Σχήμα 3.19: Γεωδαισιακή εύρεση μολυσμένων περιοχών και ανθρώπων.

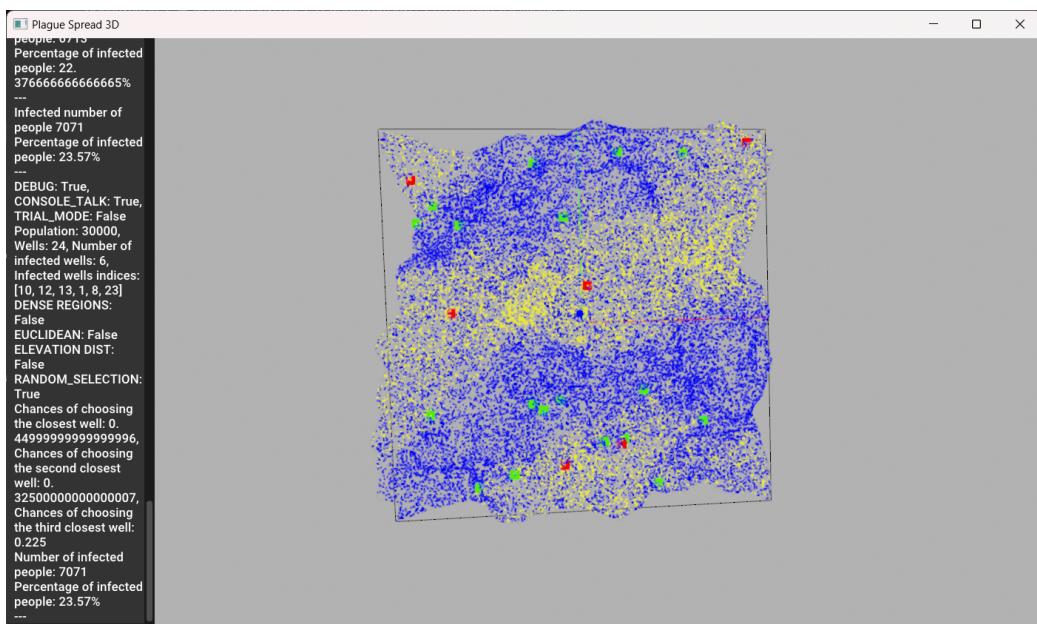


Σχήμα 3.20: Ευκλείδια εύρεση μολυσμένων περιοχών και ανθρώπων.

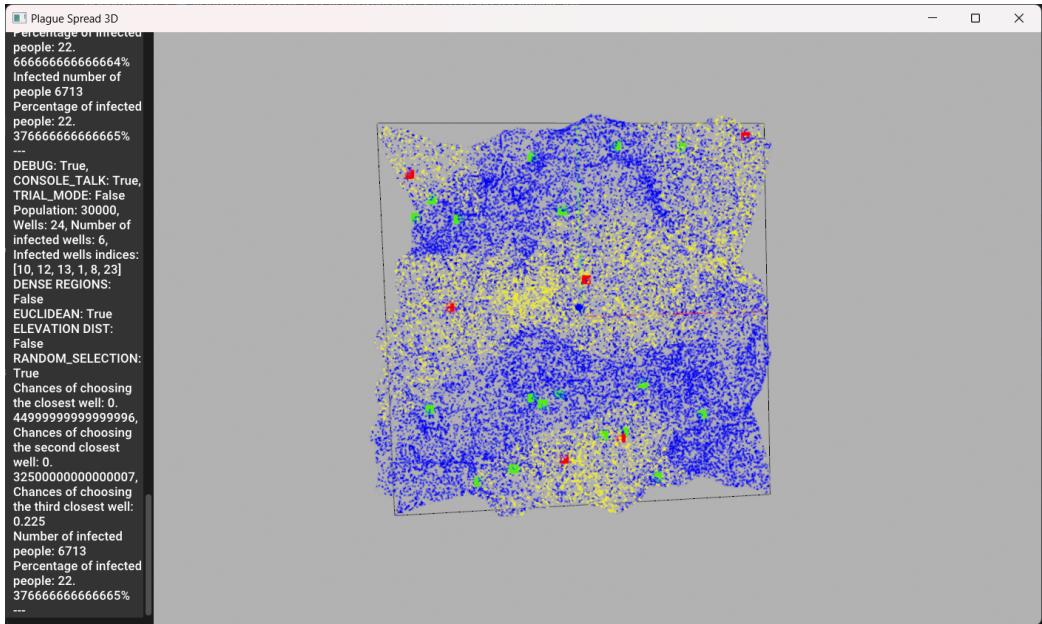
Σε γενικές γραμμές, λόγω της φύσεως του πλέγματός μας, η γεωδαισιακές αποστάσεις είναι πιο κοστοφόρες από τις ευκλείδιες. Για την ακρίβεια, οι ευκλείδιες αποστάσεις αποτελούνε γενικότερα μια “υποτίμηση” της απόστασης που θα

χρειαστεί κανείς να ταξιδέψει, αποτελεί ένα κάτω όριο, αφού δεν λαμβάνει υπόψιν ούτε τον χώρο, ούτε πιθανά εμπόδια - μόνο τις συντεταγμένες της αρχής και τις συντεταγμένες του τέλους.

Για να επαναλάβουμε το Ερώτημα 3.3.1.3, ακολουθούμε παρόμοια φιλοσοφία και αναπτύσσουμε τις συναρτήσεις: `find_stochastic_infected_geodesic`, `get_geodesic_distances_from_to_many`, οι οποίες χρησιμοποιούν το μητρώο με τα ελάχιστα κόστη και το δέντρο με τα κεντροειδή, και επιπλέον την `find_stochastic_infected_euclidean`, η οποία χρησιμοποιεί μόνο το k-d Tree για τα πηγάδια. Βρίσκουμε τα 3 κοντινότερα πηγάδια, και πιθανοτικά διαλέγουμε μεταξύ τους.



Σχήμα 3.21: Γεωδαισιακή πιθανοτική εύρεση μολυσμένων περιοχών και ανθρώπων, με χαμηλές πιθανότητες να επιλεχθεί το εγγύτερο πηγάδι.



Σχήμα 3.22: Ευκλείδια πιθανοτική εύρεση μολυσμένων περιοχών και ανθρώπων, με χαμηλές πιθανότητες να επιλεχθεί το εγγύτερο πηγάδι.

Περίπου στις ίδιες γραμμές με τη δισδιάστατη σκηνή κυμαίνονται οι παρατηρήσεις μας 3.1.3.

Κατασκευάζουμε το GIF χειροκίνητα, και παραδίδεται μαζί με την εργασία.

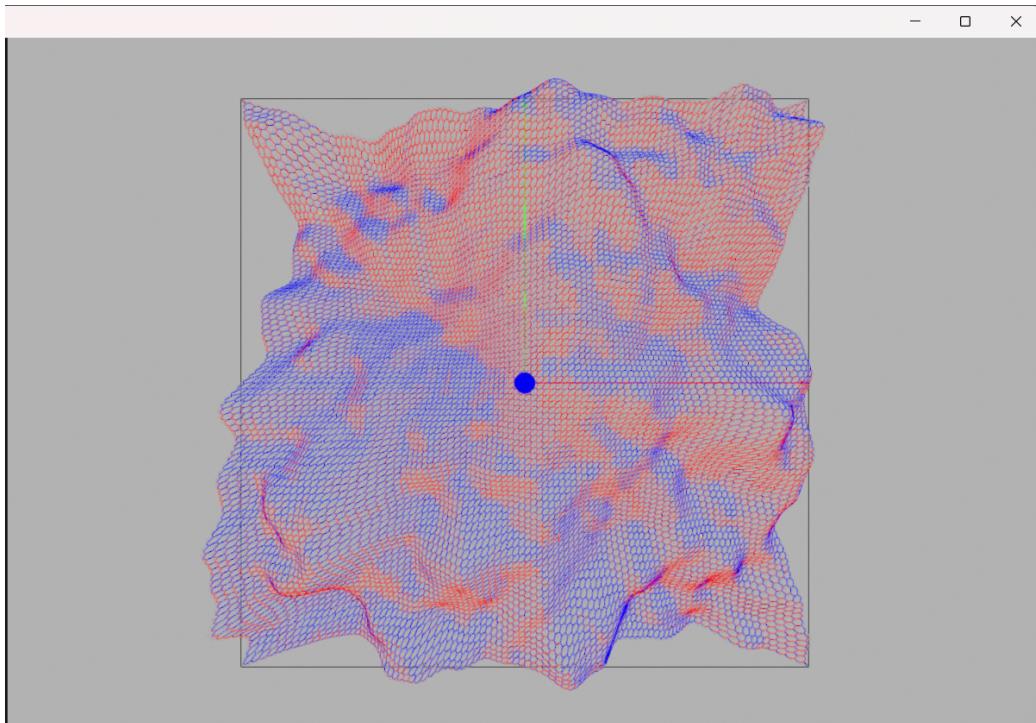
3.2.3 Ερώτημα 6

Βήμα-βήμα, μετακινούμαστε σε μια πιο πραγματική προσέγγιση της απόστασης που θα χρειαστεί να ταξιδέψει ένας άνθρωπος για να φτάσει στο πηγάδι του. Σε αυτό το ερώτημα, ζητάτε να λάβουμε υπόψιν την ανύψωση του πλέγματος, το elevation, και να ορίσουμε ανηφόρες και κατηφόρες της απόστασης.

