ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ &

ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Curvature Estimation Methods for 3D Point-Clouds

Μέθοδοι Υπολογισμού Καμπυλότητας για 3Δ Νέφη-Σημείων

Νικολαΐδης Τραϊανός

Επιβλέπων : Δρ. Αντώνιος Πρωτοψάλτης

Ιούλιος 2022, Κοζάνη

Δήλωση Πνευματικών Δικαιωμάτων

Δηλώνω ρητά ότι, σύμφωνα με το άρθρο 8 του Ν. 1599/1986 και τα άρθρα 2,4,6 παρ. 3 του Ν. 1256/1982, η παρούσα Διπλωματική Εργασία με τίτλο

"Μέθοδοι Υπολογισμού Καμπυλότητας 3Δ Νέφη-Σημείων "

καθώς και τα ηλεκτρονικά αρχεία και πηγαίοι κώδικες που αναπτύχθηκαν ή τροποποιήθηκαν στα πλαίσια αυτής της εργασίας και αναφέρονται ρητώς μέσα στο κείμενο που συνοδεύουν, και η οποία έχει εκπονηθεί στο Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Πανεπιστημίου Δυτικής Μακεδονίας, υπό την επίβλεψη του μέλους του Τμήματος κ. Αντώνιο Πρωτοψάλτη αποτελεί αποκλειστικά προϊόν προσωπικής εργασίας και δεν προσβάλλει κάθε μορφής πνευματικά δικαιώματα τρίτων και δεν είναι προϊόν μερικής ή ολικής αντιγραφής, οι πηγές δε που χρησιμοποιήθηκαν περιορίζονται στις βιβλιογραφικές αναφορές και μόνον. Τα σημεία όπου έχω χρησιμοποιήσει ιδέες, κείμενο, αρχεία ή / και πηγές άλλων συγγραφέων, αναφέρονται ευδιάκριτα στο κείμενο με την κατάλληλη παραπομπή και η σχετική αναφορά περιλαμβάνεται στο τμήμα των βιβλιογραφικών αναφορών με πλήρη περιγραφή.

Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας εργασίας, εξ ολοκλήρου ή τμήματος αυτής, για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα. Ερωτήματα που αφορούν τη χρήση της εργασίας για κερδοσκοπικό σκοπό πρέπει να απευθύνονται προς τον συγγραφέα. Οι απόψεις και τα συμπεράσματα που περιέχονται σε αυτό το έγγραφο εκφράζουν τον συγγραφέα και μόνο.

Copyright (C) Ονοματεπώνυμο Φοιτητή & Επιβλέποντα/ες, Έτος, Πόλη

Copyright (C) Νικολαΐδης Τραϊανός, Αντώνιος Πρωτοψάλτης, 2022 , Κοζάνη

Υπογραφή Φοιτητή:

ΠΕΡΙΛΗΨΗ

Υπάρχουν πολλοί τρόποι για να υπολογιστεί η καμπυλότητα σε ένα νέφος σημείων. Στην παρούσα μελέτη θα αναφερθούν διαφορετικοί τρόποι συλλογής τρισδιάστατου νέφους σημείων και διαδικασίες επεξεργασίας αυτών. Επίσης θα αναφερθούν εργαλεία και βιβλιοθήκες τα οποία χειρίζονται νέφη σημείων. Η εργασία αυτή έχει μια εκτεταμένη ανάλυση μιας μεθόδου η οποία ονομάζεται ομπρέλα. Θα εξηγηθεί το κάθε προγραμματιστικό και μαθηματικό βήμα της υλοποίησης για το πως επιλέγονται οι γείτονες του κάθε σημείου, πως υπολογίζονται τα κανονικά διανύσματα και τα εφαπτόμενα επίπεδα για να σχηματιστεί το σχήμα της ομπρέλας όπως δηλώνει η ονομασία της μεθόδου. Τα αποτελέσματα θα συγκριθούν μεταξύ τους ανάλογα με το πλήθος γειτόνων που θα χρησιμοποιηθούν ως δείγμα για να κατανοηθεί καλύτερα η ακρίβεια της καμπυλότητας πάνω στα σημεία και επιφάνειες σαν σύνολο μαζί με τους χρόνους εκτέλεσης χρησιμοποιώντας διαφορετικές τεχνικές και απλοποιήσεις στον κώδικα. Ο κώδικας είναι γραμμένος σε C++ χρησιμοποιώντας τις βιβλιοθήκες OpenGL και PCL.

ABSTRACT

There are many ways to calculate the curvature on point clouds. In the present study, different means of collecting 3D point clouds and their processing ways will be mentioned. In addition, there will also be a reference to useful tools and libraries. This work has an extensive analysis of a method called umbrella curvature. Each programming and mathematical step of the implementation will be explained along with its simplifications and results. The results will be compared with each other depending on the number of neighbors that will be used as a sample to better understand the accuracy of the curvature on the points and surfaces as a whole along with the execution times using different techniques and simplifications in the code. The code is written in C ++ using the OpenGL and PCL libraries.

Περιεχόμενα

Κατάλογος Εικόνων

ΚΕΦΑΛΑΙΟ 1ο
Εισαγωγή

Οι σύγχρονοι σαρωτές λέιζερ, όπως τα Lidars[12], και τα συστήματα CAD/CAM[15] χρησιμοποιούνται ολοένα και περισσότερο για πολύ βασικές αλλά και πολύπλοκες διαδικασίες όπως χαρτογράφηση και αναπαράσταση περιοχών και μοντελοποίηση του ανθρώπινου σώματος. Η ανάλυση της καμπυλότητας βοηθάει πολύ σε διάφορες περιπτώσεις όπως π.χ. στην αναγνώριση προσώπου και στην τμηματοποίηση ενός νέφους σημείων. Μας βοηθάει να εξηγήσουμε πως θα ήταν πραγματικά μια ελλιπής επιφάνεια δειγματοληψίας να διορθωθεί και να χωριστεί σε κομμάτια για πιο λεπτομερή περεταίρω ανάλυση ή και να προσδιορίσει 2 διαφορετικά αντικείμενα σε έναν χώρο όπου οι σαρωτές δεν μπορούν να καταλάβουν την διαφορά. Το παραγόμενο προϊόν των σαρωτών ονομάζεται νέφος σημείων και αναπαριστά ένα σύνολο σημείων στον 3Δ χώρο, που βρίσκονται στην επιφάνεια του αντικειμένου που έχει σαρωθεί. Για αυτόν τον λόγο οι απαιτήσεις για την ακρίβεια στα γεωμετρικά χαρακτηριστικά, όπως η καμπυλότητα πάνω στα σημεία, αυξάνεται συνεχώς. Η παρούσα εργασία παρουσιάζει την υλοποίηση μιας μεθόδου υπολογισμού καμπυλότητας η οποία ονομάζεται ομπρέλα.

1.1 Στόχοι και αναγκαιότητα της διπλωματικής

Η καμπυλότητα είναι μια ιδιότητα των επιφανειών η οποία χρησιμοποιείται για την αναγνώριση χαρακτηριστικών όπως κορυφογραμμές, κοιλάδες, επίπεδα κυρτά ή κοίλα ή και σχήματα σέλας. Οι επιφάνειες τμηματοποιούνται [2] σε περιοχές με βάση την καμπυλότητα ενώ τα τμήματα και τα χαρακτηριστικά που προκύπτουν χρησιμοποιούνται στη συνέχεια για την αναγνώριση και την καταχώρηση αντικειμένων. Για την επίτευξη μιας καλής προσέγγισης λείας επιφάνειας μπορεί να εκτιμηθεί η καμπυλότητας της.

Τα τρισδιάστατα μοντέλα γίνονται όλο και περισσότερο το κέντρο των τεχνικών ανάλυσης και επεξεργασίας σήματος που προηγουμένως εφαρμόζονταν σε τύπους δεδομένων όπως ήχο, εικόνα και βίντεο. Ένα σημαντικό συστατικό των αλγορίθμων ανίχνευσης χαρακτηριστικών και

φιλτράρισματος[23], είναι η διακριτή εκτίμηση διαφορικών μεγεθών η οποία εφαρμόζεται τόσο στη γεωμετρία όσο και σε άλλους τύπους δεδομένων. Στην περίπτωση ενός σχήματος, οι επιφανειακές διαφορές, όπως τα κανονικά διανύσματα και οι καμπυλότητες, χρησιμεύουν σε αλγόριθμους φωτορεαλιστικών γραφικών όπως ο φωτισμός.

Το μειωμένο κόστος των τρισδιάστατων (3D) συστημάτων σε συνδυασμό με την αυξανόμενη ποιότητά τους, λόγω της μεγαλης υπολογιστικής ισχύος που είναι διαθέσιμη στις μέρες μας, κάθιστά τα τρισδιάστατα συστήματα πραγματικού χρόνου ικανά να συνεισφερουν σε διαδικασίες αναγνώρισης προσώπου[21] το κοντινο μελλον. Η ανίχνευση και η αναγνώριση προσώπου έχουν σημαντικές εφαρμογές σε τομείς όπως η βιντεοπαρακολούθηση, η επιβολή του νόμου και οι διεπαφές ανθρώπου-υπολογιστή. Ο ανιχνευτής προσώπου αναλύει την καμπυλότητα των επιφανειών στην τρισδιάστατη αναπαράσταση της σκηνής που αποκτά ο σαρωτής. Εμφανή χαρακτηριστικά του προσώπου, όπως η μύτη και τα μάτια χαρακτηρίζονται εύκολα από την άποψη της ανάλυσης καμπυλότητας.

Οι καμπύλες είναι τα δομικά στοιχεία των σχημάτων και των σχεδίων στο γεωμετρικό σχέδιο με τη βοήθεια υπολογιστή (CAGD)[24]. Είναι σημαντικό να διασφαλίσουμε ότι αυτές οι καμπύλες είναι τόσο οπτικά όσο και γεωμετρικά αισθητικές για να ανταποκρίνονται στην υψηλή αισθητική ανάγκη για επιτυχημένο μάρκετινγκ προϊόντων. Οι καμπύλες Bezier, B-Spline και NURBS είναι τύποι εύκαμπτων καμπυλών που αναπτύχθηκαν για διάφορες σχεδιαστικές προθέσεις. Ωστόσο αυτές οι καμπύλες παράγουν πολύπλοκες συναρτήσεις καμπυλότητας που μπορούν να υποβαθμίσουν τη διαμόρφωση της αισθητικής του σχήματος. Μια λύση σε αυτό το πρόβλημα είναι η διατύπωση αισθητικών καμπυλών[22] και επιφανειών από καλά καθορισμένες καμπυλότητες για τη βελτίωση της αισθητικής ποιότητας σχεδίασης. Οι ερευνητές εξακολουθούν να εργάζονται για την τελειοποίηση των αισθητικών καμπυλών, συμβάλλοντας έτσι σε μια αύξηση σε δημοσιεύσεις στον τομέα αυτό. Ωστόσο, η έρευνα σχετικά με τη διαμόρφωση αισθητικών επιφανειών βρίσκεται ακόμη σε πρώιμο στάδιο.

Τα τριγωνικά πλέγματα είναι οι πιο συχνά χρησιμοποιούμενες αναπαραστάσεις επιφανειών σε πολλές εφαρμογές αναπαράστασης 3Δ επιφανειών. Οι ιδιότητες επιφανειακής καμπυλότητας έχουν χρησιμοποιηθεί με επιτυχία για την επίλυση διαφορετικών πρακτικών προβλημάτων[23],

όπως η εξομάλυνση ή η απλοποίηση των πλεγμάτων αυτών στη μοντελοποίηση και την κατασκευή, επίσης για ταξινόμηση επιφανειών και 3D αναγνώριση αντικειμένων στην έρευνα όρασης υπολογιστών. Άλλες διακριτές μορφές συνεχών ορισμών κάποιων διαφορικών τελεστών όπως, οι τιμές καμπυλότητας, γεωδαιτικές καμπύλες κ.α. έχουν δοθεί για αυθαίρετα τριγωνικά πλέγματα.

Έχουν δοθεί αρκετοί μαθηματικοί τύποι[5][6][7] και αρκετές πρακτικές διαδικασίες για την εύρεση καμπυλότητας πάνω σε νέφη σημείων. Κάποιες από τις διαδικασίες είναι εύκολες στην υλοποίηση με γρήγορα αποτελέσματα αλλά όχι με την μεγαλύτερη ακρίβεια και άλλες είναι χρονοβόρες με μεγαλύτερη προσοχή στην υλοποίηση αλλά και με καλύτερα και πιο ακριβή αποτελέσματα.

Στόχος της διπλωματικής είναι η διερεύνηση και αξιολόγηση μεθόδων υπολογισμού καμπυλότητας για ένα 3Δ νέφος σημείων ώστε να διευκολύνει όλες τις παραπάνω περιπτώσεις χρήσης.

1.2 Επισκόπηση κεφαλαίων

Στο 2ο κεφάλαιο θα αναφερθούν κάποιες διαδικασίες επεξεργασίας νεφών σημείων και κάποια ελαττώματα στην συλλογή ενός νέφους σημείων. Επίσης υπάρχουν βιβλιοθήκες όπως η PCL ως εργαλείο ανοιχτού κώδικα για επεξεργασία νέφους σημείων. Στο 3ο θα δοθούν κάποιες μαθηματικές έννοιες της του υπολογισμού καμπυλότητας μαζί με 2 παραδείγματα άλλων εργασιών και περαιτέρω κατανόηση.

ΚΕΦΑΛΑΙΟ 2ο
Νέφος σημείων

2.1 Τι είναι το νέφος σημείων

Τα νέφη σημείων είναι σύνολα δεδομένων που αντιπροσωπεύουν αντικείμενα και τις επιφάνειες τους. Αυτά τα σημεία αποτελούνται από τις 3 γεωμετρικές καρτεσιανές συντεταγμένες ενός μόνο σημείου σε μια επιφάνεια δειγματοληψίας. Ένας μεγάλος αριθμός μεμονωμένων χωρικών μετρήσεων σημείων αποτελούν ένα νέφος σημείων το οποίο με την σειρά του μπορεί να αποτελεί ένα υποσύνολο ενός μεγαλύτερου συνόλου νέφους σημείων. Το τρισδιάστατο νέφος σημείων μπορεί να γίνει τετραδιάστατο εισάγοντας την πληροφορία χρώματος των σημείων.

Τα νέφη σημείων παράγονται από 3D σαρωτές λέιζερ[14] και τεχνικές τεχνολογίας LiDAR[12]. Κάθε σημείο αντιπροσωπεύει μια σάρωση με λέιζερ. Για την πλήρη σάρωση μίας σκηνής, χρησιμοποιούνται διαδικασίες εγγραφής (point cloud registration) (βλ. παρ. 2.2).

2.2 Διαδικασίες επεξεργασίας νεφών σημείων

Συμπίεση νέφους σημείων

Το point cloud compression (PCC)[1] είναι ο τρόπος συμπίεσης των ογκομετρικών οπτικών δεδομένων. Τα ογκομετρικά οπτικά δεδομένα συνήθως δημιουργούνται από υπολογιστή ή συλλέγονται από τον πραγματικό κόσμο.

