ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΑΓΡΟΝΟΜΩΝ ΚΑΙ ΤΟΠΟΓΡΑΦΩΝ ΜΗΧΑΝΙΚΩΝ - ΜΗΧΑΝΙΚΩΝ ΓΕΩΠΛΗΡΟΦΟΡΙΚΗΣ
ΤΟΜΕΑΣ ΤΟΠΟΓΡΑΦΙΑΣ
ΕΡΓΑΣΤΗΡΙΟ ΦΩΤΟΓΡΑΜΜΕΤΡΙΑΣ

Διπλωματική εργασία

ΑΝΑΠΤΥΞΗ ΑΛΓΟΡΙΘΜΟΥ ΓΙΑ ΤΗΝ ΠΑΡΑΓΩΓΗ ΑΝΑΠΤΥΓΜΑΤΩΝ ΚΥΛΙΝΔΡΙΚΩΝ ΕΠΙΦΑΝΕΙΩΝ

Οδυσσέας Γαλανάκης

Επιβλέπων :

Ανδρέας Γεωργόπουλος
Καθηγητής ΕΜΠ

Αθήνα 2021

NATIONAL TECHNICAL UNIVERSITY OF ATHENS
SCHOOL OF RURAL, SURVEYING AND GEOMATICS ENGINEERING
DEPARTMENT OF TOPOGRAPHY
LABORATORY OF PHOTOGRAMMETRY

Diploma Thesis

AN ALGORITHM FOR THE DEVELOPMENT OF CYLINDRICAL SURFACES

Odysseas Galanakis

Supervisor :

Andreas Georgopoulos
NTUA Professor

Athens 2021

```
from 'Γ.Σεφέρης' import 'Ὁ βασιλιὰς τῆς  Ἀσίνης'

"""
Κι ὁ ποιητὴς ἀργοπορεῖ κοιτάζοντας τὶς πέτρες κι ἀναρωτιέται
ὑπάρχουν ἄραγε
ἀνάμεσα στὶς χαλασμένες τοῦτες γραμμὲς τὶς ἀκμὲς τὶς αἰχμὲς
    τὰ κοῖλα καὶ τὶς καμπύλες _
ὑπάρχουν ἄραγε
ἐδῶ ποὺ συναντιέται τὸ πέρασμα τῆς βροχῆς τοῦ ἀγέρα καὶ τῆς φθορᾶς
ὑπάρχουν, ἡ κίνηση τοῦ προσώπου τὸ σχῆμα τῆς στοργῆς
ἐκείνων ποὺ λιγόστεψαν τόσο παράξενα μὲς στὴ ζωή μας
"""
```

Περίληψη

Η παραγωγή αναπτυγμάτων επιφανειών και ορθοφωτογραφιών κατά την κλασική φωτογραμμετρική μέθοδο επιτυγχάνεται με την απόκτηση υφής και γεωμετρικής πληροφορίας από προσανατολισμένες εικόνες μέσω μιας διαδικασίας αντίστροφης εφαρμογής της συνθήκης συγγραμμικότητας, αφινικού μετασχηματισμού, και ραδιομετρικής παρεμβολής. Η ταχέως εξελισσόμενη τεχνολογία σάρωσης με laser έχει φτάσει στη δυνατότητα να την αντικαταστήσει ή να τη συμπληρώσει, παράγοντας πυκνά νέφη σημείων με πληροφορία υφής. Σε αυτή την διπλωματική εργασία κατασκευάζεται ένας αλγόριθμος για την παραγωγή αναπτυγμάτων κυλινδρικών επιφανειών χρησιμοποιώντας δεδομένα νεφών σημείων, και υλοποιείται σε ένα περιβάλλον ανοικτού λογισμικού. Το παραγόμενο πρόγραμμα ως μια συστοιχία διαφόρων αρχείων πηγαίου κώδικα μπορεί να χρησιμοποιηθεί στο πλαίσιο της γεωμετρικής τεκμηρίωσης κτιρίων και μνημείων, με προοπτική την συμβολή στην συντήρηση και αποκατάστασή τους. Κάθε βήμα της διαδικασίας οπτικοποιείται και καταγράφεται σε εξωτερικό αρχείο κειμένου.

Abstract

The production of surface developments and orthophotos according to standard photogrammetric method is accomplished by obtaining texture and geometric information from oriented images through a process including reverse application of the colinearity condition, affine transformation, and colour interpolation. Rapidly advancing laser scanning technology has reached the capacity to compete with or complement this by producing dense point clouds with texture information. In this diploma thesis, an algorithm is constructed for the development of cylindrical surfaces through point cloud data acquisition, and implemented in an open source environment. The resulting program as an array of various source code files can be utilized within the geometric documentation of buildings and monuments, with an eye towards contributing to their maintenance and restoration. Every step of the process is visualized and logged in an external text file.

Πίνακας Περιεχομένων

1. Εισαγωγή

Οι καμπύλες επιφάνειες στις ανθρωπογενείς κατασκευές υπάρχουν από καταβολής χρόνου και απαντώνται σε μεγάλο μέρος της αρχιτεκτονικής τέχνης των ανθρώπινων πολιτισμών καθ' όλη τη διάρκεια της ιστορίας. Πολλά δείγματα αυτής της πολιτιστικής κληρονομιάς προσεγγίζουν την γεωμετρία του κυλίνδρου και ως εκ τούτου για την βέλτιστη υποστήριξη της συντήρησης, αναστήλωσης και αποκατάστασής τους μέσα από έργα και μελέτες, απαιτούνται εργαλεία που μπορούν να στηρίξουν την άρτια αντίληψη και αποτύπωση του χώρου τους μέσω της γεωμετρικής τεκμηρίωσης και ορθής απεικόνισης της μορφολογίας τους.

Η ορθοφωτογραφία αποτελεί ένα από τα βασικότερα αυτά εργαλεία, αν και δεν προσφέρεται τόσο για τις καμπύλες επιφάνειες όσο για τις επίπεδες, καθώς η ορθή προβολή αναπόφευκτα θα εισαγάγει σοβαρές παραμορφώσεις στην απεικόνιση του αντικειμένου. Προκύπτει έτσι η ανάγκη για τη δυνατότητα παραγωγής μιας επίπεδης προβολής της τρισδιάστατης μορφής των καμπύλων επιφανειών, γνωστής και ως "ανάπτυγμα" (development).

Κατά την αναλογική (και μετέπειτα ψηφιακή) διαδικασία παραγωγής αναπτυγμάτων, εκτελούνται πολλά κοινά βήματα με την αντίστοιχη διαδικασία παραγωγής ορθοφωτογραφιών, όπως ο εξωτερικός προσανατολισμός της εικόνας, ο καθορισμός της έκτασης της προβολής, η αντίστροφη εφαρμογή της εξίσωσης της συγγραμμικότητας και του αφινικού σχηματισμού, καθώς και η ραδιομετρική παρεμβολή για τον καθορισμό του χρώματος. Πρόκειται για χρονοβόρα διαδικασία η οποία ωστόσο μπορεί ως ένα βαθμό να αυτοματοποιηθεί ή να συγκεραστεί με καινοτόμες τεχνολογίες προς τη βελτίωση του αποτελέσματος και του απαιτούμενου χρόνου επεξεργασίας.

Η τεχνολογία σάρωσης με laser (laser scanning technology, LIDAR) αναπτύσσεται ραγδαία τις τελευταίες δύο δεκαετίες βρίσκοντας εφαρμογές σε πολλά προβλήματα που μέχρι την έλευσή της προσεγγίζονταν με τις καθιερωμένες αναλογικές και ψηφιακές φωτογραμμετρικές διαδικασίες. Σε αρκετά από αυτά, όπως στην παραγωγή ψηφιακών μοντέλων επιφάνειας και εδάφους (ιδίως σε αστικές περιοχές) και την αποτύπωση αντικειμένων μικρού μεγέθους ή πλάτους, τις ξεπερνά σε αποδοτικότητα και τείνει να τις αντικαθιστά χάρη στην υπολογιστική ταχύτητα και ακρίβεια που την χαρακτηρίζει. Ακόμα και αν η διείσδυσή της στο αντικείμενο της (επίσης ραγδαία εξελισσόμενης) επιστήμης της Φωτογραμμετρίας είναι ως ένα σημείο ανταγωνιστική, σε μεγάλο βαθμό την συμπληρώνει και η ολοκλήρωσή τους επιτρέπει την ανάπτυξη αποτελεσματικότερων και αποδοτικότερων εργαλείων με μεγαλύτερη ακρίβεια και πεδίο εφαρμογών (Baltsavias, 1999).

Οι αυτοματοποιημένες φωτογραμμετρικές τεχνικές που εγκολπώνουν την υφιστάμενη τεχνολογία σάρωσης με laser έχουν επιτρέψει την παραγωγή πυκνών νεφών σημείων (dense point clouds) τα οποία μπορούν πέρα από την γεωμετρική πληροφορία (συντεταγμένες, διανυσματικά στοιχεία) να φέρουν και ραδιομετρική πληροφορία (χρώμα, ένταση). Αυτό ανοίγει νέες δυνατότητες στον τρόπο με

τον οποίο μπορεί αυτή η πληροφορία να διαβαστεί και να αξιοποιηθεί, και ειδικά σε ένα περιβάλλον προγραμματισμού που μπορεί να διαχειριστεί εύκολα τον όγκο της και να παραγάγει αποτελέσματα που καλύπτουν τις ανάγκες αρκετών, φωτογραμμετρικών και μη, εφαρμογών.

1.1. Αντικείμενο εργασίας

Η παρούσα διπλωματική εργασία έχει ως στόχο την ανάπτυξη και υλοποίηση σε περιβάλλον ανοικτού λογισμικού, με τη χρήση της προγραμματιστικής γλώσσας Python, ενός αλγορίθμου ο οποίος θα αντλεί γεωμετρική και ραδιομετρική πληροφορία από ένα πυκνό νέφος σημείων και θα επιτρέπει στον χρήστη να αναπτύξει μια κυλινδρική επιφάνεια εντός του. Μετά από μια διαδικασία υπολογισμού του μαθηματικού μοντέλου που προσαρμόζεται βέλτιστα σε αυτήν, γραμμικών μετασχηματισμών και προβολής του σχετικού υποσυνόλου του νέφους στον τρισδιάστατο χώρο, παράγεται μέσω ραδιομετρικής παρεμβολής μια εικόνα που περιέχει το ανάπτυγμα της επιλεγμένης επιφάνειας.

1.2. Δομή εργασίας

Για την αναλυτική παρουσίαση της εργασίας χωρίζεται η ύλη που καλύπτει σε 6 κεφάλαια.

1. Στο παρόν κεφάλαιο γίνεται μια εισαγωγή στο θέμα που πραγματεύεται και εκφράζεται ο στόχος της.
2. Στο δεύτερο κεφάλαιο παρατίθεται το θεωρητικό υπόβαθρο του αντικειμένου και οι μέθοδοι που χρησιμοποιήθηκαν κατά την εκπόνηση της εργασίας.
3. Στο τρίτο κεφάλαιο γίνεται μια βιβλιογραφική ανασκόπηση υφιστάμενων προσεγγίσεων του θέματος στη ΣΑΤΜ-ΜΓ ΕΜΠ.
4. Στο τέταρτο κεφάλαιο περιγράφεται αναλυτικά ο σχεδιασμός αλγορίθμου και η υλοποίηση του, με την παράθεση του σχετικού κώδικα σε κάθε βήμα της διαδικασίας.
5. Στο πέμπτο κεφάλαιο εφαρμόζεται το τελικό πρόγραμμα σε δείγματα πραγματικών δεδομένων και γίνεται μια επισκόπηση του αποτελέσματος.
6. Στο έκτο κεφάλαιο εξάγονται συμπεράσματα σχετικά με την απόδοση της υλοποίησης του αλγορίθμου και προτείνονται περαιτέρω επεκτάσεις των δυνατοτήτων του.
7. Τέλος, σε ένα παράρτημα αναγράφονται οδηγίες για την εγκατάσταση και εκτέλεση του προγράμματος.

2. Θεωρητικό Υπόβαθρο

Προκειμένου να κατανοηθούν καλύτερα οι έννοιες, ορισμοί και διαδικασίες που έχουν οργανικό ρόλο στην ανάπτυξη του ζητούμενου αλγορίθμου, πρέπει να γίνει μια επισκόπηση των χαρακτηριστικών τους.

2.1. Γεωμετρικά Στοιχεία

Ο κύλινδρος (απειριζόμενου μήκους) ορίζεται γεωμετρικά ως το σχήμα που προκύπτει από την παράλληλη μετατόπιση μιας ευθείας (γενέτειρας) κατά μήκος μιας οδηγού καμπύλης. Συγκεκριμένα για τον κυκλικό κύλινδρο, μπορεί και να οριστεί ως το σχήμα εκ περιστροφής μιας ευθείας, όπου ο άξονα περιστροφής ταυτίζεται με τον μετέπειτα άξονα [συμμετρίας] του κυλίνδρου.

Αν και η παρούσα διπλωματική εργασία ασχολείται αποκλειστικά με την παραγωγή αναπτυγμάτων κυλίνδρων, ο κώνος ως επιφάνεια έχει επίσης μεγάλη σημασία στην προσαρμογή μαθηματικών μοντέλων σε πραγματικά δεδομένα.

2.1.1. Γενική Εξίσωση

Ο κύλινδρος και ο κώνος ανήκουν στις τετραγωνικές επιφάνειες (quadratic surfaces), αλγεβρικές επιφάνειες 2ου βαθμού που προκύπτουν από τη γενική εξίσωση:

$$f(x, y, z) = Ax^2 + By^2 + Cz^2 + Dxy + Eyz + Fxz + Gx + Hy + Iz + J = 0$$

όπου οι συντελεστές A, B... F ανήκουν στους πραγματικούς αριθμούς με έναν τουλάχιστον εξ αυτών διάφορο του μηδενός (αν όλοι είναι μηδέν η σχέση καταλήγει στην Gx + Hy + Iz + J = 0 που ισοδυναμεί με την εξίσωση του επιπέδου).

Τα σημεία μιας τετραγωνικής επιφάνειας διακρίνονται σε ομαλά και μη. Τα μη ομαλά σημεία ονομάζονται ιδιόμορφα ή διπλά σημεία της επιφάνειας (π.χ. η κορυφή κωνικής επιφάνειας). Οι τετραγωνικές επιφάνειες με όλα τα σημεία ομαλά ονομάζονται γνήσιες. Εάν περιέχουν μη ομαλά σημεία ονομάζονται εκφυλισμένες.

Ο κύλινδρος και ο κώνος ανήκουν επίσης στις ευθειογενείς καμπύλες. Από κάθε σημείο μιας ευθειογενούς επιφάνειας διέρχεται μία ευθεία όλα τα σημεία της οποίας είναι σημεία της επιφάνειας. Κάθε καμπύλη που συναντά όλες τις γενέτειρες της επιφάνειας ονομάζεται οδηγός καμπύλη της επιφάνειας.

Μία ευθειογενής επιφάνεια ονομάζεται αναπτυκτή εάν μπορεί να εφαρμοστεί ισομετρικά στο επίπεδο διατηρώντας αναλλοίωτες τις αποστάσεις των σημείων της. Σε κάθε αναπτυκτή ευθειογενή επιφάνεια το εφαπτόμενο επίπεδο στα σημεία μιας γενέτειρας παραμένει σταθερό.

2.1.2. Εξισώσεις κυλίνδρου

Για τους κυλίνδρους η γενική εξίσωση μπορεί να πάρει τη μορφή

$$f_C(x, y, z) = Ax^2 + By^2 + Cz^2 + Gx + Hy + Iz + J = 0$$

Όταν μια εκ των συντεταγμένων x, y, z στην παραπάνω εξίσωση μηδενίζεται, τότε η εξίσωση εκφράζει έναν κύλινδρο του οποίου ο άξονας ταυτίζεται με τον άξονα της μηδενικής συντεταγμένης, καταλήγοντας στις παραμετρικές εξισώσεις

$f_{Hx}(y, z) = c$ (κύλινδρος παράλληλος στον άξονα x)
$f_{Hz}(x, z) = c$ (κύλινδρος παράλληλος στον άξονα y)
$f_V(x, y) = c$ (κύλινδρος παράλληλος στον άξονα z)

Σε ορθοκανονικό (καρτεσιανό) σύστημα συντεταγμένων η εξίσωση f_V παίρνει τη μορφή

$$x_2 + y_2 = r_2$$

ενώ σε πολικό σύστημα συντεταγμένων εκφράζεται ως

$$x = r\cos\theta$$
$$y = r\sin\theta$$

Αν και έχει επικρατήσει ο όρος "κύλινδρος" να παραπέμπει σημασιολογικά στον τύπο του κυκλικού κυλίνδρου, υπάρχουν περισσότερα είδη αναλόγως της γεωμετρίας της οδηγού καμπύλης.

Ο ελλειπτικός κύλινδρος ορίζεται από ελλειπτική οδηγό καμπύλη και έχει την αντίστοιχη εξίσωση f_V όταν οι παράμετροι A, B έχουν το ίδιο πρόσημο (AB > 0)

$(x/a)^2 + (y/b)^2 = 1$ όπου a, b οι άξονες της έλλειψης

Ο υπερβολικός κύλινδρος αντίστοιχα ορίζεται από υπερβολική οδηγό καμπύλη και έχει την εξίσωση f_V όταν οι παράμετροι A, B έχουν διαφορετικό πρόσημο (AB < 0)

$(x/a)^2 - (y/b)^2 = 1$ όπου a, b οι ημιάξονες της έλλειψης

Τέλος, ο παραβολικός κύλινδρος προκύπτει από παραβολική οδηγό καμπύλη όταν A=1 και B=0 (AB = 0) χαρακτηριζόμενος από την αντίστοιχη εξίσωση f_V

$$x^2 + 2ay = 0.$$

(Albert, 2016).

Το ανάπτυγμα ενός κυκλικού κυλίνδρου μη απειριζόμενου μήκους είναι σχετικά απλό, αποτελούμενο από δύο κύκλους (τις βάσεις του) και ένα ορθογώνιο παραλληλόγραμμο πλάτους ίσου με το ύψος του κυλίνδρου και μήκους ίσου με την περίμετρό του *2πr*.

Για την ανάπτυξη μόνο μέρους του κυλίνδρου, το μήκος του αναπτύγματος θα ισούται με *θr* όπου θ η γωνία του τόξου που το ορίζει.

2.1.3. Εξισώσεις κώνου.

Στις κωνικές επιφάνειες η κίνηση της γενέτειρας ευθείας περνάει πάντα από ένα σταθερό σημείο, το οποίο ορίζεται ως η κορυφή της κωνικής επιφάνειας.

Η Κωνική επιφάνεια με κορυφή ένα σημείο (x_0, y_0, z_0) έχει εξίσωση

$$f(x-x_0, y-y_0, z-z_0) = 0$$

Ο ελλειπτικός κώνος χαρακτηρίζεται από εξίσωση παρόμοια με αυτήν του ελλειπτικού κυλίνδρου:

$$(x/a)^2 + (y/b)^2 = z^2$$ όπου a, b οι άξονες της έλλειψης

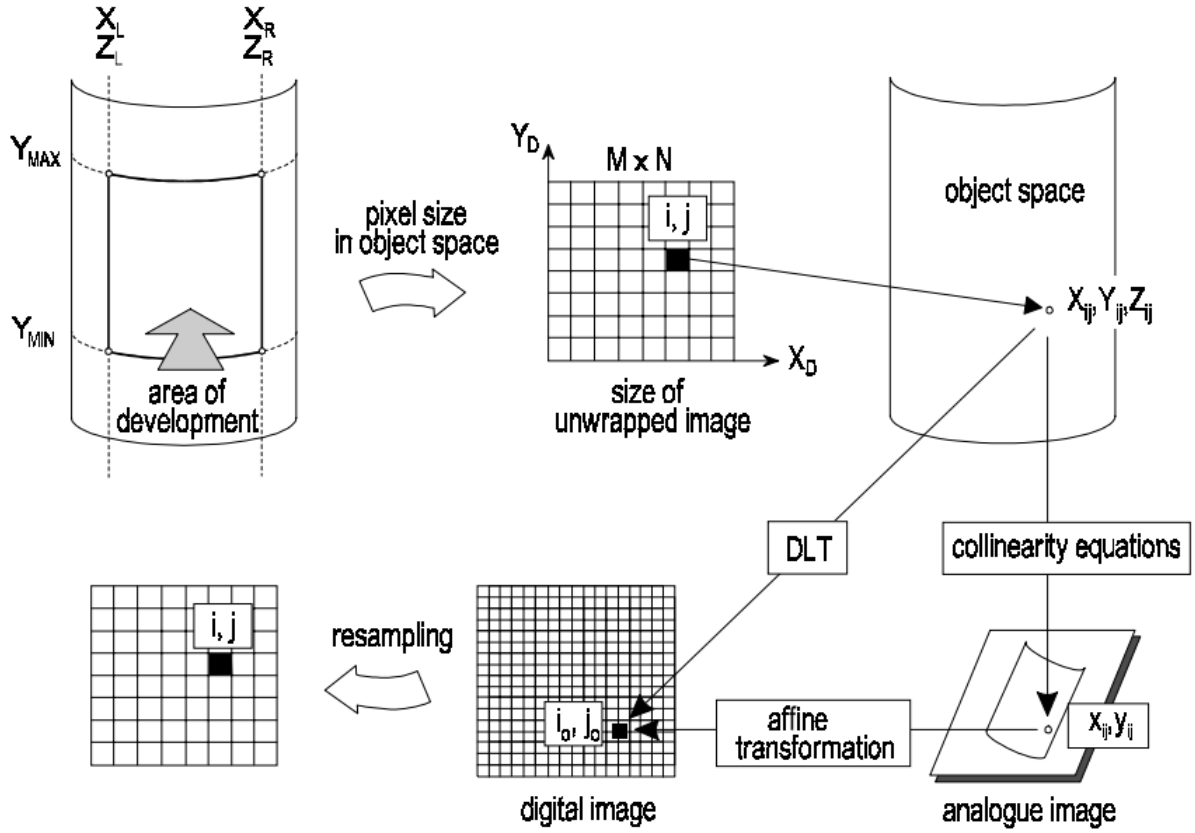
όπου αν a=b προκύπτει η εξίσωση του κυκλικού κώνου.

2.2. Φωτογραμμετρική Παραγωγή Αναπτυγμάτων

Το ανάπτυγμα στην γεωμετρία ορίζεται ως η προβολή στο επίπεδο μιας αναπτυκτής επιφάνειας ενός τρισδιάστατου στερεού ή αντικειμένου. Τα περισσότερα διακριτά είδη γεωμετρικών χώρων (όπως ο κύλινδρος, ο κώνος και τα πολυεδρικά στερεά) έχουν αναπτυκτές επιφάνειες που μπορούν με την κατάλληλη διαδικασία να αναπαρασταθούν σε δύο διαστάσεις, ενώ άλλα, όπως η σφαίρα και το ελλειψοειδές, δεν έχουν ορθό ανάπτυγμα, και από αυτήν την ιδιότητά τους πηγάζει σημαντικό μέρος των προβλημάτων που καλείται να επιλύσει ο Τοπογράφος Μηχανικός. Η διαδικασία αναπτύγματος μη αναπτυκτών επιφανειών απαιτεί την χρήση χαρτογραφικών προβολών και έχει αντιμετωπιστεί στο πρόσφατο παρελθόν (Γαλαζούλα, 2021).

Στην αναλυτική φωτογραμμετρία η μέθοδος που έχει προταθεί από τους Καρρά et al. (1996) για την παραγωγή κυλινδρικών αναπτυγμάτων από αναλογικές εικόνες ακολουθεί ως ένα βαθμό την μέθοδο παραγωγής ορθοφωτογραφίας. Με ελάχιστες τροποποιήσεις και συμπτύξεις βημάτων μπορεί να εξυπηρετήσει μια διαδικασία που χρησιμοποιεί ως δεδομένο πυκνά νέφη σημείων αντί αναλογικών φωτογραφιών:

1. Προσδιορίζονται οι παράμετροι της μαθηματικής επιφάνειας του κυλίνδρου που προσαρμόζονται βέλτιστα στο αντικείμενο.
2. Μετασχηματίζονται οι καρτεσιανές συντεταγμένες του νέφους σημείων στο σύστημα του κυλίνδρου μέσω στροφής και μετάθεσης.
3. Μετασχηματίζονται οι καρτεσιανές συντεταγμένες του νέφους σε κυλινδρικές συντεταγμένες μέσω ακτινικής προβολής.
4. Για κάθε εικόνα ορίζεται η περιοχή του αναπτύγματος από τα αριστερά $(X, Y)_L$ και δεξιά $(X, Y)_R$ σημεία, καθώς και τα ύψη Y_{max}, Y_{min}.
5. Ορίζεται το σύστημα συντεταγμένων X_DY_D του αναπτύγματος με γνωστές αντιστοιχίες στο σύστημα του αντικειμένου XYZ.
6. Επιλέγεται το μέγεθος της εδαφοψηφίδας του αναπτύγματος, ως της πραγματικής πλευράς του εικονοστοιχείου του στο χώρο του αντικειμένου, και συναρτήσει αυτής η ανάλυση M×N του αναπτύγματος.
7. Από αυτό το σημείο, για εικονοστοιχείο θεσης κέντρου (i, j) στο ανάπτυγμα υπολογίζεται η αντίστοιχη θέση του $(X, Y, Z)_{ij}$ στο χώρο του αντικειμένου, και υπάρχουν δύο επιλογές για τη συνέχεια της διαδικασίας:
    a. Μέσω αντίστροφης εφαρμογής της συνθήκης συγγραμμικότητας από το $(X, Y, Z)_{ij}$ εντοπίζεται η αντίστοιχη θέση του $(x, y)_{ij}$ στην αναλογική εικόνα, και μέσω αφινικού σχηματισμού αντιστοιχίζεται με το εικονοστοιχείο (i_o, j_o) της σαρωμένης αναλογικής εικόνας.
    b. Εναλλακτικά, πραγματοποιείται απευθείας ένας γραμμικός μετασχηματισμός μεταξύ του χώρου του αντικειμένου και της σαρωμένης εικόνας.
8. Παρεμβάλλεται χρώμα στο (i, j) από το (i_o, j_o).



Σχήμα 1: Στάδια αναπτύγματος κυλίνδρου (Karras et al. 1996)

Οι υπολογιστικές δυνατότητες διαχείρισης των νεφών σημείων καθιστούν την εναλλακτική του γραμμικού μετασχηματισμού στο βήμα 4b πολύ πιο ταχεία και εύκολη στην υλοποίηση, ενώ η παρεμβολή του χρώματος μπορεί είτε να γίνει με τον ίδιο τρόπο, αντιστοιχίζοντας τα εικονοστοιχεία του αναπτύγματος με αυτά μιας ψηφιακής εικόνας, είτε να αντληθεί από τυχόν ραδιομετρική πληροφορία που φέρουν τα σημεία του νέφους.

2.3. Μέθοδοι Προσαρμογής Επιφανειών

Για την βέλτιστη προσαρμογή μιας επιφάνειας σε ένα σύνολο σημειακών δεδομένων υπάρχουν δύο κοινότοπες μέθοδοι.

Η μέθοδος ελαχίστων τετραγώνων (least squares method), είναι μια κοινή προσέγγιση στην επίλυση συστημάτων εξισώσεων όταν υπάρχουν περισσότερες παρατηρήσεις από άγνωστες καθοριστικές

παραμέτρους. Αποτελεί τον ακρογωνιαίο λίθο των μεθόδων συνόρθωσης (regression), και αναπτύχθηκε από τους μαθηματικούς Legendre και Gauss στις αρχές του 19ου αιώνα.

Η δεύτερη μέθοδος καλείται RANSAC (random, sample and consensus) (Fischler and Bolles, 1981) και βασικό προτέρημά της είναι η αυξημένη δυνατότητα αντιμετώπισης ενός σταθερού προβλήματος των μεθόδων ελαχίστων τετραγώνων που είναι είναι η αντιπαραγωγική για την σωστή σύγκλιση του μοντέλου ύπαρξη ακραίων σημείων (outliers) στα δεδομένα.