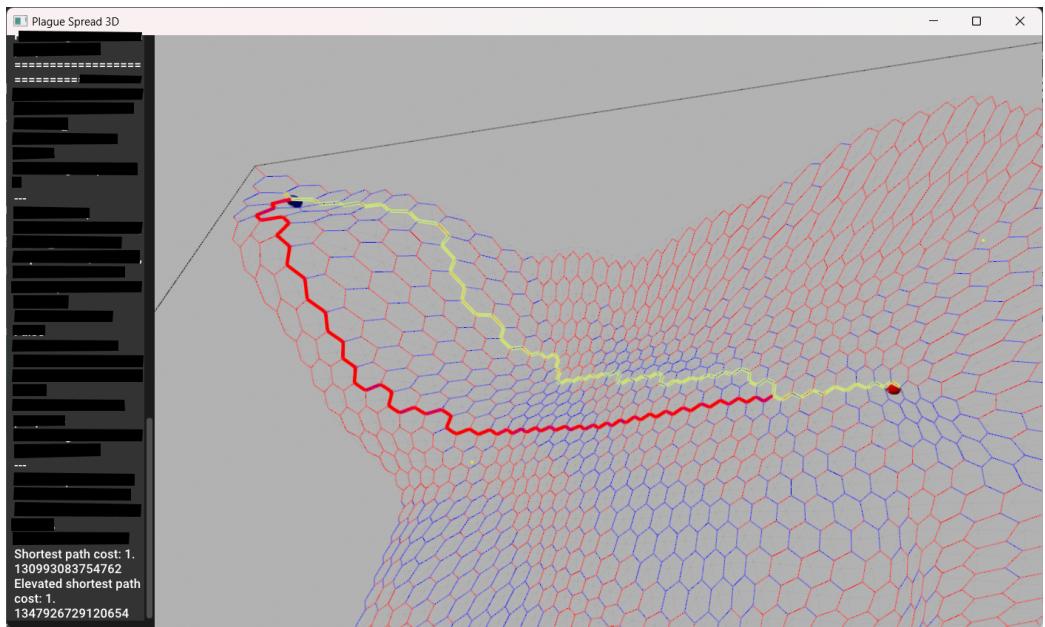
Στη προσέγγιση που ακολουθήσαμε, θεωρούμε ότι η ανηφόρες αναπτύσσονται καθώς το ύψος y και το βάθος z αυξάνονται, και κατηφόρες υπάρχουν όσο μειώνεται η συντεταγμένη z , αλλά το y συνεχίζει να αυξάνεται. Αυτό ουσιαστικά λειτουργεί ως μια μεταφορική σάρωση προς τον αύξοντα άξονα y , και ορίζει αντιστοίχως τις ανηφόρες και τις κατηφόρες.

Έχοντας αυτόν τον ορισμό υπόψιν, υπολογίζουμε την οριζόντια απόσταση μεταξύ των 2 σημείων που εξετάζουμε, και μετά πραγματοποιούμε τους δύο ελέγχους που αναφέραμε πριν. Αν πρόκειται για ανηφόρα, επιφέρουμε ένα σημαντικό κόστος, και πολλαπλασιάζουμε την κάθετη απόσταση με ένα βάρος. Αν πρόκειται για κατηφόρα, πολλαπλασιάζουμε με ένα ελαφρύ κέρδος, και μετά τα προσθέτουμε όλα μάζι για να λάβουμε τη τελική απόσταση.

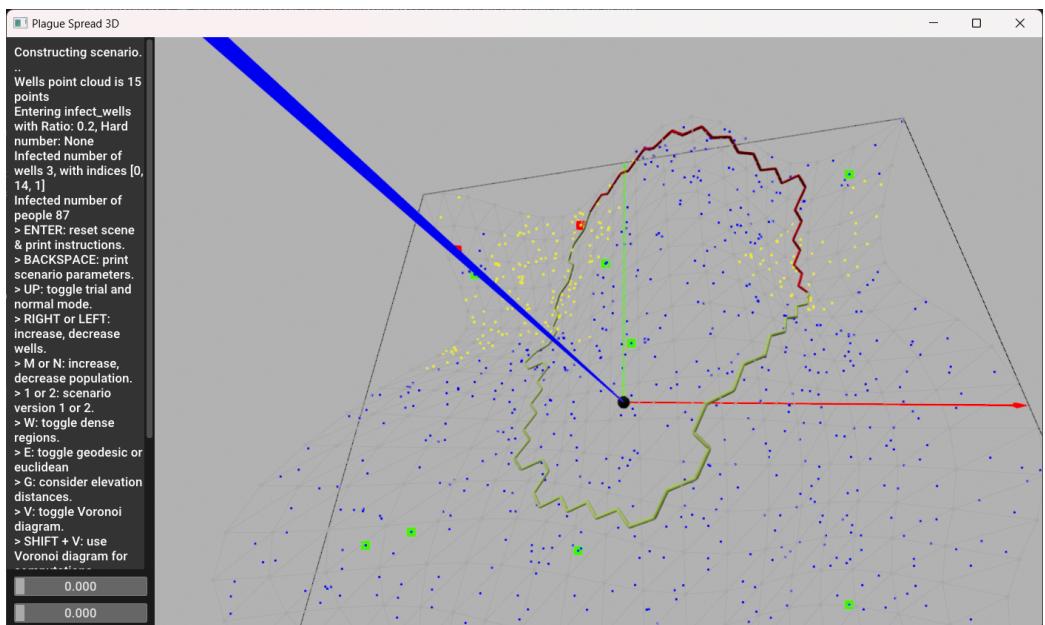
Αποθηκεύοντας τις διασυνδέσεις κεντροειδών που ορίζουν ανηφόρα ή κατηφόρα, μπορούμε να τις ζωγραφίσουμε και να τις δείξουμε στη σκηνή μας. Το κόκκινο αντιστοιχεί σε ανηφόρα και κόστος διάσχισης και το μπλε σε κατηφόρα και κέρδος διάσχισης.



Σχήμα 3.23: Ο χάρτης ανηφόρων-κατηφόρων. Μπορούμε να παρατηρήσουμε ότι αύξηση z κατά αύξηση y οδηγεί σε ανηφόρα (κόκκινο) και μείωση z σε κατηφόρα (μπλε).



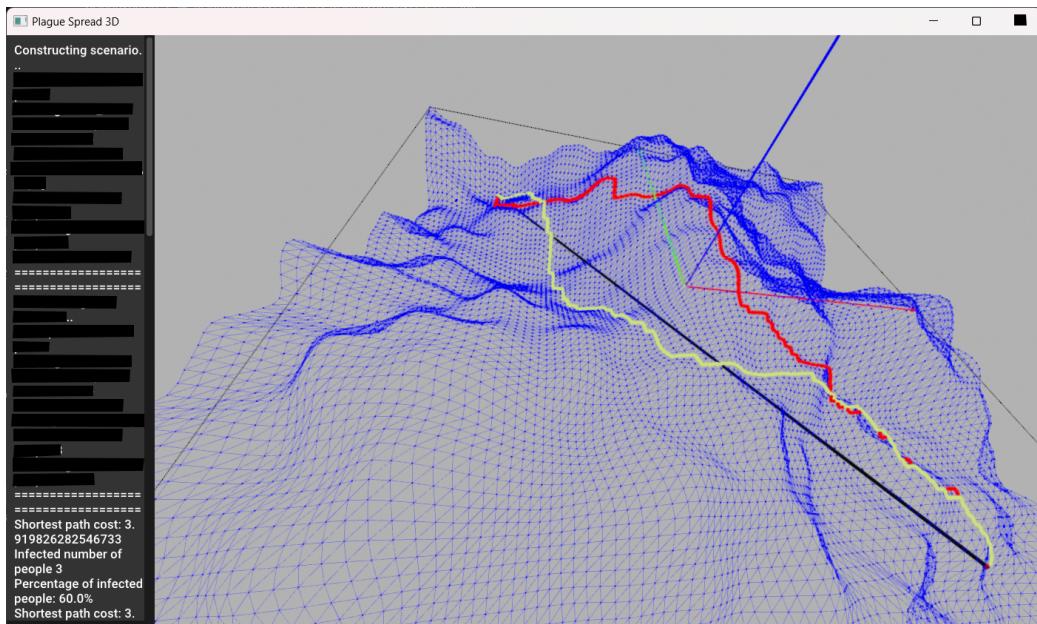
Σχήμα 3.24: Η απόσταση που λαμβάνει υπόψιν το elevation (πράσινο) προτιμά τις μπλε (κατηφόρα) διασυνδέσεις, ενώ η απλή γεωδαισιακή (κόκκινο) τα αντιμετωπίζει ισοδυναμα.



Σχήμα 3.25: Για μικρότερο πλέγμα, ενδιαφέρον διχοτομή στην απόφαση βέλτιστου μονοπατιού.

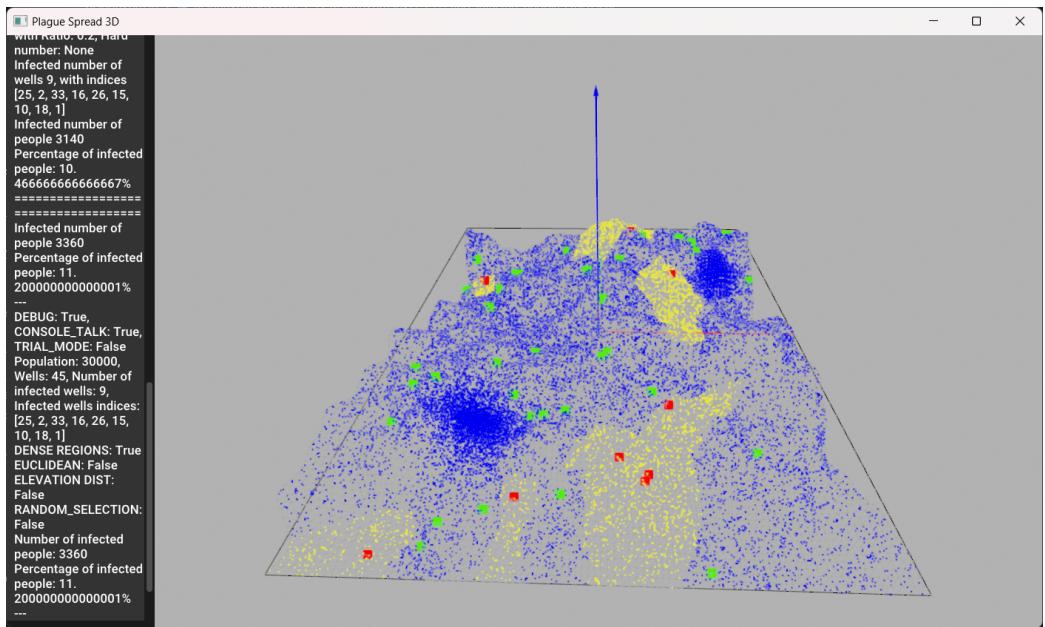
Όπως και για τη γεωδαισιακή απόσταση χωρίς αναφορά στο elevation, πρέπει να αποθηκεύσουμε τους αντίστοιχους πίνακες αποστάσεων και ελαχίστων κόστων, που πρόκειται για άλλη μία χωρική επιβάρυνση των 2.86 GB.

Παραθέτουμε και μια ολοκληρωμένη σύγκριση των 3 αποστάσεων στο συγκεκριμένο πλέγμα. Η ενκλείδια διασχίζει το πλέγμα για να φτάσει μονοσήμαντα στο σημείο-προορισμό, η απλή γεωδαισιακή έχει τη βέλτιστη διαδρομή της ως αυτή που διανύει τα λιγότερα κεντροειδή, και η γεωδαισιακή με ανύψωση επιχειρεί να διανύει κατηφόρες (που μειώνουν την απόστασή τους), ή αν δεν τα καταφέρει, να διατηρήσει το υψόμετρο στο οποίο βρίσκεται.