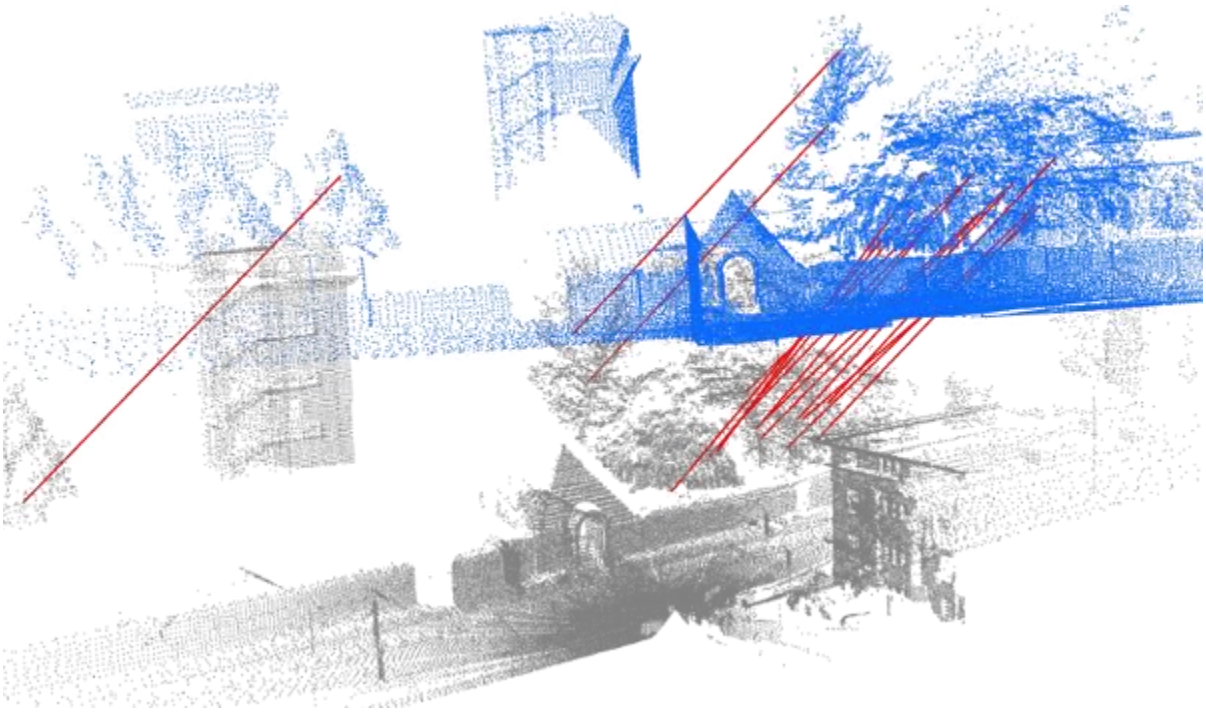
Τμηματοποίηση νέφους σημείων

Η τμηματοποίηση[2] τρισδιάστατου νέφους σημείων είναι η διαδικασία διαχωρισμού των σημειακών νεφών σε διαφορετικές ομογενείς περιοχές έτσι ώστε τα σημεία στην δική τους περιοχή να έχουν κοινά χαρακτηριστικά και ιδιότητες. Η τρισδιάστατη τμηματοποίηση είναι μια δύσκολη διαδικασία λόγω της άνισης πυκνότητας δειγματοληψίας.

Η διαδικασία τμηματοποίησης είναι χρήσιμη για την ανάλυση της σκηνής σε διάφορες εφαρμογές, όπως εντοπισμός και αναγνώριση αντικειμένων, ταξινόμηση και εξαγωγή χαρακτηριστικών.

Point Cloud Registration

Το point cloud registration[3] είναι ένα θεμελιώδες πρόβλημα στην τρισδιάστατη όραση υπολογιστή. Όταν μια περιοχή ή ένα αντικείμενο έχει σαρωθεί από διαφορετικές θέσεις ή συντεταγμένες, ο στόχος της καταχώρισης είναι να βρεθεί ο μετασχηματισμός που τα ευθυγραμμίζει καλύτερα όλα αυτά σε ένα κοινό σύστημα συντεταγμένων. Το point cloud registration παίζει σημαντικό ρόλο σε πολλές εφαρμογές όρασης, όπως η ανακατασκευή τρισδιάστατων μοντέλων, η παρακολούθηση κατολισθήσεων και η ανάλυση ηλιακής ενέργειας.



Εικόνα 1. Δεδομένα από δύο τρισδιάστατες σαρώσεις του ίδιου περιβάλλοντος που πρέπει να ευθυγραμμιστούν χρησιμοποιώντας την εγγραφή σετ σημείων (Point cloud registration).

Ανακατασκευή επιφανειών

Το πρόβλημα που αντιμετωπίζεται με την ανακατασκευή επιφάνειας[4] είναι η ανάκτηση της ψηφιακής αναπαράστασης ενός φυσικού σχήματος που έχει σαρωθεί, όπου τα σαρωμένα δεδομένα περιέχουν μια μεγάλη ποικιλία ελαττωμάτων.

- **Μη ομοιόμορφη δειγματοληψία**

Η κατανομή των σημείων δειγματοληψίας της επιφάνειας αναφέρεται ως πυκνότητα δειγματοληψίας. Οι τρισδιάστατες σαρώσεις δίνουν συνήθως μια μη ομοιόμορφη δειγματοληψία στην επιφάνεια, η οποία μπορεί να οφείλεται στην απόσταση από το σχήμα έως τη θέση του σαρωτή, τον προσανατολισμό του σαρωτή, καθώς και τα γεωμετρικά χαρακτηριστικά του σχήματος. (εικ. 2(b))

- **Θόρυβος**

Σημεία τυχαία κατανεμημένα κοντά στην επιφάνεια ονομάζονται θόρυβοι. Η κατανομή αυτή είναι μια συνηθής συνάρτηση των αντικειμένων σάρωσης όπως ο θόρυβος του αισθητήρα, ο κβαντισμός του βάθους και η απόσταση ή ο προσανατολισμός της επιφάνειας σε σχέση με τον σαρωτή. Όταν υπάρχει ένα τέτοιο είδος θορύβου, στόχος του αλγόριθμου ανοικοδόμησης επιφανειών είναι να παραχθεί μια επιφάνεια ακριβώς στα σημεία κάτω από τον θόρυβο. (εικ. 2(c))

- **Αποκομμένα σημεία**

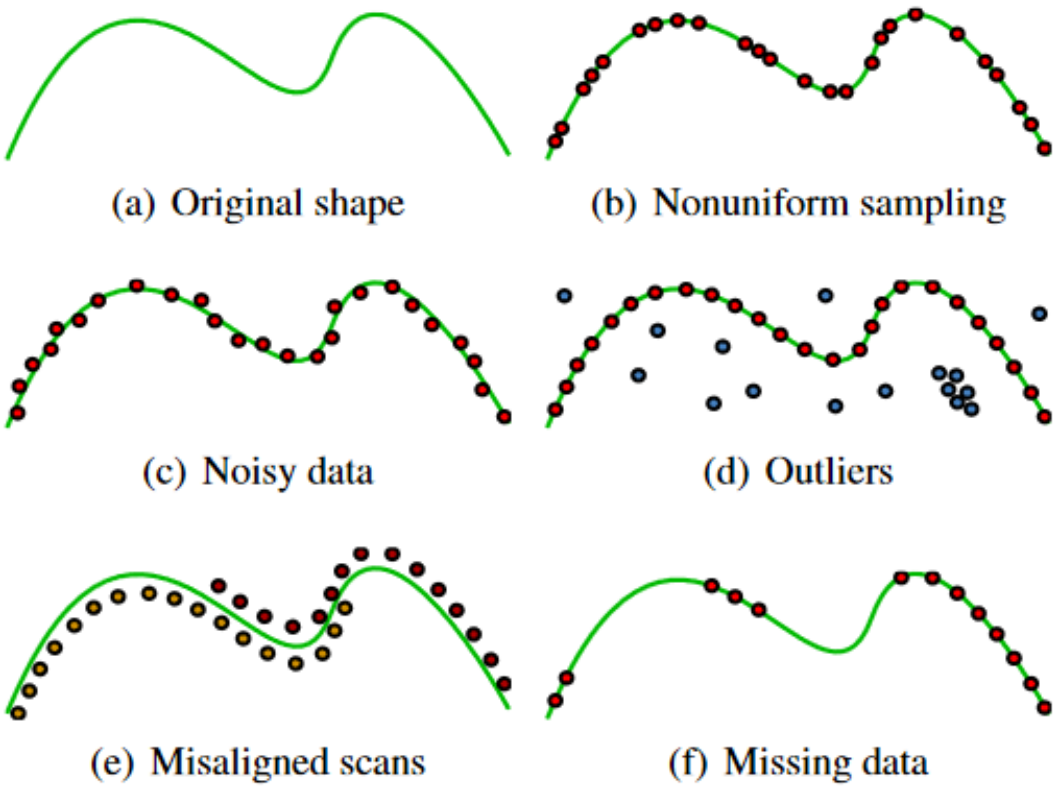
Τα αποκομμένα σημεία είναι εκείνα που απέχουν από την πραγματική επιφάνεια. Συνήθως, τα σημεία αυτά προκαλούνται στη διαδικασία σάρωσης. Σε ορισμένες περιπτώσεις, οι ακραίες τιμές διασκορπίζονται τυχαία στον όγκο, όπου η πυκνότητά τους είναι μικρότερη από την πυκνότητα των επιφανειακών σημείων δειγματοληψίας. Τα αποκομμένα σημεία μπορούν επίσης να είναι πιο δομημένα όπου μπορεί να υπάρχουν ομάδες σημείων υψηλής πυκνότητας μακριά από την επιφάνεια. (εικ. 2(d))

- **Εσφαλμένη ευθυγράμμιση**

Η εσφαλμένη ευθυγράμμιση προκύπτει από την ελλιπή καταχώρηση των σαρώσεων εύρους. Όταν η αρχική ρύθμιση ενός συνόλου σαρώσεων εύρους απέχει πολύ από το να είναι ιδανική, ένας αλγόριθμος εισαγωγής εντοπίζει εσφαλμένη ευθυγράμμιση. Η εσφαλμένη ευθυγράμμιση είναι μια σημαντική πρόκληση για την ανοικοδόμηση της επιφάνειας, καθώς εισάγει δομημένο θόρυβο μέσω σαρώσεων που είναι ελαφρώς μετατοπισμένες από την επιφάνεια. (εικ. 2(e))

- **Ελλιπή δεδομένα**

Πολλές προσεγγίσεις ανασυγκρότησης έχουν ως κίνητρο την ανάγκη αντιμετώπισης δεδομένων που λείπουν. Τα χαμένα δεδομένα προκαλούνται από περιορισμένο εύρος αισθητήρων, ισχυρή απορρόφηση φωτός και εμπόδια στη διαδικασία σάρωσης στα οποία δεν καταγράφονται τεράστια τμήματα του σχήματος. (εικ. 2(f))



Εικόνα 2. Διαφορετικές μορφές αντικειμένων νέφους σημείων στην περίπτωση καμπύλης στον δισδιάστατο χώρο.

Πριν υπολογιστεί η καμπυλότητα μπορούν να γίνουν κάποιες προ-επεξεργασίες όπως να αφαιρεθεί ο θόρυβος ή/και να γίνει ευθυγράμμιση στο νέφος. Οι μέθοδοι για τον υπολογισμό της καμπυλότητας χωρίζονται σε 3 βασικές κατηγορίες. Η πρώτη είναι τοποθέτηση πολυωνυμικών επιφανειών σε μια τοπική περιοχή. Η δεύτερη είναι ο υπολογισμός των καμπυλοτήτων σε κορυφές ενός μοντέλου πλέγματος (mesh) μετρώντας την γωνιά κάθε πολύγωνου που διέρχεται από την κορυφή. Και η τρίτη στην οποία πέφτει και η εργασία αυτή είναι χρησιμοποιώντας γειτονικά σημεία υπολογίζοντας κανονικά διανύσματα και εφαπτόμενα επίπεδα.

Επίσης να σημειωθεί ότι υπάρχουν δυσκολίες με τις προαναφερθείσες μεθόδους επειδή σε πολλές περιπτώσεις δύο διαφορετικές επιφάνειες μπορεί να βρίσκονται κοντά μεταξύ τους και να επηρεάζουν η μια την καμπυλότητα της άλλης και σαν αποτέλεσμα μια ευθεία γραμμή να απέχει αρκετά από την μηδενική καμπυλότητα. Η πληροφορία της καμπυλότητας μπορεί να μας κάνει να καταλάβουμε σε ποιες επιφάνειες υπάρχει θόρυβος ή ελλιπή και αποκομμένα σημεία έτσι ώστε να γίνουν οι αντίστοιχες διορθώσεις.

2.3 Η βιβλιοθήκη νέφους σημείων

Η PCL[16] (Point Cloud Library) είναι μια πλήρως διαμορφωμένη, σύγχρονη βιβλιοθήκη C++ για 3D επεξεργασία σημείου νέφους και η χρήση της μειώνει δραματικά το χρόνο και την δυσκολία υλοποίησης πολλών έργων και εργασιών. Η PCL χρησιμοποιείται ευρέως από πολλούς επιστήμονες, προγραμματιστές και έργα (projects) και μαζί με κάποιες τρίτες βιβλιοθήκες. Γραμμένο με αποτελεσματικότητα και απόδοση σε σύγχρονους επεξεργαστές. Οι περισσότερες μαθηματικές πράξεις υλοποιούνται με και βασίζονται στην βιβλιοθήκη Eigen[17], μια βιβλιοθήκη προτύπων ανοιχτού κώδικα για γραμμική άλγεβρα. Επιπλέον, η PCL παρέχει υποστήριξη για OpenMP και Intel Threading Building Blocks (TBB)[18] για παραλληλοποίηση πολλαπλών πυρήνων. Ο κορμός για γρήγορη αναζήτηση γειτόνων παρέχεται από το FLANN[19] (Fast Library for Approximate Nearest Neighbors). Όλες οι ενότητες και οι αλγόριθμοι της PCL περνούν δεδομένα γύρω από τη χρήση κοινών δεικτών Boost (Boost Shared Pointer) έτσι αποφεύγεται η

ανάγκη για εκ νέου αντιγραφή δεδομένων που υπάρχουν ήδη μέσα στο σύστημα. Από την έκδοση 0.6, η PCL έχει γίνει διαθέσιμη Windows, MacOS, Linux και Android.

Από αλγοριθμική άποψη, η PCL προορίζεται να ενσωματώσει ένα πλήθος αλγορίθμων επεξεργασίας 3D που λειτουργούν σε δεδομένα νέφους σημείου, συμπεριλαμβανομένων: φιλτραρίσματος, εκτίμησης χαρακτηριστικών, ανακατασκευή επιφάνειας, προσαρμογή μοντέλου, τμηματοποίηση, καταχώριση και δομές δεδομένων όπως τα kd-trees και τα octrees. Κάθε σύνολο αλγορίθμων ορίζεται μέσω βασικών κλάσεων που προσπαθούν να ενσωματώσουν όλες τις κοινές λειτουργίες που χρησιμοποιούνται σε ολόκληρο το αλυσιδωτό σύστημα, διατηρώντας έτσι τις υλοποιήσεις των πραγματικών αλγορίθμων συμπαγείς και καθαρές.