Από την Μ.Ε.Τ. προκύπτουν περαιτέρω εξειδικευμένες παραλλαγές, όπως η μέθοδος συμβατικών παρατηρήσεων (όπου οι άγνωστοι θεωρούνται οι καλύτερες τιμές των στοιχείων που μετρήθηκαν) και η μέθοδος εμμέσων παρατηρήσεων (όπου ως άγνωστοι θεωρούνται όλες οι ανεξάρτητες καθοριστικές παράμετροι του μοντέλου). Αμφότερες αποτελούν ειδικές περιπτώσεις της γενικής περίπτωσης συνόρθωσης, όπου ζητούνται συγκεκριμένες παράμετροι του μοντέλου και σχηματίζονται εξισώσεις που περιέχουν τις ζητούμενες παραμέτρους και περισσότερα από ένα μετρημένα στοιχεία. (Αγατζά-Μπαλοδήμου, 2009). Αποκτάται έτσι η καλύτερη τιμή των ζητούμενων παραμέτρων από τη λύση ενός συστήματος μέσω αυτών των εξισώσεων συνθήκης.

Για τον κύλινδρο συγκεκριμένα, η εξίσωση που τον χαρακτηρίζει είναι μη γραμμική, οπότε απαιτείται η ύπαρξη αρχικών προσεγγιστικών τιμών και η γραμμικοποίηση των συνθηκών μεταξύ αγνώστων καθοριστικών παραμέτρων και παρατηρήσεων.

Έχουν εκπονηθεί πολλές έρευνες για την προσαρμογή κυλινδρικού μοντέλου σε τρισδιάστατα δεδομένα (Nurunnabi et al., 2017), που μπορούν να κατηγοριοποιηθούν σε μη-γραμμικές και γραμμικές προσεγγίσεις ελαχίστων τετραγώνων. Οι γραμμικές βασίζονται σε γραμμικοποίηση της ευκλείδειας απόστασης, ενώ οι μη-γραμμικές ελαχιστοποιούν την γεωμετρική απόσταση των σημείων από την επιφάνεια του κυλίνδρου (Marshall et al., 2001).

2.4. Περιβάλλον Προγραμματισμού

Η ανάπτυξη του αλγορίθμου πραγματοποιήθηκε στη γλώσσα προγραμματισμού Python 3 (Python Software Foundation, 2011) μιας σύγχρονης, ανοικτού πηγαίου κώδικα (open source) και υψηλού επιπέδου γλώσσας προγραμματισμού, εύκολης στην εκμάθηση και χρήση, με ευρύτατο πεδίο εφαρμογών και ιδιαιτέρως ενεργή κοινότητα.

Μια εγκατάσταση της Python περιέχει πολλές ενσωματωμένες δυνατότητες και λειτουργίες από μόνη της. Κατά την κρίση του χρήστη και αναλόγως της εργασίας που επιθυμεί να αναπτύξει, μπορεί να επεκταθεί με ακόμα περισσότερες εξειδικευμένες συλλογές κώδικα, γνωστές και ως βιβλιοθήκες (libraries).

Οι βιβλιοθήκες που χρησιμοποιήθηκαν στα πλαίσια της παρούσας εργασίας είναι:

1. NumPy, μια βιβλιοθήκη για υψηλού επιπέδου μαθηματικές εφαρμογές, αριθμητική ανάλυση, γραμμική άλγεβρα κ.α.
2. SciPy, μια βιβλιοθήκη για επιστημονικούς υπολογισμούς και μεθόδους
3. MatPlotLib, μια βιβλιοθήκη για χάραξη γραφικών παραστάσεων και σχημάτων από δεδομένα
4. Open3D, μια βιβλιοθήκη για την επεξεργασία και διαχείριση τρισδιάστατων δεδομένων

Από αυτές οι τρεις πρώτες χρησιμοποιούνται κατά κόρον στις περισσότερες επιστημονικές και τεχνικές εφαρμογές, δίνοντας στην Python τη δυνατότητα να ανταγωνιστεί δημοφιλείς γλώσσες του χώρου όπως η C++ και η Fortran.

Η Open3D εξυπηρετεί τις πιο εξειδικευμένες ανάγκες της παρούσας εργασίας. Είναι σχετικά νέα προσθήκη στο οπλοστάσιο της Python ανάμεσα σε διάφορες άλλες εναλλακτικές, όπως η Mayavi (γενικής χρήσης αλλά με πολλά προαπαιτούμενα), Panda3D (προσανατολισμένη στην ανάπτυξη παιχνιδιών) και η PyTorch3d (προσανατολισμένη στην μηχανική μάθηση). Αναπτύχθηκε από τους Zhou et al. το 2018 και συντηρείται ενεργά έκτοτε με διαρκείς επεκτάσεις και βελτιώσεις των δυνατοτήτων της.

Για την ανάπτυξη και διαχείριση πολύπλοκων προγραμμάτων κώδικα ενδείκνυται η χρήση ενός ολοκληρωμένου περιβάλλοντος ανάπτυξης (Integrated Development Environment) το οποίο θα υποβοηθά και θα βελτιστοποιεί τη ροή της παραγωγικής διαδικασίας με τα κατάλληλα εργονομικά εργαλεία συγγραφής, επισκόπησης και αποσφαλμάτωσης του κώδικα.

Για την παρούσα υλοποίηση χρησιμοποιήθηκε κατά κόρον το Spyder 5.0.3, το οποίο προσεγγίζει περισσότερο μεταξύ των διαφόρων IDEs για Python την λειτουργικότητα και εμφάνιση του Matlab και είναι ειδικά σχεδιασμένο για εφαρμογές προγραμματισμού από επιστήμονες, μηχανικούς και data scientists.

3. Βιβλιογραφική Ανασκόπηση

Η παρούσα εργασία πάνω στην αποτύπωση και ανάπτυξη κυλινδρικών επιφανειών δεν είναι παρά μόνο η τελευταία (πιθανότατα μόνο για ένα βραχύ διάστημα χρόνου) σε μια μακρά σειρά ακαδημαϊκών εργασιών και μελετών με το ίδιο αντικείμενο.

Ειδικά από το Εργαστήριο Φωτογραμμετρίας της σχολής Αγρονόμων και Τοπογράφων Μηχανικών - Μηχανικών Γεωπληροφορικής έχει εκπονηθεί μεγάλος όγκος έρευνας ως προς αυτό τις τελευταίες τρεις δεκαετίες.

Εν όψει του περιορισμένου χρόνου εκπόνησης της παρούσας διπλωματικής και της διαχείρισης αυτού του χρόνου, δεν αναζητήθηκαν περαιτέρω περαιτέρω μελέτες επί του ίδιου αντικειμένου στην διεθνή βιβλιογραφία ως σημείο αναφοράς, προκειμένου να αφιερωθεί περισσότερος χρόνος στην κατασκευή του πηγαίου κώδικα.

Μια ιστορική σειρά των προσπαθειών του εργαστηρίου έχει ως εξής:

1995: Αναπτύχθηκε από τους Ραψομανίκη και Σαντριβανόπουλο στο πλαίσιο της διπλωματικής τους εργασίας «Προσαρμογή και Ανάπτυγμα Κυλινδρικών και Κωνικών Επιφανειών» ένα πρόγραμμα για την παραγωγή κυλινδρικών και κυκλικών αναπτυγμάτων στην γλώσσα Autobasic του περιβάλλοντος υποβοηθούμενης σχεδίασης AutoCAD.

1996: Προτάθηκε μια πρότυπη μέθοδος παραγωγής κυλινδρικών αναπτυγμάτων από αναλογικές εικόνες από τους Καρρά, Πατιά και Πέτσα, την οποία εφάρμοσαν στην αποτύπωση ενός σιδηροδρομικού πύργου δεξαμενής νερού του 19^{ου} αιώνα, και δημοσίευσαν στο άρθρο τους “Digital Monoplotting and Photo-Unwrapping of Developable Surfaces in Architectural Photogrammetry”.

1999: Δημιουργήθηκε ένα πρόγραμμα για την παραγωγή κωνικών αναπτυγμάτων ενός παλαιού μύλου της νήσου Κιμώλου από τους Ζέρβα, Ξεκαλάκη, και Ταπεινάκη στο πλαίσιο της εργασίας τους «Μονοεικονική Ψηφιακή Απόδοση Κωνικών Αντικειμένων». Οι άγνωστες καθοριστικές παράμετροι του μοντέλου υπολογίστηκαν μέσω συστημάτων που συνδύαζαν την συνθήκη συγγραμμικότητας και την εξίσωση του κύκλου παρακάμπτοντας την μέθοδο των ελαχίστων τετραγώνων.

2002: Προσεγγίστηκε η ανάπτυξη ημιχωνίων (είδος τεταρτοσφαιρικού θόλου της μεσοβυζαντινής περιόδου) μέσω χαρτογραφικών προβολών από την Α. Βαλάνη κατά την εργασία της «Αξιοποίηση των Χαρτογραφικών Προβολών στην Αρχιτεκτονική Φωτογραμμετρία – Δημιουργία Αναπτυγμάτων για τα Ψηφιδωτά των Ημιχωνίων της Μονής Δαφνίου» στο πλαίσιο του θερινού

μαθήματος «Μεγάλες Ασκήσεις Φωτογραμμετρίας». Το πρόγραμμα αναπτύχθηκε σε περιβάλλον Matlab κάνοντας χρήση της μεθόδου ελαχίστων τετραγώνων.

2006: Αναπτύχθηκε ένα πρόγραμμα με πέντε αρθρώματα σε περιβάλλον Matlab για την προσαρμογή κωνικών επιφανειών σε νέφη σημείων και την παραγωγή αναπτύγματος, από τον Ι. Σκόνδρα στο πλαίσιο της διπλωματικής εργασίας του «Εικονιστικά Προϊόντα για την Γεωμετρική Τεκμηρίωση Μνημείων – Εφαρμογή στον Τάφο Ατρέα Μυκηνών». Μετά την επισκόπηση του εσωτερικού σχήματος του, ο θόλος του τύμβου καταμερίστηκε σε επιμέρους ζώνες ώστε να προσεγγιστούν βέλτιστα από διαφορετικές κωνικές επιφάνειες.

2018: Δημιουργήθηκε ένα πρόγραμμα σε περιβάλλον Matlab που προσαρμόζει κυλινδρικές επιφάνειες σε νέφη σημείων μέσω της μεθόδου RANSAC και τις αναπτύσσει, από την Θ. Βασιλείου στο πλαίσιο της διπλωματικής εργασίας της «Αναπτύγματα Κυλινδρικών Επιφανειών», το οποίο και εφαρμόστηκε σε μέρος της οροφής του Ναού του Ηφαίστου.

2020: Πραγματοποιήθηκε η γεωμετρική τεκμηρίωση κυλινδρικών επιφανειών από τις μεσαιωνικές οχυρώσεις της Χίου και της Ρόδου για σκοπούς συντήρησης, από τους Γεωργόπουλο, Σκαμαντζάρη, και Ταπεινάκη. Παρουσιάστηκαν ιδιαίτερες προκλήσεις κατά την διαδικασία επιλογής του βέλτιστου μοντέλου προσαρμογής για κάθε ζώνη των αντικειμένων λόγω της ελαφράς κλίσης, εγγενώς ασύμμετρης κατασκευής και αρχιτεκτονικής φυσιογνωμίας τους.

4. Υλοποίηση Αλγορίθμου

Μέθοδος Διαμερισματοποίησης

Σε κάθε μεγάλο εγχείρημα ανάπτυξης ενός προγράμματος πρέπει να διερευνηθεί σε πρώιμο στάδιο η βέλτιστη δομή του συνολικού εκτελούμενου κώδικα. Από τη στιγμή που ξεκινά να απαρτίζεται από εκατοντάδες γραμμές, η συγγραφή και η εκτέλεσή του σε ένα μοναδικό ενιαίο αρχείο εγκυμονεί πρακτικές δυσκολίες για τον χρήστη και τον υπολογιστή.

Ενδείκνυται ως εκ τούτου, βάσει της προσέγγισης του δομημένου προγραμματισμού (structural programming), η διάσπασή του σε μικρότερες οντότητες και διαδικασίες που μπορούν να κληθούν από το κυρίως πρόγραμμα στα σημεία που απαιτούνται και να συρραφούν κατά το δοκούν. Εξυπηρετείται έτσι η δοκιμή, η αποσφαλμάτωση και η συντήρηση των επιμέρους βημάτων του αλγορίθμου καθώς και η αποφυγή επανάληψης τμημάτων κώδικα.

Οι εν λόγω συνθετικές οντότητες μπορούν να είναι

- συναρτήσεις (functions), επαναχρησιμοποιούμενες διαδικασίες με δυνητικά δεδομένα εισόδου και εξόδου
- αρθρώματα (modules), αυτόνομα αρχεία που περιέχουν σειρές παρεμφερούς αντικειμένου συναρτήσεων
- κλάσεις (classes), προδιαγραφές δομών αποθήκευσης και βασικό στοιχείο της προσέγγισης του Αντικειμενοστραφή Προγραμματισμού.

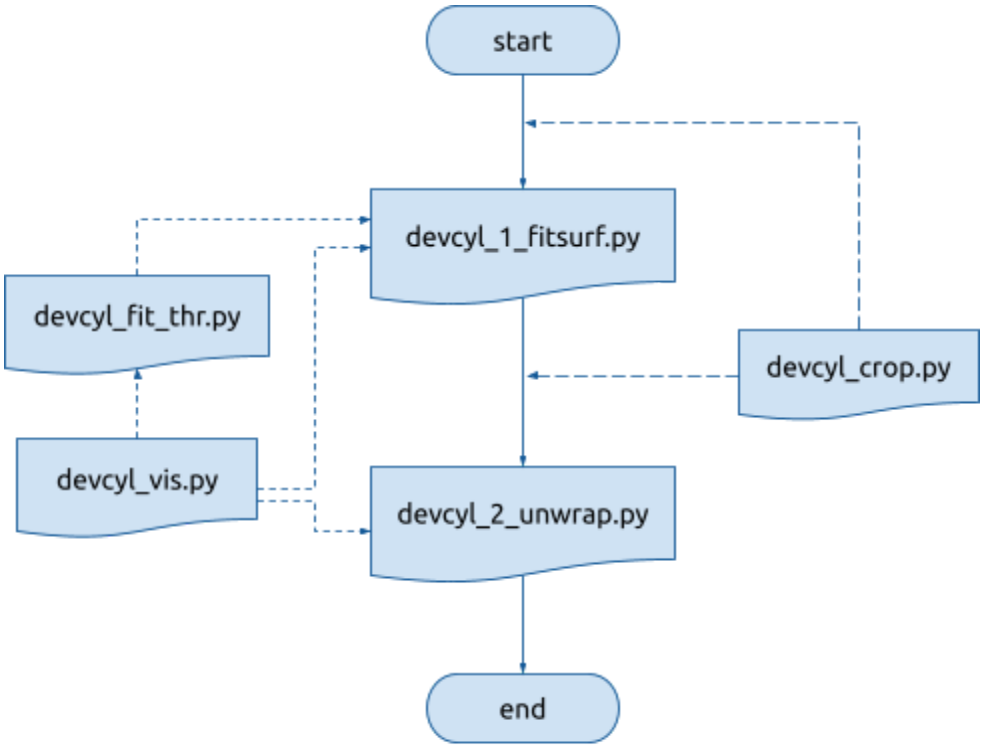
Δομή πρώτου επιπέδου

Κατόπιν μελέτης της ροής του αλγορίθμου, εκτιμήθηκε ότι πρέπει να δοθεί η ευχέρεια στον χρήστη να επαναλάβει ανεξάρτητα, προς απόκτηση διαφορετικών αποτελεσμάτων για εξειδικευμένους λόγους, τρια βασικά μέρη του προγράμματος:

1. την προσαρμογή του κυλίνδρου στο νέφος
2. τον προσδιορισμό της περιοχής αναπτύγματος
3. την παραγωγή του αναπτύγματος

Προκειμένου να μπορούν να εκτελεστούν τα παραπάνω βήματα (και δη τα 2., 3.) χωρίς να επιβάλλεται να τρέξει όλο το πρόγραμμα από την αρχή, και προκειμένου να αποφευχθούν πολύπλοκοι και επιρρεπείς σε σφάλματα βρόγχοι, κάθε ένα θα αποτελεί ξεχωριστό άρθρωμα / αρχείο κώδικα, εκτελέσιμο με τη σειρά που θα επιλέξει ο χρήστης.

Το άρθρωμα του προσδιορισμού της περιοχής αναπτύγματος ειδικά μπορεί και να μην εκτελεστεί καθόλου (εφ' όσον ο χρήστης επιθυμεί να αναπτύξει ολόκληρο το νέφος σημείων), ή ακόμα και να εκτελεστεί δις (καθώς λόγω της ευρύτερης λειτουργικότητάς του μπορεί να χρησιμοποιηθεί και πριν την προσαρμογή του κυλίνδρου για προκαταρκτικό καθαρισμό του νέφους).



Σχήμα 2: Διάγραμμα ροής αρθρωμάτων και τυπική διαδικασία εκτέλεσής τους.

Βάσει του παραπάνω σκεπτικού, το σύνολο του κώδικα συντάχθηκε σε 5 ξεχωριστά αρχεία:

1. *devcyl_1_fitsurf.py* : πρώτο βήμα, προσαρμογή του κυλίνδρου
2. *devcyl_crop.py* : δυνητικά ενδιάμεσο βήμα, αποκοπή περιοχής νέφους
3. *devcyl_2_unwrap.py* : δεύτερο βήμα, παραγωγή αναπτύγματος
4. *devcyl_fit_thr.py* : άρθρωμα που περιέχει τις διαδικασίες προσαρμογής κυλίνδρου
5. *devcyl_vis.py* : άρθρωμα που περιέχει τις διαδικασίες γεωμετρικής οπτικοποίησης

Τα 4 πρώτα θα επεξηγηθούν αναλυτικά σε αυτό το κεφάλαιο, ενώ το τελευταίο στο Παράρτημα.

Εισαγωγές και Καθολικές Παράμετροι

Στην αρχή κάθε αρχείου καθορίζονται οι εισαγωγές (imports) όπου φορτώνονται όλες οι απαραίτητες βιβλιοθήκες, συναρτήσεις και τα επικουρικά αρθρώματα στον πυρήνα (kernel) που θα τον εκτελέσει.

Συμβατικά (κατά PEP 8), αναγράφονται πρώτα οι ενσωματωμένες βιβλιοθήκες, μετά το εξωτερικό 3rd-party υλικό, και τέλος τα τοπικά επικουρικά αρθρώματα.

devcyl_1_fitsurf.py

```
import numpy as np
import time
import os
import copy

import open3d as o3d

import devcyl_fit_thr
import devcyl_vis
```

devcyl_fit_thr.py

```
import numpy as np
import copy
import time
from scipy.optimize
import leastsq
import matplotlib.pyplot
as plt

import open3d as o3d

import devcyl_vis
```

devcyl_2_unwrap.py

```
import numpy as np
import copy
import time
from PIL import Image
from scipy.interpolate
import griddata

import open3d as o3d

import devcyl_
```

Αμέσως μετά από τις εισαγωγές ορίζονται οι καθολικές παράμετροι, σταθερές που διέπουν το σύνολο του κώδικα εντός του αρθρώματος. Εν προκειμένω, ρυθμίζονται στοιχεία ποιότητας παρουσίασης και αναγνωσιμότητας όπως ο αριθμός των δεκαδικών ψηφίων εντός των πινάκων και το αρχικό μέγεθος του παραθύρου οπτικοποίησης των τρισδιάστατων στοιχείων του Open3D.

```
# display arrays in precision 4 decimals
np.set_printoptions(precision=4, suppress=True)

# set default open3d visualization window size
vww = 1600
vwh = 900
```

4.1. Προσαρμογή Κυλίνδρου

Το πρώτο εκτελέσιμο αρχείο στην αλληλουχία είναι το *devcyl_1_fitsurf.py,* το οποίο δέχεται ένα νέφος σημείων μιας κυλινδρικής επιφάνειας στο χώρο, εντοπίζει τις παραμέτρους του κυλίνδρου μέσω αλλεπάλληλων εφαρμογών της μεθόδου ελαχίστων τετραγώνων, και τις χρησιμοποιεί για να μετασχηματίσει το νέφος στο σύστημα του κυλίνδρου κατά μήκος του άξονα z.

4.1.1. Ανάγνωση Νέφους

Το πρόγραμμα ξεκινά ελέγχοντας την εγκατεστημένη έκδοση του Open3D και κατονομάζοντας το working directory, τον κατάλογο εντός του οποίου εκτελείται το πρόγραμμα και από τον οποίο αντλούνται τα τοπικά αρθρώματα.

Συνήθως το working directory ταυτίζεται με τον φάκελο που περιέχει τα εκτελούμενα αρχεία κώδικα, και τα παράγωγα της εκτέλεσης θα σώζονται εκεί.

```
print("Open3d Version: ", o3d.__version__)

# display the current working directory
cwd = os.getcwd()
print(f"\nCurrent working directory: {cwd}")
print("Files in the above folder may be simply specified by name.")
```

Ο χρήστης καλείται να δώσει το όνομα του αρχείου που περιέχει το νέφος σημείων, εφ' όσον αυτό βρίσκεται στο working directory.

> Για την ανάγνωση αρχείων από άλλο φάκελο πρέπει πριν το όνομα του αρχείου να αναγράφεται και η ακριβής διεύθυνση του φακέλου, ίσως μέσω αντιγραφής και επικόλλησης από το address bar, π.χ. *"C:\Users\Username\Documents\pointcloud.ply"* (Windows) *"/home/username/Documents/pointcloud.ply"* (Unix).
>
> Η συνάρτηση os.path.normpath φροντίζει για την αναγνωσιμότητα της διεύθυνσης ανεξάρτητα από το λειτουργικό σύστημα, αλλά δεν καλύπτει όλες τις περιπτώσεις σφαλμάτων σε άλλα σημεία. Παρατηρήθηκε λόγου χάριν, ότι η ύπαρξη ελληνικών χαρακτήρων στο file path εμποδίζει το Open3D από την επιτυχή ανάγνωση του αρχείου νέφους, οπότε ενδείκνυται τα ζητούμενα αρχεία να βρίσκονται στο working directory.

Μέσω της συνάρτησης os.path.isfile ελέγχεται αν όντως υπάρχει το δεδομένο αρχείο.

```
pcd_path = None

# ask the user to specify point cloud file
while True:
    pcd_path = os.path.normpath(input("\nEnter point cloud file path: "))
    if not os.path.isfile(pcd_path):
        print("Invalid path.")
    else:
        break
```

Ακολουθεί η ανάγνωση του νέφους σημείων. Από την βιβλιοθήκη Open3D 0.13.0 υποστηρίζονται οι ακόλουθες μορφές:

- .ply (Polygon File Format)
- .pcd (Point Cloud Data)
- .xyz (πίνακας συντεταγμένων)
- .xyzn (πίνακας συντεταγμένων με κάθετα διανύσματα)

- .xyzrgb (πίνακας συντεταγμένων με χρώμα)
- .pts (πίνακας συντεταγμένων με χρώμα ή/και τιμές έντασης του γκρι, μαζί με κεφαλίδα που περιέχει τον αριθμό των σημείων)

  Ο κοινότοπος τύπος αρχείου .obj μπορεί επίσης να διαβαστεί από την Open3D αλλά όχι με την ίδια εντολή, και μόνο για την εξαγωγή πολυγώνων και υφής από το αντικείμενο. Δεν υποστηρίζεται επί του παρόντος εξαγωγή μετρητικών πληροφοριών των σημείων, μια δυνατότητα που αποτελεί εκ των ων ουκ άνευ για την παρούσα υλοποίηση αλγορίθμου. Ενδέχεται να καταστεί διαθέσιμη στο μέλλον.

Όπως και σε πολλά επόμενα σημεία, χρονομετρείται η διάρκεια εκτέλεσης της εν λόγω εντολής με τη χρήση του time (ενσωματωμένου αρθρώματος της Python) και αναγράφεται μετά την ολοκλήρωσή της έτσι ώστε ο χρήστης να έχει μια ενημερωμένη αντίληψη του χρόνου που απαιτεί ο υπολογιστής του για κάθε βήμα.

```
# load point cloud
print(f"\nLoading '{pcd_path}' point cloud...")
tic = time.perf_counter()

pcd = o3d.io.read_point_cloud(pcd_path)
toc = time.perf_counter()
print(f"Loaded in {toc - tic:0.4f} sec.\n")
print(pcd_path, ":", pcd)
```

4.1.2. Προσαρμογή Κυλίνδρου στο Νέφος

Σε αυτό το στάδιο προσαρμόζεται στο νέφος μια κυλινδρική επιφάνεια χρησιμοποιώντας μια υλοποίηση της μεθόδου ελαχίστων τετραγώνων που καλείται από το άρθρωμα *devcyl_fit_thr.py*.

Αν η προσαρμογή είναι ανεπιτυχής δηλαδή ο εκτιμώμενος κύλινδρος δεν προσεγγίζει στοιχειωδώς την επιθυμητή θέση, κάτι που ο χρήστης μπορεί να ελέγξει εμπειρικά στην πρώτη οπτικοποίηση ή και με την επισκόπηση της κατανομής των αποστάσεων σε σχετικό ιστόγραμμα αμέσως μετά τον υπολογισμό τους, θα πρέπει ει δυνατόν να επεξεργαστεί το νέφος ώστε να είναι πιο ευδιάκριτη στον κώδικα η κυλινδρική επιφάνεια εντός του.

Αν δε η προσαρμογή είναι έστω εν μέρει επιτυχής, στις περισσότερες περιπτώσεις αναμένεται να υπάρχει μια διασπορά ακραίων σημείων (outliers) που απέχουν σημαντικά από την επιφάνεια έχοντας συμβάλει ωστόσο στους υπολογισμούς της προσαρμογής της.

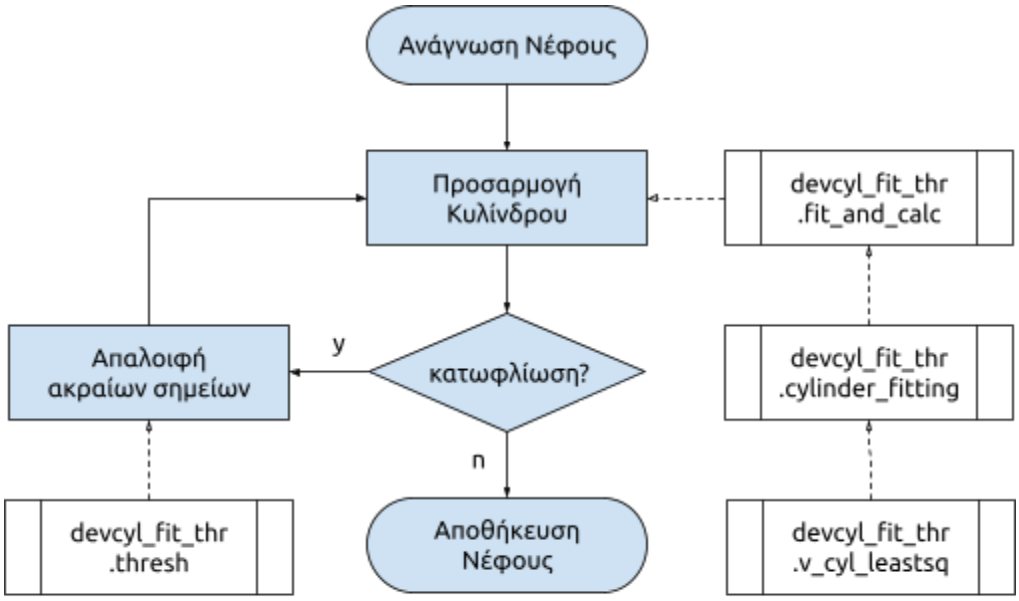
Ως εκ τούτου, ενδείκνυται η απαλοιφή των σημείων οι αποστάσεις των οποίων από την προσαρμοσμένη επιφάνεια ξεπερνούν ένα συγκεκριμένο κατώφλι (threshold). Κατόπιν τούτου, η

επιφάνεια μπορεί να προσαρμοστεί ξανά στο νέφος χωρίς να ληφθούν υπ' όψιν στους υπολογισμούς τα ακραία σημεία, με αποτέλεσμα μια πιο ακριβή προσέγγιση.

Η τιμή του κατωφλίου θα μπορούσε να οριστεί αυτόματα βάσει στατιστικής ανάλυσης της κατανομής, ως πολλαπλάσιο λ.χ. της τυπικής απόκλισης (2σ ή 3σ). Ωστόσο, κρίνεται πιο πρακτικό να έχει ο χρήστης την δυνατότητα να αλλάζει την τιμή δυναμικά, έχοντας υπ' όψιν του τόσο την κατανομή όπως την είδε στο ιστόγραμμα, όσο και το ποιες περιοχές του νέφους θα κοπούν βάσει της τιμής που θα ορίσει.