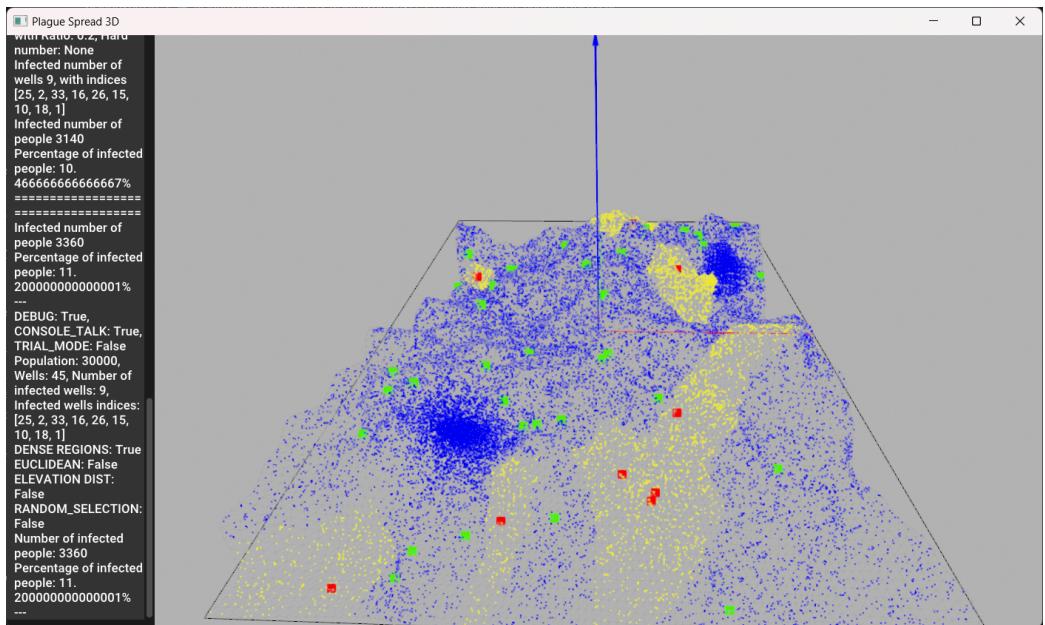


Σχήμα 3.26: Η απόσταση που λαμβάνει υπόψιν το elevation (κίτρινο) προτιμά τις μπλε (κατηφόρα) διασυνδέσεις, ενώ η απλή γεωδαισιακή (κόκκινο) τα αντιμετωπίζει ισοδυναμα.

Για να λάβουμε τις καινούργιες περιοχές μολύνσεως, απλώς στον υπολογισμό γεωδαισιακής απόστασης χρησιμοποιούμε τον πίνακα βέλτιστων κόστων με elevation, έναντι του απλού πίνακα.



Σχήμα 3.27: Περιοχές με μολυσμένο πληθυσμό με γεωδαισιακή συνάρτηση αποστάσεως λαμβάνοντας υπόψιν το elevation.

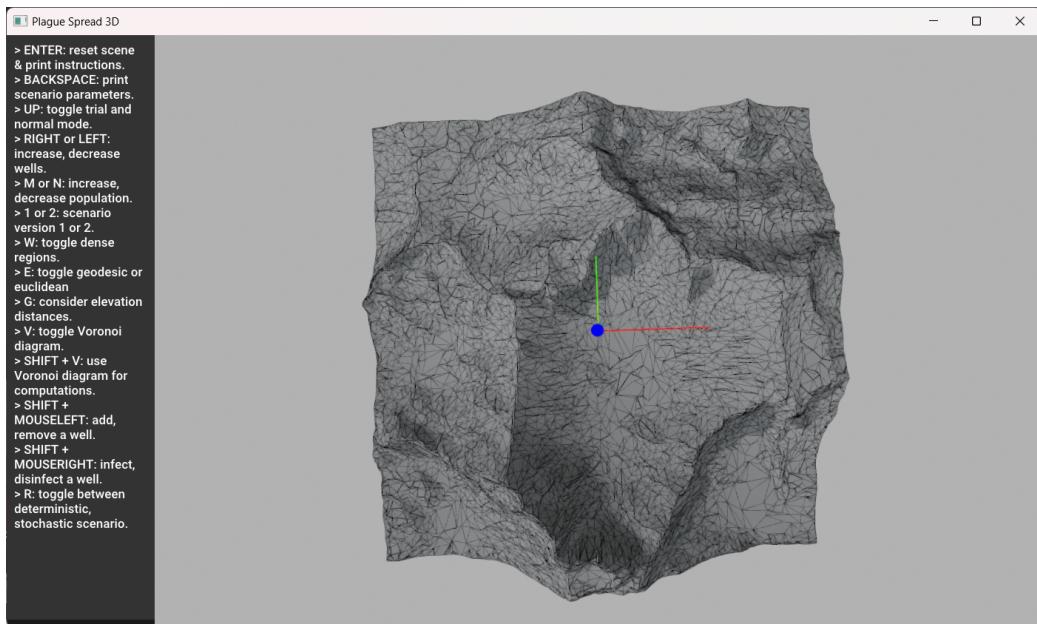


Σχήμα 3.28: Περιοχές με μολυσμένο πληθυσμό με απλή γεωδαισιακή συνάρτηση αποστάσεως.

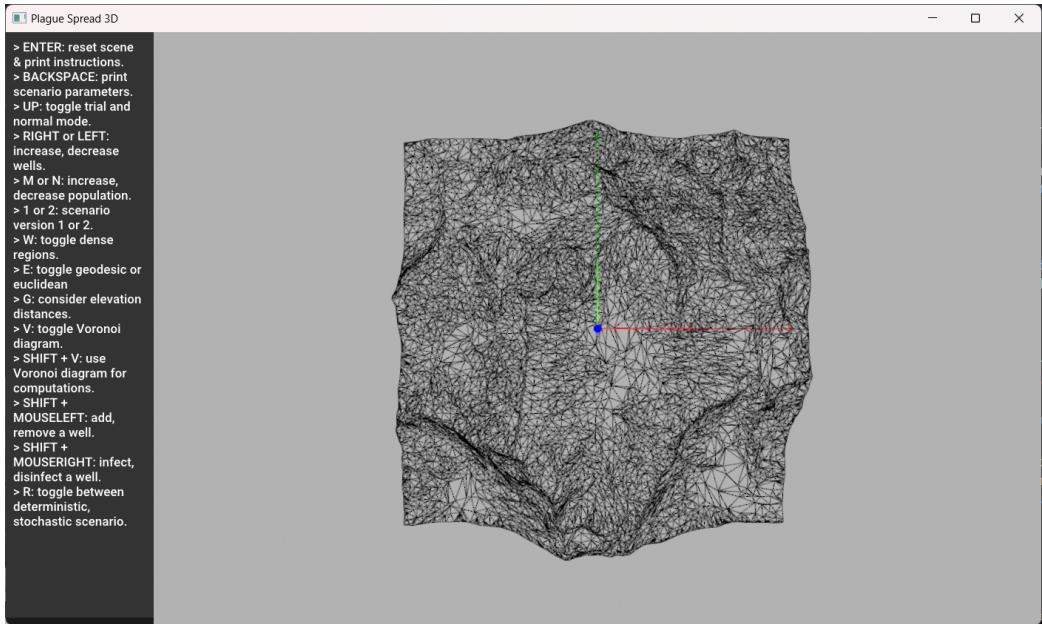
3.2.4 Ερώτημα 7

Σε αυτό το ερώτημα, ζητείται η χρήση ενός χάρτη, πιθανώς πραγματικού, και να εξατομικεύσουμε τον γράφο του οποίου αντιστοιχεί ώστε να λαμβάνει υπόψιν πραγματικά γεωγραφικά στοιχεία. Για παράδειγμα, διαδρομές λιγότερης απόστασης λόγω γραμμών τραίνων, ή διάσχιση ορεινών περιοχών.

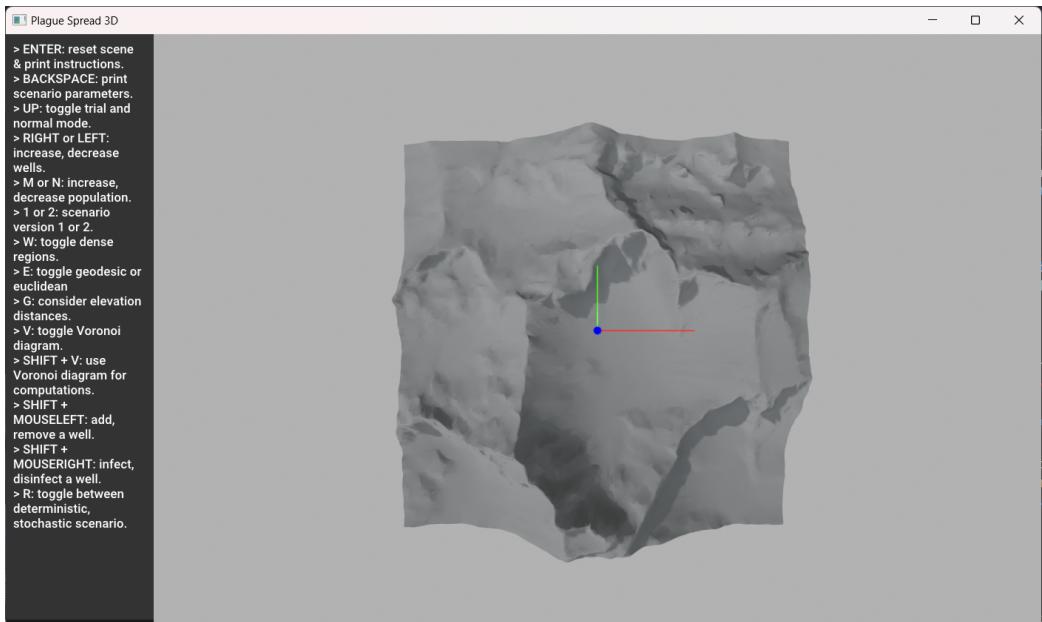
Για τον σκοπό αυτό, θα φορτώσουμε ένα ξεχωριστό μοντέλο terrain, το οποίο θα αντικαταστήσει το ανυψωμένο grid, και έτσι μεταβαίνουμε στην τρίτη έκδοση του προγράμματός μας, το 3DTerrain.



Σχήμα 3.29: Το φορτωμένο mesh - terrain.



Σχήμα 3.30: Η τριγωνοποίηση του φορτωμένου mesh - terrain.