ΚΕΦΑΛΑΙΟ 3ο
Καμπυλότητα περιοχής 3Δ νέφους σημείων

3.1 Τι είναι η καμπυλότητα

Καμπυλότητα ονομάζεται ο ρυθμός μεταβολής της κατεύθυνσης μιας καμπύλης ως προς την απόσταση κατά μήκος της καμπύλης. Η καμπυλότητα είναι μια γεωμετρική ιδιότητα και ένα καλό αμετάβλητο χαρακτηριστικό των τοπικών επιφανειών. Μπορεί να εμφανίσει αλλαγές στο σχήμα της επιφάνειας και είναι αμετάβλητη στην περιστροφή, κλιμάκωση και μετατόπιση.

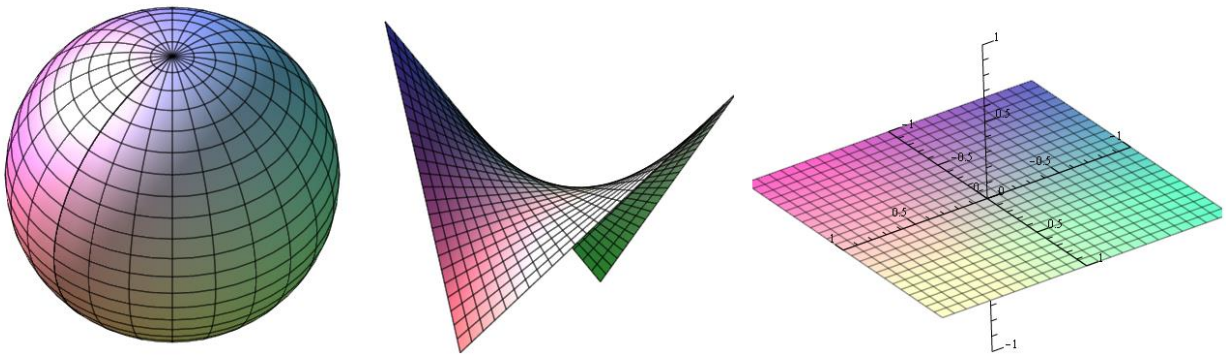
Ορισμός της Gauss και μέσης καμπυλότητας

Δεδομένου 2 πινάκων F1 και F2 οι κύριες καμπυλότητες είναι οι ιδιοτιμές του αντίστροφου F1 πολλαπλασιασμένος με τον F2 [5].

Η καμπυλότητα Gauss ορίζεται το γινόμενο των 2 κύριων καμπυλών.

Ο μέσος όρος του αθροίσματος των κυρίων καμπυλών ορίζεται ως μέση καμπυλότητα.

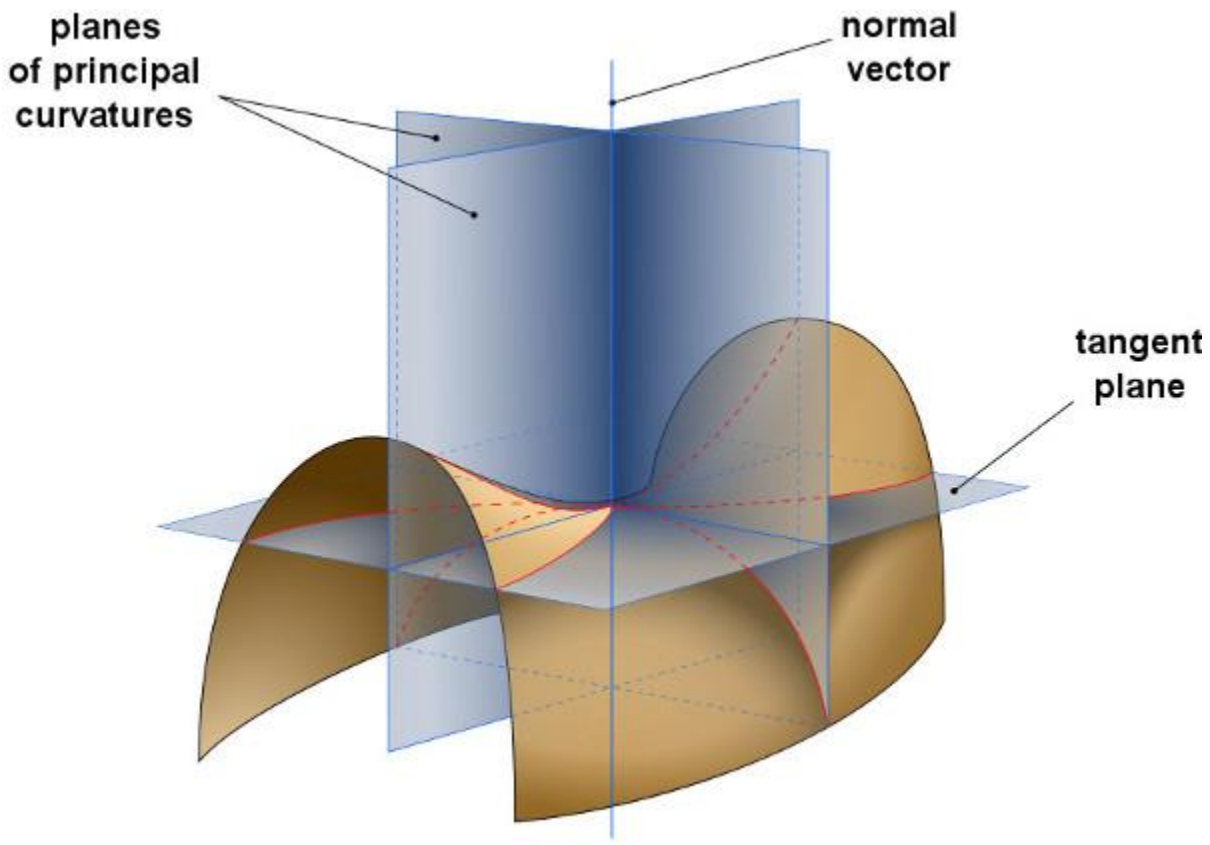
Μπορεί να οριστεί θετική καμπυλότητα π.χ. σφαίρα ή αρνητική π.χ. επιφάνεια σέλας και μηδενική καμπυλότητα όταν μιλάμε π.χ. για επίπεδο (βλ. εικόνα 3).



Εικόνα 3. Σφαίρα θετική σέλα αρνητική επίπεδο μηδενική καμπυλότητα.

Στην συνεχή χωρική καμπυλότητα κάθε σημείο p της επιφάνειας ορίζεται από δύο κύριες καμπυλότητες. Κάποιος μπορεί να επιλέξει ένα κανονικό διάνυσμα στο σημείο p και να ορίσει ένα κανονικό επίπεδο που περνά μέσα από αυτό. Κάθε επίπεδο, καθορισμένο με αυτόν τον τρόπο, θα κόψει την επιφάνεια σε μια καμπύλη επιπέδου. Κάθε καμπύλη επιπέδου που ορίζεται από διαφορετικό κανονικό επίπεδο θα έχει γενικά διαφορετική καμπυλότητα. Δύο κύριες καμπυλότητες κ1, κ2 είναι οι μέγιστες και ελάχιστες καμπυλότητες που ορίζονται από κανονικά επίπεδα γύρω από τον κανονικό διάνυσμα.

Αυτή η ιδέα μπορεί να γίνει καλυτέρα κατανοητή παρατηρώντας το *σχήμα 2* το οποίο παρουσιάζει μια επιφάνεια με δύο κύρια επίπεδα καμπυλότητας - κανονικά επίπεδα.

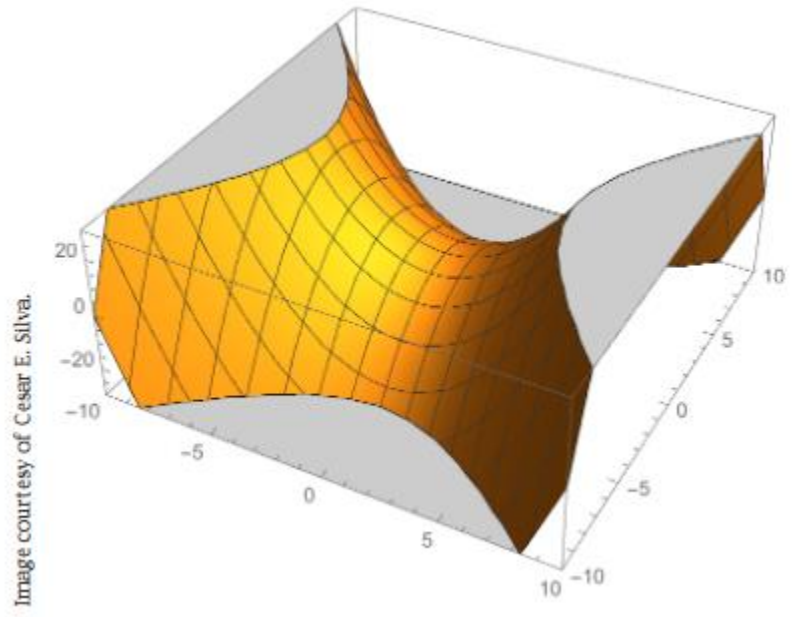


Εικόνα 4. Επιφάνεια σέλας με κανονικά επίπεδα προς τις κατευθύνσεις των κύριων καμπυλών[5]

.

Παρατηρώντας την καμπυλότητα της καμπύλης τμημάτων, φαίνεται πως υπάρχει μια καμπυλότητα προς κάθε κατεύθυνση. Οι μεγαλύτερες (θετικές, ανοδικές) και μικρότερες (πιο αρνητικές) καμπυλότητες ονομάζονται κύριες καμπυλότητες και εμφανίζονται σε ορθογώνιες κατευθύνσεις. [6]

- Ο μέσος όρος τους ονομάζεται μέση καμπυλότητα και το προϊόν τους ονομάζεται καμπύλη Gauss.
- Για τη σφαίρα μονάδας, και οι δύο κύριες καμπυλότητες είναι 1 και επομένως η καμπυλότητα Gauss είναι 1.
- Για έναν κύλινδρο μονάδας, οι κύριες καμπυλότητες είναι 1 και 0 και επομένως η καμπυλότητα Gauss είναι 0.
- Για ένα κατάλληλο υπερβολικό παραβολοειδές όπως στο Σχήμα 3, οι κύριες καμπυλότητες είναι 1 και − 1, και η καμπυλότητα Gauss είναι − 1.



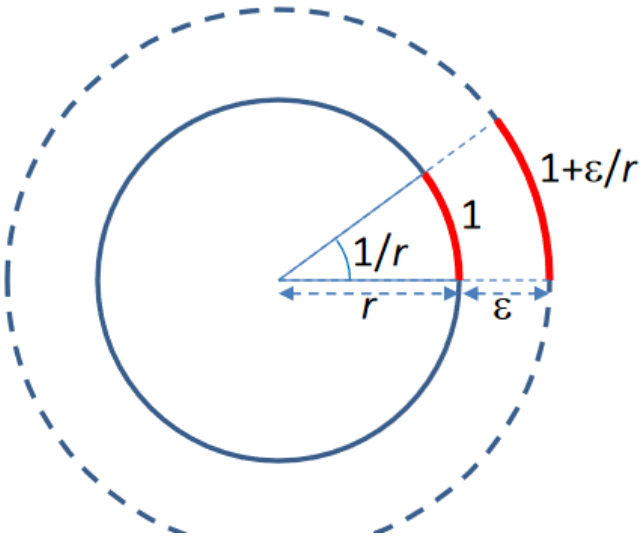
Εικόνα 5. Υπερβολικό παραβολικό, η καμπυλότητα Gauss είναι (+1)(−1) = −1[6]

3.1.1 Ομαλή καμπυλότητα

Σε κύκλο ακτίνας r, ένα τόξο μήκους μονάδας θα έχει γωνία 1 / r. Εκτείνοντας τον κύκλο σε απόσταση κατά ε στην κανονική κατεύθυνση, έχουμε έναν κύκλο με ακτίνα (r + ε) και το μήκος του αντίστοιχου τόξου γίνεται 1 + 1*ε / r . [7]

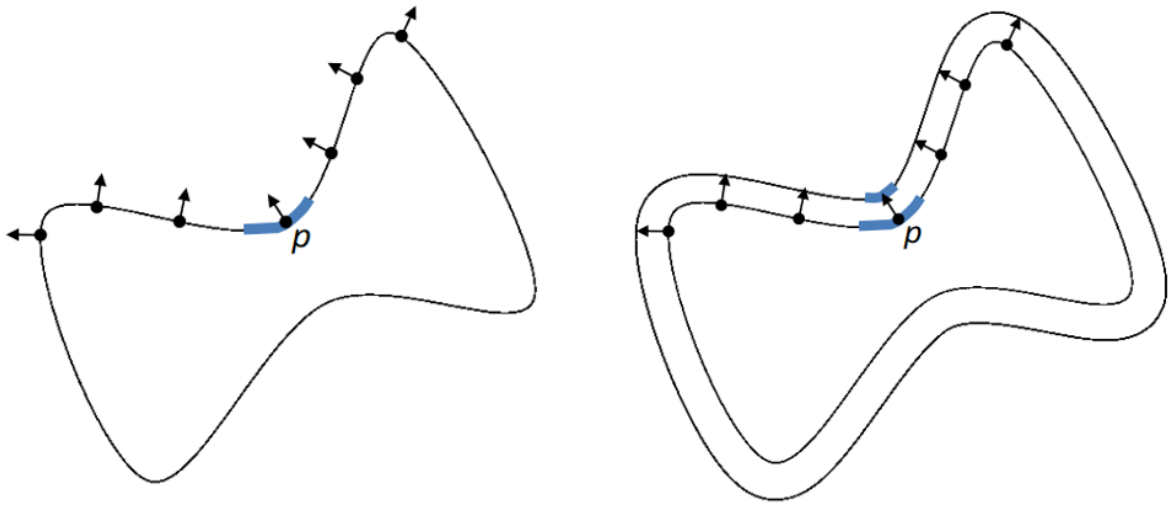
Η καμπυλότητα ορίζεται ως τον ρυθμό μεταβολής του μήκους ως συνάρτηση της απόστασης μετατόπισης ε:

$$\kappa = \frac{1}{r}$$



Εικόνα 6. Ορισμός Ομαλής Καμπυλότητας

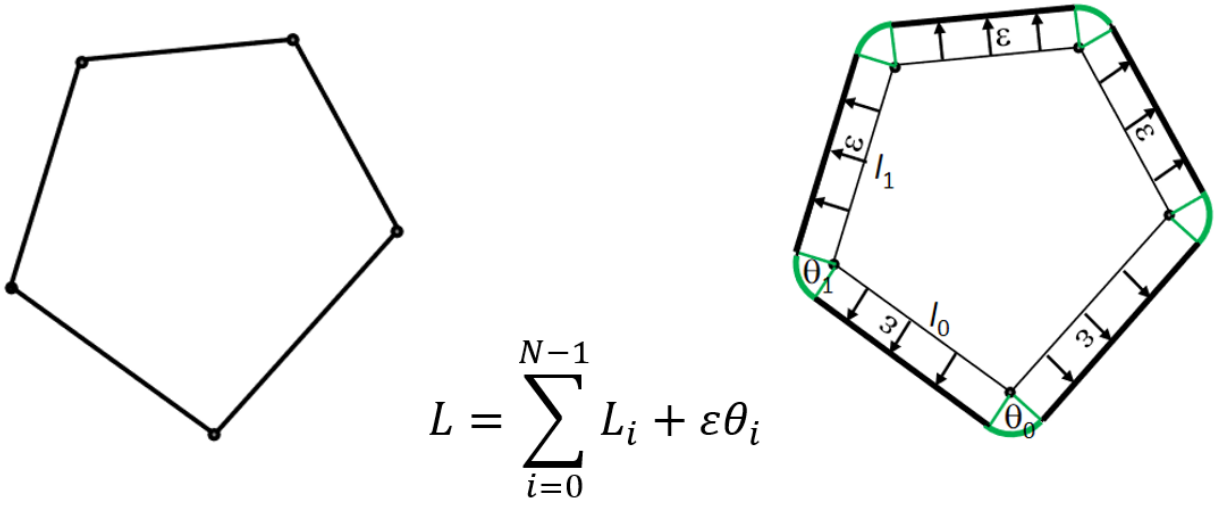
Με παρόμοιο τρόπο, μπορεί να οριστεί η καμπυλότητα σε ένα σημείο p σε μια αυθαίρετη καμπύλη λαμβάνοντας υπόψη το ρυθμό μεταβολής στο μήκος τόξου εκτείνοντας το κανονικό διάνυσμα κατά απόσταση ε.