Δεν είναι απαραίτητο να τερματιστεί η διαδικασία εδώ. Ιδεατά ο χρήστης θα πρέπει να έχει τη δυνατότητα επανάληψης του παραπάνω βήματος, με νέο κατώφλι για κάθε νέα προσαρμογή, διαρκώς προσεγγίζοντας την επιθυμητή επιφάνεια και περιορίζοντας το νέφος στην εγγύτητά της έως ότου είναι ικανοποιημένος.

Για την υλοποίηση αυτής της δυνατότητας σχεδιάστηκε ένας βρόχος (loop), κατά τον οποίο μετά την πρώτη προσαρμογή εκτελείται μια ατέρμονη αλληλουχία κατωφλίωσης-επαναπροσαρμογής, έως ότου ο χρήστης αποφασίσει ότι δεν θέλει να κατωφλιώσει άλλο, οπότε και το πρόγραμμα προχωρά στο επόμενο στάδιο.



Σχήμα 3: Διάγραμμα ροής devcyl_1_fitsurf.py

4.1.2.1. Συναρτήσεις Προσαρμογής

Το συναρτησιακό μοντέλο ενός κυλίνδρου έχει 5 ανεξάρτητες καθοριστικές παραμέτρους (Rabbani et al., 2007) στις οποίες μπορούν να αναφέρονται τα δεδομένα. Απαιτείται ένα ενιαίο μοντέλο ώστε να μπορεί να εκφράσει κυλινδρικούς γεωμετρικούς χώρους σε όλες τις κατευθύνσεις, οι οποίες ωστόσο

μπορούν να καλυφθούν από τρία επιμέρους μοντέλα ανάλογα με τον κυρίως προσανατολισμό του εκάστοτε συνόλου μετρήσεων: f_V για κατακόρυφους κυλίνδρους και f_{Hx} , f_{Hy} για οριζόντιους κατά τον άξονα x και y αντίστοιχα.

Ο κατακόρυφος κύλινδρος μπορεί να παραμετροποιηθεί με

1. το σημείο $(x_κ, y_κ, 0)$ τομής του άξονά του με το επίπεδο XY
2. την στροφή $ω$ περί τον άξονα x
3. την στροφή $φ$ περί τον άξονα y
4. και την ακτίνα του r.

Το μοντέλο του εκφράζεται ως εκ τούτου από την εξής εξίσωση:

$$f_V(x_κ, y_κ, ω, φ, r) = X^2 + Y^2 - r^2 \qquad \text{όπου} \qquad (X, Y, Z)^T = R_φ R_ω (x-x_κ, y-y_κ, z)^T$$

Η συνάρτηση στον πυρήνα της προσαρμογής επιχειρεί να ελαχιστοποιήσει το άθροισμα των τετραγώνων της απόκλισης των σημείων από τον άξονα του κυλίνδρου και της ακτίνας του,

$$\sum_{i=1}^{N} |p(x_i, y_i, z_i)^2 - r^2| = \sum_{i=1}^{N} \left| \sqrt{(X^2 + Y^2)}^2 - r^2 \right|$$

χρησιμοποιώντας το υπολογιστικό άρθρωμα της βιβλιοθήκης SciPy scipy.optimize.leastsq το οποίο δέχεται ένα πίνακα μήκους n (αριθμός μετρήσεων) και επιστρέφει m αριθμό ρητών αριθμών (ανεξάρτητες καθοριστικές παράμετροι) ως την καλύτερη τιμή των ζητούμενων μεταβλητών.

```
# %% Least Squares Fitting

def v_cyl_leastsq(xyz, p):
    """
    xyz = matrix of at least 5 rows
                        and 3 columns (x,y,z of a cylindrical surface)
    p = 5-element array, initial values of the parameter
        p[0] = Xk, x coordinate of the cylinder centre (m)
        p[1] = Yk, y coordinate of the cylinder centre (m)
        p[2] = omega, rotation angle about the x-axis (rad)
        p[3] = phi, rotation angle about the y-axis (rad)
        p[4] = r, radius of the cylinder (m)
    ------ RETURNS ------------
    est_p = 5-element array, least squares estimate of p
    cov_p = inverse Hessian matrix, not used
    """
```

Διαβάζονται οι παράμετροι (x, y, z) έκαστου σημείου του νέφους.

```
x = xyz[:, 0]
y = xyz[:, 1]
z = xyz[:, 2]
```

Υπολογίζοντας την αναλυτική μορφή του ορθοκανονικού πίνακας στροφής των σημείων στο σύστημα του κατακόρυφου κυλίνδρου, βάσει των παραπάνω εξισώσεων σχηματίζεται το πρώτο σκέλος του ζεύγους εξισώσεων και καταχωρείται σε δευτερεύουσα συνάρτηση,

$$R\varphi R\omega = \begin{bmatrix} \cos\varphi & 0 & \sin\varphi \\ 0 & 1 & 0 \\ -\sin\varphi & 0 & \cos\varphi \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\omega & -\sin\omega \\ 0 & \sin\omega & \cos\omega \end{bmatrix} = \begin{bmatrix} \cos\varphi & \sin\omega\sin\varphi & \sin\varphi\cos\omega \\ 0 & \cos\omega & -\sin\omega \\ -\sin\varphi & \cos\varphi\sin\omega & \cos\varphi\cos\omega \end{bmatrix}$$

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = R\varphi R\omega \begin{bmatrix} x_\kappa - x \\ y_\kappa - y \\ z \end{bmatrix} = \begin{bmatrix} \cos\varphi\,(x_\kappa - x) & \sin\omega\sin\varphi\,(y_\kappa - y) & z\sin\varphi\cos\omega \\ 0 & \cos\omega\,(y_\kappa - y) & -z\sin\omega \\ -\sin\varphi\,(x_\kappa - x) & \cos\varphi\sin\omega\,(y_\kappa - y) & z\cos\varphi\cos\omega \end{bmatrix}$$

$$X^2 + Y^2 = (\cos\varphi\,(x_\kappa - x) + \sin\omega\sin\varphi\,(y_\kappa - y) + z\sin\varphi\cos\omega)^2 + (\cos\omega\,(y_\kappa - y) - z\sin\omega)^2$$

```
def rotation_matrix(p, x, y, z):
    return ((np.cos(p[3])*(p[0] - x) + np.cos(p[2])*np.sin(p[3])*z
            + np.sin(p[2])*np.sin(p[3])*(p[1] - y))**2
            + (np.cos(p[2])*(p[1] - y) - np.sin(p[2])*z)**2)
```

και εισάγεται στην εξίσωση (error function) που ο αλγόριθμος θα επιχειρήσει να ελαχιστοποιήσει με το δεύτερο σκέλος να αποτελεί το τετράγωνο της ακτίνας.

```
def equations(p, x, y, z):
    return rotation_matrix(p, x, y, z) - p[4]**2

est_p, cov_p = leastsq(equations, p, args=(x, y, z))

return est_p
```

Δημιουργείται νέα συνάρτηση για να λειτουργήσει ως πλαίσιο εφαρμογής της προηγούμενης.

```
def cylinder_fitting(xyz, p):
    """
    Applies least squares estimate of the functions above
    to fit a cylinder to scattered points in space
    --- RETURNS ------------
    cyl_p = 6-element list, estimated cylinder parametres
        cyl_p[0] = xk, x coordinate of the cylinder centre (m)
        cyl_p[1] = xk, y coordinate of the cylinder centre (m)
        cyl_p[2] = omega, rotation angle about the x-axis (rad)
        cyl_p[3] = phi, rotation angle about the y-axis (rad)
```

```
        cyl_p[4] = r, radius of the cylinder (m)
        cyl_p[5] = RyRx, auxiliary rotation matrix (3x3 numpy array)
    """
```

Καλείται η βασική υπολογιστική συνάρτηση και χρονομετρείται ο χρόνος προσαρμογής.

```
    print("\nFitting cylinder... ")
    tic = time.perf_counter()

    est_p = v_cyl_leastsq(xyz, p)

    toc = time.perf_counter()
    print(f"Fitted in {toc - tic:0.4f} sec.\n")
```

Αναγράφονται στην γραμμή εντολών οι νέες εκτιμώμενες τιμές των παραμέτρων του κυλίνδρου.

```
    # print parametres
    xk = est_p[0]
    yk = est_p[1]
    rot_x = est_p[2]  # omega = roll
    rot_y = est_p[3]  # phi = pitch
    cyl_radius = abs(est_p[4])
    print(f"Estimated parameters: \nxk = {xk} m\nyk = {yk} m"
          f"\nroll = {rot_x} rad\npitch = {rot_y} rad"
          f"\ncylinder radius = {cyl_radius} m")
```

Τέλος δημιουργείται (κυρίως για πρακτική χρήση σε διάφορες εντολές της Open3D) ο πίνακας στροφής RφRω ως αντικείμενο 3×3 numpy array.

```
    cosw = np.cos(rot_x)
    sinw = np.sin(rot_x)
    cosf = np.cos(rot_y)
    sinf = np.sin(rot_y)

    # calculate RyRx rotation matrix and print it
    rot_matrix = np.array(((cosf, sinw*sinf, sinf*cosw),
                           (0, cosw, -sinw),
                           (-sinf, cosf*sinw, cosf*cosw)))
    print("\nCylinder Rotation Matrix:\n", rot_matrix)

    cyl_p = [xk, yk, rot_x, rot_y, cyl_radius, rot_matrix]
    return cyl_p
```

Ακόμα και αυτή η συνάρτηση ωστόσο θα λειτουργήσει ως δομικό στοιχείο σε μια μεγαλύτερη, συγκεκριμένα μία που θα περικλείει τη διαδικασία προσαρμογής και τη διαδικασία υπολογισμού των αποστάσεων του νέφους από την υπολογισμένη επιφάνεια του κυλίνδρου.

```
# %% Fitting

def fit_and_calc(pcd):
    """
    Loads a point cloud and uses the above functions to fit a vertical cylinder
    to it and calculates how close each point comes to the cylinder surface.
    Returns a tuple of:
        the same point cloud,
        an array containing the 5 fitting parameters (xk, yk, roll, pitch, r)
        an array containing all the point-to-surface distances
        a segment of the cylinder axis
    """
```

Εισάγεται το αντικείμενο του νέφους, διαβάζονται τα σημεία του, και αναγράφονται οι ακραίες τιμές τους στη γραμμή εντολών, πριν ζητηθεί από τον χρήστη να πληκτρολογήσει τις αρχικές τιμές των παραμέτρων.

```
    # Read point cloud from PLY
    pcd_pts = np.asarray(pcd.points)
    print(np.asarray(pcd.points), "\n")

    print('Maximum xyz:', pcd.get_max_bound())
    print('Minimum xyz:', pcd.get_min_bound())

    # %% Fitting

    # ask the user to provide initial values
    init_p = np.array([1000, 1000, 0, 0, 1])

    print("\nDefine Initial Parameters:")
    init_p[0] = input("Xk, x coordinate of the cylinder centre (m) = ")
    init_p[1] = input("Yk, y coordinate of the cylinder centre (m) = ")
    init_p[2] = input("omega, rotation angle about the x-axis (rad) = ")
    init_p[3] = input("phi, rotation angle about the y-axis (rad) = ")
    init_p[4] = input("r, radius of the cylinder (m) = ")
    print("\nInitial Parameters: ", init_p)
```

Με δεδομένα αυτές τις τιμές και τον πίνακα συντεταγμένων των σημείων εκτελείται η προηγούμενη συνάρτηση, η οποία με τη σειρά της εκτελεί την βασική υπολογιστική συνάρτηση εντός της.

```
# run the fitting script
cyl_p = cylinder_fitting(pcd_pts, init_p)
```

Δημιουργούνται οι απαραίτητες γεωμετρικές οντότητες για την οπτικοποίηση του προσαρμοσμένου κυλίνδρου, συγκεκριμένα ο άξονας και η επιφάνεια.

Για την επιφάνεια χρησιμοποιείται τριγωνικό πλέγμα (mesh) χωρίς απόδοση υφής, καθώς η Open3D δεν έχει υλοποιήσει επαρκώς ακόμα βαθμούς διαφάνειας και είναι σημαντικό να δύναται ο χρήστης να δει μέσα από την επιφάνεια του κυλίνδρου.

```
# %% Visualize Mesh

axis_max = round(pcd.get_min_bound()[2])-1
axis_min = round(pcd.get_max_bound()[2])+1

cyl_axis = devcyl_vis.rotate_line(
    cyl_p[0], cyl_p[1], axis_min, axis_max, cyl_p[5])

# visualize fitted cylinder
wireframe = devcyl_vis.cyl_mesh(pcd, cyl_p)
o3d.visualization.draw_geometries(
    [pcd, wireframe, cyl_axis], 'Fitted Cylinder', vw_width, vw_height)
```

Για τον υπολογισμό των αποστάσεων κατασκευάζεται μια απλή συνάρτηση που χρησιμοποιεί αρθρώματα της βιβλιοθήκης NumPy για τις νόρμες των διανυσμάτων και το εσωτερικό γινόμενο που εμπλέκεται στην εξίσωση απόστασης σημείου για ευθεία (για κάθε διάσταση):

$d(l, p) = || (p - a) - ((p - a) \cdot n) n ||$ όπου $l = a + tn$, η εξίσωση της ευθείας με διάνυσμα t

```
# %% Distance Calc

def p2l_distance(a, b, p):
    """
    Point to Line Distance, for any dimension N (Vector Formulation)
    ----------
    a, b: any two points on the line (np.array[N,1])
    p : arbitrary point to which the distance is measured (np.array[N,1])
    """
    n = (a-b)/np.linalg.norm(a-b)
    return np.linalg.norm((p-a)-np.dot(p-a, n)*n)
```

Διαβάζονται τα ακραία σημεία από την γραμμή του άξονα όπως αυτή οπτικοποιήθηκε αμέσως πριν, για να υπολογιστεί το διάνυσμά του.

```
# get two points on the cylinder axis
cyl_axis_pts = o3d.geometry.LineSet.get_line_coordinate(cyl_axis, 0)
```

Δημιουργείται ένας κενός πίνακας N×1 για την καταχώρηση των αποστάσεων από τον άξονα.

```
print("\nCalculating point distances from cylinder axis,")
print("(may take several minutes)...")

# create empty N*1 array to point distances from cylinder axis
cyl_axis_distances = np.zeros((np.shape(pcd_pts)[0], 1))
tic = time.perf_counter()

for i in range(np.shape(pcd_pts)[0]):
    cyl_axis_distances[i] = p2l_distance(
        cyl_axis_pts[0], cyl_axis_pts[1], pcd_pts[i])
print(cyl_axis_distances)

toc = time.perf_counter()
print(f"Completed in {toc - tic:0.4f} sec.")
```

Βάσει αυτού, υπολογίζεται το απόλυτο της διαφοράς της απόστασης κάθε σημείου από τον άξονα και της εκτιμώμενης ακτίνας του κυλίνδρου, ως η απόσταση του ίδιου σημείου από την επιφάνεια του κυλίνδρου.

```
print("\nCalculating point distances from cylinder radius...")
cyl_s2p_dist = abs(cyl_axis_distances - cyl_p[4])
print(cyl_s2p_dist)
print("\nMax. point distance from surface:\n", np.max(cyl_s2p_dist), "m")
```

Σχεδιάζεται με την χρήση της βιβλιοθήκης Matplotlib ένα ιστόγραμμα που παρουσιάζει την κατανομή των αποστάσεων.

```
plt.hist(cyl_s2p_dist, bins=100)
plt.xlabel("Distance (m)")
plt.ylabel("Number of points")
plt.title("Point distances from fitted cylinder")
plt.show()
```

Τέλος, συγκεντρώνονται τα απαιτούμενα γνωστά στοιχεία για την συνέχεια του βρόχου (αντικείμενο νέφους, εκτιμώμενες παράμετροι, πίνακας αποστάσεων, γεωμετρικό αντικείμενο άξονα) σε μια πλειάδα (tuple), και εισάγονται κατευθείαν στην συνάρτηση κατωφλίωσης.

```
return pcd, cyl_p, cyl_s2p_dist, cyl_axis
```

4.1.2.2. Συνάρτηση Κατωφλίωσης

```
def thresh(pcd, cyl_p, cyl_s2p_dist, cyl_axis):
    """
    Receives the output of the above fit-and-calc function
    and removes any point in the cloud farther than a given threshold distance.
    A preview of the cropping is given and the user decides
    whether to try a different threshold value or proceed.
    """
```

Ζητείται από τον χρήστη να εισάγει την επιθυμητή απόσταση κατωφλίωσης, ελέγχοντας παράλληλα ότι δεν είναι μεγαλύτερη από την ακτίνα του κυλίνδρου.

```
    while True:
        # request the threshold for admitting points into the next iteration
        th = float(input("\nSelection Threshold"
                         "\nDefine max. point distance to cyl. surface (m): "))
        # but not larger than the cylinder radius
        if th >= cyl_p[4]:
            print("Selection Threshold must be less than the cylinder radius!")
            continue
```

Επιλέγονται τα σημεία του νέφους για τα οποία η τιμή τους στον πίνακα αποστάσεων είναι μικρότερη ή ίση του κατωφλίου, και δημιουργείται νέο αντικείμενο νέφους από αυτά.

```
        # create new point cloud according to threshold condition
        pcd1 = pcd.select_by_index(
            np.where(cyl_s2p_dist[:, 0] <= th)[0])
```

Η σειρά που περιέχει τις παραμέτρους του κυλίνδρου αντιγράφεται εις διπλούν, με την ακτίνα να αυξάνεται κατά την τιμή το κατωφλίου στο ένα αντίγραφο και να μειώνεται ομοίως στο άλλο. Βάσει αυτών των δύο αντιγράφων οπτικοποιούνται δυο κόκκινοι κύλινδροι που περικλείουν το υποσύνολο του νέφους που βρίσκεται εντός του κατωφλίου σε ένα παράθυρο. Σε δεύτερο διαδοχικό παράθυρο οπτικοποιείται η μορφή που θα έχει το νέφος μετά την κατωφλίωση.

```
        # create two cylinders to visually convey the threshold boundaries
        cyl_p_th1 = copy.deepcopy(cyl_p)
        cyl_p_th2 = copy.deepcopy(cyl_p)
        cyl_p_th1[4] = cyl_p[4] + th
        cyl_p_th2[4] = cyl_p[4] - th

        thresh1 = devcyl_vis.cyl_mesh(pcd, cyl_p_th1, 3, True)
        thresh2 = devcyl_vis.cyl_mesh(pcd, cyl_p_th2, 3, True)

        # two consecutive visualizations convey effect of threshold cropping
```

```
            o3d.visualization.draw_geometries([pcd, thresh1, thresh2, cyl_axis],
                                              'Original cloud and cropping guides',
                                              vw_width, vw_height)
            o3d.visualization.draw_geometries([pcd1, cyl_axis],
                                              'Cropped point cloud preview',
                                              vw_width, vw_height)
```

Ο χρήστης ερωτάται μετά τις δύο διαδοχικές οπτικοποιήσεις αν καλύπτεται από το αποτέλεσμα που μόλις είδε. Αν απαντήσει αρνητικά ο αλγόριθμος ξαναζητά τιμή κατωφλίωσης, ειδάλλως το νέφος που προέκυψε τροφοδοτείται στη συνάρτηση προσαρμογής για τη δεύτερη επανάληψη του κυρίως βρόχου, όπως αναλύεται αμέσως παρακάτω.

```
            # "continue or redefine?"
            thr_chk = input("\nRecalibrate cylinder on cropped cloud? (y/n)"
                            "\nEnter (y) to continue, (n) to redefine threshold: ")
            if thr_chk == "y":
                break
            elif thr_chk == "n":
                continue
            else:
                print("Invalid user input.")

        return pcd1, th
```

4.1.2.3. Υλοποίηση βρόχου στο κυρίως πρόγραμμα

Δημιουργούνται δύο κενές λίστες όπου θα καταχωρούνται οι παράμετροι της προσαρμογής και η τιμή του κατωφλίου αντίστοιχα.

```
# %% Fitting & Threshold Loop

# loop logging
parameter_log = []
thresh_log = []
```

Εντός ενός ατέρμονος βρόχου εκτελούνται η προσαρμογή και η κατωφλίωση καλώντας τις αντίστοιχες συναρτήσεις από το άρθρωμα. Στο ενδιάμεσό τους ο χρήστης ερωτάται αν θέλει να απαλείψει ακραία σημεία μέσω κατωφλίωσης, και αν δεν απαντήσει θετικά, ο βρόχος λύεται και καταγράφονται οι παράμετροι της τελευταίας προσαρμογής ως οι τελικές.

```
# fitting and thresholding loop
while True:

    # least squares fitting and axis-to-points distance calculation
    fitcyl = devcyl_fit_thr.fit_and_calc(pcd)
    parameter_log.append(fitcyl[1][:-1])

    # threshold
    thresh_q = input("\nApply distance threshold to crop off outliers?"
                     "\nEnter (y) to crop, any other key to continue: ")
    if thresh_q == "y":
        pcd1, thresh = devcyl_fit_thr.thresh(
            fitcyl[0], fitcyl[1], fitcyl[2], fitcyl[3])
        print("\nRepeating cylinder fitting on the new cloud...")
        pcd = pcd1
        thresh_log.append(thresh)
    else:
        break

pcd = fitcyl[0]
cyl_p = fitcyl[1]
mesh = devcyl_vis.cyl_mesh(pcd, cyl_p)

xk = fitcyl[1][0]
yk = fitcyl[1][1]
rot_x = fitcyl[1][2]
rot_y = fitcyl[1][3]
cyl_radius = fitcyl[1][4]
cyl_axis = fitcyl[3]
```

4.1.3 Μετασχηματισμός του Νέφους

Προκειμένου να μετατραπούν εύκολα οι καρτεσιανές συντεταγμένες του νέφους στο σύστημα του κυλίνδρου, ενδείκνυται η μετατόπιση και στροφή του νέφους και του προσαρμοσμένου κυλίνδρου του ούτως ώστε ο άξονάς του να ταυτίζεται με τον άξονα z.

Αυτό επιτυγχάνεται άμεσα εφ' όσον είναι γνωστές πλέον οι παράμετροι του κυλίνδρου: μετατόπιση xyz κατά (-xκ, -yκ, 0), και στροφή ωφκ κατά (-ω, -φ, 0).

```
# %% Alignment to Z Axis

# transform cloud and fitted cylinder to align with z-axis
R = pcd.get_rotation_matrix_from_xyz((-rot_x, -rot_y, 0))
pcd_z = copy.deepcopy(pcd).rotate(R, center=(xk, yk, 0))
mesh_z = copy.deepcopy(mesh).rotate(R, center=(xk, yk, 0))
pcd_z.translate((-xk, -yk, 0))
mesh_z.translate((-xk, -yk, 0))
```

Για την οπτική επαλήθευση σχεδιάζεται ανεξάρτητα ο άξονας z στην περιοχή των τιμών z του νέφους και οπτικοποιείται μαζί με το νέφος και τον άξονά του, προκειμένου να επιβεβαιωθεί ότι ταυτίζονται.

```
# visualize purple z_axis to check alignment
zamin = pcd_z.get_min_bound()[2] - 1
zamax = pcd_z.get_max_bound()[2] + 1
z_axis = devcyl_vis.create_line((0, 0, zamin), (0, 0, zamax), (.7, 0, .7))

o3d.visualization.draw_geometries(
    [pcd_z, mesh_z, z_axis], 'Final (Z-axis is purple)', vww, vwh)
```

Το νέφος αποθηκεύεται με την παρούσα του θέση και προσανατολισμό προς περαιτέρω ανεξάρτητη χρήση από τα υπόλοιπα εκτελέσιμα αρχεία του προγράμματος. Προκειμένου να εξυπηρετηθεί η πιο εύκολη εύρεση και διαχείριση των παραγώγων αρχείων του προγράμματος, το όνομά του δημιουργείται αυτόματα ως "Pointcloud-YYYYMMDD-HHMMSS.ply" καταχωρώντας ημερομηνία και ώρα με ακρίβεια δευτερολέπτου στα αντίστοιχα ψηφία.

Ο χρήστης μπορεί αργότερα να μετονομάσει το αρχείο νέφους κατά το δοκούν μέσα από το πρόγραμμα περιήγησης, συστήνεται ωστόσο να διατηρείται η χρονική υπογραφή για πρακτικούς λόγους ελέγχου και τήρησης αρχείων.

```
# save aligned cloud
timestamp = time.strftime("%Y%m%d-%H%M%S")
pcd_name = f"PointCloud-{timestamp}.ply"
o3d.io.write_point_cloud(pcd_name, pcd_z)
print(f"\nAligned cloud saved as {pcd_name}.")
```

4.1.4. Καταγραφή διαδικασίας

Με την ίδια χρονική υπογραφή στο όνομα αποθηκεύεται και ένα αρχείο κειμένου, συνοδευτικό στο νέο αρχείο νέφους, που καταχωρεί το όνομα του αρχικού αρχείου νέφους, τις υπολογισμένες παραμέτρους του προσαρμοσμένου κυλίνδρου με πλήρη αριθμό δεκαδικών ψηφίων (ούτως ώστε να μπορούν να διαβαστούν και να χρησιμοποιηθούν χωρίς απώλεια ακρίβειας μετά τον τερματισμό του παρόντος αρχείου), καθώς και τις τιμές που πήραν σε κάθε επανάληψη του βρόχου.

```
# %% Logging

# save fitting log, with cylinder radius on 3rd line

with open(f"Log-Fit-{timestamp}.txt", "w") as log:
    log.write("Source point cloud : " + pcd_path + "\n")
    log.write("Fitted Cylinder Radius (m):\n" + str(cyl_radius))

    log.write("\n\n--------------------------\nFitted cylinder parameters:\n")
    log.write("xk (m): \n" + str(parameter_log[-1][0]) + "\n")
    log.write("yk (m): \n" + str(parameter_log[-1][1]) + "\n")
    log.write("roll (rad): \n" + str(parameter_log[-1][2]) + "\n")
    log.write("pitch (rad): \n" + str(parameter_log[-1][3]) + "\n")

    log.write("\n-------------------------\nFitting loop iterations:\n")
    for i in range(len(parameter_log)):
        log.write("\nIteration " + str(i+1) + "\n")
        log.write("xk : " + "{:.3f}".format(parameter_log[i][0]) + "\n")
        log.write("yk : " + "{:.3f}".format(parameter_log[i][1]) + "\n")
        log.write("roll : " + "{:.3f}".format(parameter_log[i][2]) + "\n")
        log.write("pitch : " + "{:.3f}".format(parameter_log[i][3]) + "\n")
        log.write("radius : " + "{:.3f}".format(parameter_log[i][4]) + "\n")
        i += 1

    log.write("\n-----------------\nThresholds applied:\n\n")
    for i in range(len(thresh_log)):
        log.write("iteration " + str(i+1) + " : " + str(thresh_log[i]) + "m\n")
        i += 1

print(f"Log file saved as Log-Fit-{timestamp}.txt")
```

Τέλος, αναγράφονται υποδείξεις προς τον χρήστη αναφορικά με τα υπόλοιπα βήματα της διαδικασίας και τα αρχεία κώδικα που θα πρέπει να τρέξει για να προχωρήσει, και το παράθυρο γραμμής εντολών παραμένει ανοιχτό έως ότου ο χρήστης το τερματίσει πατώντας enter.

```
print("\nFURTHER STEPS:")
```

```
print("Execute 'devcyl_2_unwrap.py' and provide it with the above saved")
print(".ply file and .txt log to develop it to a 2D image. ")
print("In order to only develop a specific area within the point cloud")
print("execute 'devcyl_crop.py' before that and use its product .ply instead.")

input("Press Enter to exit.")
```

4.2. Προσδιορισμός περιοχής αναπτύγματος

Αν ο χρήστης επιθυμεί να αντιγράψει και να αποθηκεύσει προς μελλοντική χρήση μόνο ένα συγκεκριμένο τμήμα ενός νέφους σημείων, το devcyl_crop.py είναι ένα σύντομο script που μπορεί να εξυπηρετήσει αυτήν την λειτουργικότητα. Ο προσανατολισμός και οι τυχούσες υπολογισμένες κυλινδρικές παράμετροι του θυγατρικού νέφους δεν αλλάζουν, οπότε μπορεί αργότερα (στην παραγωγή αναπτύγματος) να χρησιμοποιηθεί σε σύζευξη με το ίδιο αρχείο καταγραφής παραμέτρων που χαρακτηρίζει και το νέφος από το οποίο προήλθε.