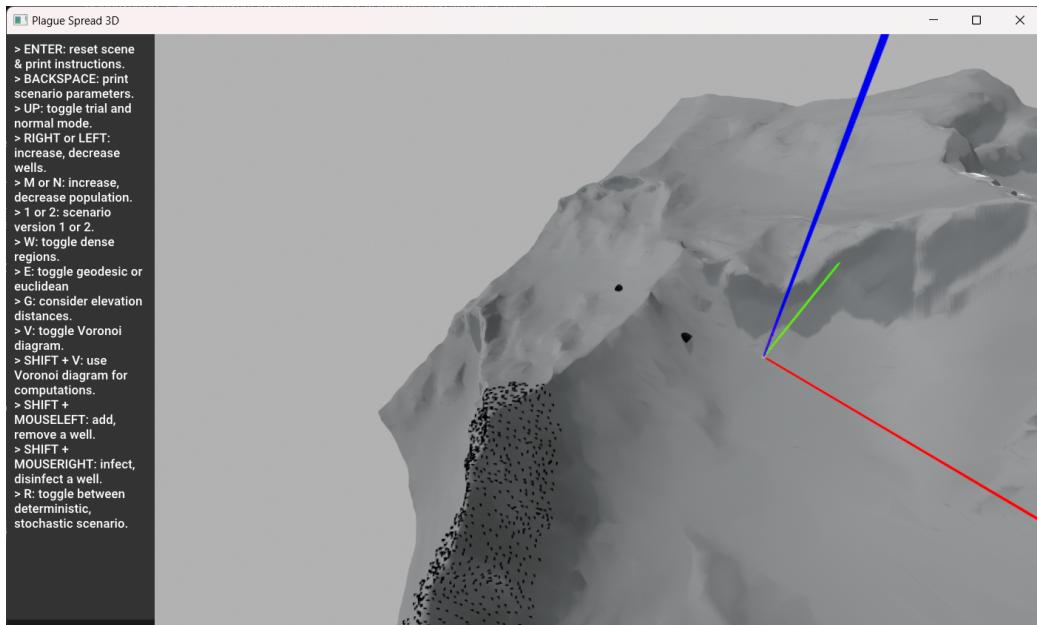


Σχήμα 3.31: Γεμισμένο χωρίς τρίγωνα mesh.

Δεν μπορούμε πλέον να βασιστούμε στην υπόθεση για το τρίγωνο στο οποίο ανήκει ένα σημείο που κάναμε στο Ερώτημα 5 3.2.2, οπότε στον καινούργιο

υπολογισμό της βαρυκεντρικής παρεμβολής 4.6 για τα ύψη πρέπει να κάνουμε αναζήτηση και μέσα σε ποιο τρίγωνο ανήκει ένα σημείο. Υπολογίζουμε επίσης τους πίνακες που αντιστοιχούν στο κατινούργιο πλέγμα και τους αποθηκεύουμε.

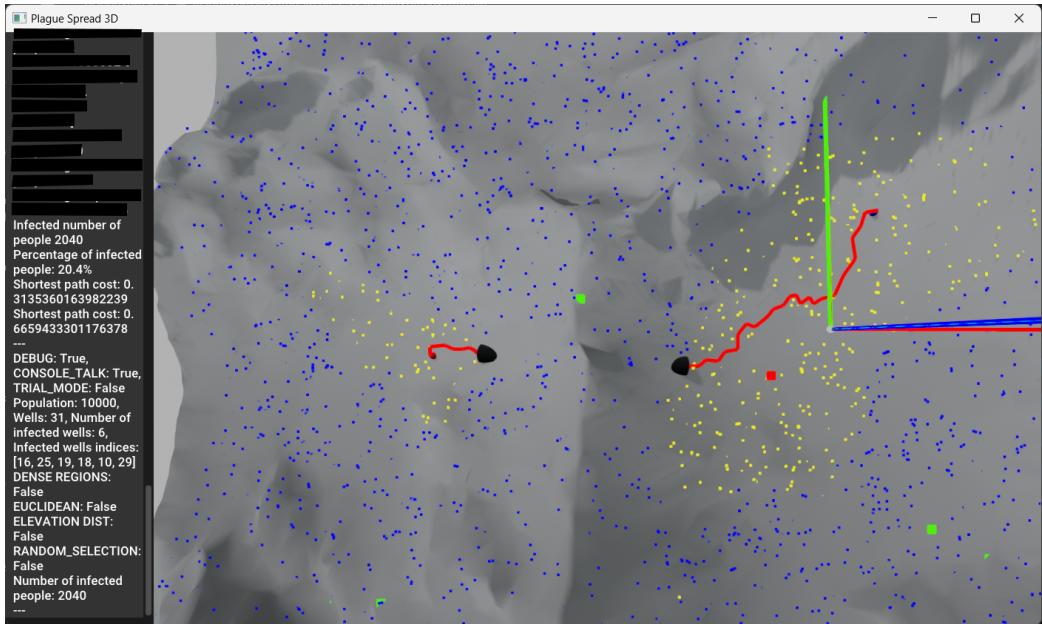
Λόγω έλλειψης χρόνου, αναπτύχθηκαν μόνο δύο γεωγραφικά χαρακτηριστικά: ένα τούνελ που θα κάνει γειτονικά δύο παραδοσιακά μη-γειτονικά τρίγωνα, και μια βουνώδης περιοχή - εμπόδιο, δια της οποίας είναι απίθανη η διέλευση



Σχήμα 3.32: Γεωγραφικά χαρακτηριστικά της σκηνής: τούνελ και ορεινή, μη-προσβάσιμη περιοχή με τα μαύρα σημεία.

Για να το καταφέρουμε αυτό, στην κατασκευή των πινάκων adjacency, centroid_distances, κλπ., ελέγχουμε αν εξετάζουμε το συγκεκριμένο index κεντροειδούς, και εφαρμόζουμε αναλόγως τις αλλαγές - ιδιαιτερότητες. Για τον σκοπό αυτό, αναπτύσσουμε την συνάρτηση `setup_geography_specifics`, η οποία αποθηκεύει τα indexes που μας ενδιαφέρουν.

Για το τούνελ, στη συνάρτηση που κατασκευάζει το adjacency matrix, αν εξετάσουμε έναν από τους δύο δείκτες κεντροειδών που αφορά το τούνελ, τότε θέτουμε τις αντίστοιχες θέσεις ως 1, κανοντάς τες με αυτόν τον τρόπο γειτονικές. Όταν έρθει η ώρα να υπολογίσουμε την απόσταση μεταξύ τους, μειώνουμε ιδιαίτερα το κόστος απόστασης, ώστε να εκφράσουμε τις επιπτώσεις μιας σήραγγας στο ταξίδι.



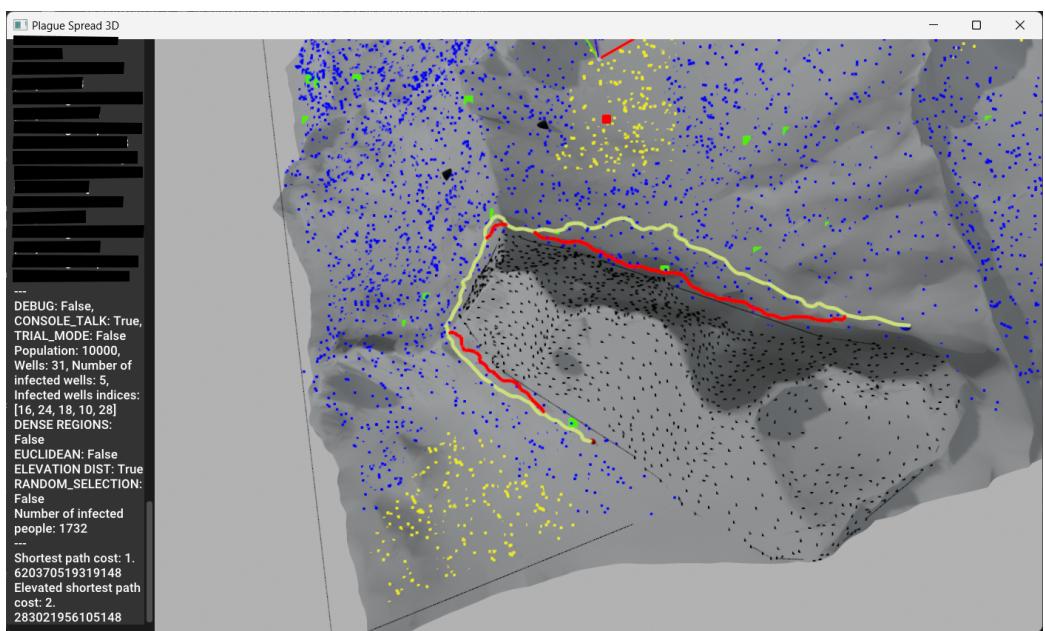
Σχήμα 3.33: Η επιρροή του τούνελ στην εύρεση αποστάσεων. Το μολυσμένο πηγάδι επεκτείνει την ευρεση σε δυνάμει περιοχή του μέσα από το μη-μολυσμένο.

Για την ορεινή περιοχή αντιθέτως, δεν είναι αυθαίρετη η τοποθεσία της όπως η σήραγγα (αν και θα μπορούσε αν το επιθυμούσαμε). Βρίσκουμε τα 50 ψηλότερα σημεία του mesh μας, αφαιρούμε τα σημεία που δεν βρίσκονται στη κοντινή περιοχή του υψηλότερου σημείου (σε ακτίνα 0.3), και για κάθε σημείο που μας απομένει, εντοπίζουμε τα κεντροειδή του που βρίσκονται εντός μιας ακτίνας 0.2. Έτσι καταλήγουμε με μια ομάδα σημείων κεντροειδών που αναπαριστούνε μια ορεινή περιοχή.

Καθώς κατασκευάζουμε το μητρώο αποστάσεων, αν ελέγχουμε ένα από αυτά τα ορεινά κεντροειδή, θέτουμε την απόσταση τους μεταξύ των γειτόνων τους σε άπειρο. Παρόμοιες προσεγγίσεις θα αποτελούσαν η αφαίρεση γειτνίασης στο adjacency matrix, ή και η πολύ μεγάλη απόσταση έναντι του απείρου για τους γείτονες.