Εικόνα 7. Ομαλή Καμπυλότητα

3.1.2 Διακριτή καμπυλότητα

Σε ένα σχήμα/αντικείμενο που αποτελείται από ευθείες και απότομες γωνίες εκτείνουμε το σχήμα προς την κανονική κατεύθυνση κατά ε. Το μήκος της καμπύλης μετατόπισης είναι το μήκος της παλιάς καμπύλης συν τα μήκη των τόξων. [7]



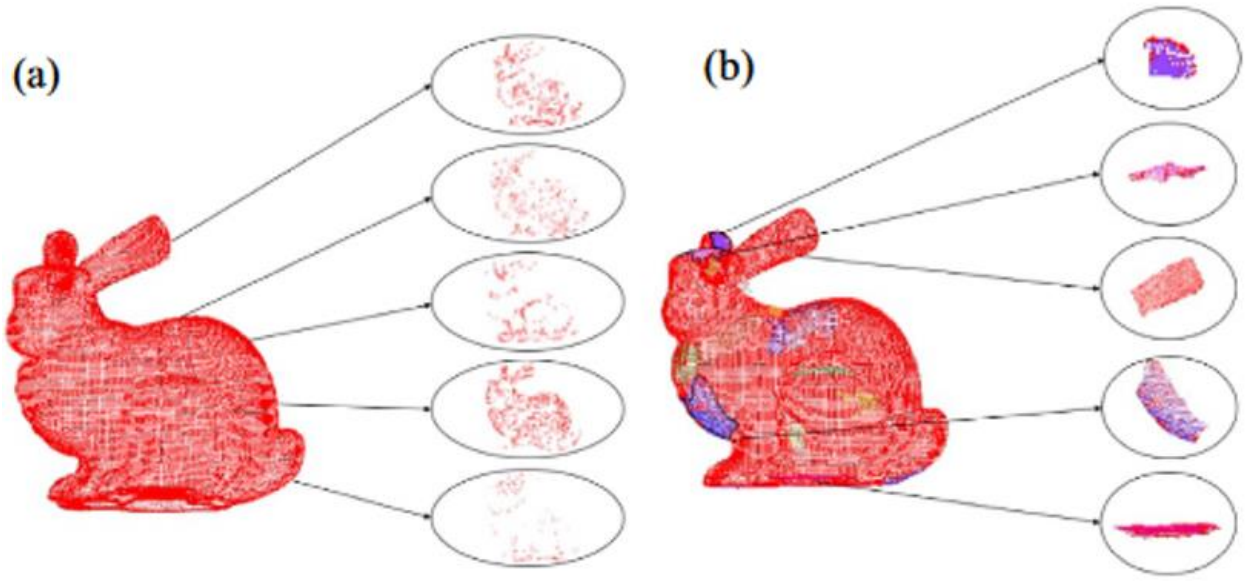
$$L = \sum_{i=0}^{N-1} L_i + \varepsilon\theta_i$$

Εικόνα 8. Διακριτή καμπυλότητα

3.2 Αλγόριθμος ομαδοποίησης διχοτομίας

Η ουσία του αλγόριθμου διχοτομίας[8] είναι να χωρίζει το κάθε κομμάτι/σύννεφο σημείων σε 2 μικρότερα και να υπολογίζει σε κάθε κομμάτι τον μέσο όρο των καμπυλοτήτων πάνω σε κάθε σημείο του αντίστοιχου κομματιού. Αυτός ο χωρισμός σταματάει όταν ο μέσος όρος της καμπυλότητας έχει φτάσει σε ένα όριο που έχει ορισθεί νωρίτερα.

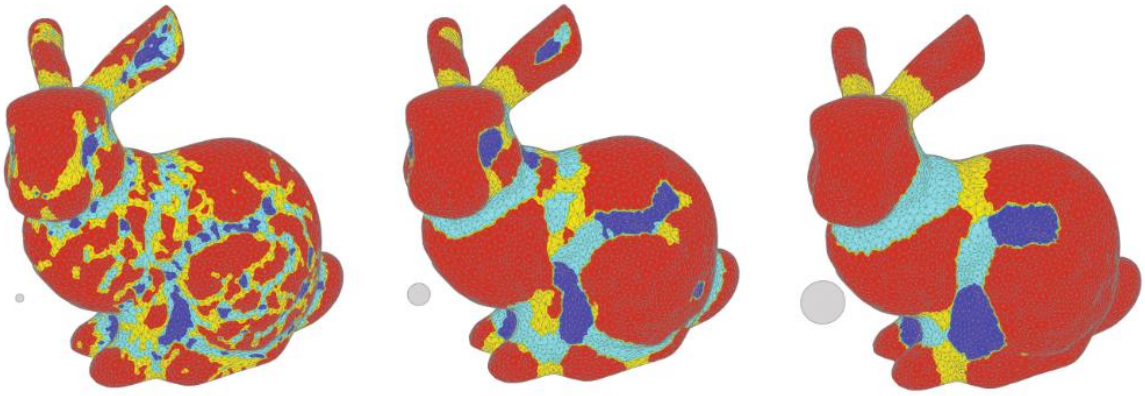
Όταν ο μέσος όρος της καμπυλότητας φτάσει στο κατώτατο όριο από τον αλγόριθμο ομαδοποίησης διχοτομίας τα σημεία αυτά απλώνονται στο κομμάτι τους επειδή αυτή η τιμή της καμπυλότητας θα μπορούσε να υπάρχει και εκεί. Σε αντίθεση άλλοι αλγόριθμοι ομαδοποίησης θα παγίδευαν τα σημεία τους τοπικά.



Εικόνα 9. Σύγκριση αλγορίθμων ομαδοποίησης: (a) ομαδοποίηση με διχοτομία – (b) ομαδοποίηση με αριθμό γειτόνων[8]

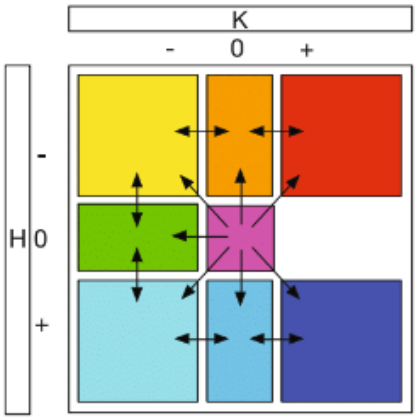
3.3 Shape-Curvature-Graph (SCG)

Σε αυτήν την εργασία[9] για τον υπολογισμό της μέσης καμπυλότητας και της καμπυλότητας Gauss χρησιμοποιείται η μέθοδος του Meyer. Για την επιλογή των γειτόνων χρησιμοποιείται ένα όριο απόστασης από το επιλεγμένο σημείο. Στην εικόνα 10 φαίνεται το 1ο παράδειγμα με μικρό όριο και το 3ο με μεγάλο όριο απόστασης.

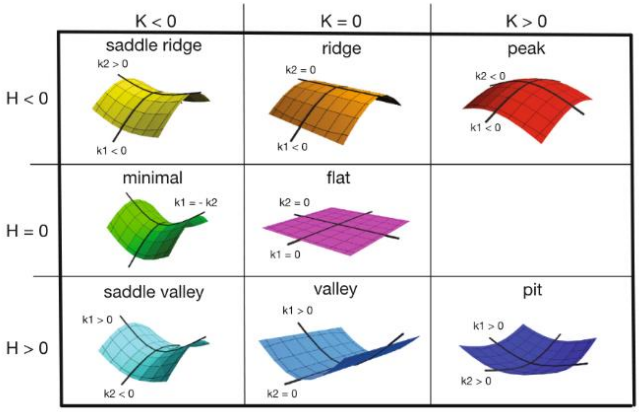


Εικόνα 10. Επίδραση του ορίου απόστασης στο πλέγμα λαγουδάκι, η απόσταση εμφανίζεται ως γκρι δίσκος[9]

Σε σημεία στα οποία η καμπυλότητα μεταβάλλεται απότομα έχουν δημιουργηθεί κάποιοι κανόνες έτσι ώστε να υπάρχει μια ομαλή συνέχεια (εικ 12). Οι κανόνες αυτοί είναι τα χρώματα που βρίσκονται στην εικόνα 12 και αντιστοιχούν με τις καμπυλότητες στην εικόνα 11. Τα αποτελέσματα φαίνονται στην εικόνα 10.



Εικόνα 12. Κανόνες συνέχειας[9]

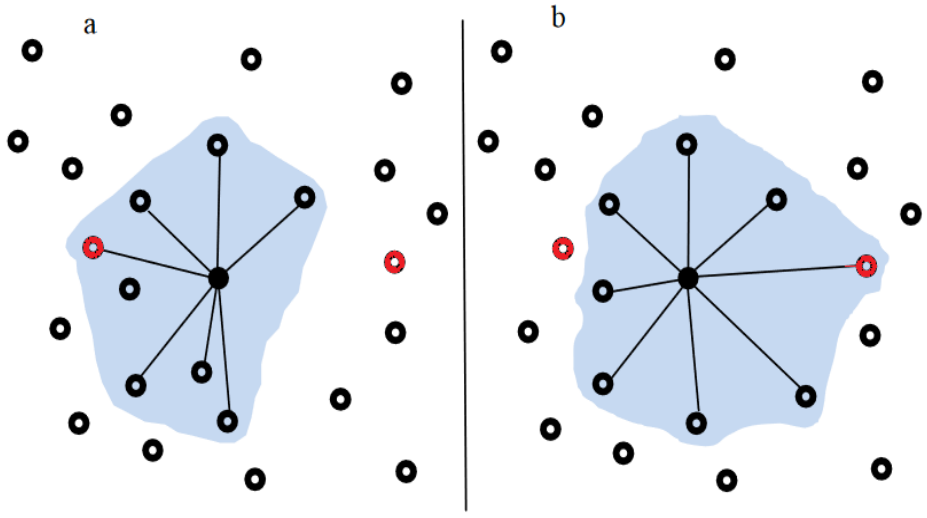


Εικόνα 11. Τιμές καμπυλότητας συνέχειας[9]

ΚΕΦΑΛΑΙΟ 4ο
Υλοποίηση Μεθόδου Ομπρέλα

4.1 Εισαγωγή

Η μέθοδος ομπρέλα [10] αφορά ένα αλγόριθμο υπολογισμού της καμπυλότητας ενός νέφους σημείων. Η μέθοδος αυτή υπολογίζει, για κάθε επιλεγμένο σημείο, 20 αρχικούς γείτονες με τη χρήση του αλγορίθμου K-nearest [13] (KNN). Ακολούθως, ο αλγόριθμος εντοπίζει τα 8 γειτονικά σημεία, που είναι οι κοντινότεροι και ομοιόμορφα κατανεμημένοι γείτονες γύρω από το επιλεγμένο σημείο. Τα 8 ομοιόμορφα κατανεμημένα γειτονικά σημεία διαμορφώνουν ομογενή γειτονιά, η οποία σχηματικά μοιάζει με ομπρέλα. Κατά τη διαδικασία επιλογής των 8 ομογενών γειτόνων κάποιοι κοντινοί, μη ομογενείς, γείτονες απορρίπτονται από τον αλγόριθμο έτσι ώστε να μείνουν οι τελικοί 8 με τις καλύτερα ομοιόμορφες θέσεις γύρω από επιλεγμένο σημείο. Στο τέλος θα υπολογισθεί το νέο κανονικό διάνυσμα για το επιλεγμένο σημείο χρησιμοποιώντας την ομογενή γειτονιά του και τελικά να υπολογισθεί η τελική τιμή της καμπυλότητας ομπρέλα. Το παράδειγμα στην εικόνα 13 δείχνει την απόρριψη ενός σημείου και την επιλογή ενός άλλου γείτονα, σε μεγαλύτερη απόσταση από τον απορριφθέντα γείτονα, έτσι ώστε να σχηματιστεί η ομογενής γειτονιά.