Ως ξεχωριστό αρχείο κώδικα μπορεί να εκτελεστεί ανεξάρτητα από την διαδικασία προσαρμογής κυλίνδρου και παραγωγής αναπτύγματος, ως μέρος της, ή και καθόλου.

Κατασκευάζεται ωστόσο σε μορφή συνάρτησης, ώστε να μπορεί να κληθεί από άλλα προγράμματα ή αρθρώματα, αν χρειαστεί (αυτή η δυνατότητα δεν αξιοποιήθηκε στην ροή του παρόντος προγράμματος, αλλά αποτελεί καλή προγραμματιστική πρακτική η υλοποίησή της).

Δέχεται ένα αναγνωσμένο νέφος σημείων ως δεδομένο, ανιχνεύει τα όριά του σε κάθε διάσταση, κατασκευάζει σύμφωνα με αυτά ένα ορθογώνιο στερεό που το περικλείει και το οπτικοποιεί.

```
# %% Development Area Selection

def crop_pcd_area(pcd):
    """
    User can crop given point cloud to an area of interest
    through use of an adjustable bounding box.
    """

    print("\nPoint cloud max. xyz:", pcd.get_max_bound())
    print("Point cloud min. xyz:", pcd.get_min_bound())
```

Κατόπιν ο χρήστης καλείται να εισαγάγει τα όρια της περιοχής του νέφους που θέλει να αποθηκεύσει, πληκτρολογώντας διαδοχικά καθ' υπόδειξιν τις μέγιστες και ελάχιστες συντεταγμένες του σε κάθε άξονα. Το στερεό που ορίζουν οπτικοποιείται μαζί με το νέφος, συν ένα βοηθητικό σύμβολο για την υπόδειξη του προσανατολισμού των αξόνων.

```
# loop visualizations until user satisfied with cropping box
while True:
    xmax = float(input("xmax = "))
    xmin = float(input("xmin = "))
    ymax = float(input("ymax = "))
    ymin = float(input("ymin = "))
    zmax = float(input("zmax = "))
    zmin = float(input("zmin = "))
    cmin = [xmin, ymin, zmin]
    cmax = [xmax, ymax, zmax]
    cbox = o3d.geometry.AxisAlignedBoundingBox(
        min_bound=cmin, max_bound=cmax)

    ax = o3d.geometry.TriangleMesh.create_coordinate_frame()
    ax.translate((xmin, ymin, zmin))
    o3d.visualization.draw_geometries(
        [pcd, cbox, ax], "Selected Area is within bounding box", vww, vwh)
```

Αφ' ότου κλείσει το παράθυρο της οπτικοποίησης, ο χρήστης αποφασίζει αν το τμήμα του νέφους εντός της περικλειόμενης περιοχής είναι αποδεκτό, ή αν θέλει να ορίσει ξανά τα όριά της.

```
    bound_q = input("Enter any key to redefine bounding box."
                    "\n Enter (q) to save (or not) and exit: ")
    if bound_q == "q":
        break
    else:
        continue

cpcd = copy.deepcopy(pcd).crop(cbox)
timestamp = time.strftime("%Y%m%d-%H%M%S")
```

Πλέον το νέο νέφος υπάρχει ως αντικείμενο, και δίνεται η επιλογή αποθήκευσής του ως ανεξάρτητο αρχείο νέφους. Ο χρήστης έχει επίσης την επιλογή να μην το αποθηκεύσει, για περιπτώσεις, φερ' ειπείν, όπου η συνάρτηση καλείται από άλλο πρόγραμμα και απαιτείται μόνο να χρησιμοποιηθεί το αντικείμενο του νέφους (ως παράγωγο της συνάρτησης) αμιγώς εντός του κώδικα.

```
# crop-and-save bounded area, or exit
while True:
    crop_q = input("\nCrop cloud to selected area and save it? (y/n)\n"
                   "Enter (y) to crop and save, (n) to exit: ")
    if crop_q == "y":
        pcd_name = f"CroppedPointCloud-{timestamp}.ply"
        o3d.io.write_point_cloud(pcd_name, cpcd)
        print("Saved as", pcd_name)
```

```
        elif crop_q == "n": break
        else: print("Invalid user input.")
    return cpcd
```

Πέρα από την παραπάνω συνάρτηση, το *devcyl_crop.py* περιέχει περαιτέρω κώδικα που δεν μπορεί να κληθεί από άλλο πρόγραμμα αλλά εκτελείται μόνο κατά την εκτέλεση του ίδιου του *devcyl_crop.py* ως κυρίως προγράμματος. Αυτό επιτυγχάνεται με τη χρήση της (κοινότυπης και εύχρηστης για τέτοιες περιπτώσεις) εντολής *if __name__ == "__main__"*. Θα ζητήσει την εισαγωγή ενός αρχείου νέφους και θα το αναγνώσει προκειμένου να εφαρμοστεί η συνάρτηση σε αυτό.

```
# %% Execute

if __name__ == "__main__":
    pcd_file = input("Please select point cloud file to load.")

    print(f"\nLoading '{pcd_file}' point cloud...")
    tic = time.perf_counter()

    pcd = o3d.io.read_point_cloud(pcd_file)
    toc = time.perf_counter()
    print(f"Loaded in {toc - tic:0.4f} sec.\n")
    print(pcd_file, ":", pcd)
```

Ο χρήστης μπορεί επίσης αντί για την παραπάνω διαδικασία, να χρησιμοποιήσει το διαδραστικό παράθυρο οπτικοποίησης του Open3D, το οποίο του επιτρέπει να την καθορίσει σχεδιάζοντας ορθογώνια ή πολύγωνα με το ποντίκι. Οδηγίες για τα κατάλληλα πλήκτρα αναγράφονται στην γραμμή εντολών κατά την δημιουργία του παραθύρου.

```
    box_or_lasso = input("\nTo select area by specific bounds, enter (1).\n"
                         "To select area manually, enter (2).\n")
    if box_or_lasso == "1":
        crop_pcd_area(pcd)
    elif box_or_lasso == "2":
        print("Press H to print help message.")
        print("Press [/] to increase/decrease field of view.")
        print("Press 'X'/'Y'/'Z' to enter orthogonal view along axis, "
              "   press again to flip.")
        print("Press 'K' to lock screen and to switch to selection mode:")
        print("   Drag for rectangle selection,")
        print("   or use ctrl + left click for polygon selection")
        print("Press 'C' to get a selected geometry and to save it.")
        print("Press 'F' to switch to freeview mode.")
        print("Press 'Q' to close the window.")
        o3d.visualization.draw_geometries_with_editing(
```

```
            [pcd], "Manual area selection", vww, vwh)
    else:
        print("Invalid user input.")

    input("Enter any key to exit.")
```

Αποτελεί μια πιο γρήγορη μέθοδο επιλογής περιοχής, καθώς ο χρήστης μπορεί εύκολα να επιτύχει παρόμοια αποτελέσματα με την πρώτη μέθοδο απλά πατώντας τα κατάλληλα πλήκτρα για ορθογραφική προβολή του νέφους στην οθόνη (“X”, “Y”, “Z”, ή “[“ έως το μέγιστο πεδίο όρασης) και σχεδιάζοντας ένα ορθογώνιο επίπεδο με το ποντίκι ή επιλέγοντας ένα πολύγωνο επίπεδο με πολλαπλά ctrl-click.

Με το πλήκτρο “C” ανοίγει ένα παράθυρο γραφικού περιβάλλοντος που αποθηκεύει όλα τα σημεία που προβάλλονται στο εν λόγω επίπεδο ως νέο νέφος (για την αποφυγή σφαλμάτων, ενδείκνυται η αποθήκευση να γίνεται εντός του wdir).

Τέλος, πέρα από την λειτουργικότητα της επιλογής περιοχής, το διαδραστικό παράθυρο οπτικοποίησης μπορεί να χρησιμοποιηθεί για την γρήγορη εμπειρική επισκόπηση της γεωμετρίας ενός νέφους εφ’ όσον δεν υπάρχουν εύκαιρα εργαλεία για αυτό.

4.3. Παραγωγή Αναπτύγματος

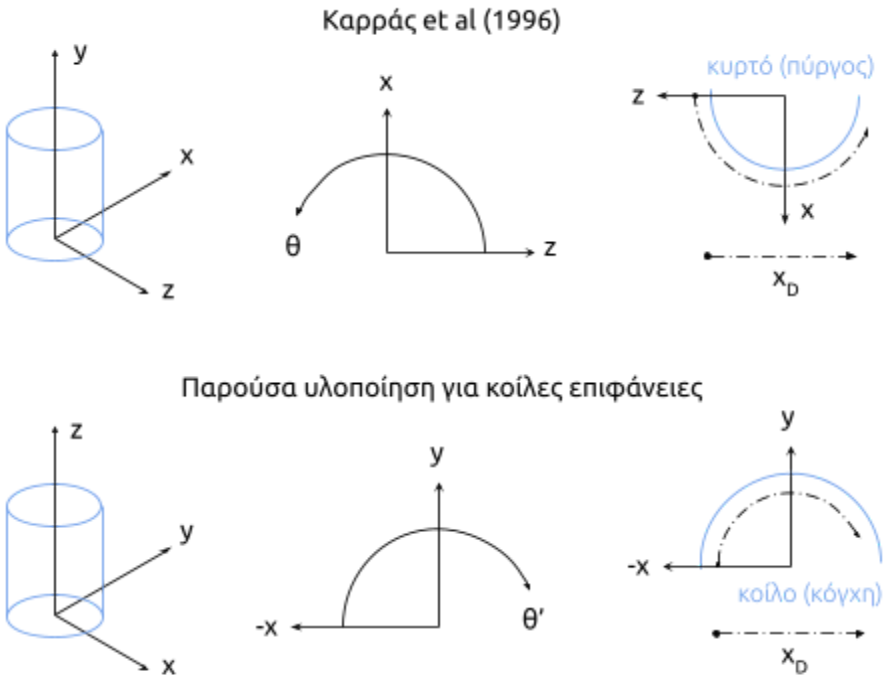
Πλέον υπάρχει ένα νέφος σημείων κυλινδρικής επιφάνειας με γνωστή την ακτίνα του προσαρμοσμένου κυλίνδρου, προσανατολισμένο κατά μήκος του άξονα z. Μπορεί να εκτελεστεί επί τούτου λοιπόν το *devcyl_2_unwrap.py,* που περιέχει το δεύτερο και τελευταίο σκέλος του συνολικού αλγορίθμου.

Οι Καρράς et al. (1996) προτείνουν για την μεταφορά από το σύστημα του κυλίνδρου στο δισδιάστατο σύστημα του αναπτύγματός του τον εξής μετασχηματισμό:

$$x_D = \alpha_i R$$
$$y_D = y_i$$

όπου (x_D, y_D) οι αναπτυκτές συντεταγμένες για κάθε σημείο (x_i, y_i) και α_i η γωνία που σχηματίζεται μεταξύ της ακτίνας του σημείου CQ_i και του θετικού άξονα z.

Λαμβάνοντας υπ' όψιν τον διαφορετικό προσανατολισμό αξόνων τους και το κυρτό της επιφάνειας που ανέπτυξαν (ενός πύργου), ορίζεται ένα παρόμοιο σύστημα πιο κατάλληλο για κοίλες επιφάνειες το οποίο θα επιτρέπει στο πρόγραμμα να διαβάζει τις τετμημένες x του αναπτύγματος δεξιόστροφα από το κέντρο του κυλίνδρου.



Σχήμα 4: Μετασχηματισμός από σύστημα κυλίνδρου

Η γωνία συναρτήσει της οποίας θα υπολογιστούν οι αναπτυκτές τετμημένες ορίζεται ως το είδωλο της θ κατά τον άξονα y, ξεκινώντας δεξιόστροφα από τον αρνητικό άξονα x:

$$\theta' = -\theta + \pi$$

με εύρος τιμών [0, 2π) όταν η θ κινείται στο (-π, +π].

Οι αναπτυκτές συντεταγμένες του και το εύρος τιμών τους (άρα και οι διαστάσεις του αναπτύγματος) υπολογίζονται βάσει του εξής μετασχηματισμού:

$$x_D = \theta'_i r$$
$$y_D = z_i$$

4.3.1. Προβολή Σημείων στον Κύλινδρο

Η διαδικασία εκκινεί με την εισαγωγή από τον χρήστη του ονόματος ή της διεύθυνσης του προσανατολισμένου νέφους και του συνοδευτικού αρχείου κειμένου με τις παραμέτρους προσαρμογής.

```
# %% Load

pcd_path = input("Enter point cloud file path: ")
log_path = input("Enter fitting log file path: ")

print(f"\nLoading '{pcd_path}' point cloud...")
tic = time.perf_counter()

pcd = o3d.io.read_point_cloud(pcd_path)
toc = time.perf_counter()
print(f"Loaded in {toc - tic:0.4f} sec.\n")
print(pcd_path, ":", pcd)
```

Δημιουργούνται δύο πίνακες διαστάσεων N×3 με τις συντεταγμένες (x, y, z) και χρώμα (r, g, b) κάθε σημείου αντίστοιχα.

```
poi = np.asarray(pcd.points)
col = np.asarray(pcd.colors)
```

Μέχρι στιγμής τα περισσότερα σημεία του νέφους έχουν διατηρήσει τις αυθεντικές σχετικές μεταξύ τους θέσεις και δεν ανήκουν στο γεωμετρικό χώρο της κυλινδρικής επιφάνειας, επομένως πρέπει να προβληθούν ακτινικά σε αυτή προτού η επιφάνεια αναπτυχθεί σε επίπεδο.

Διαβάζεται από το συνοδευτικό αρχείο κειμένου, όπου βρίσκεται στην 3η γραμμή συγκεκριμένα, η ακτίνα του προσαρμοσμένου κυλίνδρου.

```
# %% Projection to Cylinder

# read fitted radius from log file
with open(log_path, "r") as log:
    list_of_lines = list(log)
    r = float(list_of_lines[2])
print("Fitted Cylinder Radius =", r)
```

Δημιουργείται νέο αντικείμενο για το νέφος που θα απαρτίζεται από τις προβολές των σημείων του δεδομένου νέφους στον κύλινδρο, κι άλλο ένα για τις συντεταγμένες των προβολών.

```
# "p" for "projected"
ppcd = copy.deepcopy(pcd)
ppoi = np.asarray(ppcd.points)
```

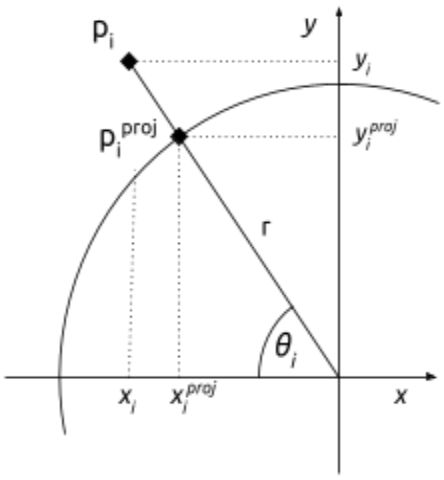
Η τομή ενός κυκλικού κυλίνδρου σε κάθε επίπεδο κάθετο στον άξονα συμμετρίας του είναι κύκλος, όπου κάθε σημείο του αρχικού νέφους poi_i και η προβολή του στον κύκλο $ppoi_i$ θα έχουν την ίδια γωνιακή συντεταγμένη θ_i αλλά και την ίδια τιμή z_i καθώς ο άξονας συμμετρίας ταυτίζεται με τον z.

Οι συντεταγμένες του poi_i είναι γνωστές, όποτε η φ_i δύναται να υπολογιστεί από αυτές ως

$$\theta = \text{atan2}(y, x)$$

όπου *atan2* μια πρακτική παραλλαγή της αντίστροφης εφαπτομένης *arctan* τέτοια ώστε να λαμβάνει υπ' όψιν το τεταρτημόριο της γωνίας και να έχει πραγματική τιμή για κάθε πιθανό x, y:

$$\operatorname{atan2}(y,x) = \begin{cases} \arctan\left(\frac{y}{x}\right) & \text{if } x > 0 \\ \arctan\left(\frac{y}{x}\right) + \pi & \text{if } x < 0 \text{ and } y \geq 0 \\ \arctan\left(\frac{y}{x}\right) - \pi & \text{if } x < 0 \text{ and } y < 0 \\ \frac{\pi}{2} & \text{if } x = 0 \text{ and } y > 0 \\ -\frac{\pi}{2} & \text{if } x = 0 \text{ and } y < 0 \\ \text{undefined} & \text{if } x = 0 \text{ and } y = 0. \end{cases}$$



Σχήμα 5: Προβολή σημείου στον κύκλο

Η ακτινική συντεταγμένη του $ppoi_i$ είναι επίσης γνωστή ως ίση με την ακτίνα του κυλίνδρου *r*.

Με γνωστές τις πολικές συντεταγμένες των προβολών μπορούν να υπολογιστούν οι καρτεσιανές:

$$x = r\cos\theta$$
$$y = r\sin\theta$$

```
print(f"\nProjecting '{pcd_path}' point cloud to cylinder surface...")
```

```
tic = time.perf_counter()

# polar into cartesian coordinates with quadrant-aware arctan function
for i in range(len(poi)):
    x = poi[i, 0]
    y = poi[i, 1]
    ppoi[i, 0] = r * np.cos(np.arctan2(y, x))
    ppoi[i, 1] = r * np.sin(np.arctan2(y, x))

toc = time.perf_counter()
print(f"Projected in {toc - tic:0.4f} sec.\n")
```

Για την επισκόπηση του αποτελέσματος σχεδιάζονται οι άξονες στην περιοχή του αναπτύγματος και οπτικοποιούνται μαζί με το προβεβλημένο νέφος.

```
# %% Visualization

# draw axes with 1m extra length beyond coordinate bounds
xamin = pcd.get_min_bound()[0] - 1
xamax = pcd.get_max_bound()[0] + 1
yamin = pcd.get_min_bound()[1] - 1
yamax = pcd.get_max_bound()[1] + 1
zamin = pcd.get_min_bound()[2] - 1
zamax = pcd.get_max_bound()[2] + 1

# cylinder centre lies at mean point height
zavg = (zamax + zamin) / 2

x_axis = devcyl_vis.create_line(
    (xamin, 0, zavg), (xamax, 0, zavg), (1, 0, 0))
y_axis = devcyl_vis.create_line(
    (0, yamin, zavg), (0, yamax, zavg), (0, 1, 0))
z_axis = devcyl_vis.create_line((0, 0, zamin), (0, 0, zamax), (0, 0, 1))

# draw axis unit vectors
ax_note = o3d.geometry.TriangleMesh.create_coordinate_frame()
ax_note.translate((0, 0, zavg), relative=False)
zavg_decimals = '{0:.3f}'.format(zavg)
print(f"Visualizing: Unit vector indicator point at (0, 0, {zavg_decimals})")

o3d.visualization.draw_geometries([ppcd, x_axis, y_axis, z_axis, ax_note],
                                  "Projected to Cylinder Surface", vww, vwh)
```

4.3.2. Αναζήτηση αρχής νέφους

Κατά την μέθοδο των Καρρά et al. (1996) οι διαστάσεις της περιοχής ανάπτυξης ορίζονται από τα σημεία (X_L, Y_L, Z_{max}), (X_L, Y_L, Z_{min}), (X_R, Y_R, Z_{max}), (X_R, Y_R, Z_{min}). Στην παρούσα υλοποίηση το αναγνωσμένο νέφος του κάθετου κυλίνδρου αναμένεται να είναι ήδη φυσικά οριοθετημένο από αυτήν σε προηγούμενο στάδιο, οπότε τα Z_{max}, Z_{min} ταυτίζονται με τα ακρότατα του νέφους κατά z. Δεν ισχύει το ίδιο για τα X_L, Y_L, X_R, Y_R, καθώς δεν ταυτίζονται απαραίτητα με τα X_{min}, Y_{min}, X_{max}, Y_{max}.

Σε έναν κατακόρυφο κύλινδρο, για κάθε σημείο του η συνάρτηση του X_i προς το Y_i είναι αμφιμονοσήμαντη, οπότε ενδιαφέρουν στην προκειμένη περίπτωση το αριστερότερο (X_L, Y_L) και το δεξιότερο σημείο (X_R, Y_R). Το δεξιότερο μπορεί δε, να υπολογιστεί συναρτήσει του αριστερότερου εφ' όσον είναι γνωστό το εύρος τιμών της θ', οπότε για την περαιτέρω ροή του αλγορίθμου απαιτείται μόνο ο εντοπισμός του αριστερότερου σημείου, ήγουν της ελάχιστης γωνίας από την οποία ξεκινά το νέφος.

```
# %% Leftmost point alignment

# "r" for "rotated"
rpcd = copy.deepcopy(ppcd)
print("\nDetecting leftmost point...")
```

Υλοποιείται μια διαδικασία κατά την οποία ο κώδικας θα περιστρέφει το νέφος κατά την αρνητική φορά στροφής γύρω από τον άξονα z ελέγχοντας αν υπάρχουν σημεία του που κείνται στον αρνητικό άξονα x, μέχρι να εντοπίσει κενό.

Το γωνιακό βήμα με το οποίο γίνεται η περιστροφή της αναζήτησης δίνεται ως κλάσμα της περιφέρειας. Εκ προεπιλογής είναι 2π/8 (45°), αλλά ο χρήστης μπορεί εύκολα να αλλάξει την τιμή του παρονομαστή στον κώδικα με έναν άλλο ακέραιο αν επιθυμεί (για ειδικές περιπτώσεις όπως λ.χ. νέφη κυλίνδρων με περιφέρεια άνω του 3π/2 ή με πολλά κενά κάποια από τα οποία θέλει να υπερπηδήσει).

```
# the cloud will turn clockwise around z-axis by this increment
pie_slice = 12
yaw_incr = 2*np.pi / pie_slice

# until a gap is detected : no point of the cloud lies on the negative x-axis
R = ppcd.get_rotation_matrix_from_xyz((0, 0, -yaw_incr))
xmin = rpcd.get_min_bound()[0]
incr_rotations = 0

for i in range(pie_slice):
    rpcd.rotate(R, center=(0, 0, 0))
    xmin = rpcd.get_min_bound()[0]
    if abs(abs(xmin)-r) <= 0.000000001:
        incr_rotations += 1
```

```
            continue
        else:
            z_searching_rot = incr_rotations * (- yaw_incr)
            break

    rpoi = np.asarray(rpcd.points)
```

Αν κατά τη διάρκεια της αναζήτησης δεν εντοπιστεί κάποιο κενό και πραγματοποιηθεί μια πλήρης περιστροφή, τότε το ανάπτυγμα θα καλύπτει όλη την περιφέρεια του κύκλου, θεωρώντας το (-r, 0) ως το αριστερότερο σημείο με αναπτυκτή τετμημένη 0.

```
    if incr_rotations == pie_slice:  # came full circle, did not find a gap
        print("\nLeftmost point not detected.\
              \nDevelopment width will span 360 degrees.")
        tot_rot = 0  # no rotation
```

Αν όμως εντοπιστεί κενό, η περιστροφή σταματά και το σημείο με το ελάχιστο *x* θεωρείται ως το αριστερότερο, υπολογίζεται η γωνιακή συντεταγμένη *θ* του και το νέφος περιστρέφεται αντίστροφα κατά *π - θ* ούτως ώστε το σημείο να βρεθεί επί του αρνητικού άξονα x με συντεταγμένες (-r, 0).

```
    # find polar angle theta and use it to rotate the leftmost point onto -x axis
    else:
        thetamin = np.arccos(xmin / r)  # because now y>0 for the xmin point
        R = ppcd.get_rotation_matrix_from_xyz((0, 0, (np.pi-thetamin)))
        rpcd.rotate(R, center=(0, 0, 0))
```

Καταγράφεται στην γραμμή εντολών η συνολική περιστροφή του νέφους, ο αύξων αριθμός του σημείου και οι συντεταγμένες του πρίν και μετά την περιστροφή.

```
        # total angle by which cloud was rotated
        tot_rot = z_searching_rot + (np.pi - thetamin)
        print(f"Leftmost point detected after turning {tot_rot} rad.")

        rpoi_minindex = np.argmin(rpoi, axis=0)
        print("  Point index: ", rpoi_minindex[0])
        xmin_ppoi = np.asarray(ppcd.points)[rpoi_minindex[0]]
        print("  Before rotation: ", xmin_ppoi)
        xmin_rpoi = np.asarray(rpcd.points)[rpoi_minindex[0]]
        print("  After rotation: ", xmin_rpoi)
```

Οπτικοποιείται επίσης το περιστραμμένο νέφος, με το αριστερότερο σημείο να υποδεικνύεται από μια μωβ ακτίνα, παράλληλη προς τον άξονα x.

```
        # check visually
```

```
    xmin_poi_line = devcyl_vis.create_line(
        xmin_rpoi, (0, 0, xmin_rpoi[2]), (1, 0, 1))
    o3d.visualization.draw_geometries(
        [rpcd, x_axis, y_axis, z_axis, ax_note, xmin_poi_line],
        "Aligned : leftmost point (purple) must lie on the XZ plane", vww, vwh)
```

4.3.3. Μετατροπή καρτεσιανών σε πολικές συντεταγμένες

Ορίζεται η γωνιακή συντεταγμένη θ' που θα χρησιμοποιηθεί και δημιουργείται ένας άδειος πίνακας N×2 όπου υπολογίζονται και καταχωρούνται για κάθε σημείο οι πολικές συντεταγμένες του (r, θ'_i).

```
# %% Conversion to Polar

def nega_theta(x, y):
    # conventional : counter-clockwise from positive x-axis, range (-pi,pi]
    theta = np.arctan2(y, x)
    # its mirror, but better : clockwise from negative x-axis, range [0, 2*pi)
    nega_theta = - theta + np.pi
    return(nega_theta)

# get a table with the polar coords of the cloud points
print("\nCalculating polar coordinates...")
tic = time.perf_counter()

rpoi_polar = np.empty((len(rpoi), 2), float)
for i in range(len(rpoi)):
    rho = r
    theta = nega_theta(rpoi[i][0], rpoi[i][1])
    rpoi_polar[i] = [rho, theta]

toc = time.perf_counter()
print(f"Calculated in {toc - tic:0.4f} sec.\n")
```

Αναγράφονται στην γραμμή εντολών οι πίνακες των καρτεσιανών και πολικών συντεταγμένων (ή η αρχή και το τέλος τους, αν τα σημεία είναι περισσότερα από 100), καθώς και το εύρος τιμών της γωνιακής συντεταγμένης $[\theta'_{min}, \theta'_{max}]$ όπου η θ'_{min} αναμένεται να είναι 0.

```
print("Cartesian coords: ", rpoi)
print("Polar coords: ", rpoi_polar)

thetamin = np.amin(rpoi_polar, axis=0)[1]
thetamax = np.amax(rpoi_polar, axis=0)[1]
print("Minimum theta: ", '{0:.4f}'.format(thetamin))
print("Maximum theta: ", '{0:.4f}'.format(thetamax))
```

4.3.4. Καθορισμός διαστάσεων εικόνας αναπτύγματος

Προκειμένου να υπολογιστεί η ανάλυση (resolution) της τελικής εικόνας ζητείται από τον χρήστη να δώσει το μέγεθος της εδαφοψηφίδας (ground sampling distance) το οποίο εκφράζει την πραγματική απόσταση στο αντικείμενο μεταξύ δύο κέντρων εικονοστοιχείων (pixels) της εικόνας του, και πρέπει να είναι ίσο ή μεγαλύτερο από το μέγεθος της εδαφοψηφίδας στις προσανατολισμένες εικόνες από τις οποίες αντλείται η χρωματική πληροφορία.

```
# %% Pixel Size in Object Space

# ask for GSD (real distance between pixel centers)
gsd = float(input("\nDefine Ground Sampling Distance (m): "))
```

Προσδιορίζονται βάσει των υπολογισμών των προηγούμενων βημάτων οι πραγματικές διαστάσεις του αναπτύγματος, και από αυτές η ανάλυση της εικόνας του Nx×My για την δεδομένη εδαφοψηφίδα.

```
# real dimensions of developed image (in metres)
devh = rpcd.get_max_bound()[2] - rpcd.get_min_bound()[2]
devw = (thetamax - thetamin) * r

# developed image resolution (in pixels)
Nx = round(devw / gsd)
My = round(devh / gsd)
```