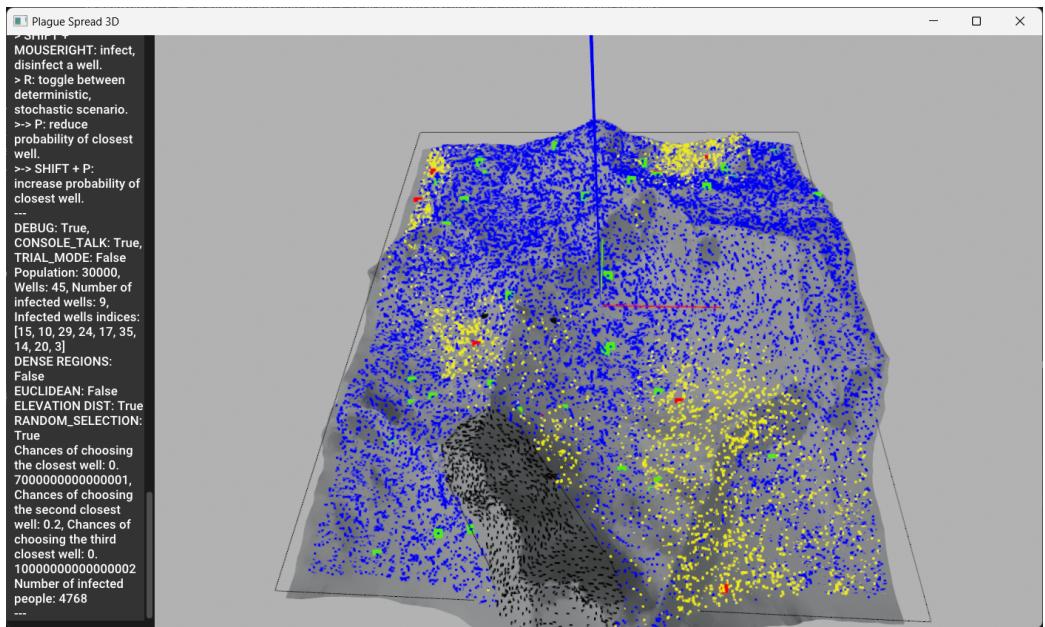
Όμως, στην δημιουργία των πληθυσμών μέσω των μεθόδων `createRandom`, `createRandomWeighted`, ο αλγόριθμος δεν έχει τρόπο να γνωρίζει την απαγορευμένη περιοχή. Επομένως, κατασκευάζουμε άλλη μία συνάρτηση `move_from_impassable_areas`, η οποία υπολογίζει το κυρτό περίβλημα των σημείων της ορεινής περιοχής, εντοπίζει μέσω της συνάρτησης `is_inside_polygon_2d` 5.7 τα σημεία που βρίσκονται εντός του, και τα μετακινεί αλλού τυχαία εντός του bounding box.

Έτσι τελικά, έχουμε κατασκευάσει την αδιάβατη, ορεινή περιοχή μας.

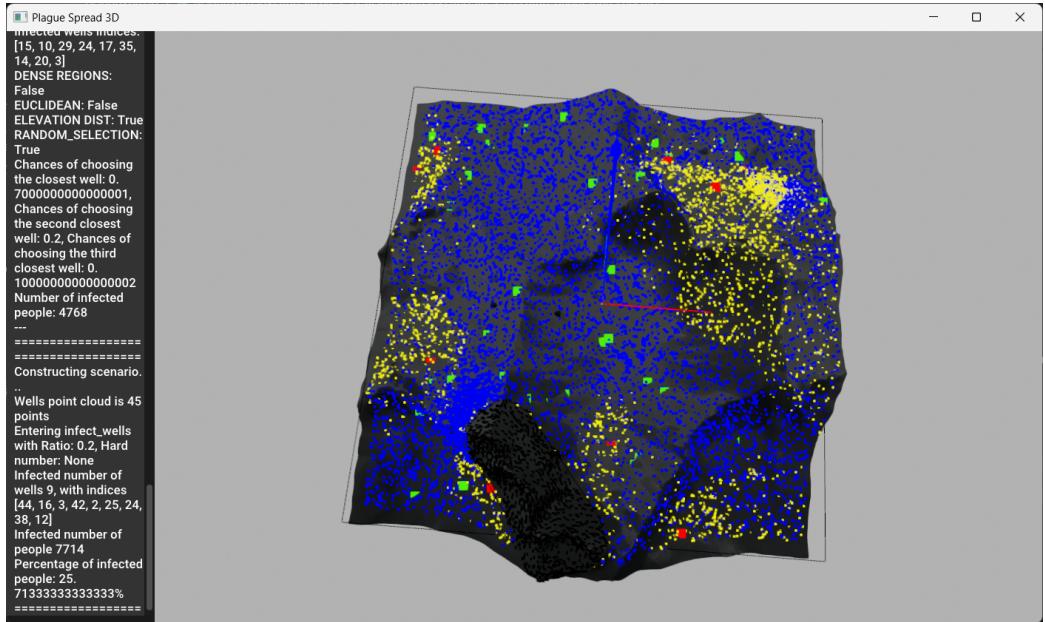


Σχήμα 3.34: Οι άνθρωποι δεν μπορούν να ταξιδέψουν δια της ορεινής περιοχής. Κόκκινη η απλή γεωδαισιακή και κίτρινη η απόσταση με elevation

Ολοκληρώνουμε λοιπόν το πρόγραμμά μας, αν και υπάρχει ακόμη πολύς χώρος για επέκταση και βελτίωση του 6.2.



Σχήμα 3.35: Η τελική σκηνή μας με τυχαιότητα επιλογής, αποστάσεις που λαμβάνουν υπόψιν την ανύψωση του terrain, και ιδιαιτερότητες της γεωγραφίας.



Σχήμα 3.36: Η τελική σκηνή μας με τυχαιότητα επιλογής, αποστάσεις που λαμβάνουν υπόψιν την ανύψωση του terrain, ιδιαιτερότητες της γεωγραφίας και αστικές περιοχές.

4 Θεωρία/Θεωρητικό Υπόβαθρο

4.1 Διάγραμμα Voronoi

...

4.1.1 Αλγόριθμος του Fortune

Ο αλγόριθμος που επιχειρήσαμε να εφαρμόσουμε, είναι στη πραγματικότητα αυτός που περιγράφεται στο βιβλίο των Mark de Berg et al 10, ο οποίος διαφέρει λίγο από την αρχική του επινόηση, και ταιριάζει με αυτή των Guibas, Stolfi 10.

Ο αλγόριθμος είναι sweep line αλγόριθμος, και τα βήματά του είναι τα εξής:

Algorithm VORONOIDIAGRAM(P)

Input. A set $P := \{p_1, \dots, p_n\}$ of point sites in the plane.

Output. The Voronoi diagram $\text{Vor}(P)$ given inside a bounding box in a doubly-connected edge list \mathcal{D} .

1. Initialize the event queue \mathcal{Q} with all site events, initialize an empty status structure \mathcal{T} and an empty doubly-connected edge list \mathcal{D} .
2. **while** \mathcal{Q} is not empty
3. **do** Remove the event with largest y -coordinate from \mathcal{Q} .
4. if the event is a site event, occurring at site p_i
5. **then** HANDLESITEEVENT(p_i)
6. **else** HANDLECIRCLEEVENT(γ), where γ is the leaf of \mathcal{T} representing the arc that will disappear
7. The internal nodes still present in \mathcal{T} correspond to the half-infinite edges of the Voronoi diagram. Compute a bounding box that contains all vertices of the Voronoi diagram in its interior, and attach the half-infinite edges to the bounding box by updating the doubly-connected edge list appropriately.
8. Traverse the half-edges of the doubly-connected edge list to add the cell records and the pointers to and from them.

HANDLESITEEVENT(p_i)

1. If \mathcal{T} is empty, insert p_i into it (so that \mathcal{T} consists of a single leaf storing p_i) and return. Otherwise, continue with steps 2–5.
2. Search in \mathcal{T} for the arc α vertically above p_i . If the leaf representing α has a pointer to a circle event in Ω , then this circle event is a false alarm and it must be deleted from Ω .
3. Replace the leaf of \mathcal{T} that represents α with a subtree having three leaves. The middle leaf stores the new site p_i and the other two leaves store the site p_j that was originally stored with α . Store the tuples $\langle p_j, p_i \rangle$ and $\langle p_i, p_j \rangle$ representing the new breakpoints at the two new internal nodes. Perform rebalancing operations on \mathcal{T} if necessary.
4. Create new half-edge records in the Voronoi diagram structure for the edge separating $\mathcal{V}(p_i)$ and $\mathcal{V}(p_j)$, which will be traced out by the two new breakpoints.
5. Check the triple of consecutive arcs where the new arc for p_i is the left arc to see if the breakpoints converge. If so, insert the circle event into Ω and add pointers between the node in \mathcal{T} and the node in Ω . Do the same for the triple where the new arc is the right arc.

HANDLCIRCLEVENT(γ)

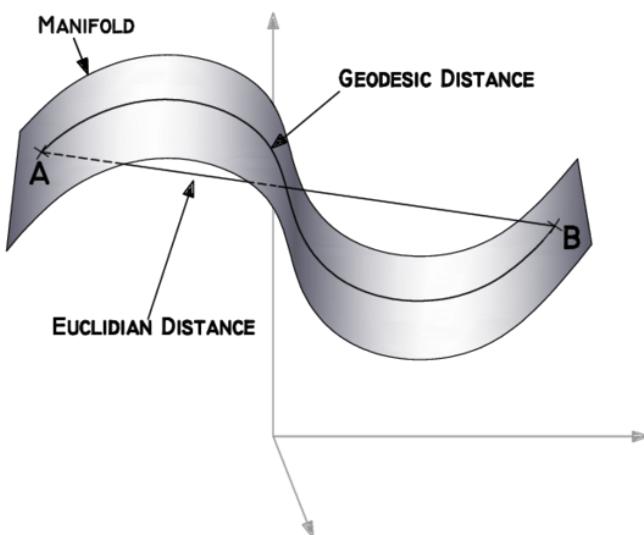
1. Delete the leaf γ that represents the disappearing arc α from \mathcal{T} . Update the tuples representing the breakpoints at the internal nodes. Perform rebalancing operations on \mathcal{T} if necessary. Delete all circle events involving α from Ω ; these can be found using the pointers from the predecessor and the successor of γ in \mathcal{T} . (The circle event where α is the middle arc is currently being handled, and has already been deleted from Ω .)
2. Add the center of the circle causing the event as a vertex record to the doubly-connected edge list \mathcal{D} storing the Voronoi diagram under construction. Create two half-edge records corresponding to the new breakpoint of the beach line. Set the pointers between them appropriately. Attach the three new records to the half-edge records that end at the vertex.
3. Check the new triple of consecutive arcs that has the former left neighbor of α as its middle arc to see if the two breakpoints of the triple converge. If so, insert the corresponding circle event into Ω and set pointers between the new circle event in Ω and the corresponding leaf of \mathcal{T} . Do the same for the triple where the former right neighbor is the middle arc.

Σχήμα 4.1: Ψευδοκώδικας για την υλοποίηση του scan line αλγορίθμου.

...

4.2 Γεωδαισιακές Αποστάσεις

an intrinsic dimension of two in a three-dimensional Euclidean space.



Σχήμα 4.2: Διαφορά ευκλίδειας και γεωδαισιακής απόστασης.

...

4.3 Δυϊκός Γράφος

- Τι είναι δυϊσμός, σημείο/ευθεία - Τριγωνοποίηση Delaunay δυϊκό του διαγράμματος Voronoi - Γιατί διάνυση κεντροειδών κι όχι ακμές τριγώνων.