Εικόνα 13. (a) Γείτονες με K-nearest, (b) ομογενείς γείτονες[10]

4.2 Εργαλεία Υλοποίησης

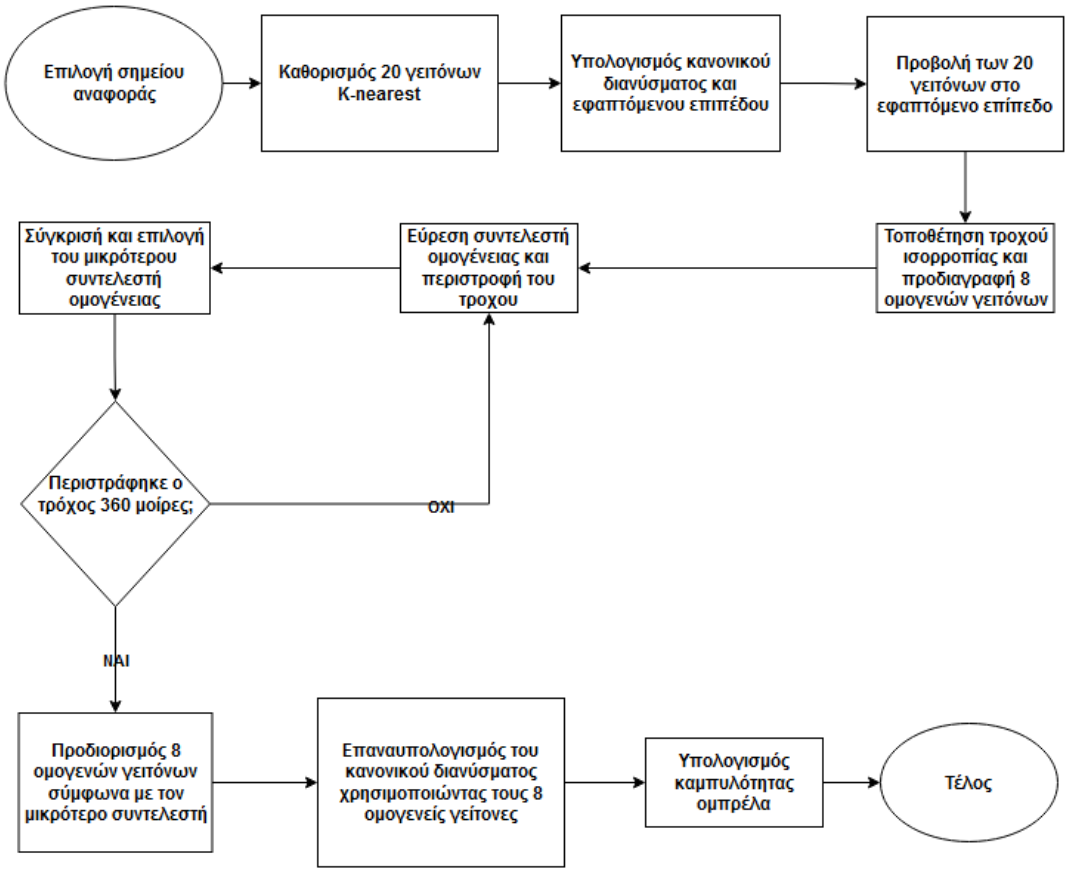
Για την υλοποίηση της μεθόδου ομπρέλα ο κώδικας γράφτηκε σε C++ και χρησιμοποιήθηκαν οι βιβλιοθήκες PCL και OPENGL.

Η επιλογή της C++ έγινε διότι ο χρόνος εκτέλεσης παίζει σημαντικό ρόλο. Η μέθοδος ομπρέλα, όπως και οι άλλοι αλγόριθμοι υπολογισμού καμπυλότητας, απαιτεί ακρίβεια αποτελεσμάτων και αποδοτικότητα στην εκτέλεση των υπολογιστικά δαπανηρών υπολογισμών. Για αυτό το λόγο η υλοποίηση έγινε με κατάλληλους βελτιστοποιημένους αλγορίθμους ώστε η επεξεργασία να ολοκληρώνεται στον καλύτερο δυνατό χρόνο και με την μεγαλύτερη ακρίβεια στα αποτελέσματα.

Η βιβλιοθήκη PCL [16] χρησιμοποιήθηκε για διαδικασίες νεφών σημείων, εύρεση αποστάσεων μεταξύ σημείων και υπολογισμό κανονικού διανύσματος. Η βιβλιοθήκη OPENGL[20] χρησιμοποιείται για την γραφική αναπαράσταση των αποτελεσμάτων.

4.3 Υλοποίηση

Ξεκινώντας από τις συντεταγμένες όλων των σημείων, η καμπυλότητα θα υπολογιστεί για κάθε σημείο ξεχωριστά με τη χρήση των 20 κοντινότερων γειτόνων του. Αρχικός στόχος της μεθόδου είναι η επιλογή της ομογενούς γειτονιάς κάθε σημείου, που απαρτίζεται από 8 τελικούς ομογενείς γείτονες, ανάμεσα από τους 20 αρχικούς. Ως εκ' τούτου η ομογενής γειτονιά θα απαρτίζεται από τους τελικούς γείτονες οι οποίοι θα είναι ομοιόμορφα κατανεμημένοι γύρω από το επιλεγμένο σημείο. Στο παρακάτω διάγραμμα ροής (εικόνα 14) φαίνονται τα βήματα επεξεργασίας του νέφους σημείου με τη μέθοδο ομπρέλα που υλοποιήθηκαν στα πλαίσια αυτής της διπλωματικής εργασίας.



Εικόνα 14. Διάγραμμα ροής για τον υπολογισμο καμπυλότητας

Η μέθοδος ομπρέλα θα υπολογίσει την καμπυλότητα για κάθε σημείο του αρχικού νέφους σημείων και θα αποθηκευτεί σε έναν πίνακα για να χρησιμοποιηθεί αργότερα.

```
for (int i = 0; i < total_points; i++) {
    curvatures[i] = umbrella(i);
}
```

4.3.1 Καθορισμός αρχικών γειτόνων

Το πρώτο βήμα της μεθόδου Ομπρέλα εντοπίζει, για ένα επιλεγμένο σημείο *P* του νέφους *PC* (Point Cloud), ένα σύνολο σημείων *NN* (Nearby Neighbors) με τα 20 κοντινότερα γειτονικά

σημεία. Η συγκεκριμένη διαδικασία είναι η πιο υπολογιστικά δαπανηρή από τις υπόλοιπες της μεθόδου. Για το λόγο αυτό υλοποιήθηκαν 2 διαφορετικές υλοποιήσεις της μεθόδου kNN (k-nearest-neighbors)[13],με τη χρήση του αλγορίθμου Selection-sort, και των δομών δεδομένων KD-tree[13].

κNN με Selection-sort

Για κάθε σημείο *P* υπολογίζονται οι τετραγωνικές αποστάσεις του di^2 από όλα τα σημεία του νέφους *PC*. Για 2 σημεία Α και Β η τετραγωνική απόσταση υπολογίζεται με τον εξής τύπο:

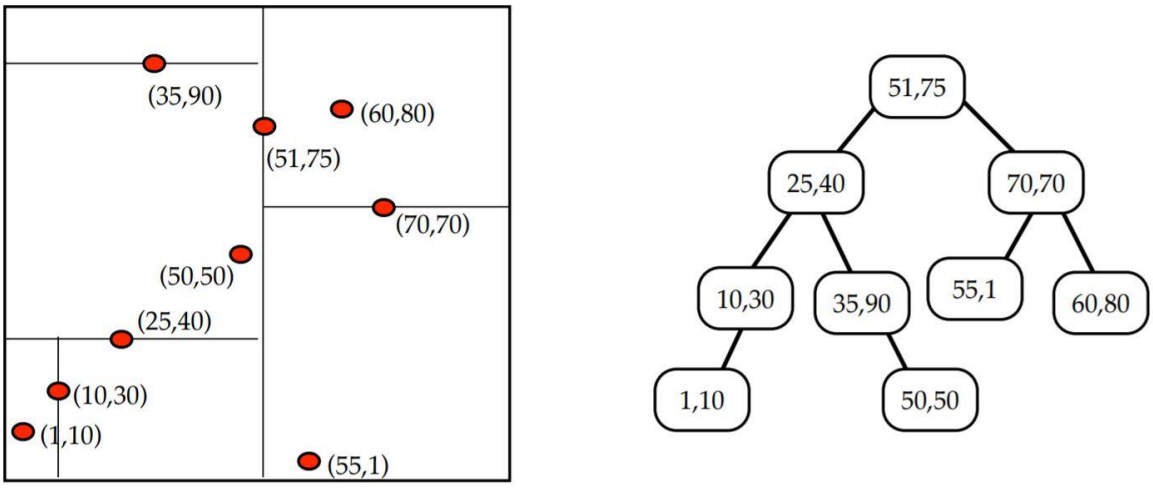
$$d^2 = (A_x - B_x)^2 + \left(A_y - B_y\right)^2 + (A_z - B_z)^2 \qquad (1)$$

Ακολούθως, γίνεται ταξινόμηση των αποστάσεων αυτών χρησιμοποιώντας τον αλγόριθμο selection sort. Αφού η μέθοδος ομπρέλα χρησιμοποιεί μόνο τους 20 κοντινότερους γείτονες του σημείου δεν είναι απαραίτητη η πλήρης ταξινόμηση όλων των αποστάσεων αλλά μόνο ο εντοπισμός των 20 κοντινότερων. Έτσι, ο selection sort θα εντοπίσει μέσα στο σύνολο του νέφους σημείων τα 20 με την μικρότερη απόσταση από το επιλεγμένο σημείο και θα σταματήσει εκεί αγνοώντας όλα τα υπόλοιπα. Όλες οι συντεταγμένες των γειτόνων αποθηκεύονται 2D πίνακες.

```
//υπολογισμος αποστασεων σημειων με το κεντρικο
for (int i = 0; i < total_points; i++) distances[i] =
    ((points[num][0] - points[i][0])*(points[num][0] - points[i][0]) +
     (points[num][1] - points[i][1])*(points[num][1] - points[i][1]) +
     (points[num][2] - points[i][2])*(points[num][2] - points[i][2]));

double copiedPoints[total_points][3];
start1 = std::clock();
/*  δημιουργια ενος αντιτυπου πινακα με τον αρχικο που περιεχει ολες τις
    συντεταγμενες ετσι ωστε καθε αλλαγη/επεξεργασια να μην επηρεασει τον αρχικο*/
for (int i = 0; i < total_points; i++) {
    copiedPoints[i][0] = points[i][0];
    copiedPoints[i][1] = points[i][1];
    copiedPoints[i][2] = points[i][2];
}
//ταξινομηση για την ευρεση τον 40 κοντινοτερων γειτονων
selectionSort(distances, copiedPoints);
```

kNN με KD-tree.

Τα kd-trees είναι δομή δεδομένων δυαδικού δέντρου για διαχωρισμό χώρου που καταλαμβάνει ένα 3D νέφος σημείων. Η δομή αυτή διαιρεί τον χώρο βάσει των σημείων του νέφους και χρησιμοποιείται κυρίως για την εύρεση κοντινότερων γειτόνων με απόσταση ή μέσα σε μια δεδομένη ακτίνα. Τα σημεία στα αριστερά αυτού του χώρου αντιπροσωπεύονται από το αριστερό υποδέντρο και τα σημεία στα δεξιά του χώρου αντιπροσωπεύονται από το δεξί υποδέντρο. Το σημείο αναφοράς θα έχει επίπεδο στοίχισης x και θα χωρίζουν τον χώρο κατά x (δηλαδή κάθετα) σε 2, στην συνέχεια θα επιλεχθούν ως τα 2 παιδιά τα σημεία με το μέσο y αριστερά και το μέσο y δεξιά και αυτά με την σειρά τους θα χωρίσουν το καθένα το δικό τους χώρο στα 2 κατά y (δηλαδή οριζόντια) και θα επιλεχθούν ως 2 παιδιά του καθενός το μέσο x στο πάνω κομμάτι και το μέσο x στο κάτω κομμάτι. Η διαδικασία θα επαναλαμβάνεται εναλλάξ μέχρι να έχουν μπει όλα τα σημεία στο δέντρο. Όπως και με την selection-sort όλες οι συντεταγμένες αποθηκεύονται σε 2D πίνακα.



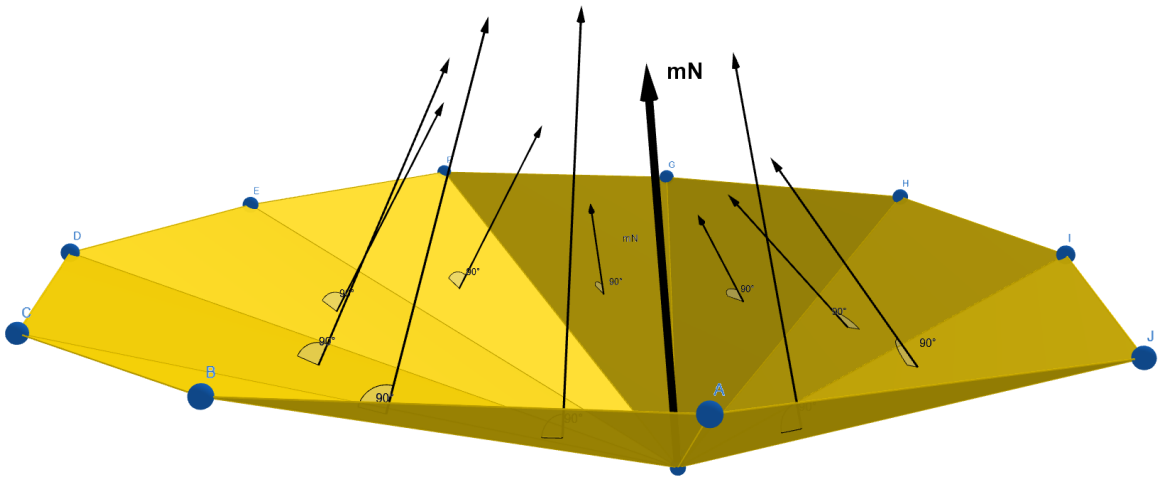
Εικόνα 15. Αναπαράσταση ενός kdtree

```
// K nearest neighbor search
std::vector<int> pointIdxKNNSearch(K+1);
std::vector<float> pointKNNSquaredDistance(K+1);

kdtree.nearestKSearch(num, K + 1, pointIdxKNNSearch,
    pointKNNSquaredDistance);
double point[3] = { cloud[pointIdxKNNSearch[0]].x,
    cloud[pointIdxKNNSearch[0]].y,cloud[pointIdxKNNSearch[0]].z };
for (std::size_t i = 1; i < pointIdxKNNSearch.size(); ++i) {
    twentyNeighbors[i - 1][0] = cloud[pointIdxKNNSearch[i]].x;
    twentyNeighbors[i - 1][1] = cloud[pointIdxKNNSearch[i]].y;
    twentyNeighbors[i - 1][2] = cloud[pointIdxKNNSearch[i]].z;
}
```

Η βιβλιοθήκη PCL προσφέρει κλάσεις για τις δομές δεδομένων του KD-tree. Σε επόμενη υπο-ενότητα παρατίθεται η συγκριτική μελέτη των χρόνων εκτέλεσης των 2 διαφορετικών υλοποιήσεων της μεθόδου kNN.

4.3.2 Υπολογισμός μέσου κανονικού διανύσματος

Το επιλεγμένο σημείο *P* σε συνδυασμό με δύο κάθε φορά σημεία από τους 20 αρχικούς



Εικόνα 16. Μέσο κανονικό διάνυσμα (mN)

γείτονες *NN* σχηματίζει 20 μικρές τριγωνικές επιφάνειες όπως το παράδειγμα εικόνας 16.

Σε αυτή τη διαδικασία υπολογίζονται τα κανονικά διανύσματα *Ni* των 20 τριγωνικών επιφανειών και στη υπολογίζεται το μέσο κανονικό διάνυσμα *mN*.