Δημιουργείται ένας κενός πίνακας με τις διαστάσεις της εικόνας, κάθε στοιχείο του οποίου θα αποθηκεύσει χρωματική πληροφορία για ένα εικονοστοιχείο σε τρεις φασματικές περιοχές (κόκκινο, πράσινο, μπλε) εύρους πληροφορίας 8 bit (δεκαδικές τιμές από 0 έως 255) έκαστη.

```
# empty table for the image -- M rows, N columns, each cell to hold [R,G,B]
devimg = np.zeros((My, Nx, 3), dtype=np.uint8)
```

Για κάθε εικονοστοιχείο της εικόνας υπολογίζονται οι οι συντεταγμένες στο ανάπτυγμα του κέντρου του και καταχωρούνται σε δεύτερο πίνακα, ο οποίος αναγράφεται έπειτα στην γραμμή εντολών ώστε ο χρήστης να επαληθεύσει την ορθότητά του για τη δεδομένη εδαφοψηφίδα.

```
# developed image pixel coordinates -- M rows, N columns, each cell holds [x,y]
devxy = np.zeros((My, Nx, 2))

print("\nCalculating developed image pixel coordinates...")
tic = time.perf_counter()

# table ij to pixel centre xy
for i in range(My):
    for j in range(Nx):
```

```
            xij = (gsd/2) + j*gsd
            yij = (gsd/2) + My*gsd - (i+1)*gsd
            devxy[i][j] = [xij, yij]

    toc = time.perf_counter()
    print(f"Calculated in {toc - tic:0.4f} sec.")
    print("\nDeveloped image pixel coordinates (x, y):\n", devxy)
```

Υπολογίζεται επικουρικά και η θέση στο χώρο του νέφους που αντιστοιχεί στο κέντρο κάθε pixel και καταχωρείται σε τρίτο πίνακα. Στην παρούσα υλοποίηση δεν είναι απαραίτητος για την παραγωγή του αναπτύγματος αλλά μπορεί να χρησιμοποιηθεί σε αλγορίθμους ελέγχου ή διαδικασίες επαναδειγματοληψίας (resampling).

```
    # object space cart. coordinates -- M rows, N columns, each cell holds [X,Y,Z]
    devXYZ = np.zeros((My, Nx, 3))
    zmin = rpcd.get_min_bound()[2]

    print("\nCalculating corresponding pixel coordinates in object space...")
    tic = time.perf_counter()

    for i in range(My):
        for j in range(Nx):
            xij = devxy[i][j][0]
            yij = devxy[i][j][1]

            thetaij = -(xij/r) + np.pi  # conventional counter-clockwise theta
            Xij = np.cos(thetaij) * r
            Yij = np.sin(thetaij) * r
            Zij = yij + zmin
            devXYZ[i][j] = [Xij, Yij, Zij]

    toc = time.perf_counter()
    print(f"Calculated in {toc - tic:0.4f} sec.")
    print("\nCorresponding pixel coordinates in object space (X, Y, Z):\n", devXYZ)
```

4.3.5. Γραμμικός Μετασχηματισμός νέφους στο XY επίπεδο

Εφαρμόζονται σε αντίγραφο του περιστραμμένου νέφους οι σχέσεις $x_D = \theta'_i r$, $y_D = z_i$, έτσι ώστε κάθε σημείο του να μεταφερθεί από τον τρισδιάστατο χώρο του κυλίνδρου στις αναπτυκτές του συντεταγμένες επί του επιπέδου XY.

```
    # %% Linear Transformation to 2D

    # "d" for "developed"
```

```
dpcd = copy.deepcopy(rpcd)
dpoi = np.asarray(dpcd.points)

print("\nDeveloping by direct linear transformation...")
tic = time.perf_counter()

for i in range(len(rpoi)):
    xd = rpoi_polar[i][1] * r
    yd = rpoi[i][2] - zmin
    dpoi[i] = [xd, yd, 0]

toc = time.perf_counter()
print(f"Unwrapped in {toc - tic:0.4f} sec.")
```

Το ανεπτυγμένο πλέον νέφος οπτικοποιείται μαζί με βοηθητικό σύμβολο στην αρχή των αξόνων ώστε να επιβεβαιωθεί η σωστή θέση του αναπτύγματος στο επίπεδο XY (με το κάτω αριστερά άκρο του επί της αρχής των αξόνων) και να εντοπιστούν τυχόν ατέλειες, ούτως ώστε ο χρήστης να κρίνει αν επιθυμεί να προχωρήσει στην χρωματική παρεμβολή ή να διακόψει το πρόγραμμα και να επαναλάβει από κάποιο προηγούμενο στάδιο προς διόρθωσή τους.

```
ax_origin = o3d.geometry.TriangleMesh.create_coordinate_frame()
o3d.visualization.draw_geometries(
    [dpcd, ax_origin], "Unwrapped on XY plane", vww, vwh)
```

4.3.6. Παρεμβολή χρώματος

Στο τελικό στάδιο της διαδικασίας προκειμένου να ανασυσταθεί το αποτέλεσμα σε αρχείο εικόνας πρέπει να καθοριστεί το χρώμα κάθε εικονοστοιχείου της. Αυτή η ραδιομετρική πληροφορία μπορεί να αντληθεί από τα γειτονικά στις καρτεσιανές συντεταγμένες του κέντρου του σημεία του ανεπτυγμένου νέφους με διάφορες μεθόδους παρεμβολής (interpolation).

Η παρούσα υλοποίηση επαφίεται στην ύπαρξη χρωματικής πληροφορίας εντός του αρχείου νέφους, το οποίο την έλαβε από μια συστοιχία αρχικών ψηφιακών εικόνων. Ωστόσο, με όχημα τον προηγουμένως υπολογισμένο πίνακα συντεταγμένων κάθε εικονοστοιχείου του αναπτύγματος (*devXYZ*), το χρώμα θα μπορούσε δυνητικά να παρεμβληθεί και από τις προσανατολισμένες αρχικές εικόνες με επαναδειγματοληψία όπως στην μέθοδο παραγωγής ορθοφωτογραφίας.

Δύο αρκετά διαδεδομένες και σχετικά χαμηλών υπολογιστικών απαιτήσεων είναι η Παρεμβολή Κοντινότερου Γείτονα (Nearest Neighbour) και η Διγραμμική Παρεμβολή (Bilinear Interpolation). Στην πρώτη αποδίδεται η τιμή του κοντινότερου σημείου, με σχετικά ελάχιστο υπολογιστικό χρόνο, ενώ στη δεύτερη χρησιμοποιούνται οι τιμές των τεσσάρων κοντινότερων σημείων σε έναν διγραμμικό

μετασχηματισμό με συντελεστές βάρους αντιστρόφως ανάλογους με τις αποστάσεις τους από το παρεμβαλλόμενο σημείο.

Για την υλοποίησή τους χρησιμοποιείται η συνάρτηση του πακέτου SciPy scipy.interpolate.griddata.

Δίδεται μέσω βρόχου η δυνατότητα στον χρήστη να αποθηκεύσει μια εικόνα του αναπτύγματος με την μια ή την άλλη μέθοδο ή και με αμφότερες. Οι εικόνες αποθηκεύονται στο working directory σε μη συμπιεσμένη μορφή .png διατηρώντας την αρχική ανάλυση με τίτλο που περιέχει χρονική υπογραφή και τη μέθοδο παρεμβολής που χρησιμοποιήθηκε.

```
# %% Colour Interpolation

dcol = np.asarray(dpcd.colors)
timestamp = time.strftime("%Y%m%d-%H%M%S")

while True:
    nn_or_2l = input("Select method to resample developed image colour by:\n\
                     enter (1) for Nearest Neighbour\n\
                     enter (2) for Bilinear Interpolation\n\
                     or enter (3) to log and exit: ")

    if nn_or_2l == "1":
        print("Drawing by nearest neighbour...")
        tic = time.perf_counter()

        interpol = griddata(dpoi[:, :2], dcol, devxy, method='nearest')

        img = Image.fromarray(np.uint8(interpol*255), 'RGB')
        img.show()

        toc = time.perf_counter()
        print(f"Drawn in {toc - tic:0.4f} sec.\n")

        img.save(f'Dev-NN-{timestamp}.png')
        print(f"Saved as 'Dev-NN-{timestamp}.png'")
        continue

    elif nn_or_2l == "2":
        print("Drawing with bilinear interpolation...")
        tic = time.perf_counter()

        interpol = griddata(dpoi[:, :2], dcol, devxy,
                            method='linear', fill_value=0)

        img = Image.fromarray(np.uint8(interpol*255), 'RGB')
```

```
            img.show()

            toc = time.perf_counter()
            print(f"Drawn in {toc - tic:0.4f} sec.\n")

            img.save(f'Dev-Bilinear-{timestamp}.png')
            print(f"Saved as 'Dev-Bilinear-{timestamp}.png'")
            continue

        elif nn_or_2l == "3":
            break
        else:
            print("Invalid user input.\n")
```

4.3.6. Καταγραφή διαδικασίας

Τέλος, όπως και στο *devcyl_fitsurf.py,* αποθηκεύεται και ένα αρχείο κειμένου που καταγράφει τις παραμέτρους που δόθηκαν ή προέκυψαν κατά τη διάρκεια της παραγωγής αναπτύγματος. Αποθηκεύεται επίσης σε αρχείο .npy προσβάσιμο από άλλα προγράμματα σε Python ο πίνακας των αντίστοιχων συντεταγμένων των εικονοστοιχείων στο χώρο του κυλίνδρου, χάριν περαιτέρω αξιολόγησης ή ελέγχου.

```
# %% Logging

# save devXYZ for any further evaluation
np.save(f"devXYZ-{timestamp}", devXYZ)
print("Saved corresponding pixel coordinates in devXYZ-{timestamp}.npy")

# log unwrapping variables
with open(f"Log-Unwrap-{timestamp}.txt", "w") as log:
    log.write("Source point cloud : " + pcd_path + "\n")
    log.write("Source fitting log file : " + log_path + "\n")
    log.write("Leftmost point alignment rotation angle (rad):\n" + str(tot_rot))

    log.write("\nGround Sampling Distance (m):\n" + str(gsd))
    log.write("\nDeveloped Area Width (m):\n" + str(devw))
    log.write("\nDeveloped Area Height (m):\n" + str(devh))
    log.write("\nDeveloped Image Width (pixels):\n" + str(Nx))
    log.write("\nDeveloped Image Height (pixels):\n" + str(My))

input("Enter any key to exit.")
```

5. Εφαρμογή και αξιολόγηση

Για την δοκιμή των σταδίων και την τελική εφαρμογή του αλγορίθμου χρησιμοποιήθηκαν νέφη σημείων από τρεις κυλινδρικές επιφάνειες, κόγχες του αρχαιολογικού ενδιαφέροντος ναού του Προφήτη Ηλία στο Γεράκι Λακωνίας. Τα δεδομένα προέκυψαν από τις εργασίες υπαίθρου Μεταπτυχιακής εργασίας του ΔΠΜΣ "Προστασία Μνημείων" του ΕΜΠ.

Εξ αυτών η κόγχη του ιερού είναι ελαφρώς πιο ιδιόμορφη και επιρρεπής σε σφάλματα από τις άλλες δύο, οπότε χάριν της εκτίμησης της αντοχής του αλγορίθμου στην επεξεργασία του "χειρότερου" σεναρίου και της αξιολόγησης των αποτελεσμάτων του, θα παρουσιαστεί αναλυτικότερα η ανάπτυξη αυτής και συνοπτικά των άλλων δύο.

Τα στοιχεία που πληκτρολογούνται από τον χρήστη θα επισημαίνονται με μπλε χρώμα.

5.1.1. Προσαρμογή Κυλίνδρου - devcyl_1_fitsurf.py

Η εκτέλεση κάθε ενός από τα κυρίως αρχεία ξεκινά αναγράφοντας στην γραμμή εντολών τον τίτλο, τα πακέτα κώδικα που χρησιμοποιήθηκαν, και το working directory.

```
AN ALGORITHM FOR THE DEVELOPMENT OF CYLINDRICAL SURFACES
  diploma thesis of Odysseus Galanakis
  supervised by Dr. Andreas Georgopoulos
  School of Rural, Surveying and Geoinformatics Engineering
  National Technical University of Athens, 2021

  Utilizing Python 3.8 with NumPy, SciPy, and Open3d 0.13

First Step : Fitting the Cylinder

Open3d Version:  0.13.0
Current working directory: /home/odhynn/github/Develop-Cylindrical-PCloud
Files in the above folder may be simply specified by name.
```

Με την εκκίνηση της εκτέλεσης του devcyl_1_fitsurf.py εισάγεται το όνομα του νέφους (ονομαστικά εν προκειμένω καθώς βρίσκεται στο working directory) και διαβάζεται από το πρόγραμμα.

```
Enter point cloud file path: Kogxi_Ierou_Mesh_Trimmed.ply

Loading 'Kogxi_Ierou_Mesh_Trimmed.ply' point cloud...
Loaded in 1.9254 sec.
Kogxi_Ierou_Mesh_Trimmed.ply : PointCloud with 8375491 points.
```

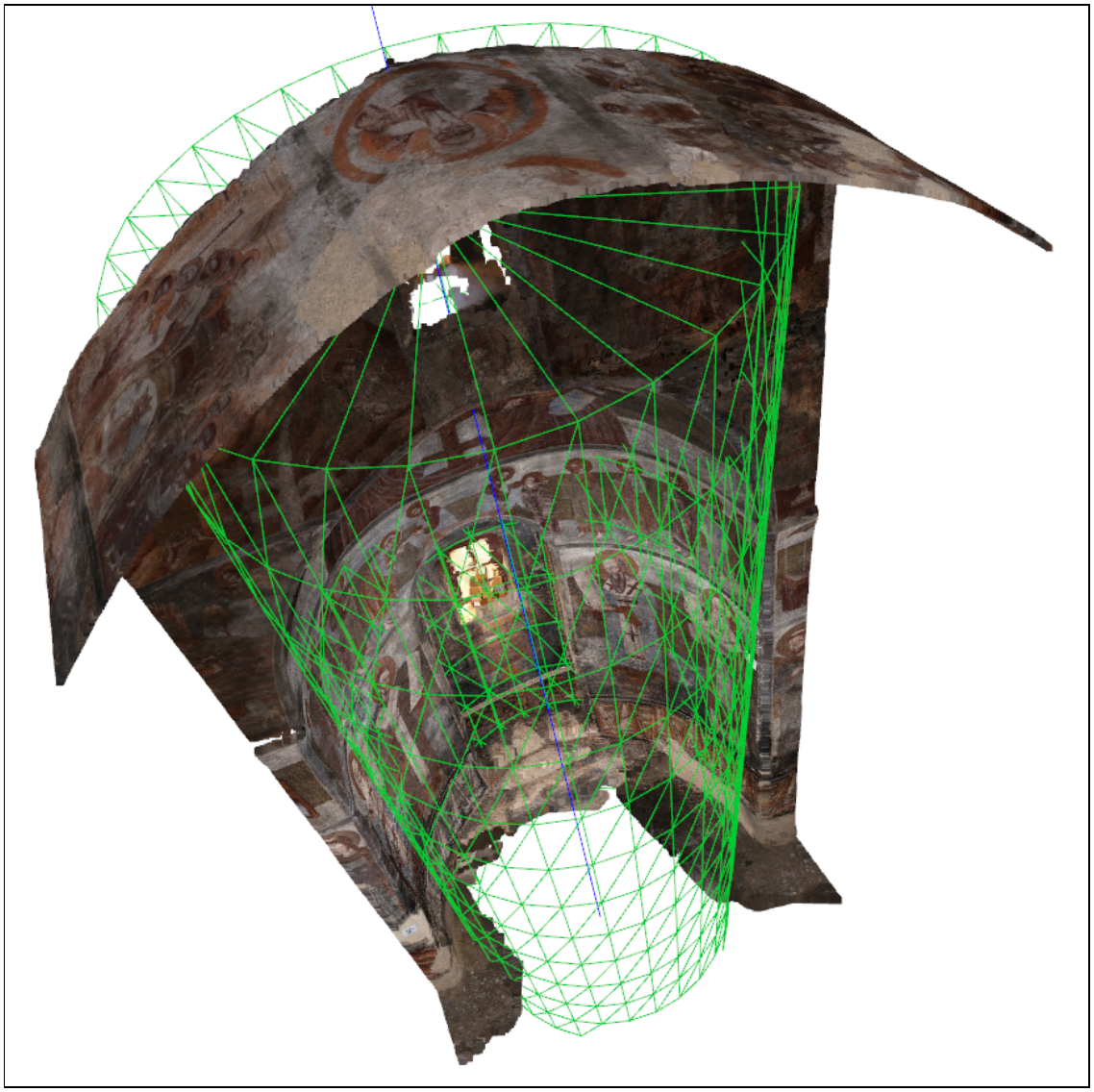
```
[[ 986.1174 1012.7862   99.9228]
 [ 986.9636 1012.8925   99.9215]
 [ 986.9627 1012.892  99.9217]
 ...
 [ 985.3237 1011.743  99.9461]
 [ 985.3239 1011.7431   99.9461]
 [ 985.3239 1011.7432   99.9461]]

Maximum xyz: [ 987.3254 1013.4197  105.3195]
Minimum xyz: [ 984.6466 1010.8058   99.9215]
```

Ζητείται από τον χρήστη να εισάγει τις αρχικές παραμέτρους του κυλίνδρου. Εισάγονται προσεγγιστικά 1000 για τις συντεταγμένες (έχοντας υπ' όψιν το εύρος των συντεταγμένων του νέφους που έχει αναγραφεί στις δυο προηγούμενες γραμμές), μηδέν για τον πίνακα στροφής (κατακόρυφος κύλινδρος), και 1 μέτρο ακτίνα.

```
Define Initial Parameters:
Xk, x coordinate of the cylinder centre (m) = 1000
Yk, y coordinate of the cylinder centre (m) = 1000
omega, rotation angle about the x-axis (rad) = 0
phi, rotation angle about the y-axis (rad) = 0
r, radius of the cylinder (m) = 1

Initial Parameters:  [1000 1000     0      0      1]

Fitting cylinder...
Fitted in 66.0233 sec.
```

Οπτικοποιείται η πρώτη προσαρμογή του κυλίνδρου σε νέο παράθυρο και παρατίθενται οι εκτιμώμενες τιμές των παραμέτρων του.

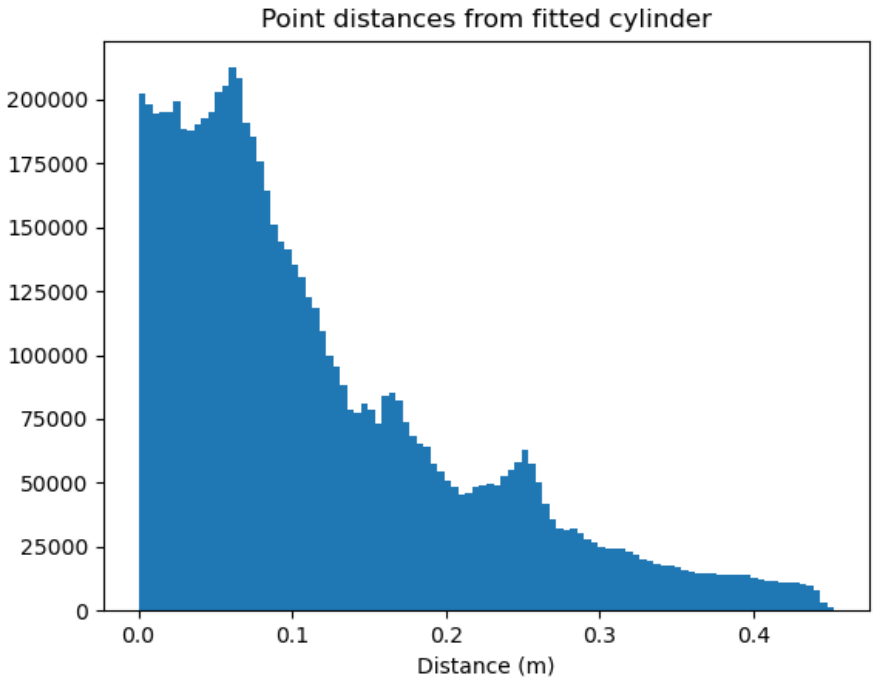
```
Estimated parameters:
xk = 979.3616589340986 m
yk = 1017.8708288172177 m
roll = 0.05378159930294736 rad
pitch = 0.06212967553459921 rad
cylinder radius = 0.9105948341867885 m

Cylinder Rotation Matrix:
 [[ 0.9981  0.0033  0.062 ]
 [ 0.   0.9986 -0.0538]
 [-0.0621  0.0537  0.9966]]
```



Υπολογίζονται οι αποστάσεις των σημείων του νέφους από την επιφάνεια του κυλίνδρου και δημιουργείται ιστόγραμμα της κατανομής τους σε νέο παράθυρο.

```
Calculating point distances from cylinder axis,
(may take several minutes)...
[[0.6196]
 [1.4441]
 [1.4432]
 ...
 [0.7797]
 [0.7795]
 [0.7795]]
Completed in 347.6423 sec
```

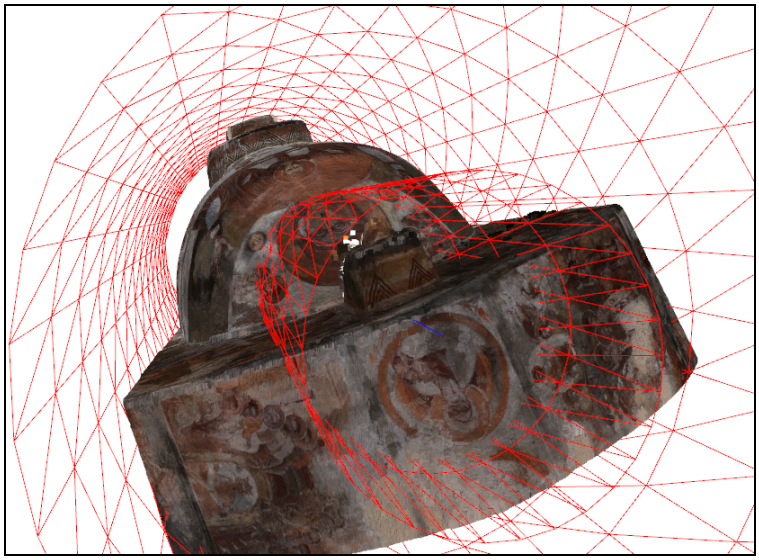
```
Calculating point distances from cylinder radius...
[[0.291 ]
 [0.5335]
 [0.5326]
 ...
 [0.1309]
 [0.1311]
 [0.1311]]

Max. point distance from surface:
 0.9097290816987008 m
```

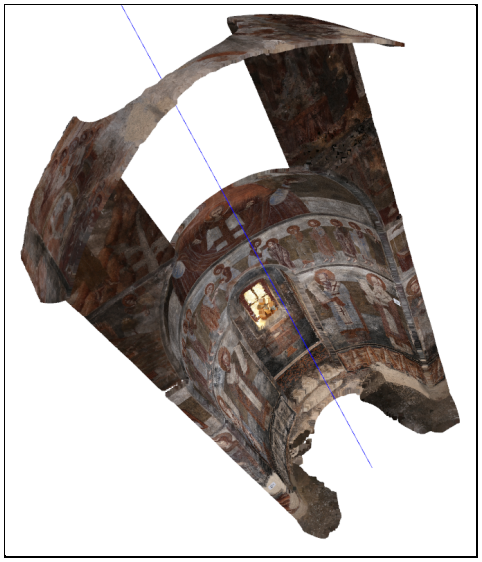


Ενημερωμένος για την κατανομή ο χρήστης αποφασίζει αν θα ορίσει τιμή κατωφλίωσης. Έστω ότι εισάγει 0.4 (εν προκειμένω χονδρική προσέγγιση της τριπλής τυπικής απόκλισης).

```
Apply distance threshold to crop off outliers?
Enter (y) to crop, any other key to continue: y

Selection Threshold
Define max. point distance to cyl. surface (m): .4
```

Δύο διαδοχικά παράθυρα οπτικοποίησης παρουσιάζουν πρώτα τους δυο κυλίνδρους που περικλείουν την περιοχή εντός του κατωφλίου που θα διατηρηθεί, και έπειτα την μορφή του νέφους μετά την κατωφλίωση. Όταν κλείσουν αμφότερα, ερωτάται ο χρήστης αν τον καλύπτει το αποτέλεσμα.