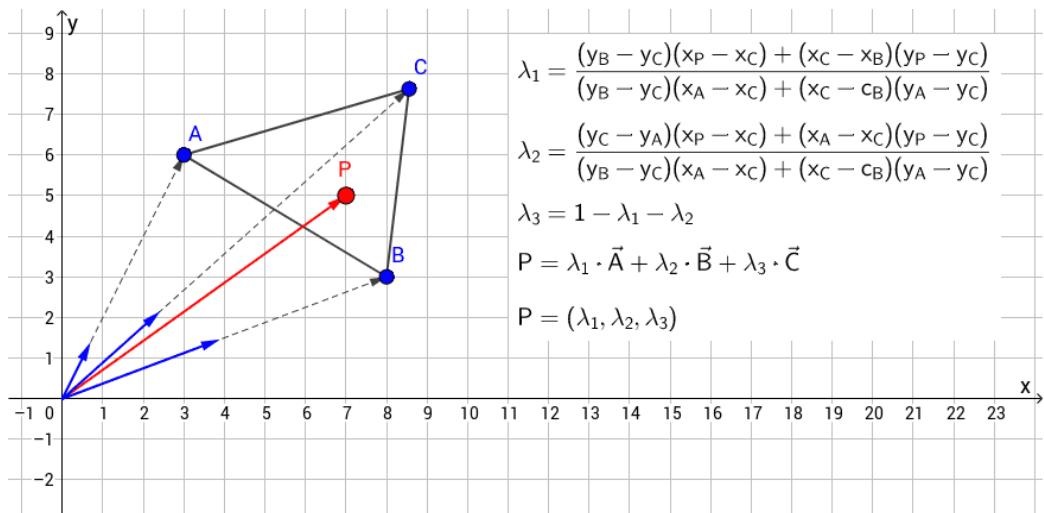
4.4 Αλγόριθμος του Dijkstra - Αναζήτηση Βέλτιστου Μονοπατιού

- Ιστορικά - Βήματα Αλγορίθμου - Σημασία, χρήση

4.5 Χωρική αναζήτηση, k-d Trees

...

4.6 Βαρυκεντρική Παρεμβολή



Σχήμα 4.3: Περιγραφή των βαρυκεντρικών συντεταγμένων και της παρεμβολής 10

- Περιγραφή, εξισώσεις

4.7 Κυρτό περίβλημα

5 Υλοποιήσεις Υπολογιστικής Γεωμετρίας στον Κώδικα

5.1 Η κλάση Voronoi

...

5.2 Η κλάση KdNode

...

5.3 Η κλάση Dijkstra

...

5.4 Το αρχείο *GeometryUtils*

...

5.5 Η μέθοδος `createRandomWeighted`

...

5.6 Η συνάρτηση `triangulate_grid`

...

5.7 Η συνάρτηση is_inside_polygon_2d

...

6 Αποτελέσματα & Συζήτηση

6.1 Αποδόσεις

Το σύστημα στο οποίο αναπτύχθηκε η εργασία είναι ένα ASUS VivoBook, 11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.42GHz, 16 GB RAM, τύπου ενσωματωμένης κάρτας γραφικών Intel(R) Iris(R) Xe Graphics. Όλες οι παρατηρήσεις πέρι αποδοτικότητας λαμβάνουν χώρα σε αντό το πλαίσιο υπολογιστικού συστήματος.

6.1.1 Υπολογισμός Μητρώων

Για τη 2Δ σκηνή, μπορούμε να πούμε ότι το πρόγραμμα λειτουργεί αρκετά αποδοτικά. Δεν υπάρχουν ιδιαίτερες απαιτήσεις στη μνήμη, το διάγραμμα Voronoi υπολογίζεται γρήγορα λόγω των βελτιστοποιήσεων της scipy και χειρίζεται εύκολα και γρήγορα πληθυσμούς της τάξης 1,000 και 10,000. Στους 30,000 ανθρώπους και 45 πηγάδια, γίνονται αισθητές οι απαιτήσεις στο πεδίο του χρόνου και καθυστερεί η εύρεση του μολυσμένου πληθυσμού περίπου 6 δευτερόλεπτα.

Για τη 3Δ σκηνή, οι απαιτήσεις του προγράμματος εξαρτώνται σε μεγάλο βαθμό από το μέγεθος του grid που θα επιλέξουμε. Για ένα πλέγμα χαμηλής ανάλυσης GRID_SIZE = 20, οι πίνακες που απαιτούνται υπολογίζονται πολύ γρήγορα, καθώς και για την δημιουργία των πίνακα αποστάσεων με αναφορά στο elevation.

```

FFngine (64 bits) created at 000002385F9DC050 (threading is enabled)
FFngine resolved backend: OpenGL
Checking if the grid file exists...
Checking if the grid is the same as the stored grid...
Stored grid loaded in 0.010783195495605469 seconds.
The instance grid is the same as the stored grid? False
Grid has changed, storing new one...
-----
Saving the centroids...
Centroids saved in 0.0 seconds.
Shape of the centroids array: (722, 3)
Creating the KD-Tree for centroids...
KD-Tree for centroids created in 0.019975900650024414 seconds.
Adjacency matrix: calculating and storing it...
Calculating adjacency matrix: 100%|██████████| 1121/1121 [00:00<00:00, 89751.56it/s]
Saving the adjacency matrix...
Adjacency matrix saved in 0.007986307144165039 seconds.
Need to calculate the distances matrix...
Calculating distances: 100%|██████████| 722/722 [00:00<00:00, 5338.09it/s]
Shape of the distances matrix: (722, 722)
Distances matrix calculated in 0.1352543830871582 seconds.
Saving the distances matrix...
Distances matrix saved in 0.008359999057617188 seconds.
Calculating the shortest paths matrix...
Calculating shortest paths: 100%|██████████| 722/722 [00:01<00:00, 412.93it/s]
Shortest paths matrix calculated in 1.7549092769622803 seconds.
Saving the shortest paths matrix...

```

Σχήμα 6.1: Υπολογισμός και αποθήκευση των πινάκων για τετράγωνο πλέγμα μεγέθους 20×20 .

```

Calculating the elevation distance matrix...
Calculating elevation distances: 100%|██████████| 722/722 [00:00<00:00, 3601.29it/s]
Saving the uphill and downhill indices matrices...
Shape of the elevation distance matrix: (722, 722)
Elevation distance matrix calculated in 0.2027289867401123 seconds.
Saving the elevation distance matrix...
Elevation distance matrix saved in 0.0014805793762207031 seconds.
Calculating the shortest paths matrix...
Calculating shortest paths: 100%|██████████| 722/722 [00:01<00:00, 407.38it/s]
Elevated shortest paths matrix calculated in 1.7818431854248047 seconds.
Saving the elevated shortest paths matrix...
Shortest paths matrix saved in 0.0498356819152832 seconds.
Shape of the elevated shortest paths matrix: (722, 722)

```

Σχήμα 6.2: Υπολογισμός και αποθήκευση των πινάκων πέρι elevation για τετράγωνο πλέγμα μεγέθους 20×20 .

Οι πιο χωρικά κοστοφόροι πίνακες που αποθηκεύονται, είναι αυτοι των ελαχίστων κόστων (*shortest_paths.npy*, *elevated_shortest_paths.npy*) με μέγεθος των 3.97 MB. Οι λοιποί (adjacency, κεντροειδών, ...) είναι της τάξης μερικών KB, κυρίως λόγο στη sparse δομή για πίνακες που χρησιμοποιούμε. Αποθηκεύονται σε συμπιεσμένο format *.npz*, για ένα συνολικό μεγέθος όλων των αρχείων 8.01 MB.

Ένα τέτοιο πλέγμα είναι σχετικά χαμηλής ανάλυσης. Ένα πιο λογικό μέγεθος πλέγματος λαμβάνουμε άμα θέσουμε GRID_SIZE = 60.

```

FFngine (64 bits) created at 0000017A62288050 (threading is enabled)
FFngine resolved backend: OpenGL
Checking if the grid file exists...
Checking if the grid is the same as the stored grid...
Stored grid loaded in 0.009412527084350586 seconds.
The instance grid is the same as the stored grid? False
Grid has changed, storing new one...
-----
Saving the centroids...
Centroids saved in 0.003921985626220703 seconds.
Shape of the centroids array: (6962, 3)
Creating the KD-Tree for centroids...
KD-Tree for centroids created in 0.06142020225524902 seconds.
Adjacency matrix: calculating and storing it...
Calculating adjacency matrix: 100% | 10561/10561 [00:00<00:00, 95605.93it/s]
Saving the adjacency matrix...
Adjacency matrix saved in 0.015778303146362305 seconds.
Need to calculate the distances matrix...
Calculating distances: 100% | 6962/6962 [00:00<00:00, 7230.62it/s]
Shape of the distances matrix: (6962, 6962)
Distances matrix calculated in 0.9675607681274414 seconds.
Saving the distances matrix...
Distances matrix saved in 0.021934986114501953 seconds.
Calculating the shortest paths matrix...
Calculating shortest paths: 100% | 6962/6962 [02:32<00:00, 45.65it/s]
Shortest paths matrix calculated in 153.05073046684265 seconds.
Saving the shortest paths matrix...
Shortest paths matrix saved in 0.6600704193115234 seconds.
Shape of the shortest paths matrix: (6962, 6962)
Grid set up
-----
```

Σχήμα 6.3: Υπολογισμός και αποθήκευση των πινάκων για τετράγωνο πλέγμα μεγέθους 60×60 .

```

Calculating the elevation distance matrix...
Calculating elevation distances: 100% | 6962/6962 [00:02<00:00, 3301.45it/s]
Saving the uphill and downhill indices matrices...
Shape of the elevation distance matrix: (6962, 6962)
Elevation distance matrix calculated in 2.1502981185913086 seconds.
Saving the elevation distance matrix...
Elevation distance matrix saved in 0.01613616943359375 seconds.
Calculating the shortest paths matrix...
Calculating shortest paths: 100% | 6962/6962 [02:45<00:00, 42.14it/s]
Elevated shortest paths matrix calculated in 165.83991312980652 seconds.
Saving the elevated shortest paths matrix...
Shortest paths matrix saved in 0.9930527210235596 seconds.
Shape of the elevated shortest paths matrix: (6962, 6962)
```

Σχήμα 6.4: Υπολογισμός και αποθήκευση των πινάκων πέρι elevation για τετράγωνο πλέγμα μεγέθους 60×60 .

Μπορούμε να δούμε ότι οι δύο πιο υπολογιστικά ακριβοί πίνακες μπορούν να χρειαστούν έως και 2:45 λεπτά για να υπολογιστούν, και για να αποθηκευτούν περίπου 1 επιπλέον δευτερόλεπτο.