Από την εξίσωση(2) του επιπέδου κάθε τριγωνικής επιφάνειας υπολογίζεται το κάθε διάνυσμα:

$$P_i = ax_i + by_i + cz_i + d \qquad (2)$$

$$Ni = (a, b, c) \qquad (3)$$

$$και \;\; mN = \frac{\sum_{n=20}^{i=1} N_i}{n} \qquad (4)$$

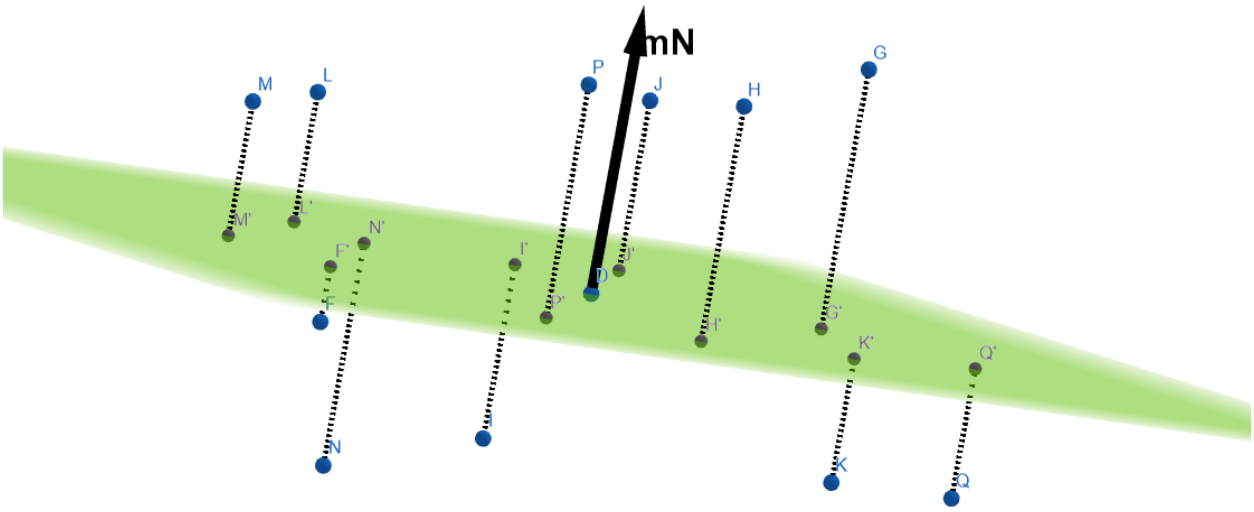
Η παραπάνω διαδικασία θα επαναληφθεί για όλα τα σημεία του νέφους *PC*. Όλες οι τιμές των κανονικών διανυσμάτων αποθηκεύονται σε στοιχεία της κλάσης Normal της PCL.

```
pcl::NormalEstimation<pcl::PointXYZ, pcl::Normal> ne;
pcl::PointCloud<pcl::Normal>::Ptr
    cloud_normals(new pcl::PointCloud<pcl::Normal>);
ne.setInputCloud(cloudptr);
ne.setKSearch(K+1);
ne.compute(*cloud_normals);
```

Για τον υπολογισμό των κανονικών διανυσμάτων στο επιλεγμένο σημείο *P* του νέφους *PC*, χρησιμοποιείται η κλάση kd-tree μαζί με τις συναρτήσεις υπολογισμού κανονικού διανύσματος της PCL.

4.3.3 Προβολή των 20 γειτόνων στο εφαπτόμενο επίπεδο

Το μέσο κανονικό διάνυσμα *mN* που αντιστοιχίζεται στο σημείο *P* του νέφους *PC* ορίζει ένα εφαπτόμενο επίπεδο στο οποίο θα προβληθούν τα 20 γειτονικά σημεία *NN*. Η εικόνα 17 δείχνει το διάνυσμα *mN*, το εφαπτόμενο επίπεδο, και τις προβολές των 20 γειτόνων *NN* στο εφαπτόμενο επίπεδο. Από τη διαδικασία αυτή προκύπτουν οι 20 προβολές *PN* των γειτόνων *NN*.



Εικόνα 17. Προβολή σε εφαπτόμενο επίπεδο

```
double pro[K][3];

//προβολη των K γειτονων
double metro = NormalVector[0] * NormalVector[0] +
    NormalVector[1] * NormalVector[1] +
    NormalVector[2] * NormalVector[2];
for (int i = 0; i < K; i++) {
    double proj = NormalVector[0] * (point[0] - twentyNeighbors[i][0]) +
        NormalVector[1] * (point[1] - twentyNeighbors[i][1]) +
        NormalVector[2] * (point[2] - twentyNeighbors[i][2]);
    proj /= metro;
    pro[i][0] = twentyNeighbors[i][0] + (proj*NormalVector[0]);
    pro[i][1] = twentyNeighbors[i][1] + (proj*NormalVector[1]);
    pro[i][2] = twentyNeighbors[i][2] + (proj*NormalVector[2]);
}
```

Στο παραπάνω κομμάτι κώδικα κανονικοποιείται το κάθετο διανύσμα και προβάλλονται οι 20 γείτονες πάνω στο εφαπτόμενο επίπεδο που δημιουργείται αποθηκεύοντας τις συντεταγμένες τους στον πίνακα pro[K][3].

4.3.4 Τοποθέτηση τροχού ισορροπίας

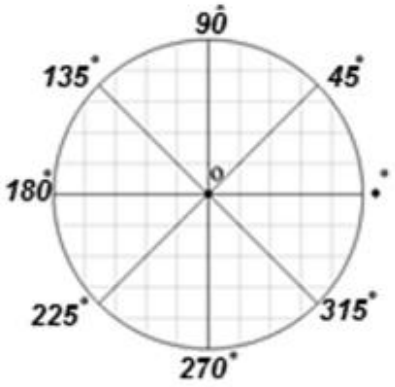
Για κάθε μία από τις προβολές PN των 20 γειτόνων *NN* ορίζεται ένα διάνυσμα από το επιλεγμένο σημείο *P* και την αντίστοιχη προβολή γείτονα *PNi*.

$$\mathrm{Vi} = \mathrm{PNi} - \mathrm{P} \qquad (5)$$

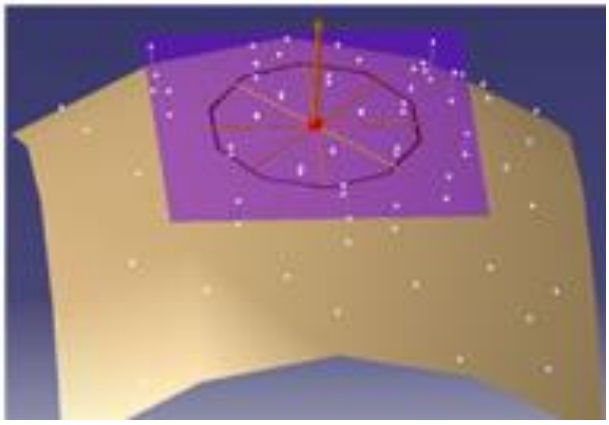
Ο τροχός ισορροπίας είναι ένας εικονικός κύκλος διαιρεμένος σε οκταμόρια (εικόνα 18), που τοποθετείται πάνω στο εφαπτόμενο επίπεδο με κέντρο το επιλεγμένο σημείο *P*, ευθυγραμμισμένος έτσι ώστε ο άξονας X να εφάπτεται με το διάνυσμα V_1 του πρώτου γείτονα PN_1.

Για κάθε διάνυσμα *Vi* υπολογίζεται η γωνιακή του διαφορά φ_i με τον άξονα X του τροχού, δηλαδή με το διάνυσμα V_1.

$$\varphi_{\mathrm{i}} = \cos^{-1}\left(\frac{\overrightarrow{\mathrm{V_1}}\cdot\overrightarrow{\mathrm{V_{1+1}}}}{\|\overrightarrow{\mathrm{V_1}}\|\cdot\|\overrightarrow{\mathrm{V_{1+1}}}\|}\right) \qquad (6)$$



Εικόνα 18. Τροχός ισορροπίας για οκτώ ομογενείς γείτονες[11]



Εικόνα 19. Τροχός ισορροπίας στο εξεταζόμενο σημείο[11]

```
double angles[20];
double dist[20];
double LorR[20];

//διαδικασια τροχου ισοροπιας
double ab[3] = { point[0] * pro[0][0],  point[1] * pro[0][1],   point[2] * pro[0][2] };
for (int i = 0; i < 20; i++) {


    double ad[3] = { point[0] * pro[i][0],point[1] * pro[i][1],point[2] * pro[i][2] };
    double myvec[3];
    crossProduct(ab, ad, myvec);
    //συνάρτηση αν ο γείτονας βρισκεται δεξία (L(eft)) ή (or) δεξία (R(ight)) του δινύσματος
    LorR[i] = dotProduct(myvec, NormalVector);

    angles[i] = calculateAngle(pro[0][0], pro[0][1], pro[0][2],
        point[0], point[1], point[2],
        pro[i][0], pro[i][1], pro[i][2]
        );

    angles[i] = acos(angles[i]) * 180 / PI;

    if (LorR[i] < 0) angles[i] = 360 - angles[i];
    else if (LorR[i] == 1) angles[i] = 180;
    if (angles[i] == 360) {
        angles[i] = 0;
    }\
    //απόσταση του κάθε γειτονα με το κεντρο
    dist[i] = sqrt((point[0] - pro[i][0])*(point[0] - pro[i][0]) +
        (point[1] - pro[i][1])*(point[1] - pro[i][1]) +
        (point[2] - pro[i][2])*(point[2] - pro[i][2]));

}
```

Οι 20 γωνιακές διαφορές φ_i που θα υπολογισθούν θα αποθηκευτούν σε έναν πίνακα. Αφού τα αποτελέσματα του acos βρίσκονται στο διάστημα [0-180] μοιρών, πρέπει πρώτα να προσδιοριστεί αν ο κάθε γείτονας είναι πάνω ή κάτω από τον άξονα X. Αν είναι πάνω το αποτέλεσμα της γωνίας μένει ως έχει, αν είναι κάτω τότε η γωνία αλλάζει σε 360- φ_i . Ακολούθως τα *PN* θα ταξινομηθούν με αύξουσα σειρά γωνιών *φι*.

4.3.5 Περιστροφή και ευθυγράμμιση τροχού ισορροπίας

Ο τροχός ισορροπίας θα περιστραφεί 20 φορές ώστε σε κάθε περιστροφή να ευθυγραμμίσει διαδοχικά τον άξονα X με κάθε μία από τις 20 προβολές PNi των γειτόνων *NNi* στο εφαπτόμενο επιπέδο. Σε κάθε θέση του τροχού ισορροπίας θα επιλεγούν οι 7 προβολές από τα *PNi* που σχηματίζουν ελάχιστες αποκλίσεις από τους άξονες των οκταμορίων του τροχού. Με τους 8 επιλεγμένους γείτονες (1+7), σε κάθε μια από τις 20 (κ=[1,20]) ευθυγραμμίσεις, θα υπολογισθεί ο παράγοντας ομογένειας με την παρακάτω σχέση:

$$H_\kappa = (\sum_{i=1}^{8} |\alpha_i|) \cdot (\sum_{i=1}^{8} d_i) \quad (7)$$

Το α_i αντιπροσωπεύει την γωνία του κάθε προβεβλημένου γείτονα με τον αντίστοιχο άξονα οκταμορίου και το d_i είναι η απόσταση της κάθε προβολής *PNi* με το επιλεγμένο σημείο.

```
for (int i = 0; i < 20; i++) {
    float distance = 0, total_angle = 0;
    float newang[20];
    float newang1[20];
    for (int j = 0; j < 20; j++) {
        newang[j] = angles[j] - angles[i];
        if (newang[j] < 0) newang[j] += 360;
    }
    int positions[8];
```

```
    for (int k = 0; k < 8; k++) {

        minangle = 361;
        for (int j = 0; j < 20; j++) {
            newang1[j] = abs(newang[j] - k * 45);
        }

        for (int j = 0; j < 20; j++) {
            int myflag = 0;
            if (newang1[j] < minangle) {
                minangle = newang1[j];
                positions[k] = j;
            }
        }
    }
    for (int i1 = 0; i1 < 7; i1++) {
        for (int j1 = i1 + 1; j1 < 8; j1++) {
            if (positions[j1] == positions[i1]) {
                if (positions[j1] < 19)
                    positions[j1] = positions[i1] + 1;
                else positions[j1] = 0;
            }
        }
    }
    for (int j = 0; j < 8; j++) {
        distance += dist[positions[j]];
        total_angle += abs(newang[positions[j]] - 45 * j);
    }
    if (total_angle * distance < homogFactor || homogFactor == NULL) {
        homogFactor = total_angle * distance;
        for (int j = 0; j < 8; j++) finalpositions[j] = positions[j];
    }
}
```

4.3.6 Σύγκριση συντελεστών ομογένειας

Από τη διαδικασία αυτή θα προκύψουν 20 συντελεστές ομογένειας *Hi* για κάθε μία θέση του τροχού ισορροπίας. Ο μικρότερος συντελεστής ομογένειας *Hi* είναι αυτός που θα καθορίσει τους 8 τελικούς ομογενείς γείτονες *Ni* του επιλεγμένου σημείου *P*.

4.3.7 Τελικός υπολογισμός κανονικού διανύσματος και καμπυλότητας ομπρέλα

Με τους 8 ομογενείς γείτονες *Ni* υπολογίζεται το τελικό κανονικό διάνυσμα *n*, με την ιδια μέθοδο που χρησιμοποιεί και η PCL (point cloud library), το οποίο χρησιμοποιείται για τον υπολογισμό της καμπυλότητας "ομπρέλα".