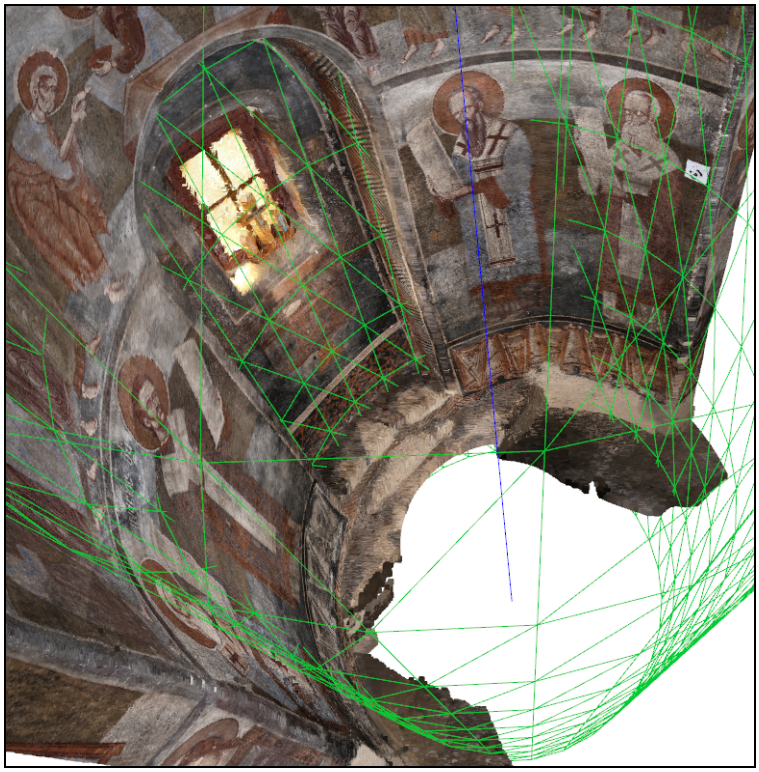


```
Recalibrate cylinder on cropped cloud? (y/n)
Enter (y) to continue, (n) to redefine threshold: y
```

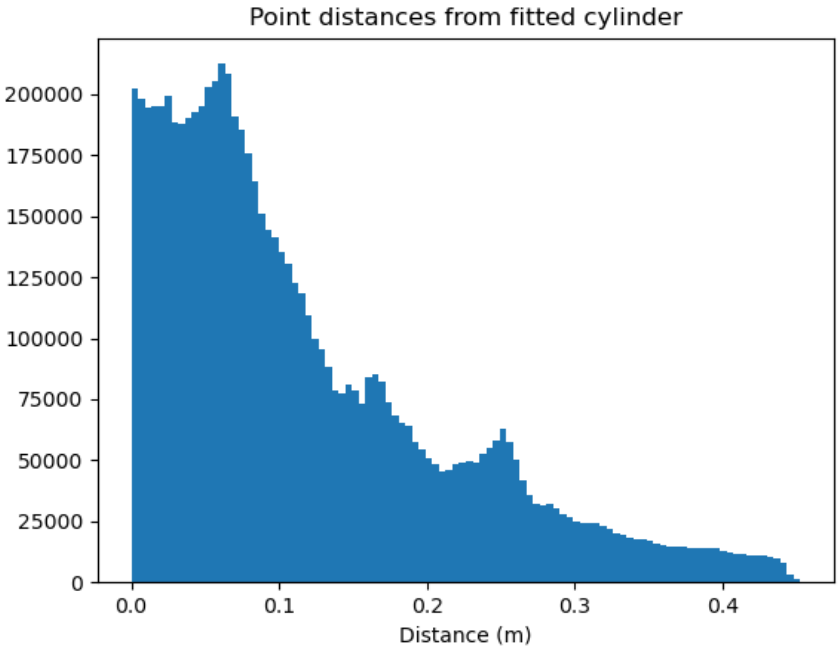
Η θετική απάντηση οδηγεί σε δεύτερη επανάληψη της ανωτέρω διαδικασίας.

```
Repeating cylinder fitting on the new cloud...
[[ 986.1174 1012.7862   99.9228]
 [ 986.1168 1012.7858   99.9236]
 [ 986.0483 1012.7838   99.9238]
 ...
 [ 985.3237 1011.743  99.9461]
 [ 985.3239 1011.7431   99.9461]
 [ 985.3239 1011.7432   99.9461]]

Maximum xyz: [ 987.1793 1013.4197  105.3002]
Minimum xyz: [ 984.6466 1010.9251   99.9228]

Define Initial Parameters:
Xk, x coordinate of the cylinder centre (m) = 1000
Yk, y coordinate of the cylinder centre (m) = 1000
omega, rotation angle about the x-axis (rad) = 0
phi, rotation angle about the y-axis (rad) = 0
r, radius of the cylinder (m) = 1

Initial Parameters:  [1000 1000     0      0      1]

Fitting cylinder...
Fitted in 29.6216 sec.
```

```
Estimated parameters:
xk = 980.1632431697354 m
yk = 1016.9332337470461 m
roll = 0.04483370826509805 rad
pitch = 0.054503547764166986 rad
cylinder radius = 0.9583449437253447 m

Cylinder Rotation Matrix:
 [[ 0.9985  0.0024  0.0544]
 [ 0.   0.999  -0.0448]
 [-0.0545  0.0448  0.9975]]

Calculating point distances
from cylinder axis,
(may take several
minutes)...
[[0.6081]
 [0.6074]
 [0.5509]
 ...
 [0.7581]
 [0.7579]
 [0.7579]]
Completed in 300.1244 sec.

Calculating point distances
from cylinder radius...
[[0.3502]
 [0.351 ]
 [0.4074]
 ...
 [0.2003]
 [0.2004]
 [0.2005]]

Max. point distance from surface:
 0.4527607400431207 m
```

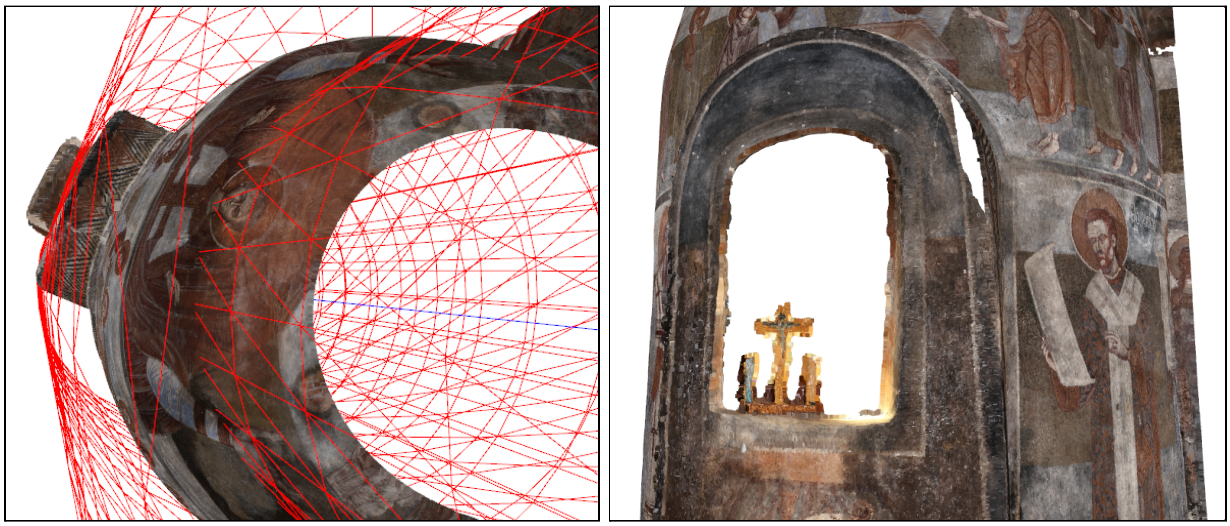


Κατά τη δεύτερη επανάληψη παρατηρείται ότι ο εκτιμώμενος κύλινδρος προσέγγισε καλύτερα την φυσική επιφάνεια, κάτι που αποτυπώνεται στην διαφορά των νέων τιμών των υπολογισμένων παραμέτρων από τις προηγούμενες.



Παρατηρείται ωστόσο ότι η κατανομή των αποστάσεων των σημείων από την επιφάνεια δεν άλλαξε πολύ, οπότε αυτή τη φορά ο χρήστης αποφασίζει να κατωφλιώσει σε απόσταση 0.2.

```
Apply distance threshold to crop off outliers?
Enter (y) to crop, any other key to continue: y

Selection Threshold
Define max. point distance to cyl. surface (m): .2
```



Αντιλαμβάνεται ωστόσο από τα παράθυρα οπτικοποίησης ότι αποκόπτεται μια μικρή ακμή της φυσικής επιφάνειας που επιθυμεί να αποτυπωθεί στο τελικό ανάπτυγμα, οπότε όταν ερωτηθεί αν θέλει να

συνεχίσει με την επαναπροσαρμογή, απαντά αρνητικά για να ορίσει ξανά την τιμή του κατωφλίου 1 cm μεγαλύτερη.

```
Recalibrate cylinder on cropped cloud? (y/n)
Enter (y) to continue, (n) to redefine threshold: n

Selection Threshold
Define max. point distance to cyl. surface (m): .21
```

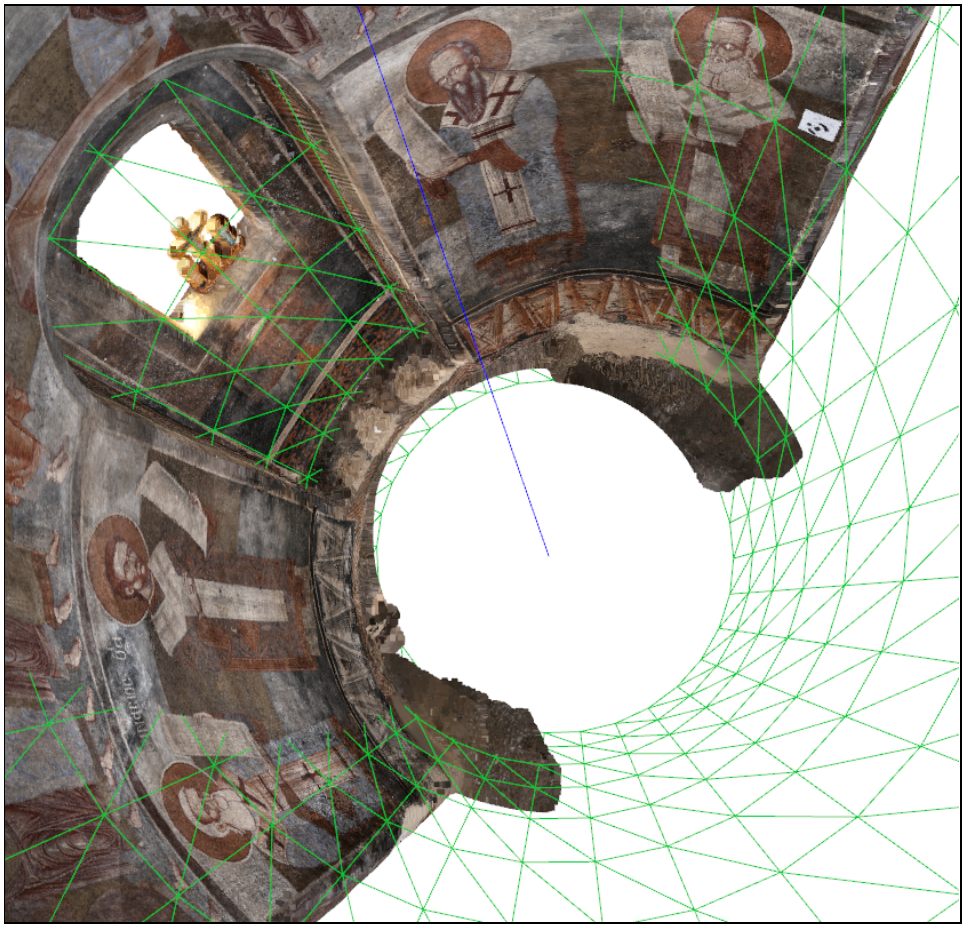
Το αποτέλεσμα όπως οπτικοποιείται σε δύο νέα παράθυρα τον καλύπτει οπότε στην επόμενη ερώτηση απαντά θετικά και συνεχίζει προς την τρίτη επανάληψη προσαρμογής.

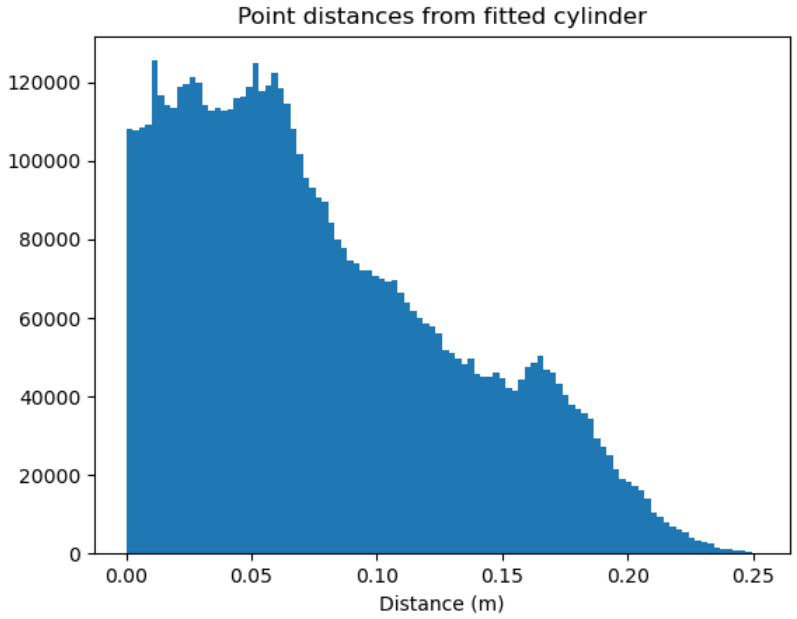
```
Recalibrate cylinder on cropped cloud? (y/n)
Enter (y) to continue, (n) to redefine threshold: y

Repeating cylinder fitting on the new cloud...
[[ 986.3002 1012.7783   99.923 ]
 [ 986.3015 1012.7781   99.923 ]
 [ 986.3003 1012.7773   99.923 ]
 ...
 [ 985.3237 1011.743  99.9461]
 [ 985.3239 1011.7431   99.9461]
 [ 985.3239 1011.7432   99.9461]]

Maximum xyz: [ 987.0566 1013.4197  105.2806]
Minimum xyz: [ 984.6466 1011.0538   99.923 ]
```

Αυτή τη φορά ο χρήστης αποφασίζει να χρησιμοποιήσει ως αρχικές τιμές της νέας προσαρμογής τις εκτιμώμενες τιμές της προηγούμενης, αντιγράφοντας και επικολλώντας από την γραμμή εντολών.

```
Define Initial Parameters:
Xk, x coordinate of the cylinder centre (m) = 980.1632431697354
Yk, y coordinate of the cylinder centre (m) = 1016.9332337470461
omega, rotation angle about the x-axis (rad) = 0.04483370826509805
phi, rotation angle about the y-axis (rad) = 0.054503547764166986
r, radius of the cylinder (m) = 0.9583449437253447

Initial Parameters:  [ 980.1632431697354 1016.9332337470461      0.04483370826509805
0.054503547764166986  0.9583449437253447]

Fitting cylinder...
Fitted in 11.4326 sec.
```

```
Estimated parameters:
xk = 980.099750454089 m
yk = 1017.0870932498698 m
roll = 0.046477608575266255 rad
pitch = 0.05522538968377905 rad
cylinder radius = 0.9767084137530524 m

Cylinder Rotation Matrix:
 [[ 0.9985  0.0026  0.0551]
 [ 0.  0.9989 -0.0465]
 [-0.0552  0.0464  0.9974]]

Calculating point distances from cylinder axis,
(may take several minutes)...
[[0.7595]
 [0.7606]
 [0.7591]
 ...
 [0.7513]
 [0.7511]
 [0.751 ]]
Completed in 255.5869 sec.

Calculating point distances from cylinder radius...
[[0.2172]
 [0.2161]
 [0.2176]
 ...
 [0.2254]
 [0.2256]
 [0.2257]]

Max. point distance from surface:
 0.252080999824694 m
```





Η τρίτη προσαρμογή δεν εφάπτεται τέλεια στην φυσική επιφάνεια, και ο χρήστης μπορεί είτε να συνεχίσει να εκτελεί επαναλήψεις προσεγγίζοντάς την καλύτερα κάθε φορά, είτε να τερματίσει το

πρόγραμμα αποθηκεύοντας το νέο νέφος όπως έχει διαμορφωθεί μετά τις διαδοχικές κατωφλιώσεις μαζί με τις παραμέτρους του προσαρμοσμένου κυλίνδρου του.

Για να επιλέξει το δεύτερο, εισάγει μη θετική απάντηση με ένα τυχαίο πλήκτρο ή συμβολοσειρά. Το πρόγραμμα τον ενημερώνει για την επιτυχή αποθήκευση των αρχείων καθώς και για τα περαιτέρω βήματα της διαδικασίας.

```
Apply distance threshold to crop off outliers?
Enter (y) to crop, any other key to continue: zxcv
Aligned cloud saved as PointCloud-20210630-175248.ply.
Log file saved as Log-Fit-20210630-175248.txt

FURTHER STEPS:
Execute 'devcyl_2_unwrap.py' and provide it with the above saved
.ply file and .txt log to develop it to a 2D image.
In order to only develop a specific area within the point cloud
execute 'devcyl_crop.py' before that and use its product .ply instead.
Press Enter to exit.
```

Το παραγόμενο αρχείο κειμένου έχει το εξής περιεχόμενο:

```
Source point cloud : Kogxi_Ierou_Mesh_Trimmed.ply
Fitted Cylinder Radius (m):
0.9767084137530524

---------------------------
Fitted cylinder parameters:
xk (m):
980.099750454089
yk (m):
1017.0870932498698
roll (rad):
0.046477608575266255
pitch (rad):
0.05522538968377905

---------------------------
Fitting loop iterations:

Iteration 1
xk : 979.362
yk : 1017.871
roll : 0.054
```

```
pitch : 0.062
radius : 0.911

Iteration 2
xk : 980.163
yk : 1016.933
roll : 0.045
pitch : 0.055
radius : 0.958

Iteration 3
xk : 980.100
yk : 1017.087
roll : 0.046
pitch : 0.055
radius : 0.977

-------------------
Thresholds applied:

iteration 1 : 0.4m
iteration 2 : 0.21m
```

5.1.2. Προσδιορισμός Περιοχής Αναπτύγματος - devcyl_crop.py

```
Intermediate Step : Cropping to Desired Area
```

Έστω ότι ο χρήστης επιθυμεί να μην αναπτύξει ολόκληρο το νέο νέφος ως έχει, και το εισάγει σε μια εκτέλεση του devcyl_crop.py.

```
Select point cloud file to load: PointCloud-20210630-175248.ply
Loading 'PointCloud-20210630-175248.ply' point cloud...
Loaded in 1.2716 sec.

PointCloud-20210630-175248.ply : PointCloud with 6162602 points.
```

Για να κόψει το νέφος στην επιθυμητή περιοχή αναπτύγματος μπορεί να χρησιμοποιήσει είτε την μέθοδο του προσδιορισμού ακραίων συντεταγμένων, είτε το παράθυρο διαδραστικής οπτικοποίησης.

5.1.2.1. Προσδιορισμός ακραίων συντεταγμένων

```
To select area by specific bounds, enter (1).
To select area manually, enter (2).
1
```

Ο χρήστης ενημερώνεται για τα άκρα του νέφους ως έχουν, και ορίζει τις συντεταγμένες των κορυφών ενός ορθογωνίου στερεού που θα περικλείει την επιθυμητή περιοχή.

```
Point cloud max. xyz: [  1.15229316   1.06361282 105.60753797]
Point cloud min. xyz: [ -1.06870738  -1.14458857 100.16501787]
xmax = 1
xmin = -1
ymax = 1
ymin = -1
zmax = 105
zmin = 100
```

Το πρώτο στερεό είναι πολύ μεγάλο σε ύψος και περιέχει περιοχές του νέφους που δεν επιθυμεί να αναπτύξει, οπότε ζητά να τροποποιήσει τα όρια.

```
Enter any key to redefine bounding box.
Enter (q) to exit: a;sdkfj

xmax = 1
xmin = -1
ymax = 1
ymin = -1
zmax = 103.4
zmin = 100.6
```

Αν τον καλύπτει το αποτέλεσμα όπως το βλέπει στο παράθυρο οπτικοποίησης, εισάγει “q” για να συνεχίσει και απαντά θετικά στην επόμενη ερώτηση του προγράμματος για να αποθηκευτεί το νέο κομμένο νέφος προς ανάπτυξη (εν προκειμένω ως CroppedPointCloud-20210630-183820.ply).

```
Enter any key to redefine bounding box.
Enter (q) to save (or not) and exit: q
Crop cloud to selected area and save it? (y/n)
Enter (y) to crop and save, (n) to exit: y
Saved as CroppedPointCloud-20210630-183820.ply
```

Αν απαντήσει αρνητικά, το νέφος δεν θα αποθηκευτεί. Σε κάθε περίπτωση το αρχείο τερματίζεται.

5.1.2.2. Παράθυρο διαδραστικής οπτικοποίησης

```
To select area by specific bounds, enter (1).
To select area manually, enter (2).
2


Press H to print help message.
Press [/] to increase/decrease field of view.
Press 'X'/'Y'/'Z' to enter orthogonal view along axis,
   press again to flip.
Press 'K' to lock screen and to switch to selection mode:
   Drag for rectangle selection,
   or use ctrl + left click for polygon selection
Press 'C' to get a selected geometry and to save it.
Press 'F' to switch to freeview mode.
Press 'Q' to close the window.
```

Σε αυτό το παράθυρο οπτικοποίησης του Open3D υπάρχουν κάποιες παραπάνω λειτουργίες που ελέγχονται με συγκεκριμένα πλήκτρα (hotkeys) όσο ο χρήστης βρίσκεται στο παράθυρο. Τα βασικά από αυτά αναγράφονται στην γραμμή εντολών πριν ανοίξει το παράθυρο, και μπορούν να αναγραφούν όλα αναλυτικά αν ο χρήστης πιέσει “H”.

Για την επιλογή της περιοχής του οπτικοποιημένου νέφους που θα αποθηκευτεί ως νέο νέφος, πρώτα στρέφεται η γωνία θέασης στο κατάλληλο σημείο (με δυνατότητα αυξομείωσης του πεδίου όρασης) και “κλειδώνει” εκεί με το hotkey “K”. Κατόπιν επιλέγεται με το ποντίκι η επιθυμητή περιοχή είτε ως ορθογώνιο είτε ως πολύγωνο. Τέλος πιέζοντας “C” ανοίγει νέο παράθυρο αποθήκευσης της επιλεγμένης περιοχής ως νέου νέφους.

Εν προκειμένω για να επιτευχθούν παρόμοια αποτελέσματα με την πρώτη μέθοδο, πιέστηκε “X” δις για μια ορθογραφική προβολή κάθετα στον άξονα x, “K” για να κλειδώσει η γωνία θέασης, μετά επιλέχθηκε ένα ορθογώνιο με mouse drag, και τέλος πιέστηκε “C” για την αποθήκευση της περιοχής.

5.1.3. Παραγωγή Αναπτύγματος - devcyl_2_unwrap.py

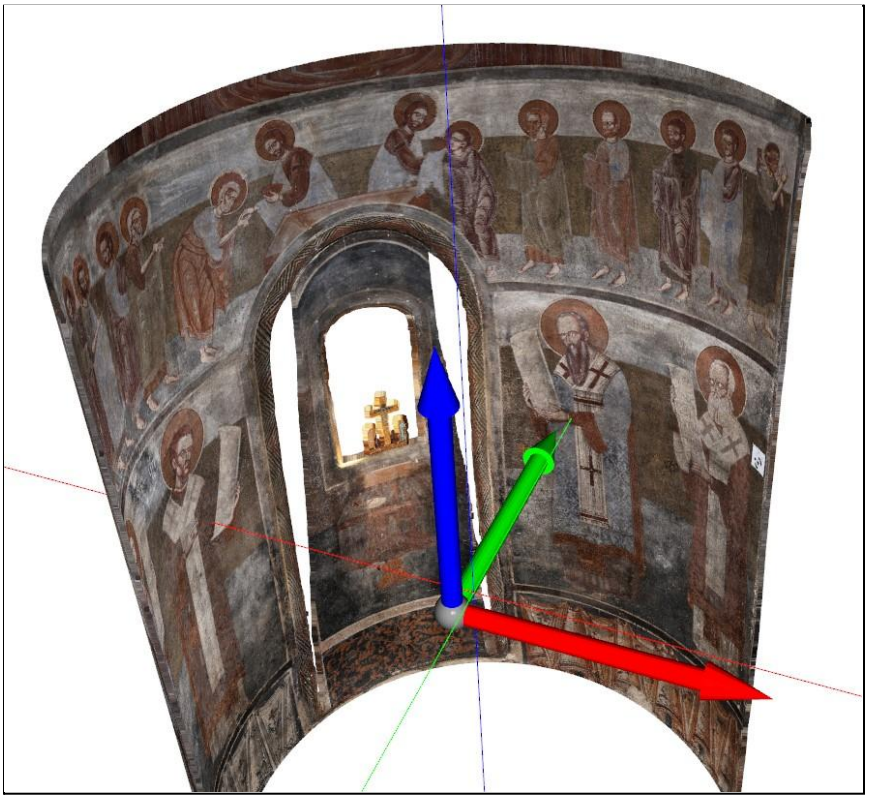
```
Second Step : Unwrapping the Cylinder
```

Για το τελευταίο στάδιο της διαδικασίας εκτελείται το devcyl_2_unwrap.py, με δεδομένα το κατακόρυφο και κομμένο στην περιοχή αναπτύγματος νέφος καθώς και του αρχείου κειμένου που περιέχει τις παραμέτρους του κυλίνδρου, από τις οποίες μόνο η ακτίνα διαβάζεται.

```
Enter point cloud file path: CroppedPointCloud-20210630-183820.ply
Enter fitting log file path: Log-Fit-20210630-175248.txt

Loading 'CroppedPointCloud-20210630-183820.ply' point cloud...
Loaded in 0.7658 sec.

CroppedPointCloud-20210630-183820.ply : PointCloud with 4037591 points.
Fitted Cylinder Radius = 0.9767084137530524
```

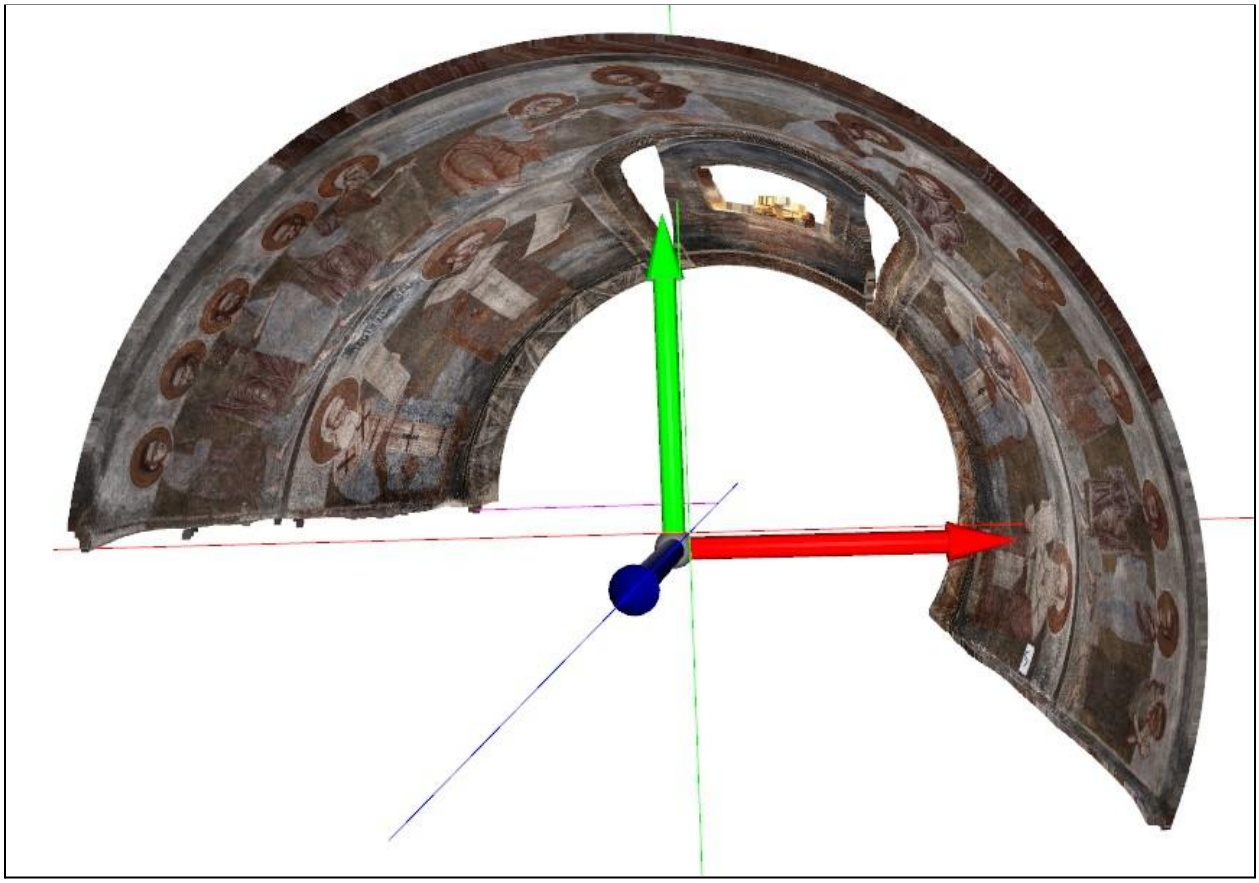
Το νέφος προβάλλεται ακτινικά στην επιφάνεια του κυλίνδρου και οπτικοποιείται.

```
Projecting 'CroppedPointCloud-20210630-183820.ply' point cloud to cylinder surface...
Projected in 50.1995 sec.

Visualizing: Unit vector indicator point at (0, 0, 102.000)
```



Μετά στρέφεται κατά τη φορά των δεικτών του ρολογιού έως ότου ανιχνεύσει κενό στον αρνητικό άξονα x και, μέσω αυτού το αριστερότερο σημείο από όπου θεωρεί ο αλγόριθμος ότι ξεκινά το νέφος. Στο παράθυρο οπτικοποίησης το εν λόγω σημείο υποδεικνύεται από την ακτίνα του με τον άξονα z, χρωματισμένη μωβ, η οποία θα εμπίπτει στο επίπεδο XZ.

```
Detecting leftmost point...
Leftmost point detected after turning -0.2776040913051272 rad.
  Point index:  408391
  Before rotation:  [ -0.4749  -0.8535 100.8793]
  After rotation:  [ -0.9767   0.   100.8793]
```



Κατόπιν υπολογίζονται οι πολικές συντεταγμένες των σημείων του νέφους και μέσω αυτών το εύρος των τιμών της γωνιακής συντεταγμένης.

```
Calculating polar coordinates...
Calculated in 28.8652 sec.

Cartesian coords:  [[  0.8435  -0.4924 100.6002]
 [  0.8435  -0.4925 100.6002]
 [  0.8436  -0.4922 100.6002]
```

```
 ...
 [ -0.9756   0.0474 100.6015]
 [ -0.9756   0.0474 100.6011]
 [ -0.9756   0.0474 100.6005]]

Polar coords:  [[0.9767 3.6699]
 [0.9767 3.6701]
 [0.9767 3.6698]
 ...
 [0.9767 0.0486]
 [0.9767 0.0486]
 [0.9767 0.0485]]
Minimum theta:  0.0000
Maximum theta:  3.6729
```