ΣΗ χωρική και χρονική πολυπλοκότητα που απαιτεί η κατασκευή αυτών των πινάκων αυξάνεται δραματικά για υψηλότερης ανάλυσης πλέγματα, τουλάχιστον με τις υπολογιστικές ικανότητες που συζητήσαμε 6.1. Αν θέσουμε GRID_SIZE = 110, οι υπολογισμοί τους είναι οι εξής:

```

FFngine (64 bits) created at 000002839BBC7050 (threading is enabled)
FFngine resolved backend: OpenGL
Checking if the grid file exists...
Checking if the grid is the same as the stored grid...
Stored grid loaded in 0.016420602798461914 seconds.
The instance grid is the same as the stored grid? False
Grid has changed, storing new one...
-----
Saving the centroids...
Saving the centroids...
Centroids saved in 0.0 seconds.
Saving the centroids...
Saving the centroids...
Centroids saved in 0.0 seconds.
Shape of the centroids array: (23762, 3)
Creating the KD-Tree for centroids...
KD-Tree for centroids created in 0.1479790210723877 seconds.
Adjacency matrix: calculating and storing it...
Calculating adjacency matrix: 100%[██████████] 35861/35861 [00:00<00:00, 106821.42it/s]
Saving the adjacency matrix...
Adjacency matrix saved in 0.03432774543762207 seconds.
Need to calculate the distances matrix...
Calculating distances: 100%[██████████] 23762/23762 [00:03<00:00, 7712.45it/s]
Shape of the distances matrix: (23762, 23762)
Distances matrix calculated in 3.080991744995117 seconds.
Saving the distances matrix...
Distances matrix saved in 0.06330204010009766 seconds.
Calculating the shortest paths matrix...
Calculating shortest paths: 100%[██████████] 23762/23762 [30:01<00:00, 13.19it/s]
Shortest paths matrix calculated in 1813.5166411399841 seconds.
Saving the shortest paths matrix...
Shortest paths matrix saved in 48.887428283691406 seconds.
Shape of the shortest paths matrix: (23762, 23762)
Grid set up

```

Σχήμα 6.5: Υπολογισμός και αποθήκευση των πινάκων για τετράγωνο πλέγμα μεγέθους 110×110 .

```

Calculating the elevation distance matrix...
Calculating elevation distances: 100%[██████████] 23762/23762 [00:05<00:00, 4414.11it/s]
Saving the uphill and downhill indices matrices...
Shape of the elevation distance matrix: (23762, 23762)
Elevation distance matrix calculated in 5.849992036819458 seconds.
Saving the elevation distance matrix...
Elevation distance matrix saved in 0.35263943672180176 seconds.
Calculating the shortest paths matrix...
Calculating shortest paths: 100%[██████████] 23762/23762 [26:43<00:00, 14.81it/s]
Elevated shortest paths matrix calculated in 1650.9451744556427 seconds.
Saving the elevated shortest paths matrix...
Shortest paths matrix saved in 21.53224353790283 seconds.
Shape of the elevated shortest paths matrix: (23762, 23762)

```

Σχήμα 6.6: Υπολογισμός και αποθήκευση των πινάκων πέρι elevation για τετράγωνο πλέγμα μεγέθους 110×110 .

Η χωρική επιβάρυνση των πλήρη πινάκων συντομότερης διαδρομής είναι στα 4,20 GB, και για δύο τέτοιους πίνακες έχουμε τουλάχιστον απασχολούμενο χώρο στο δίσκο 8,40 GB. Εύκολα συμπεραίνουμε ότι η φόρτωση τους στη μνήμη σε runtime είναι επίσης κοστοφόρα για το λειτουργικό σύστημα και πολύ απαιτητικό ως προς τους πόρους.

6.1.2 Σε χρόνο εκτέλεσης

Αν εν τέλει, τρέξουμε το 3Δ πρόγραμμα με πλέγμα μεγέθους 100, δοκιμάζουμε μερικές βασικές λειτουργίες.

Αν μεταβούμε από το σενάριο 1 στο 2, το οποίο προϋποθέτει επανυπολογισμό των pointcloud πληθυσμών, μόλυνση τυχαίων πηγαδιών και εύρεση μολυσμένων ανθρώπων, η εκτέλεση είναι η εξής:

```
=====
Constructing scenario...
Creating the population...
Adjusting the height of the population points...
Creating the wells...
Building the KD-Tree for wells...
KD-Tree for wells built.
Wells point cloud is 30 points
Entering infect_wells with Ratio: 0.2, Hard number: None
Infected number of wells 6, with indices [13, 7, 14, 18, 8, 0]
For all people, infect:: 100% | 10000/10000 [00:04<00:00, 2325.67it/s]
Infected number of people 2247
Percentage of infected people: 22.47000000000002%
=====
```

Σχήμα 6.7: Κατασκευή σεναρίου 2, 10,000 ανθρώπων και 30 πηγαδιών

Απαιτούνται 4 δευτερόλεπτα για τη μόλυνση ανθρώπων, και οι υπόλοιποι υπολογισμοί είναι αμελητέου χρόνου.

Για την κατασκευή του 3ου σεναρίου, 30,000 πληθυσμού ανθρώπων και 45 πηγάδια, έχουμε:

```
=====
Constructing scenario...
Creating the population...
Adjusting the height of the population points...
Creating the wells...
Building the KD-tree for wells...
KD-Tree for wells built.
Wells point cloud is 45 points
Entering infect_wells with Ratio: 0.2, Hard number: None
Infected number of wells 9, with indices [14, 18, 5, 44, 6, 24, 17, 29, 23]
For all people, infect:: 100% | 30000/30000 [00:12<00:00, 2432.27it/s]
Infected number of people 5300
Percentage of infected people: 17.66666666666668%
=====
```

Σχήμα 6.8: Κατασκευή σεναρίου 3, 30,000 ανθρώπων και 45 πηγαδιών

Σε γενικές γραμμές, οι ίδιες παρατηρήσεις ισχύουν. Ο ορισμός μολυσμένων περιοχών απαιτεί 12 δευτερόλεπτα, άρα η συνολική κατασκευή λίγο παραπάνω από αυτό.

Έχοντας κατασκευασμένο το σενάριο 3, η μετάβαση σε τυχαία επιλογή πηγαδιών απαιτεί επίσης 12 δευτερόλεπτα, που υποδηλώνει ότι εξαρτάται μόνο από τον αριθμό του πληθυσμού.

```
For all people, infect:: 100% | 30000/30000 [00:12<00:00, 2430.39it/s]
Infected number of people 5300
```

Σχήμα 6.9: Εύρεση μολυσμένου πληθυσμού που επιλέγει τυχαία το πηγάδι που θα επισκεφθεί

Αν μολύνουμε ένα πηγάδι, το σενάριο επαναφέρεται και υπολογίζεται εκ νέου:

```

Mouse released at (0.044745225459337234, -0.14482422173023224, 0.014799266122281551)
Entering infect_single_well with index 32
Infected well with Point3D object <vvrpywork.shapes.Point3D object at 0x000001DB0F6CFD90>, value [ 0.04546566 -0.14491796  0.01491147], index 32
For all people, infect:: 100%[██████████] 30000/30000 [00:12<00:00, 2410.04it/s]
Infected number of people: 6075
Percentage of infected people: 20.25%
Percentage impact: 2.5833333333333335

```

Σχήμα 6.10: Διαδικασία μολύνσεως ενός πηγαδιού και υπολογισμός της σκηνής εκ νέου

Πάλι απαιτούνται 12 δευτερόλεπτα για αυτόν τον υπολογισμό.

6.2 Αδυναμίες - Πιθανές Βελτιώσεις

Από την προηγούμενη ενότητα, βλέπουμε ότι μια μεγάλη αδυναμία του προγράμματός μας είναι οι μεγάλες χρονικές και χωρικές απαιτήσεις που έχει για τους πίνακες στη 3Δ σκηνή. Μελλοντικές βελτιώσεις θα πρέπει να εστιάσουν σε αυτόν τον τομέα, ώστε να είναι πιο ελαφρύ στην εκτέλεση.

Επιπλέον, καθυστερεί έναν σημαντικό αριθμό δευτερολέπτων για μεγάλο πλέγμα ή μεγάλους πληθυσμούς, που κάνει πιο δυσμενή την εμπειρία χρήσης του προγράμματος. Αντί να κατασκευάζεται εξ' ολοκλήρου η σκηνή σε κάθε τυχόν αλλαγή, μπορούν να αναπτυχθούν δομές για χωρικές διαμερίσεις ώστε να γίνονται στοχευμένες, μικρές αλλαγές (να ελέγχεται ένα υποσύνολο του πληθυσμού).

Μια έλλειψη είναι η απώλεια ανάλυσης των 3Δ σκηνών στο επίπεδο διαγράμματος Voronoi, ενώ έχει λάβει χώρα στη 2Δ. Η κατασκευή του διαγράμματος σε manifold είναι μια ενδιαφέρουσα περιοχή για βελτίωση του προγράμματος. Αναμένει ανεξερεύνητο ερώτημα για την εργασία, αν αλγόριθμοι υπολογισμού Voronoi, όπως ο αλγόριθμος Fortune 4.1.1, απλά απαιτούν αλλαγή στη συνάρτηση απόστασης (από ευκλείδια σε γεωδαισιακή), ώστε να υπολογιστεί ορθά.

Στην μοντελοποίηση της σκηνής μας, μπορεί να βρει κατάλληλη εφαρμογή και τα weighted Voronoi διαγράμματα, τα οποία αναθέτουν διαφορετικά βάρη στις εστίες, αντί να είναι όλες ισοδύναμες.

Άλλος τομέας πιθανής βελτίωσης αποτέλει το ερώτημα ορισμού του elevation, ανηφόρας και κατηφόρας. Μια πιο ενδελεχής προσέγγιση θα μπορούσε να λαμβάνει υπόψιν το gradient του mesh, και πιο αυστηρό ορισμό της ανωφέρειας/κατωφέρειας.

Στο πνεύμα σύγκρισης συναρτήσεων απόστασης, ενδιαφέρον θα παρουσίαζει η εφαρμογή απόστασης Manhattan, η οποία χρησιμοποιείται για μοντελοποίηση ταξιδιού εντός πόλεων που παρουσιάζουν ορθές γωνίες και διασταυρώσεις στους δρόμους.

Τέλος, το Ερώτημα 7 3.2.4 και η προσαρμογή του γράφου, δεν έχει αναπτυχθεί επαρκώς, και μένει αρκετός χώρος για περαιτέρω, ενδιαφέρουσες αλλαγές και παρατήρησης συμπεριφορών.

7 Συμπεράσματα

Με την ολοκλήρωση της εκπόνησης της εργασίας, έχουμε μάθει να εργαζόμαστε με και να χειρίζόμαστε μεγάλα νέφη σημείων. Μαθαίνουμε τις δυσκολίες που παρουσιάζονται για τη μετάβαση από τις 2 διαστάσεις στις 3, τη σημασία των γεωδαισιακών αποστάσεων και τη σύγκρισή τους με την ευκλείδια, τη κομβικότητα των δομών δεδομένων για αποτελεσματική χωρική αναζήτηση καθώς και τη σημασία του διαγράμματος Voronoi.