Η καμπυλότητα θα υπολογισθεί με τον παρακάτω τύπο (εξίσωση 8) [10] της οποίας το αποτέλεσμα πρέπει να βρίσκεται στο διάστημα [0,8].

$$k_{um} = \sum_{i=1}^{8} ABS\left(\frac{N_i - p}{|N_i - p|} \cdot n\right) \qquad (8)$$

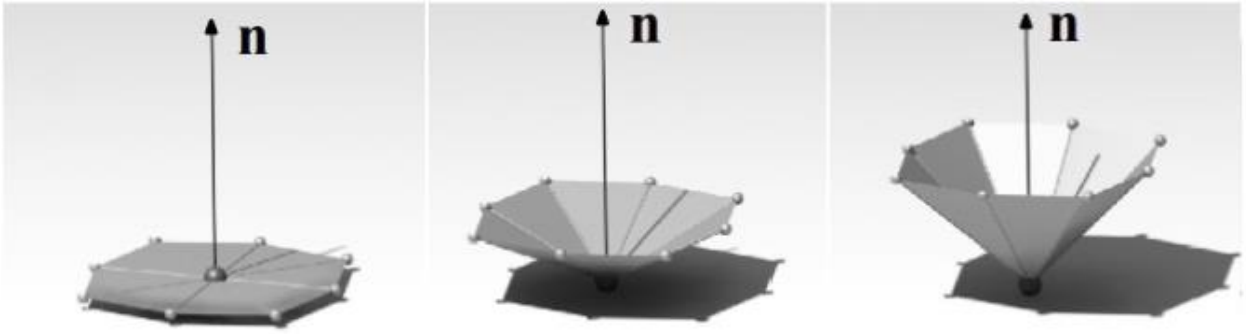
```
double b[2][3];
double dist1, dist2;
double eightNeighbors[spokes][3];
for (int i = 0; i < spokes-1; i++) {
    b[0][0] = a[i][0] - point[0];
    b[0][1] = a[i][1] - point[1];
    b[0][2] = a[i][2] - point[2];
    b[1][0] = a[i + 1][0] - point[0];
    b[1][1] = a[i + 1][1] - point[1];
    b[1][2] = a[i + 1][2] - point[2];

    eightNeighbors[i][0] = b[0][1] * b[1][2] - b[0][2] * b[1][1];
    eightNeighbors[i][1] = b[0][2] * b[1][0] - b[0][0] * b[1][2];
    eightNeighbors[i][2] = b[0][0] * b[1][1] - b[0][1] * b[1][0];
}

b[0][0] = a[spokes-1][0] - point[0];
b[0][1] = a[spokes-1][1] - point[1];
b[0][2] = a[spokes-1][2] - point[2];
b[1][0] = a[0][0] - point[0];
b[1][1] = a[0][1] - point[1];
b[1][2] = a[0][2] - point[2];

eightNeighbors[spokes-1][0] = b[0][1] * b[1][2] - b[0][2] * b[1][1];
eightNeighbors[spokes-1][1] = b[0][2] * b[1][0] - b[0][0] * b[1][2];
eightNeighbors[spokes-1][2] = b[0][0] * b[1][1] - b[0][1] * b[1][0];
```

```
double b[2][3];
double dist1, dist2;
double eightNeighbors[spokes][3];
for (int i = 0; i < spokes-1; i++) {
    b[0][0] = a[i][0] - point[0];
    b[0][1] = a[i][1] - point[1];
    b[0][2] = a[i][2] - point[2];
    b[1][0] = a[i + 1][0] - point[0];
    b[1][1] = a[i + 1][1] - point[1];
    b[1][2] = a[i + 1][2] - point[2];

    eightNeighbors[i][0] = b[0][1] * b[1][2] - b[0][2] * b[1][1];
    eightNeighbors[i][1] = b[0][2] * b[1][0] - b[0][0] * b[1][2];
    eightNeighbors[i][2] = b[0][0] * b[1][1] - b[0][1] * b[1][0];
}

b[0][0] = a[spokes-1][0] - point[0];
b[0][1] = a[spokes-1][1] - point[1];
b[0][2] = a[spokes-1][2] - point[2];
b[1][0] = a[0][0] - point[0];
b[1][1] = a[0][1] - point[1];
b[1][2] = a[0][2] - point[2];

eightNeighbors[spokes-1][0] = b[0][1] * b[1][2] - b[0][2] * b[1][1];
eightNeighbors[spokes-1][1] = b[0][2] * b[1][0] - b[0][0] * b[1][2];
eightNeighbors[spokes-1][2] = b[0][0] * b[1][1] - b[0][1] * b[1][0];
```

Στο παραπάνω κομμάτι κώδικα έχει υπολογίζεται το τελικό κανονικό διάνυσμα με τους 8 τελικούς ομογενείς γείτονες και η καμπυλότητα ομπρέλα σύμφωνα με τον μαθηματικό τύπο (εξίσωση 7).

```
double paranom[8];
for (int i = 0; i < 8; i++) {
    paranom[i] = sqrt((a[i][0] - point[0])*(a[i][0] - point[0]) +
        (a[i][1] - point[1])*(a[i][1] - point[1]) + (a[i][2] - point[2])*(a[i][2] - point[2]));
    N[i][0] = (NormalVector[0] * ((a[i][0] - point[0]) / paranom[i]));

    N[i][1] = (NormalVector[1] * ((a[i][1] - point[1]) / paranom[i]));

    N[i][2] = (NormalVector[2] * ((a[i][2] - point[2]) / paranom[i]));

    endresult += abs(N[i][0] + N[i][1] + N[i][2]);
}

dist1 = (NormalVector[0] - midPoint[0])*(NormalVector[0] - midPoint[0]) +
    (NormalVector[1] - midPoint[1]) + (NormalVector[2] - midPoint[2]);
dist2 = (-NormalVector[0] - midPoint[0])*(-NormalVector[0] - midPoint[0]) +
    (-NormalVector[1] - midPoint[1]) + (-NormalVector[2] - midPoint[2]);
if (dist2 < dist1) {
    NormalVector[0] *= -1;
    NormalVector[1] *= -1;
    NormalVector[2] *= -1;
}
return endresult;
```

Στην εικόνα 20 φαίνεται το 1ο παράδειγμα να δείχνει την καμπυλότητα για το επιλεγμένο σημείο με τιμή 0 ενώ στο 2ο και 3ο παράδειγμα και όσο η καμπυλότητα αυξάνει τόσο η ομπρέλα κλείνει.



Εικόνα 20. Ένα σημείο και οι 8 ομογενείς γείτονές του σε διαφορετικές θέσεις[10]

4.4 Περιπτώσεις χρήσης με νέφη σημείων και αποτελέσματα

Για τη δοκιμή της υλοποιημένης μεθόδου έχουν χρησιμοποιηθεί 5 διαφορετικά νέφη σημείων σε αρχεία τύπου .off (Object File Format) τα οποία περιέχουν τις συντεταγμένες των σημείων.

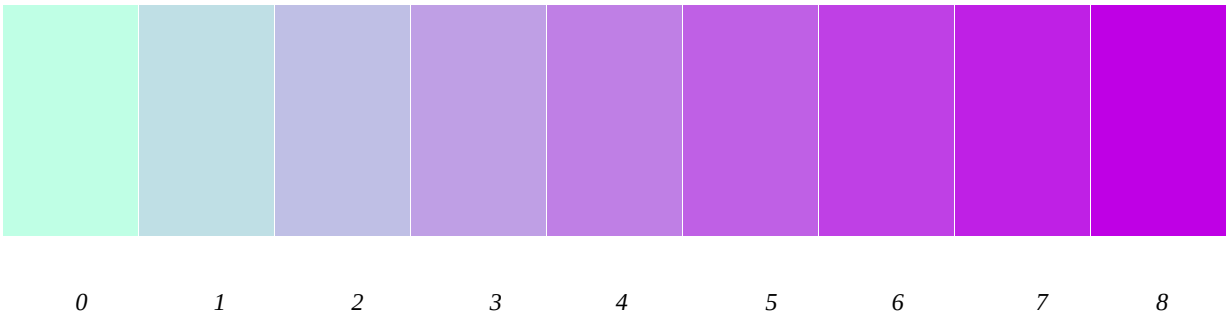
Όνομα	**Αρκούδος**	**Κουνέλι**	**Άνθρωπος**	**Μανιτάρι**	**Δεσμός**
Πλήθος σημείων	13826	34834	17495	2337	5760
Αρχική εικόνα					

Πίνακας 1.Περιπτώσεις χρήσης

Ο κώδικας τρέχει σε ένα νήμα και έναν πυρήνα του επεξεργαστή (single-threaded CPU) και τα χαρακτηριστικά του υπολογιστή στον οποίο δοκιμάστηκε η υλοποιημένη μέθοδος είναι τα εξής:

1. CPU Ryzen 3600
2. RAM DDR4 3200 MHz
3. GTX 1660 SUPER
4. SSD Read 560 MB/s - Write 530 MB/s

Στα αποτελέσματα των δοκιμών που παραθέτουμε παρακάτω χρησιμοποιήθηκαν χρώματα ανάλογα με την τιμή της υπολογισμένης καμπυλότητας, που συνοψίζονται στην εικόνα 21.



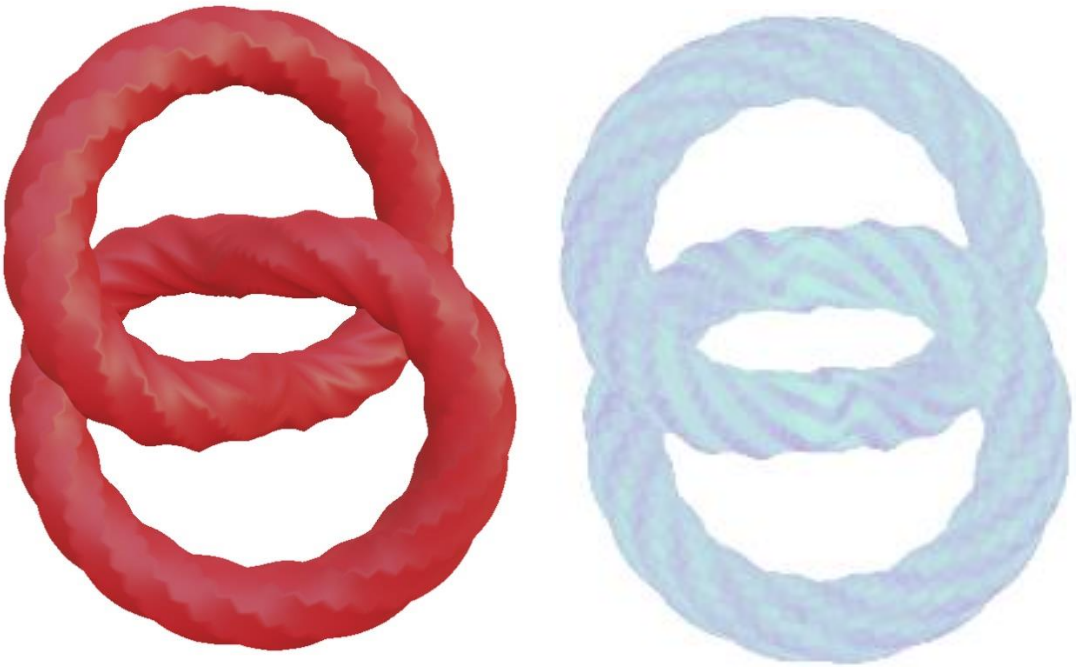
Εικόνα 21 Αποχρώσεις αριστερά/γαλάζιο μικρής με δεξιά/μωβ μεγάλης καμπυλότητας [0,8].

Ως αποτελέσματα θα αναλυθούν 5 διαφορετικά νέφη σημείων με διαφορετικές παραμέτρους και τους χρόνους εκτέλεσης τους. Οι τιμές της καμπυλότητας και οι χρόνοι θα συγκριθούν μεταξύ αυτών των 5 παραδειγμάτων για να δειχθεί η διαφορά στην επιλογή κάθε παραμέτρου. Οι 2 βασικοί παράμετροι είναι η επιλογή των γειτόνων και η επιλογή των ακτινών του τροχού ισορροπίας έτσι ώστε να υπάρχει μεγαλύτερη ακρίβεια στα αποτελέσματα αλλά με επιβάρυνση στον χρόνο εκτέλεσης. Τα πρώτα παραδείγματα και η υλοποίηση τους γίνεται με 20 γείτονες και 8 ακτίνες στον τροχό.

Στις παρακάτω εικόνες παρατηρούμε το λαιμός του αρκουδιού και τη μέση του μανιταριού, να εμφανίζονται διαφορετικές αποχρώσεις του πράσινου σε σχέση με τα αυτιά, το κέντρο της κοιλιάς του αρκουδιού. Αντίστοιχες περιοχές φαίνονται στον δεσμό, το μανιτάρι, τον άνθρωπο, και το Stanford bunny.

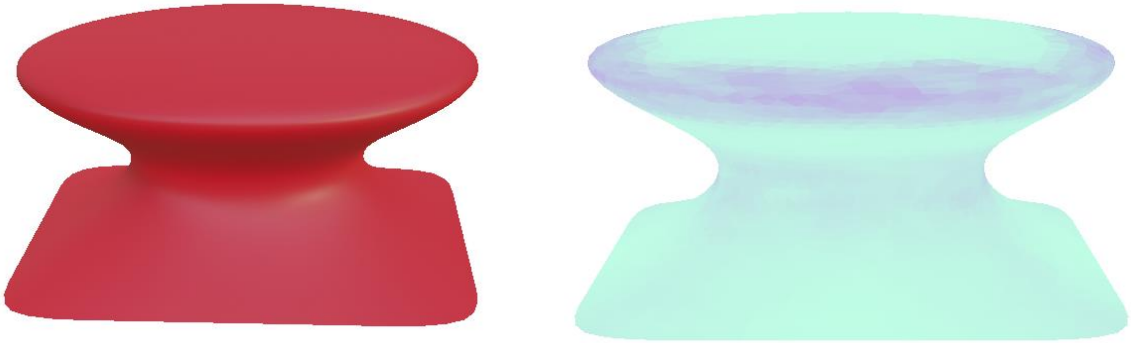
Εικόνα 22. Φιγούρα αρκουδιού με 13826 σημεία, (α) αρχικό - (β) καμπυλότητα ομπρέλα

(α) (β)



Εικόνα 23. Φιγούρα δεσμού (2 κρίκοι) με 5760 σημεία, (α) αρχικό - (β) καμπυλότητα

(α) (β)



Εικόνα 24. Φιγούρα πυρηνικού μανιταριού με 2337 σημεία, (α) αρχικό - (β) καμπυλότητα ομπρέλα

(α) (β)

Εικόνα 25. Φιγούρα ανθρώπου με 17495 σημεία, (α) αρχικό - (β) καμπυλότητα ομπρέλα

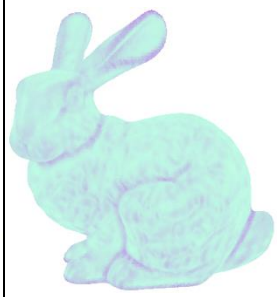
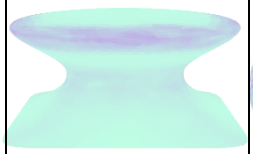
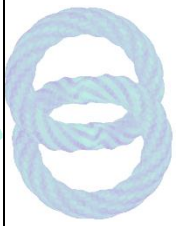
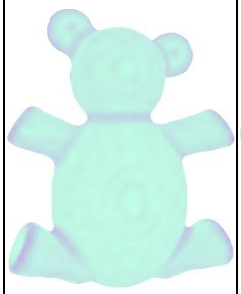
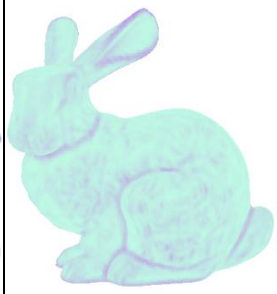
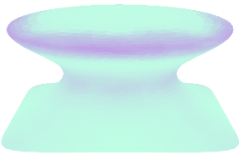
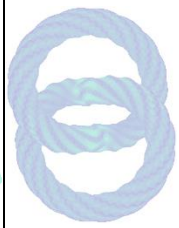
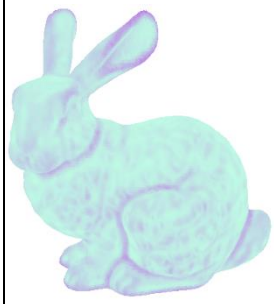
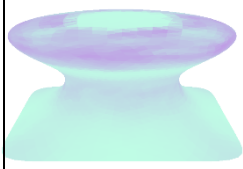
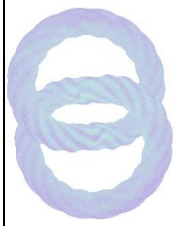
(α) (β)

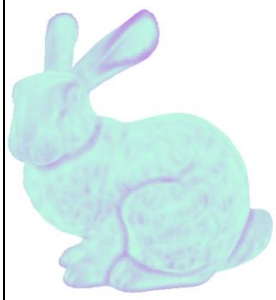
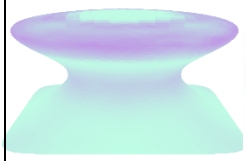
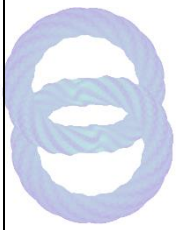
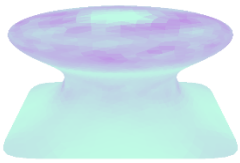
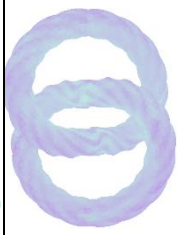
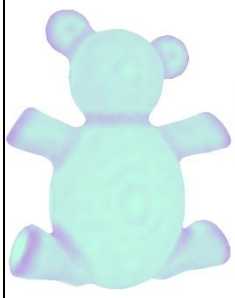
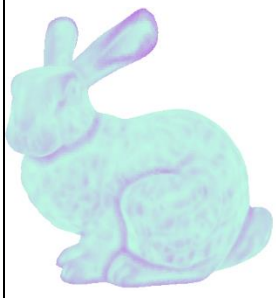
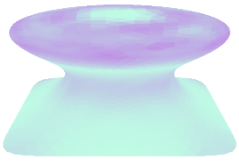
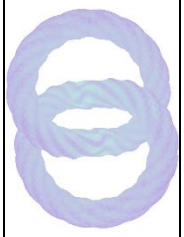
Εικόνα 26. Φιγούρα Stanford Bunny με 34834 σημεία, (α) αρχικό - (β) καμπυλότητα ομπρέλα

(α) (β)

Στον πίνακα 1 συνοψίζονται οι διάφορες εκτελέσεις της υλοποιημένης μεθόδου με διαφορετικές ρυθμίσεις στους αρχικούς γείτονες (20, 40, 55) και στον αριθμό διαμερίσεων του εφαπτόμενου επιπέδου (8, 18, 36). Όσο μεγαλύτερες επιλεχθούν αυτές οι τιμές τόσο μεγαλύτερη γίνεται και η περιοχή δειγματοληψίας γειτόνων με πιο εμφανή και ακριβή αποτελέσματα ομαλής συνέχειας στη καμπυλότητα των επιφανειών.