Ο χρήστης καλείται να ορίσει το μέγεθος της εδαφοψηφίδας, το οποίο έχοντας υπ' όψιν του την ανάλυση των αρχικών εικόνων, θέτει στο μισό εκατοστό του μέτρου. Υπολογίζονται κατευθείαν βάσει αυτού οι αναπτυκτές συντεταγμένες των εικονοστοιχείων.

```
Define Ground Sampling Distance (m): .005

Calculating developed image pixel coordinates...
Calculated in 0.8792 sec.

Developed image pixel coordinates (x, y):
 [[[0.0025 2.7975]
  [0.0075 2.7975]
  [0.0125 2.7975]
  ...
  [3.5725 2.7975]
  [3.5775 2.7975]
  [3.5825 2.7975]]

 [[0.0025 2.7925]
  [0.0075 2.7925]
  [0.0125 2.7925]
  ...
  [3.5725 2.7925]
  [3.5775 2.7925]
  [3.5825 2.7925]]

 ...
```

```
 [[0.0025 0.0075]
  [0.0075 0.0075]
  [0.0125 0.0075]
  ...
  [3.5725 0.0075]
  [3.5775 0.0075]
  [3.5825 0.0075]]

 [[0.0025 0.0025]
  [0.0075 0.0025]
  [0.0125 0.0025]
  ...
  [3.5725 0.0025]
  [3.5775 0.0025]
  [3.5825 0.0025]]]
```

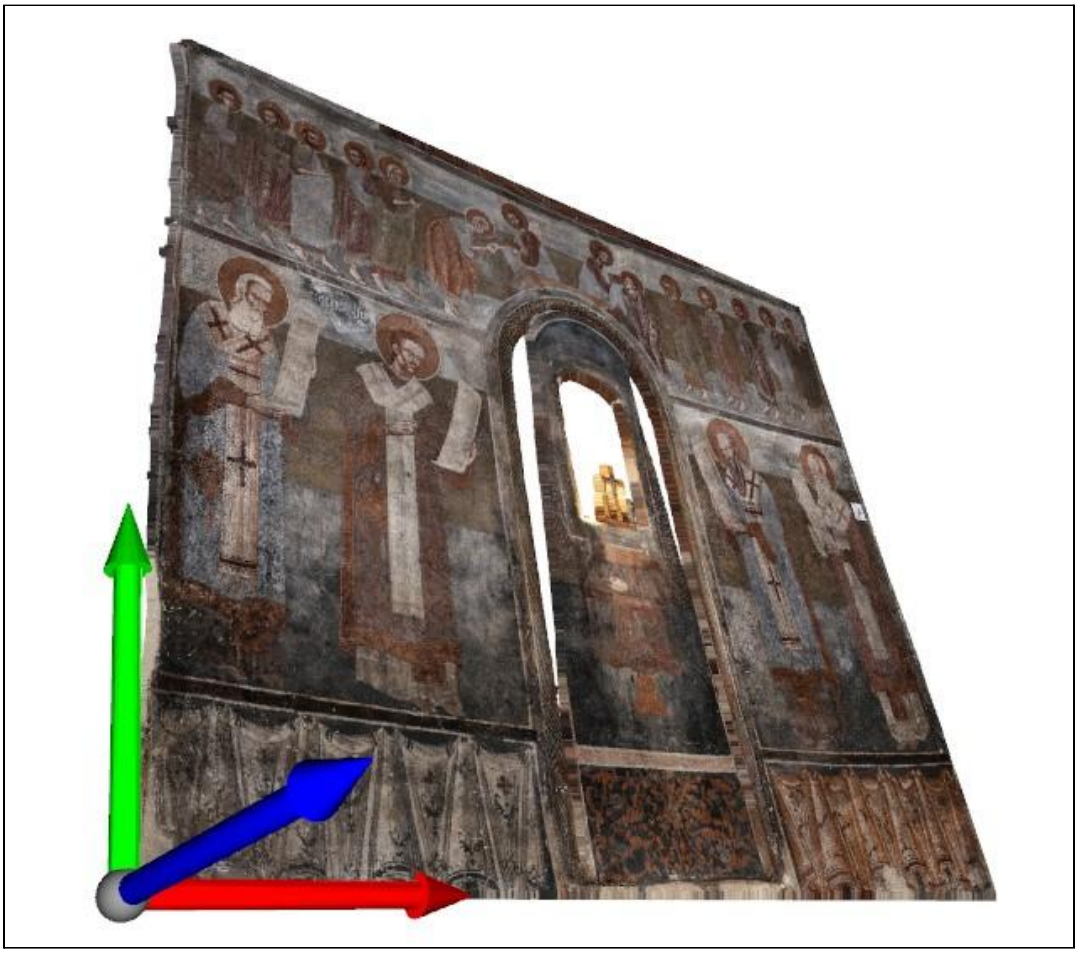
Ακολουθούν οι αντίστοιχες συντεταγμένες κάθε εικονοστοιχείου στον χώρο.

```
Calculating corresponding pixel coordinates in object space...
Calculated in 4.0857 sec.

Corresponding pixel coordinates in object space (X, Y, Z):
 [[[ -0.9767   0.0025 103.3975]
  [ -0.9767   0.0075 103.3975]
  [ -0.9766   0.0125 103.3975]
  ...
  [  0.8495  -0.482  103.3975]
  [  0.847   -0.4863 103.3975]
  [  0.8445  -0.4907 103.3975]]

 [[ -0.9767   0.0025 103.3925]
  [ -0.9767   0.0075 103.3925]
  [ -0.9766   0.0125 103.3925]
  ...
  [  0.8495  -0.482  103.3925]
  [  0.847   -0.4863 103.3925]
  [  0.8445  -0.4907 103.3925]]

 ...

 [[ -0.9767   0.0025 100.6075]
  [ -0.9767   0.0075 100.6075]
```

```
 [ -0.9766   0.0125 100.6075]
 ...
 [  0.8495  -0.482  100.6075]
 [  0.847   -0.4863 100.6075]
 [  0.8445  -0.4907 100.6075]]

[[ -0.9767   0.0025 100.6025]
 [ -0.9767   0.0075 100.6025]
 [ -0.9766   0.0125 100.6025]
 ...
 [  0.8495  -0.482  100.6025]
 [  0.847   -0.4863 100.6025]
 [  0.8445  -0.4907 100.6025]]]
```

Εφαρμόζεται ο άμεσος γραμμικός μετασχηματισμός του προβεβλημένου νέφους στο επίπεδο XY και οπτικοποιείται.

```
Developing by direct linear transformation...
Unwrapped in 11.4494 sec.
```



Τελευταίο εκτελείται το στάδιο της παρεμβολής. Ο χρήστης επιλέγει πρώτα να παράξει το ανάπτυγμα με τη μέθοδο Κοντινότερου Γείτονα.

```
Select method to resample developed image colour by:
                    enter (1) for Nearest Neighbour
                    enter (2) for Bilinear Interpolation
                    or enter (3) to log and exit: 1
Drawing by nearest neighbour...
Drawn in 8.1353 sec.

Saved as 'Dev-NN-20210630-235235.png'
```

Αφότου αποθηκευτεί η νέα εικόνα στο working directory, ο βρόχος επιστρέφει στην επιλογή μεθόδου παρεμβολής. Ο χρήστης επιλέγει τη Διγραμμική Παρεμβολή.

```
Select method to resample developed image colour by:
                    enter (1) for Nearest Neighbour
```

```
                    enter (2) for Bilinear Interpolation
                    or enter (3) to log and exit: 2
Drawing with bilinear interpolation...
Drawn in 64.2101 sec.

Saved as 'Dev-Bilinear-20210630-235235.png'
```

Έχοντας αποθηκεύσει το ανάπτυγμα και από τις δυο εικόνες, εν προκειμένω ανάλυσης 717×560 pixels για εδαφοψηφίδα 0.005m, ο χρήστης επιλέγει να τερματίσει το πρόγραμμα, αποθηκεύοντας το αρχείο κειμένου *Log-Unwrap-20210630-235235.txt* και το αρχείο NumPy πίνακα *devXYZ-20210630-235235.npy* με τις συντεταγμένες των εικονοστοιχείων.

```
Select method to resample developed image colour by:
                    enter (1) for Nearest Neighbour
                    enter (2) for Bilinear Interpolation
                    or enter (3) to log and exit: 3
Saved corresponding pixel coordinates in devXYZ-{timestamp}.npy
Press Enter to exit.
```

Το αρχείο κειμένου περιέχει τα εξής:

```
Source point cloud : CroppedPointCloud-20210630-183820.ply
Source fitting log file : Log-Fit-20210630-175248.txt
Leftmost point alignment rotation angle (rad):
-0.2776040913051272
Ground Sampling Distance (m):
0.005
Developed Area Width (m):
3.5873909256257264
Developed Area Height (m):
2.7999985411976525
Developed Image Width (pixels):
717
Developed Image Height (pixels):
560
```

Το αρχείο πίνακα NumPy δεν είναι αναγνώσιμο παρά μόνο από την βιβλιοθήκη NumPy, οπότε πρέπει να εκτελεστούν οι κατάλληλες εντολές κώδικα για να διαβαστεί (και δυνητικά να αποθηκευτεί και σε μορφές αναγνώσιμες από επεξεργαστές κειμένου, εαν οι διαστάσεις του πίνακα το επιτρέπουν).

Αξιολόγηση

Δεν παρατηρείται κάποια ιδιαίτερη απώλεια ραδιομετρικής πληροφορίας, αν και οι περιοχές στο κέντρο της περιοχής αναπτύγματος που είχαν μείνει κενές από σημεία (λόγω κατωφλίωσης / καθαρισμού του νέφους ή προβολής), έλαβαν τιμές χρώματος από τα εγγύτερά τους σημεία μέσω της παρεμβολής ενώ θα έπρεπε ιδεατά να παραμείνουν κενές και στην εικόνα.

Ως επί το πλείστον η διαδικασία παραγωγής αναπτύγματος φαίνεται να είναι επιτυχής.

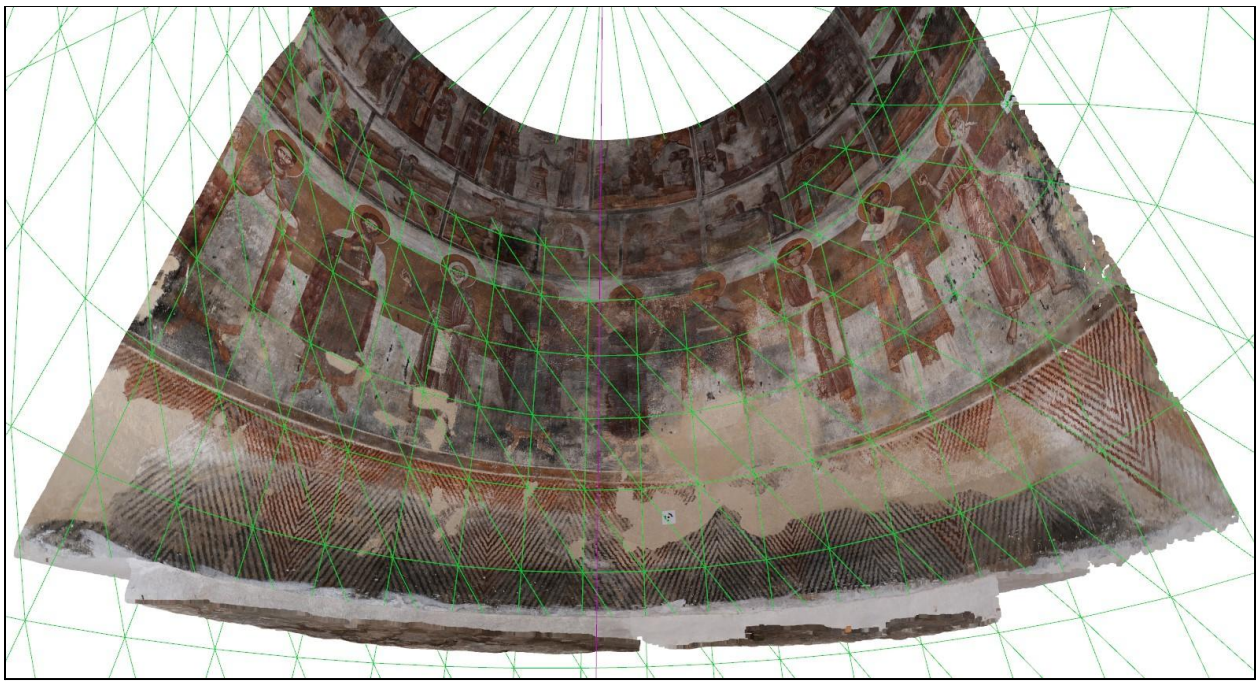
Υπάρχει ωστόσο μια εμφανής παραμόρφωση: το ανάπτυγμα καμπυλώνεται ελαφρά κατά πλάτος, με σταθερή γωνία εκατέρωθεν των πλευρών του. Αυτό θα μπορούσε ενδεχομένως να οφείλεται και στο σχήμα του αρχικού νέφους, σε περίπτωση που είχε περισσότερα σημεία στη βάση του κυλίνδρου από ότι στην κορυφή, αλλά σε αυτήν την περίπτωση θα αναμενόταν οι πλευρές κατά πλάτος του αναπτύγματος να έχουν πιο ακανόνιστα όρια. Αντ' αυτού, εν προκειμένω οι πλευρές είναι ευθείες υπό γωνία, το οποίο συνηγορεί ισχυρά υπέρ ενός άλλου σεναρίου: της ατελούς προσαρμογής.

Εκτιμάται ότι η προσαρμογή του κυλίνδρου τον έστρεψε ώστε να παρουσιάζει μια ελαφρά κλίση προς την αντίθετη κατεύθυνση από την υψηλότερη ζώνη του νέφους, και όταν αυτό προβλήθηκε σε αυτόν οι πλευρές του πήραν αυτήν την κλίση.

5.2. Περαιτέρω εφαρμογές

Προκειμένου να κατανοηθούν καλύτερα οι ατέλειες του αναπτύγματος της πιο ιδιόμορφης κόγχης, και οι τυχόν συστηματικές αδυναμίες του κώδικα που τις προκαλούν, θα παρουσιαστούν πιο συνοπτικά τα αποτελέσματα του αλγορίθμου στις άλλες δύο κόγχες του ναού, αποκαλούμενες “Κόγχη 1” και “Κόγχη 2” στο πρωτογενές υλικό των εργασιών πεδίου.

5.2.1. Κόγχη 2



Στην Κόγχη 2 του Προφήτη Ηλία δοκιμάστηκε ο αλγόριθμος με δυο παραλλαγές ως προς το στάδιο της προσαρμογής.

Στην πρώτη, εφαρμόστηκαν 4 επαναλήψεις του βρόχου προσαρμογής-κατωφλίωσης ως εξής:

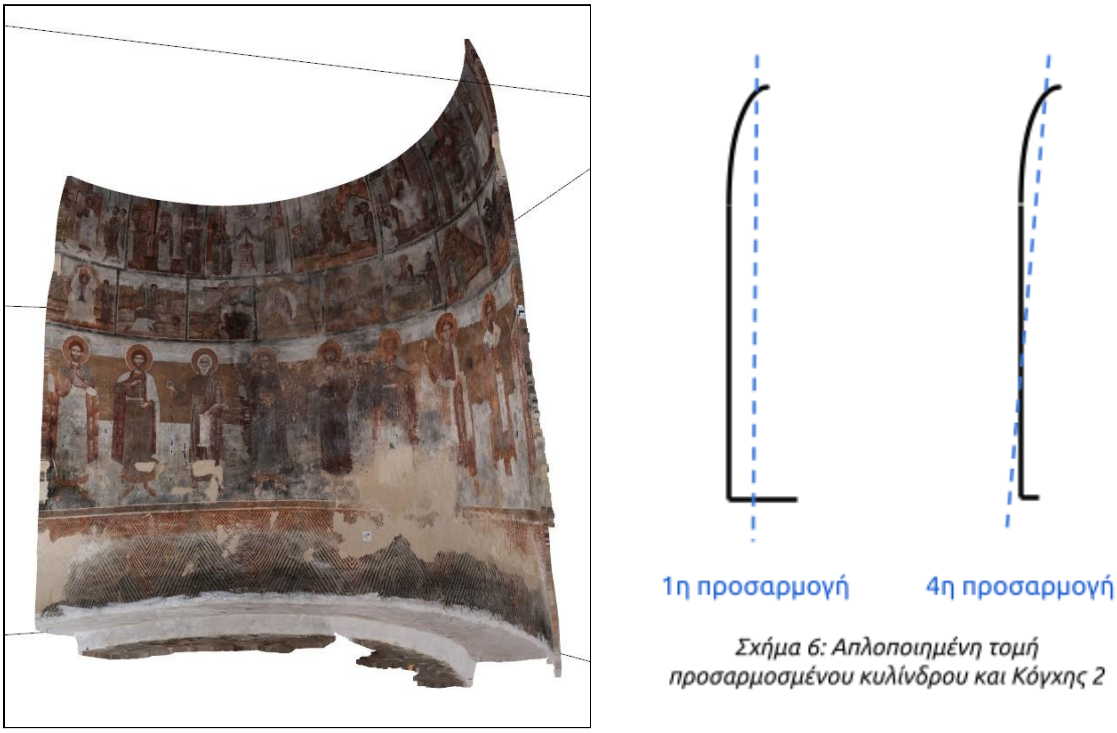
Iteration 1	Iteration 2	Iteration 3	Iteration 4	Thresholds
xk : 986.402 yk : 1010.133 roll : 0.021 pitch : 0.010 radius : 1.811	xk : 983.581 yk : 1005.341 roll : -0.026 pitch : 0.038 radius : 1.892	xk : 983.042 yk : 1004.219 roll : -0.037 pitch : 0.043 radius : 1.904	xk : 982.992 yk : 1004.076 roll : -0.039 pitch : 0.044 radius : 1.910	1 : 0.25m 2 : 0.17m 3 : 0.12m

Δεν προσδιορίστηκε συγκεκριμένη περιοχή αναπτύγματος με την χρήση του *devcyl_crop.py*, αντ’ αυτού εισήχθησαν τα παράγωγα αρχεία κατευθείαν στο *devcyl_2_unwrap.py* το οποίο και κατέληξε στο ακόλουθο ανάπτυγμα.

Η δεύτερη παραλλαγή προχώρησε στα επόμενο στάδια μόνο με την αρχική προσαρμογή του κυλίνδρου χωρίς να εφαρμόσει καθόλου κατωφλίωση, και το παράγωγο ανάπτυγμα είχε την εξής μορφή:

Παρατηρείται ότι στη μέθοδο των πολλαπλών επαναλήψεων καμπυλώνεται αισθητά το ανάπτυγμα, όπως συνέβη με την κόγχη του ιερού.

Αυτό δεν οφείλεται στην μεγαλύτερη περιοχή αναπτύγματος, καθώς το στάδιο του προσδιορισμού της έπεται της προσαρμογής του κυλίνδρου και του μετασχηματισμού του νέφους επί του άξονα z, και η κλίση των πλευρών κατά πλάτος είναι ομοιόμορφη.



Σχήμα 6: Απλοποιημένη τομή προσαρμοσμένου κυλίνδρου και Κόγχης 2

Κατόπιν επισκόπησης της γεωμετρίας της κόγχης, παρατηρείται ότι προς την κορυφή της εμφανίζει τάσεις μείωσης της ακτίνας, τείνει δηλαδή σε κωνική επιφάνεια. Εκτελώντας την μέθοδο ελαχίστων τετραγώνων ο αλγόριθμος εκλαμβάνει αυτή την ιδιομορφία ως κλίση και επιχειρεί να την προσεγγίσει, με αποτέλεσμα κατά την προβολή να υπάρχει παραμόρφωση ανάλογη της ζώνης ύψους κάθε εικονοστοιχείου.

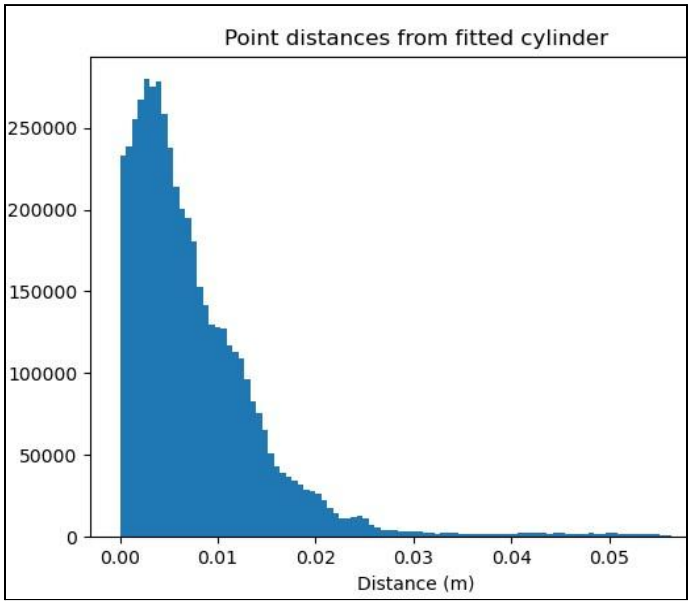
5.2.2. Κογχη 1

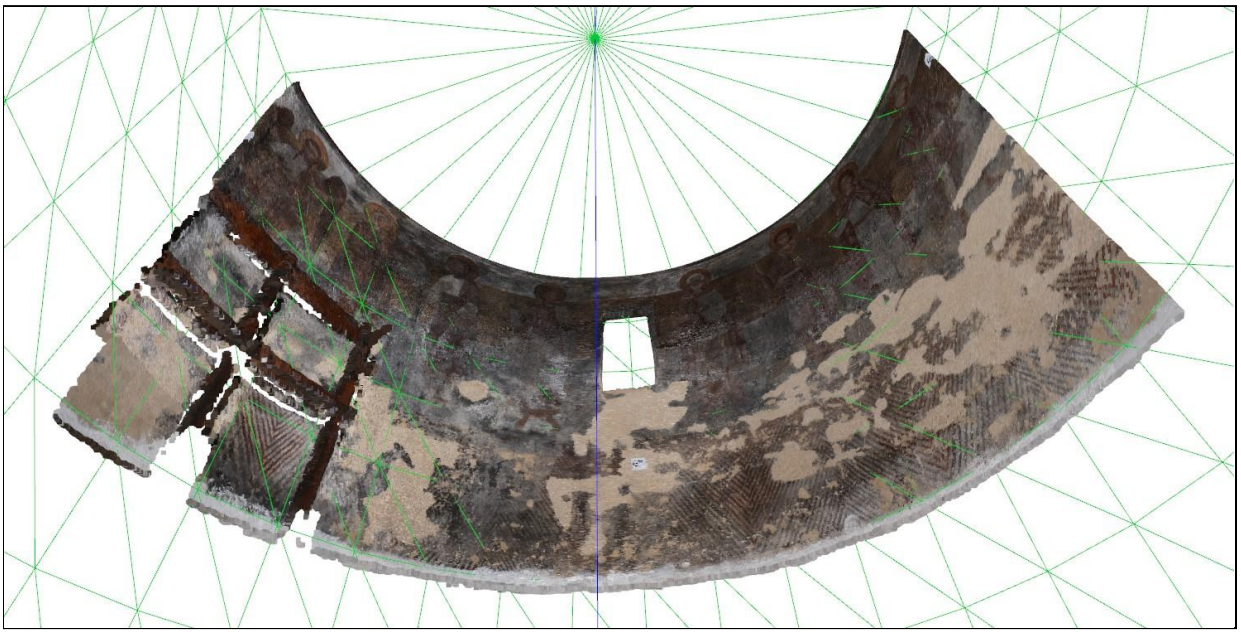
Μια γρήγορη επισκόπηση της Κόγχης 1 μέσω του *devcyl_crop.py* δείχνει ότι η Κόγχη 1 παρουσιάζει την ίδια φυσική ιδιομορφία με τις άλλες δύο, καμπυλώνεται αισθητά στην κορυφή, και αυτό επιβεβαιώνεται από την πρώτη προσαρμογή του κυλίνδρου σε αυτήν.

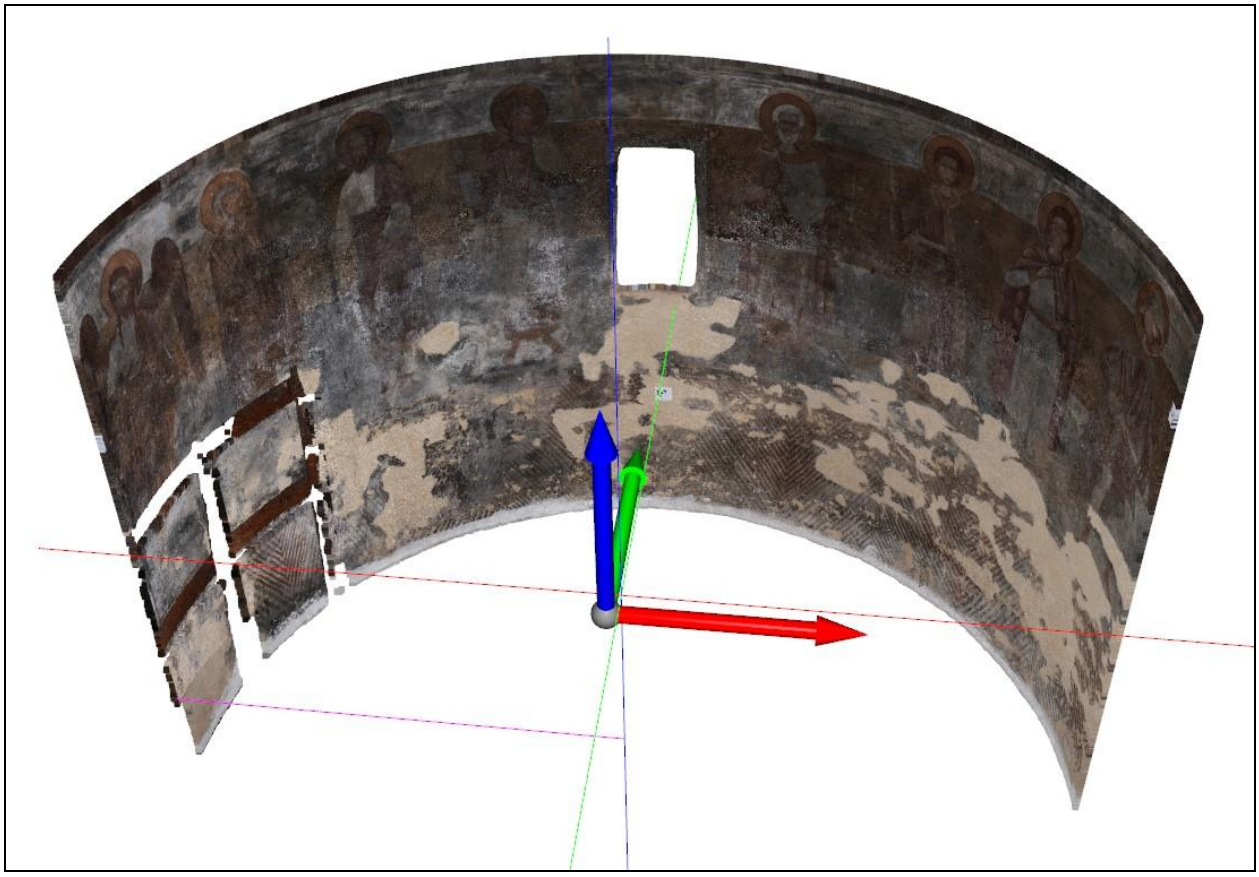
Αυτήν τη φορά το νέφος επεξεργάστηκε από πριν μέσω της 2ης μεθόδου *devcyl_crop.py* ώστε να απαλειφθούν τα περισσότερα σημεία που δεν ανήκουν στην κυλινδρική επιφάνεια, αλλά εφ' όσον διατηρήθηκαν οι υψηλές ζώνες της οι οποίες παρουσιάζουν σταδιακή μείωση της ακτίνας (και άρα προσεγγίζονται καλύτερα από κώνο), ο αλγόριθμος προσπαθεί να τις προσεγγίσει και το ανάπτυγμα παρουσιάζει παρόμοια παραμόρφωση με τα άλλα δύο.

Σε μια δεύτερη προσέγγιση της ίδιας κόγχης επιλέχθηκε προς ανάπτυξη μόνο η κατώτερη ζώνη της Κόγχης 1, η οποία προσεγγίζει φυσικά μια αμιγώς κυλινδρική επιφάνεια και δεν παρουσιάζει τις κωνικές τάσεις της υψηλότερης ζώνης.

Με δυο επαναλήψεις του βρόχου προσαρμογής, το αποτέλεσμα, όπως ήταν αναμενόμενο, παρουσίασε την μικρότερη απόκλιση σημείων και τις μικρότερες παραμορφώσεις από οποιοδήποτε προηγούμενο κατά τις δοκιμαστικές εφαρμογές του κώδικα.