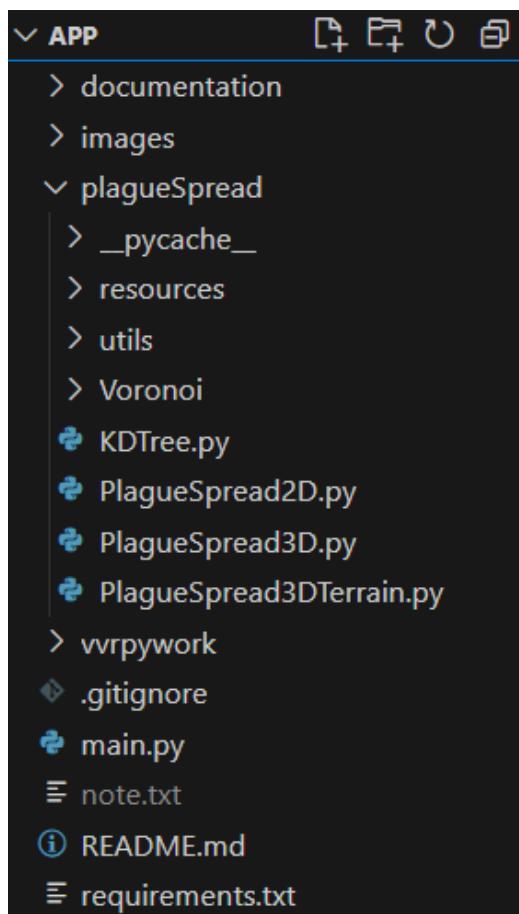
Μαθαίνουμε μέσω της μοντελοποίησης να κατασκευάζουμε/να προβλέπουμε μαζικές συμπεριφορές μεγάλων πληθυσμών εν συναρτήσει με μερικά, κεντρικά σημεία - εστίες, καθώς και σε ένα ανυψωμένο πλέγμα να βρίσκουμε τη βέλτιστη μη-ευκλείδια απόσταση μέσω από τον αλγόριθμο του Dijkstra. Επίσης, μαθαίνουμε να διαχειρίζόμαστε αυτούς τους πληθυσμούς επάνω σε manifolds, με τον δυϊκό τους γράφο και τις γεωδαισιακές αποστάσεις, καθώς και να τον τροποποιούμε ώστε να λαμβάνουμε υπόψιν το elevation και άλλα χαρακτηριστικά της γεωγραφίας.

Τέλος, εξοικειωνόμαστε με την διαχείριση, αποθήκευση και φόρτωση αρχείων μεγάλων μητρώων, και την εφαρμογή sparse matrices όπου κρίνεται δυνατό.

8 Οδηγός Κώδικα

Το σημείο εισόδου του προγράμματος είναι το αρχείο *main.py*, το οποίο το τρέχουμε και αλληλεπιδρούμε μαζί του μέσω του τερματικού. Επιλέγουμε μία από τις 3 σκηνές να τρέξουμε, το *PlagueSpread2D*, το *PlagueSpread3D*, ή το *PlagueSpread3DTerrain*.

Η ιεραρχία αρχείων του προγράμματος είναι η εξής:



Σχήμα 8.1: Ο φάκελος και τα αρχεία του προγράμματός μας.

Τα αρχεία των 3 σκηνών βρίσκονται εντός του φακέλου *plagueSpread*, αντιστοίχως ονομασμένα. Η ημιτελής κλάση για κατασκευή του διαγράμματος *Voronoi* βρίσκεται στον φάκελο *Voronoi*. Ο φάκελος *resources* περιέχει 3 υποφακέλους *grid*, *grid_100*, *terrain*, εκάστοτε εκ των οποίων περιέχει τους αντίστοιχους πίνακες που υπολογίζονται από τον πρόγραμμα για το πλέγμα (σημείωση: Για *GRID_SIZE = 100*, οι πίνακες θα αποθηκευτούν στον φάκελο *grid*. Πρέπει χειροκίνητα να μετακινηθούν στον φάκελο *grid_100* για διατήρηση αυτών των στοιχείων, και ύστερα το πρόγραμμα όπως αρμόζει θα τα ανακτά).

Και τα 3 αρχεία *PlagueSpread* περιέχουν έναν βασικό σκελετό συναρτήσεων που μοιράζεται μεταξύ τους.

- Τον constructor `__init__`, ο οποίος αρχικοποιεί τη σκήνη.
- Τη συνάρτηση `constructor_scenario`, η οποία κατασκευάζει το βασικό σενάριο.
- Τη συνάρτηση `infect_wells`, η οποία μολύνει τυχαία μερικά πηγάδια.
- Τη συνάρτηση `find_infected_people`, η οποία βρίσκει όλους τους ανθρώπους που πρέπει να έχουν μολυνθεί λόγω απόστασης από μολυσμένο πηγάδι ντετερμινιστικά.
- Τη συνάρτηση `find_infected_people_stochastic` η οποία βρίσκει όλους τους ανθρώπους που πρέπει να έχουν μολυνθεί λόγω απόστασης από μολυσμένο πηγάδι στοχαστικά.
- Τη συνάρτηση `on_key_press`, μέσω της οποίας μπορούμε να αλληλεπιδράμε με τη σκηνή με το πληκτρολόγιο.
- Τη συνάρτηση `on_mouse_press`, μέσω της οποίας μπορούμε να αλληλεπιδράμε με τη σκηνή με το ποντίκι.

Στο αρχείο *GeometryUtils*, το οποίο βρίσκεται στο φάκελο *utils*, περιέχει συναρτήσεις που εκτελούν γενικότερες γεωμετρικές λειτουργίες, όπως ορισμός δομής *LineEquation2D* (όπως ορίστικε στο εργαστήριο), *is_inside_polygon_2d*, που ελέγχει αν τα δοθέντα σημεία βρίσκονται εντός ενός πολυγώνου, και *barycentric_interpolate_height*, η οποία υπολογίζει μέσω βαρυκεντρικής παρεμβολής τα ύψη των δοθέντων σημείων, και άλλα.

9 Οδηγίες Χρήσης Προγράμματος

- Κατεβάζουμε το anaconda και το εγκαθιστούμε
- Ανοίγουμε ένα anaconda prompt και δημιουργούμε καινούργιο περιβάλλον με conda create -n <envname> python=3.10.13.
- Ενεργόποιούμε το περιβάλλον conda activate <envname>
- Λαμβάνουμε τον φάκελο του vvrpywork, όπου βρίσκεται το αρχείο *requirements.txt*
- Κάνουμε μέσω pip install -r requirements.txt τα πακέτα που απαιτούνται.
- Μέσω VSCode, ως python kernel επιλέγουμε το περιβάλλον που δημιουργήσαμε
- Εκτελούμε το main.py
- Ακολουθούμε τις οδηγίες, όπως εμφανίζονται στο τερματικό, για να τρέξουμε τη σκηνή που επιθυμούμε.

9.1 Για τη 2Δ σκηνή:

```
--> Press ENTER to reset the scene & print instructions.  
--> Press BACKSPACE to print the scenario parameters.  
--> Press UP to toggle between trial mode and normal mode.  
--> Press RIGHT or LEFT to increase or decrease the number of wells.  
--> Press M or N to increase or decrease the population.  
--> Press 1 or 2 or 3 to set the scenario to version 1 or 2 or 3.  
--> Press W to toggle dense regions of the population.  
--> Press V to toggle the Voronoi diagram.  
--> Press SHIFT + V to use the Voronoi diagram for computations.  
--> Press LEFT MOUSE BUTTON to add or remove a well.  
--> Press RIGHT MOUSE BUTTON to infect or disinfect a well.
```

--> Press R to toggle between deterministic and stochastic scenario.
-->--> Press P to reduce the probability of choosing the closest well.
-->--> Press SHIFT + P to increase the probability of choosing the closest well.

9.2 Για τη 3Δ σκηνή:

--> Press ENTER to reset the scene & print instructions.
--> Press BACKSPACE to print the scenario parameters.
--> Press UP to toggle between trial mode and normal mode.
--> Press RIGHT or LEFT to increase or decrease the number of wells.
--> Press M or N to increase or decrease the population.
--> Press 1 or 2 to set the scenario to version 1 or 2.
--> Press W to toggle dense regions of the population.
--> Press E to toggle between geodesic and euclidean distances.
--> Press G to consider the elevation in distance calculations.
--> Press V to toggle the Voronoi diagram.
--> Press SHIFT + V to use the Voronoi diagram for computations.
--> Press SHIFT + LEFT MOUSE BUTTON to add or remove a well.
--> Press SHIFT + LEFT MOUSE BUTTON to infect or disinfect a well.
--> Press R to toggle between deterministic and stochastic scenario.
Debug: -----
--> Press ALT + UP to show matrices of the grid. Default: uphills, downhills.
--> Press ALT + LEFT to set the start point of the path (euclidean, geodesic or elevation aware).
--> Press ALT + RIGHT to set the end point of the path (euclidean, geodesic or elevation aware).
--> Press ALT + LEFT or RIGHT to change the color of the grid.
--> Press ALT + SPACE to clear the debug shapes.

9.3 Για τη 3Δ σκηνή με terrain:

Όπως και η απλή 3Δ σκηνή με το grid, και επίσης:
--> Press ALT + C to toggle the visibility mode of the mesh.

10 Βιβλιογραφία

- de Berg, M., Cheong, O., van Kreveld, M., & Overmars, M. (2008). Computational geometry: Algorithms and applications (3rd ed.). Springer-Verlag.
- van Hoof, Jaenisch, Huang, Foronoi, Documentation.
- Fortune's Algorithm: An intuitive explanation on jacquesheunis.com
- Fortune's Algorithm and implementation on blog.ivank.net.
- VVR Group, Computational Geometry Lecture - Duality - Youtube
- Moustakas K, Tzovaras D, Strintzis MG. SQ-Map: efficient layered collision detection and haptic rendering. IEEE Trans Vis Comput Graph. 2007 Jan-Feb;13(1):80-93. doi: 10.1109/TVCG.2007.20. PMID: 17093338.
- GeeksforGeeks, Dijkstra's Shortest Path Algorithm using priority_queue of STL
- L. J. Guibas and J. Stolfi. Ruler, compass and computer: The design and analysis of geometric algorithms. In R. A. Earnshaw, editor, Theoretical Foundations of Computer Graphics and CAD. NATO ASI Series F, vol. 40, pages 111–165. Springer-Verlag, 1988.
- Geogebra, barycentric coordinates