Όνομα	**Αρκούδος**	**Κουνέλι**	**Άνθρωπος**	**Μανιτάρι**	**Δεσμός**
Πλήθος σημείων	13826	34834	17495	2337	5760

Αρχική εικόνα					
Γείτονες 20 Ακτίνες 8					
Γείτονες 20 Ακτίνες 18					
Γείτονες 40 Ακτίνες 18					

Γείτονες 40 Ακτίνες 36					
Γείτονες 55 Ακτίνες 18					
Γείτονες 55 Ακτίνες 36					

Πίνακας 2.Εκτελέσεις μεθόδου με ρυθμίσεις αρχικών γειτόνων και διαμερίσεις εφαπτόμενου επιπέδου

Στον πίνακα 2 φαίνονται οι χρόνοι (σε δευτερόλεπτα) για κάθε βήμα της μεθόδου όπως η εισαγωγή των σημείων σε νέφος, υπολογισμός των κανονικών διανυσμάτων, το kd-tree, octree, selection-sort για τον υπολογισμό των 20 γειτόνων, υπολογισμός του συντελεστή ομογένειας, και υπολογισμός του τελικού κανονικού διανύσματος μαζί με τον υπολογισμό της καμπυλότητας κάθε παραδείγματος.

	Αρκούδος	**Κουνέλι**	**Άνθρωπος**	**Μανιτάρι**	**Δεσμός**
Αριθμός σημείων	13826	34834	17495	2337	5760
Εισαγωγή του Point cloud	2.997s | 34%	8.029s | 35.4%	4.277s | 33%	0.514s | 33.5%	1.311s | 34%
Υπολογισμός 20-NN με Kd-tree	1.325s | 15%	3.443s | 15.1%	1.948s | 15%	0.223s | 14.5%	0.564s | 14.6%
Υπολογισμός 20-NN με Selection-Sort	6.209s | 43.8%	39.23s | 66.1%	9.7s | 49.6%	0.145s | 0.92%	1s | 23.7%
Υπολογισμός όλων των κανονικών διανυσμάτων	1.939s | 22%	5.264s | 23.2%	3.085s | 23.4%	0.351s | 22.9%	0.874s | 22.7%
Προβολή 20 γειτόνων	0.018s | 0.2%	0.05s | 0.2%	0.024s | 0.18%	0.003s | 0.2%	0.007s | 0.18%
Τοποθέτηση, περιστροφή και υπολογισμός *Hi*	2.462s | 28%	6.419s | 28.3%	3.545s | 27.2%	0.42s | 27.4%	1.057s | 27.4%

Υπολογισμός n και υπολογισμός καμπυλότητας ομπρέλας	0.011s | 0.1%	0.037s | 0.16%	0.014s | 0.1%	0.003s | 0.2%	0.006s | 0.15%
Συνολικός χρόνος (kd-tree 20-NN)	8.799s | 100%	22.67s | 100%	12.986s | 100%	1.531s | 100%	3.848s | 100%

Πίνακας 3. Χρόνοι εκτέλεσης περιπτώσεων νεφών σημείων με 20 αρχικούς και 8 τελικούς γείτονες.

Ο πίνακας 3 αποτυπώνει τους χρόνους μόνο για την επιλογή των 20 αρχικών και 8 τελικών γειτόνων. Αυτές οι 2 τεχνικές, Selection-sort και KD-tree, δεν επηρεάζουν το τελικό αποτέλεσμα/ακρίβειά της καμπυλότητας αλλά υλοποιήθηκαν μόνο για σύγκριση χρόνου εκτέλεσης ώστε να επιλεχθεί η καλύτερη τεχνική, η οποία είναι το kd-tree.

ΚΕΦΑΛΑΙΟ 5ο
Σύνοψη και μελλοντικές επεκτάσεις

Στην παρούσα εργασία υλοποιήθηκε η μέθοδος ομπρέλα και χρησιμοποιήθηκε πάνω σε 5 περιπτώσεις χρήσεις με γραφική αναπαράσταση έτσι ώστε να γίνουν κατανοητά τα αποτελέσματα και οι χρόνοι της μεθόδου. Τα νέφη σημείων που χρησιμοποιήθηκαν περιέχουν ένα μέσο αριθμό σημείων σε σχέση με σαρώσεις σκηνής αληθινού κόσμου (π.χ. ένα ολόκληρο δωμάτιο). Όλα τα αποτελέσματα και οι χρόνοι που αναφέρθηκαν έχουν υπολογισθεί από σύστημα με σχετικά περιορισμένους υπολογιστικούς πόρους ενώ ο κώδικας τρέχει αποκλειστικά σε ένα νήμα του επεξεργαστή (single-threaded CPU).

Στόχος αυτής της εργασίας είναι να προσφέρει καλύτερα και πιο ακριβή αποτελέσματα στην καμπυλότητα επιφανειών πάνω σε κάθε σημείο θυσιάζοντας όμως τον χρόνο εκτέλεσης. Η συγκεκριμένη μέθοδος λειτουργεί καλύτερα σε νέφη σημείων στα οποία έχει γίνει κάποια είδους προ-επεξεργασία λόγω το ότι λειτουργεί με βάση τα γειτονικά σημεία. Εάν το νέφος σημείων έχει αποκομμένα ή/και ελλιπή σημεία τότε η κάθε γειτονιά που θα επιλεχθεί θα επηρεάζει αντίστοιχα την καμπυλότητα του κάθε σημείου.

Στη μελέτη που έγινε συγκρίναμε τα αποτελέσματα της μεθόδου Ομπρέλα χρησιμοποιώντας διαφορετικό πλήθος γειτόνων ή/και διαφορετικό πλήθος ομογενών γειτόνων. Η επιλογή των γειτόνων θα πρέπει να οριστεί από τον χρήστη ως παράμετρος. Ως συμπέρασμα φαίνεται ότι όσο περισσότεροι είναι οι αρχικοί ή/και οι τελικοί γείτονές τόσο αυξάνει ο χώρος δειγματοληψίας και η ακρίβεια στα αποτελέσματα της καμπυλότητας αλλά ταυτόχρονα και ο χρόνος εκτέλεσης. Οι δύο τεχνικές υπολογισμού αρχικών γειτόνων που χρησιμοποιήθηκαν παράγουν ίδια αποτελέσματα, αλλά με σημαντική διαφορά στο χρόνο εκτέλεσης. Αυτό το βήμα της μεθόδου είναι το πιο υπολογιστικά ακριβό και για αυτό προτάθηκαν 2 διαφορετικές τεχνικές. Μελλοντικά θα πρέπει να βελτιωθεί η υλοποίηση αυτού του βήματος με λιγότερο υπολογιστικά δαπανηρή υλοποίηση ενώ θα πρέπει να διερευνηθεί και η μετατροπή του αλγορίθμου σε παράλληλης επεξεργασίας για την εκμετάλλευση πολλαπλών νημάτων CPU.

Βελτίωση σε επίπεδα χρόνου εκτέλεσης αλλά και σε επίπεδο ποιότητας αποτελεσμάτων θα μπορούσε να επιτευχθεί και στο βήμα τοποθέτησης και περιστροφής του τροχού ισορροπίας αποφεύγοντας τους δαπανηρούς υπολογισμούς acos. Επίσης, η χρήση πιο σύνθετων δομών δεδομένων, πχ. hash table για την δυναμική αποθήκευση των αρχικών γειτόνων θα βελτιώσει τη χρόνο εκτέλεσης της μεθόδου. Επιπλέον, βελτιώσεις μπορούν να γίνουν για κάθε ένα από τα βήματα του πίνακα 3 ώστε να επιτευχθεί βελτίωση στον χρόνο εκτέλεσης χρησιμοποιώντας διαφορετικούς τρόπους και τεχνικές όπως πχ. η χρήση octree.

Η εργασία σε θέμα δυνατοτήτων συστημάτων και κώδικα μπορεί να βελτιωθεί αρκετά χρησιμοποιώντας πολυνηματικούς πολυπύρηνους επεξεργαστές, προγραμματισμό OpenMP και προγραμματισμό σε GPU/CUDA. Ο χρόνος εκτέλεσης μπορεί να μειωθεί ισάξια με τους πυρήνες που παρέχονται από το σύστημα (π.χ. για 8 πυρήνες ο χρόνος μειώνεται έως και 8 φορές) .

Βιβλιογραφία

[1] Schwarz, Sebastian, et al. "Emerging MPEG standards for point cloud compression." *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 9.1 (2018): 133-148.

[2] A. Nguyen and B. Le, "3D point cloud segmentation: A survey," *2013 6th IEEE Conference on Robotics, Automation and Mechatronics (RAM)*, 2013, pp. 225-230, doi: 10.1109/RAM.2013.6758588.

[3] Pan, Yue, et al. "Iterative global similarity points: A robust coarse-to-fine integration solution for pairwise 3d point cloud registration." *2018 International Conference on 3D Vision (3DV)*. IEEE, 2018.

[4] Matthew Berger, Andrea Tagliasacchi, Lee Seversky, Pierre Alliez, Joshua Levine, et al.. State of the Art in Surface Reconstruction from Point Clouds. Eurographics 2014 - State of the Art Reports, Apr 2014, Strasbourg, France. 1 (1), pp.161-185, 2014, EUROGRAPHICS star report. <http://diglib.eg.org/EG/DL/conf/EG2014/stars/161-185.pdf>. <10.2312/egst.20141040>. <hal-01017700>

[5] Polthier, Konrad. *Polyhedral surfaces of constant mean curvature*. Diss. Habilitationsschrift TU Berlin, 2002.

[6] Lee, Suk-Ho, and Jin Keun Seo. "Noise removal with Gauss curvature-driven diffusion." *IEEE Transactions on Image Processing* 14.7 (2005): 904-909.

[7] Dyn, N., Hormann, K., Kim, S.J. and Levin, D., 2001. Optimizing 3D triangulations using discrete curvature analysis. Mathematical methods for curves and surfaces, 1, pp.135-146.

[8] Wang, Guolin, et al. "Point cloud simplification algorithm based on the feature of adaptive curvature entropy." *Measurement Science and Technology* 32.6 (2021): 065004.

[9] Polette, Arnaud, Jean Meunier, and Jean-Luc Mari. "“Shape-Curvature-Graph”: Towards a New Model of Representation for the Description of 3D Meshes." *International Conference on Augmented Reality, Virtual Reality and Computer Graphics*. Springer, Cham, 2017.

[10] Foorginejad, A., & Khalili, K. (2014). Umbrella curvature: a new curvature estimation method for point clouds. *Procedia Technology*, *12*, 347-352.

[11] Zeraatkar, Mojtaba & Khalili, Khalil & Foorginejad, Abolfazl. (2015). High-Precision Laser Scanning System for Three-Dimensional Modeling of Saffron Flower. Journal of Food Process Engineering. 39. n/a-n/a. 10.1111/jfpe.12248.

[12] Collis, R. T. H. "Lidar." *Applied optics* 9.8 (1970): 1782-1788.

[13] Peterson, Leif E. "K-nearest neighbor." *Scholarpedia* 4.2 (2009): 1883.

[14] Daneshmand, Morteza, et al. "3d scanning: A comprehensive survey." *arXiv preprint arXiv:1801.08863* (2018).

[15] Alghazzawi, Tariq F. "Advancements in CAD/CAM technology: Options for practical implementation." *Journal of prosthodontic research* 60.2 (2016): 72-84.

[16] Rusu, R. B., & Cousins, S. (2011, May). 3d is here: Point cloud library (pcl). In *2011 IEEE international conference on robotics and automation* (pp. 1-4). IEEE.

[17] G. Guennebaud, B. Jacob, et al., “Eigen v3,” http://eigen.tuxfamily.org, 2010.

[18] J. Reinders, Intel threading building blocks : outfitting C++ for multi-core processor parallelism. O’Reilly, 2007.

[19] M. Muja and D. G. Lowe, “Fast approximate nearest neighbors with automatic algorithm configuration,” in International Conference on Computer Vision Theory and Application VISSAPP’09). INSTICC Press, 2009, pp. 331–340.

[20] Woo, M., Neider, J., Davis, T., & Shreiner, D. (1999). OpenGL programming guide: the official guide to learning OpenGL, version 1.2. Addison-Wesley Longman Publishing Co., Inc.

[21] Colombo, Alessandro, Claudio Cusano, and Raimondo Schettini. "3D face detection using curvature analysis." *Pattern recognition* 39.3 (2006): 444-455.

[22] Miura, Kenjiro T., and Gobithaasan RU. "Aesthetic curves and surfaces in computer aided geometric design." *International Journal of Automation Technology* 8.3 (2014): 304-316.

[23] Rusinkiewicz, Szymon. "Estimating curvatures and their derivatives on triangle meshes." *Proceedings. 2nd International Symposium on 3D Data Processing, Visualization and Transmission, 2004. 3DPVT 2004.*. IEEE, 2004.

[24] Farin, Gerald E., and Gerald Farin. *Curves and surfaces for CAGD: a practical guide*. Morgan Kaufmann, 2002.