6. Συμπεράσματα

6.1. Συνολική Αποτίμηση

Μετά τις δοκιμαστικές εφαρμογές του κώδικα, εμφανισιακά η μεγαλύτερη αδυναμία του είναι η ατελής προσαρμογή σε επιφάνειες μη αμιγώς κυλινδρικές και η παραμόρφωση που αυτή συνεπάγεται. Ωστόσο, οφείλεται ως επί το πλείστον στην υπερβολικά φιλόδοξη συμπερίληψη κωνικών ζωνών του αντικειμένου σε διαδικασία παραγωγής ενός κυλινδρικού αναπτύγματος, από έναν αλγόριθμο που ασχολείται αποκλειστικά με κυλινδρικές επιφάνειες. Ως εκ τούτου, δεν μπορεί να του καταλογιστεί ιδιαίτερα.

Προηγούμενες προσεγγίσεις όπως στον Τάφο Ατρέα Μυκηνών (Σκόνδρας 2006) και στο Κάστρο της Ρόδου (Γεωργόπουλος et al. 2020) έχουν δείξει η βέλτιστη προσέγγιση στις σύνθετες επιφάνειες περιλαμβάνει καταμερισμό της περιοχής ενδιαφέροντος σε διαφορετικές ζώνες οι οποίες θα προσεγγιστούν επιλεκτικά από διαφορετικά μαθηματικά μοντέλα.

Ως προς το αντικείμενο που πραγματεύεται, η παρούσα υλοποίηση του αλγορίθμου πετυχαίνει τον στόχο της, παραδίδοντας επαρκή αποτελέσματα από την επεξεργασία διαφορετικών δειγμάτων υλικού σε ένα προσβάσιμο πλαίσιο ανοικτού λογισμικού που μπορεί εύκολα να βελτιωθεί στο μέλλον με διορθώσεις, αναδιαρθρώσεις και προσθήκες περαιτέρω κώδικα και χαρακτηριστικών.

6.2. Περαιτέρω Προτεινόμενες Βελτιώσεις

6.2.1. Εφαρμογή μάσκας κατά την παραγωγή αναπτύγματος

Προκειμένου να μην παρεμβάλλεται χρώμα σε εικονοστοιχεία της τελικής εικόνας που δεν αποτυπώνουν σημείο του αντικειμένου, μπορούν να προστεθούν νέα ορίσματα στον κώδικα παρεμβολής σε επίπεδο σχεδιασμού αλγορίθμου (εφ' όσον βέβαια αυτός αναπτύσσεται ανεξάρτητα και δεν χρησιμοποιείται αντ' αυτού η υλοποίηση της υφιστάμενης βιβλιοθήκης SciPy, όπως στο παρόν πρόγραμμα).

Μια αμεσότερη λύση θα ήταν η υλοποίηση μιας μεθόδου που θα εντοπίζει τις κενές περιοχές εντός του αντικειμένου του νέφους, και θα δημιουργεί μια "μάσκα", ήγουν έναν πίνακα ίσων διαστάσεων με τον πίνακα της εικόνας όπου θα αποθηκεύεται δυαδική πληροφορία: κάθε στοιχείο θα περιέχει 0 ή 1 ανάλογα με το αν η αντίστοιχη θέση του στο χώρο του αντικειμένου βρίσκεται ή όχι σε κενή περιοχή.

Για να εφαρμοστεί η μάσκα στην εικόνα, κάθε στοιχείο του πίνακα της εικόνας θα πολλαπλασιάζεται με το αντίστοιχό του στον πίνακα της μάσκα, αφαιρώντας έτσι την ραδιομετρική πληροφορία από τα εικονοστοιχεία που αποτυπώνουν τις εν λόγω κενές περιοχές.

6.2.2. Ποιοτική αξιολόγηση

Προκειμένου να μην επαφίεται ο χρήστης σε μια εμπειρική εκτίμηση της απώλειας ή παραμόρφωσης ραδιομετρικής πληροφορίας κατά τη συνολική διαδικασία του αναπτύγματος, πρέπει να υλοποιηθεί μια μέθοδος που θα την ελέγχει βάσει ανάλυσης των φασματικών τιμών πριν και μετά την διαδικασία του αναπτύγματος.

Μια άμεση προσέγγιση οπτικού ελέγχου θα παρήγαγε δυο ιστογράμματα χρώματος, ένα από τα σημεία του νέφους που συμμετέχουν στην παραγωγή του αναπτύγματος, και ένα από την εικόνα του αναπτύγματος. Αυτό ωστόσο προυποθέτει ότι θα έχει εφαρμοστεί μάσκα σε αυτήν όπως περιγράφηκε ανωτέρω προκειμένου να μην συνυπολογιστούν οι χρωματικές τιμές των παρεμβαλλόμενων κενών εικονοστοιχείων.

Περαιτέρω στατιστική ανάλυση της ραδιομετρικής πληροφορίας είναι εφικτή σε διάφορα μοντέλα ερμηνείας χρώματος, και μπορεί να αναπτυχθεί από υφιστάμενες τεχνικές Τηλεπισκόπησης.

6.2.3 Ποσοτική Αξιολόγηση

Για να τεκμηριωθεί η γεωμετρική αρτιότητα του αναπτύγματος ως προς την πηγή του μπορεί να δημιουργηθεί ένα αρχείο πηγαίου κώδικα το οποίο θα διαβάζει το αρχικό νέφος, την εικόνα του αναπτύγματος, και όλες τις τιμές μεταβλητών των μετασχηματισμών που πραγματοποιήθηκαν κατά τη συνολική διαδικασία.

Με τη χρήση του αποθηκευμένου (στο προηγουμένως αναφερθέν αρχείο *devXYZ-{timestamp}.npy*) πίνακα συντεταγμένων των εικονοστοιχείων στον χώρο του προβεβλημένου κυλινδρικού νέφους, μπορούν να υπολογιστούν οι ακτίνες τους προς τα αντίστοιχα σημεία στον χώρο του αρχικού νέφους, εφαρμόζοντας αντίστροφα τους μετασχηματισμούς.

Αντιστοιχίζεται έτσι το αρχικό νέφος αμφιμονοσήμαντα με την εικόνα του αναπτύγματος, και μπορούν λ.χ. σημεία με γνωστές συντεταγμένες επί του κυλίνδρου όπως τα φωτοσταθερά (ground control points) να ελεγχθούν ως προς την αντίστοιχη θέση τους στο ανάπτυγμα και να εκτιμηθεί το *a posteriori* σφάλμα του.

6.2.4. Εναλλακτική μέθοδος εισαγωγής αρχείων στον αλγόριθμο

Στην παρούσα υλοποίηση κάθε αρχείο εισάγεται στον αλγόριθμο μέσω πληκτρολογίου με το όνομά του, ή και τη διεύθυνσή του αν δεν βρίσκεται στο working directory. Για μεγάλα και πολύπλοκα ονόματα

αρχείων όπως αυτά που παράγονται από τον ίδιο τον αλγόριθμο, αυτή η μέθοδος επιλογής είναι χρονοβόρα και επιρρεπής σε σφάλματα.

Η πιο πρακτική, ταχεία και ασφαλής τεχνική εισαγωγής ονόματος αρχείου με τον κώδικα ως έχει είναι:

1. Alt-Tab - αλλαγή ενεργού παραθύρου στο πρόγραμμα περιήγησης (π.χ. explorer.exe)
2. Left Click - η επιλογή του αρχείου
3. F2 - για να επιλεχθεί η συμβολοσειρά του ονόματος ως μέρος της λειτουργίας μετονομασίας
4. Ctrl-A - επιλογή και της επέκτασης αρχείου
5. Ctrl-C - αντιγραφή
6. Alt-Tab - αλλαγή ενεργού παραθύρου στη γραμμή εντολών
7. Ctrl-V - επικόλληση στη γραμμή εντολών

Προκειμένου να μην επαναλαμβάνεται τόσο συχνά η παραπάνω αλληλουχία για λόγους φιλικότητας προς τον χρήστη, θα μπορούσε να διερευνηθούν δύο εναλλακτικές μέθοδοι επιλογής αρχείου.

- η επιλογή αρχείων μέσω παραθύρου γραφικού περιβάλλοντος (υφίστανται Python βιβλιοθήκες που το υλοποιούν αυτό)
- η υλοποίηση αυτόματης συμπλήρωσης (command line completion) για την γρήγορη εναλλαγή ονομάτων αρχείων εντός γραμμής εντολής με τη χρήση του πλήκτρου *Tab.*

6.2.5. Αναζήτηση αρχής νέφους

Η διαδικασία περιστροφής προς ανίχνευση αριστερότερου σημείου όπως υλοποιήθηκε είναι σχετικά απλή και καλύπτει τις περισσότερες περιπτώσεις αλλά με δυνητικά σφάλματα εσφαλμένης ανάγνωσης όσο μικραίνει το γωνιακό βήμα.

Ένα σφάλμα που παρατηρήθηκε λόγου χάριν ήταν με γωνιακό βήμα 2π/12 στο νέφος της Κόγχης Ιερού όπου η αριστερότερη πλευρά δεν είναι πλήρως συνεκτική, υπήρχαν δηλαδή σημεία με αυξανόμενα κενά μεταξύ τους κοντά στο άκρο. Ανιχνεύοντας το πρόγραμμα ένα από αυτά τα κενά σταμάτησε θεωρώντας ότι όλο το νέφος είχε περιστραφεί πέρα από τον αρνητικό άξονα x, αλλά στην πραγματικότητα είχε παραβλέψει τρια σημεία που δεν τον είχαν διασχίσει ακόμα. Το παραγόμενο ανάπτυγμα θα κάλυπτε ως εκ τούτου πολύ μεγαλύτερο μήκος από το προβλεπόμενο.

Μια πιο συμπαγής αλλά πιθανότατα και πιο αργή προσέγγιση θα μετέτρεπε προηγουμένως σε πολικές συντεταγμένες και θα εξέταζε το εύρος τιμών της θ, επιλέγοντας βάσει της κατανομής τους την ουσιαστικά μικρότερη τιμή ως το σημείο εκκίνησης του νέφους.

Εναλλακτικά, θα μπορούσε να οπτικοποιηθεί το νέφος και να κινεί χειρωνακτικά ο χρήστης την ακτίνα που θα ορίζει το κενό πριν την αρχή του νέφους, ώστε να μετρηθούν οι γωνιακές συντεταγμένες

ξεκινώντας από εκείνο το σημείο. Αυτό πιθανόν να είναι σύντομα εφικτό καθώς αναπτύσσονται περαιτέρω οι δυνατότητες διαδραστικής οπτικοποίησης του Open3D.

6.2.6. Γρήγορη αναζήτηση σημείων κατά την παρεμβολή

Ο υπολογιστικός χρόνος παρεμβολής στις παραπάνω εφαρμογές είναι σχετικά μικρός, αλλά μπορεί να αυξηθεί εκθετικά για μεγαλύτερα datasets (περισσότερα σημεία νέφους) ή μικρότερες εδαφοψηφίδες (μεγαλύτερη ανάλυση, περισσότερα εικονοστοιχεία). Στην περίπτωση της βραδύτερης μεθόδου, ο αλγόριθμος θα προσπαθεί να εντοπίσει τα γειτονικά σημεία στις συντεταγμένες κάθε παρεμβαλλόμενου εικονοστοιχείου ελέγχοντας την απόσταση κάθε σημείου του νέφους από αυτό.

Προκειμένου να βελτιστοποιηθεί η μέθοδος ανίχνευσης των κοντινότερων σημείων, πρέπει η δομή της αναζήτησης να σχεδιαστεί πιο έξυπνα. Μια κοινή μέθοδος για αυτό είναι το k-d trees (Freidman et al., 1977), μια δενδρική δομή δεδομένων, γενίκευση του δυαδικού δέντρου αναζήτησης (binary search tree) και αποδοτικότατη στις μικρότερες διαστάσεις (Muja and Lowe, 2009).

Ο καταμερισμός του νέφους σε περιοχές αναζήτησης βάσει της μεθόδου k-d trees θα μπορούσε να υλοποιηθεί με τη χρήση των σχετικών αλγορίθμων των SciPy και scikit-learn (βιβλιοθήκη python για μηχανική μάθηση). Η Open3D επίσης την υλοποιεί σε σχετικό της module χρησιμοποιώντας την μέθοδο FLANN.

6.3. Github Online Repository

Ο κώδικας έχει ανέβει ως αποθετήριο κώδικα στο Github, ενός δημοφιλούς στον προγραμματιστικό κόσμο εξυπηρετητή που φιλοξενεί διάφορα projects ανοιχτού λογισμικού και παρέχει έλεγχο πρόσβασης και δυνατότητες συνεργασίας και συμμετοχής στην ανάπτυξή τους.

Είναι ιδιαίτερα εύχρηστο για σκοπούς διαχείρισης εκδόσεων (version control), επιτρέποντας τη δοκιμή παραλλαγών του πηγαίου κώδικα (branches) χωρίς να επηρεάζεται η τρέχουσα κατάσταση του.

Ο αλγόριθμος που αναπτύχθηκε ως αντικείμενο της παρούσας διπλωματικής εργασίας έχει πολλά περιθώρια για περαιτέρω βελτίωση και προσθήκη νέων δυνατοτήτων ή εργαλείων όπως αναλύθηκε παραπάνω. Ως αποθετήριο στο github μπορεί να συνεχίσει να επικαιροποιείται και να συντηρείται στο μέλλον, είτε από τον γράφοντα είτε από όποιο άλλο πρόσωπο επιθυμεί να συμβάλλει.

Γίνεται δυνατή επιπλέον η προσθήκη αναλυτικών επεξηγήσεων και τόσο wiki του αποθετηρίου όσο και στο output του αλγορίθμου ώστε να καθίσταται φιλικότερο προς τον χρήστη.

6.4. Απόδοση Συστήματος Υπολογιστή

Το πρόγραμμα αναπτύχθηκε και εκτελέστηκε σε δύο διαφορετικούς υπολογιστές σε περιβάλλον διαχείρισης πακέτων Conda, οι τεχνικές προδιαγραφές των οποίων είναι:

- custom-built Desktop, κατασκευής 2018, Windows 10, 8gb RAM, Nvidia GTX-1050ti GPU (υψηλής απόδοσης), AMD Ryzen 3 1300X 3.5GHz CPU
- Dell Studio XPS Laptop, μοντέλο 2012, Ubuntu 20.04, 16gb RAM, Nvidia GTX-555m GPU (μεσαίας απόδοσης), Intel Core i7-2600 2.20GHz CPU

Οι χρόνοι εκτέλεσης των διαφορετικών διαδικασιών του προγράμματος κινούνται στα ίδια επίπεδα εντός του ίδιου υπολογιστή, αλλά παρατηρήθηκαν υπολογίσιμες αποκλίσεις μεταξύ των υπολογιστών για τις ίδιες λειτουργίες.

Η γενική απόδοση φαίνεται να εξαρτάται κατά κύριο λόγο από την ισχύ του επεξεργαστή, με την μνήμη ταχείας προσπέλασης (RAM) να σηκώνει τον φόρτο της ανάγνωσης και των υπολογισμών συνόρθωσης. Η χρήση της κάρτας γραφικών από το Open3D ήταν αμελητέα για νέφη της τάξης των 9 εκ. σημείων.

Ενδεικτικά για την Κόγχη του Ιερού:

Processing time in seconds	Windows 10 Desktop (high CPU / GPU, low RAM)	Ubuntu 20.04 Laptop (low CPU / GPU, high RAM)
Ανάγνωση νέφους	2.1880	1.9254
Πρώτη προσαρμογή κυλίνδρου	111.5101	66.0233
Πρώτος υπολογισμός απόστασης σημείων από άξονα	180.3412	347.6423
Ακτινική προβολή στον κύλινδρο	31.5010	50.1995
Υπολογισμός πολικών συντεταγμένων	17.5520	28.8652
Γραμμικός Μετασχηματισμός στο επίπεδο XY	8.3096	11.4494
Παρεμβολή Κοντινότερου Γείτονα	7.0315	8.1353
Διγραμμική Παρεμβολή	61.9580	64.2101

Παράρτημα

Οδηγίες εγκατάστασης βιβλιοθηκών

Όλες οι εκδόσεις της γλώσσας προγραμματισμού Python είναι διαθέσιμες προς λήψη και εγκατάσταση από τον ιστότοπό της στη διεύθυνση https://www.python.org/downloads/.

Για την εκτέλεση των αρχείων πηγαίου κώδικα απαιτούνται 4 βιβλιοθήκες πέρα από τις ενσωματωμένες στην Python αυτή καθ' αυτή, που πρέπει να εγκατασταθούν. Οι οδηγίες βρίσκονται στους ιστοτόπους τους:

1. NumPy: https://numpy.org/install/
2. SciPy: https://www.scipy.org/install.html
3. Matplotlib: https://matplotlib.org/stable/users/installing.html
4. Open3D: http://www.open3d.org/docs/release/getting_started.html

Κατά κανόνα υπάρχουν δύο παραλλαγές διαχείρισης της εγκατάστασης.

1. Η μία είναι η απευθείας εγκατάσταση μέσω **pip**, η οποία κατεβάζει το εκάστοτε πακέτο από το Python Package Index (PyPI), το κατ' εξοχήν διαδικτυακό αποθετήριο υλικού για την γλώσσα Python, και το συνδέει άμεσα με την έκδοση Python που θα εντοπίσει στο σύστημα.
2. Η άλλη είναι η εγκατάσταση μέσω **conda**, η οποία καλεί το πακέτο μέσω του καταλόγου του περιβάλλοντος διαχείρισης πακέτων Anaconda, ώστε να μπορεί να χρησιμοποιηθεί ευέλικτα σε διαφορετικά περιβάλλοντα πακέτων ανάλογα με τις απαιτήσεις της εκάστοτε εργασίας.

Οδηγίες εκτέλεσης αρχείων

Προσβάσιμα στο αποθετήριο Github: https://github.com/Odhynn/Develop-Cylindrical-PCloud

Ο τρόπος εκτέλεσης των αρχείων πηγαίου κώδικα διαφέρει ελαφρά αναλόγως του λειτουργικού συστήματος και της εγκατάστασης της Python και των ζητούμενων βιβλιοθηκών.

Στην πιο απλή περίπτωση, πραγματοποιείται εντός του τερματικού, κονσόλας ή στοιχειώδους γραμμής εντολών του λειτουργικού συστήματος (console / command prompt / terminal / [bash] shell).

Πέραν τούτου όμως, δύναται να εκτελεστεί και εντός ενός κατάλληλου IDE καλύτερα προσαρμοσμένου για τους σκοπούς και την ερευνητική μέθοδο του χρήστη, π.χ. Spyder για προσομοίωση λειτουργικότητας Matlab, Jupyter Notebook για σύνταξη διαδραστικής παρουσίασης/διάλεξης, VS Code για επί τόπου καταχώρηση προτάσεων αλλαγής στο αποθετήριο.

Windows

Αν η Python έχει εγκατασταθεί ως ξεχωριστό πρόγραμμα ή app (μέσω του Windows Store) και έχει τεθεί ως προεπιλογή για την εκτέλεση αρχείων .py, και οι απαιτούμενες βιβλιοθήκες έχουν συνδεθεί με αυτήν την έκδοση, τότε πιθανότατα αρκεί ένα διπλό κλικ εντός του windows explorer.

Εναλλακτικά, ο χρήστης μπορεί να ανοίξει μια γραμμή εντολών *Command Prompt* (cmd.exe) ή *Windows Powershell*, και να μεταβεί στο working directory με την εντολή αλλαγής καταλόγου *cd* (change directory).

Έστω ότι η δομή της διεύθυνσης του καταλόγου είναι *…\username\folder1\folder2\wdir* όπου *username* ο βασικός συστημικός φάκελος του χρήστη και *wdir* ο φάκελος που περιέχει τα αρχεία κώδικα.

Εκτελούμενη η γραμμή εντολών καταλήγει στην εξής γραμμή:

```
C:\Users\username>_
```

Ο χρήστης πληκτρολογεί *cd fo* και πιέζει *Tab* για αυτόματη συμπλήρωση (command line completion). Αν ο πρώτος αλφαβητικά φάκελος με όνομα που αρχίζει από "fo" είναι λ.χ. ο *folder0,* τότε η γραμμή εντολών θα συμπληρώσει:

```
C:\Users\username> cd folder0_
```

Πιέζοντας ξανά *Tab*, ανιχνεύεται ο επόμενος αλφαβητικά φάκελος, που τυχαίνει να είναι ο *folder1:*

```
C:\Users\username> cd folder1_
```

Ο χρήστης εισάγει \ (backslash) για να μεταβεί εντός του *folder1* και επαναλαμβάνει:

```
C:\Users\username> cd folder1\folder2\_
```

Έστω ότι ο φάκελος *wdir* είναι δεύτερος στην αλφαβητική σειρά εντός του *folder2*. Ο χρήστης πιέζει *Tab* δύο φορές οπότε η γραμμή εντολών βρίσκει και συμπληρώνει το σωστό όνομα. Πιέζοντας *Enter*, εκτελείται η αλλαγή καταλόγου και πλέον η γραμμή εντολών τρέχει εντός του *wdir*.

```
C:\Users\username\folder1\folder2\wdir>_
```

Για την εκτέλεση ενός αρχείου Python εντός του φακέλου, πληκτρολογείται η λέξη-κλειδί της εγκατάστασης Python (συνήθως *python* ή *python3*) ακολουθούμενη από το όνομα του αρχείου, το οποίο επίσης μπορεί να συμπληρωθεί αυτόματα με τη χρήση του Tab.

```
C:\Users\username\folder1\folder2\wdir> python3 devcyl_1_fitsurf.py
```

Ο χρήστης πιέζει *Enter* για να εκτελέσει το αρχείο.

Conda

Αν δε οι απαραίτητες βιβλιοθήκες έχουν εγκατασταθεί σε περιβάλλον διαχείρισης πακέτων Anaconda, πρέπει να χρησιμοποιηθεί η γραμμή εντολών εντός αυτού του περιβάλλοντος.

Στα Windows εκκινείται η γραμμή εντολών Anaconda Command Prompt (anaconda3):

```
(base) C:\Users\username> _
```

Για να αλλάξει το περιβάλλον πακέτων από το γενικό (base) σε κάποιο ειδικό που επιθυμεί ο χρήστης (π.χ. myenv) χρησιμοποιείται η εντολή conda activate myenv:

```
(base) C:\Users\username> conda activate myenv
(myenv) C:\Users\username> _
```

Ακολουθείται η ίδια διαδικασία που περιγράφηκε παραπάνω.

```
(base) C:\Users\username\folder1\folder2\wdir> python devcyl_1_fitsurf.py
```

Επισημαίνεται ότι πρέπει να είναι γνωστό σε ποιά εγκατάσταση Python είναι εγκατεστημένες οι απαραίτητες βιβλιοθήκες, και σε ποιό περιβάλλον διαχείρισης πακέτων αν αυτό υπάρχει.

Ενδεικτικά, στο σύστημα Windows όπου δοκιμάστηκε η εκτέλεση του κώδικα υπάρχουν δυο εγκατεστημένες εκδόσεις Python, η Python 3.9.5 (από Windows App Store) και η Python 3.8.10 (από Anaconda). Οι βιβλιοθήκες NumPy, SciPy και MatPlotLib είναι προεγκατεστημένες στο περιβάλλον Anaconda, και η Open3D εγκαταστάθηκε επίσης μέσω αυτού.

Επομένως, όταν καλείται η Python 3.9.5 (ως *python3*) η οποία δεν είναι συνδεδεμένη με τις εν λόγω βιβλιοθήκες, δεν τις εντοπίζει κατά τα imports και παρουσιάζει σφάλμα.

```
(base) C:\Users\Captain\github\Develop-Cylindrical-PCloud> python3 devcyl_1_fitsurf.py
Traceback (most recent call last):
       File "C:\Users\Odhynn\github\Develop-Cylindrical-PCloud\devcyl_1_fitsurf.py",
       line 3, in <module> import numpy as np
ModuleNotFoundError: No module named 'numpy'
```

Καθώς επίσης είναι και η ίδια εγκατάσταση που διαχειρίζεται εκ προεπιλογής τα αρχεία .py, αν εκτελεστεί π.χ. το devcyl_1_fitsurf.py μέσω διπλού κλικ, το παράθυρο γραμμής εντολών ανοίγει στιγμιαία και κλείνει αμέσως λόγω του σφάλματος, μη δίνοντας περαιτέρω πληροφορίες ως προς την αιτία.

Ενώ όταν καλείται η Python 3.8.10 (ως *python*) η οποία συνδέεται με τις βιβλιοθήκες ως οργανικό μέρος του ίδιου περιβάλλοντος πακέτων, το πρόγραμμα εκτελείται επιτυχώς.

```
(base) C:\Users\Captain\github\Develop-Cylindrical-PCloud> python devcyl_1_fitsurf.py
```

```
AN ALGORITHM FOR THE DEVELOPMENT OF CYLINDRICAL SURFACES
[ ... ]
```

Unix

Στα λειτουργικά συστήματα ανοικτού λογισμικού βασισμένα στο Unix το πρόγραμμα γραμμής εντολών είναι συχνά γνωστό ως τερματικό (terminal). Χρησιμοποιείται σε γενική βάση από τους χρήστες αυτών των συστημάτων για την εκτέλεση διαφόρων εντολών και λειτουργιών μακράν περισσότερο από το αντίστοιχό του στα Windows command prompt.

Σε περιβάλλον Ubuntu καταλήγει στην εξής γραμμή περιμένοντας το input του χρήστη:

```
username@computername:~$ _
```

Ακολουθείται η ίδια μέθοδος που περιγράφηκε παραπάνω για την αλλαγή καταλόγου.

```
username@computername:~$ cd /home/username/folder1/folder2/wdir
username@computername:~/folder1/folder2/wdir$ _
```

Συνήθως υπάρχει και η δυνατότητα έναρξης του τερματικού εντός του επιθυμητού φακέλου μέσω εντολών στο γραφικό περιβάλλον του. Η αλληλουχία *right click > Open in Terminal* επιτελεί την ίδια λειτουργία με το *cd*.

Το αρχείο πηγαίου κώδικα εκτελείται επίσης με τον ίδιο τρόπο:

```
username@computername:~/folder1/folder2/wdir$ python3 devcyl_1_fitsurf.py
```

Conda

Οι διανομές Anaconda για Unix-based systems δεν διαθέτουν ούτε δικό τους terminal όπως στα Windows ούτε γραφικό περιβάλλον, οπότε η διαχείρησή τους γίνεται με την βοήθεια του τερματικού.

Για την ενεργοποίηση ενός συγκεκριμένου περιβάλλοντος πακέτων, πρέπει να δοθεί πρώτα η εντολή *conda activate* οπότε ακολουθείται η ίδια διαδικασία όπως παραπάνω για να περιέλθει το τερματικό στο περιβάλλον conda και να ζητηθεί η εκτέλεση του αρχείου.

```
username@computername:~/folder1/folder2/wdir$ conda activate base
(base) username@computername:~/folder1/folder2/wdir$ python3 devcyl_1_fitsurf.py
```

Άρθρωμα οπτικοποίησης devcyl_vis.py

Παρατίθεται παρακάτω η σύνταξη του πέμπτου αρχείου κώδικα, ενός αρθρώματος που δεν έχει κάποια ιδιαίτερη υπολογιστική σημασία στην διαδικασία αλλά περιέχει τρεις βασικές λειτουργίες που χρησιμοποιούνται ως δομικό στοιχείο για την κάλυψη των αναγκών της οπτικοποίησης όπως αυτή υλοποιείται από την Open3D 0.13.

```
import copy

import open3d as o3d
```

Η πρώτη συνάρτηση ορίζει μια γραμμή στον χώρο και της προσδίδει χρώμα. Το εύρος [0, 255] κάθε φάσματος του χρωματικού χώρου RGB κανονικοποιούνται στο εύρος [0, 1].

```
def create_line(a, b, colour=[0, 0, 0]):
    """
    Creates a line segment.
    ---------------------
    a, b = points, 3-element array containing x,y,z
    colour = 3-element array containing r,g,b (normalized to [0,1] range )
    """

    # create cylinder axis at work area
    line_points = a, b
    line_lines = [[0, 1]]
    line_colour = [colour for i in range(len(line_lines))]

    line = o3d.geometry.LineSet()
    line.points = o3d.utility.Vector3dVector(line_points)
    line.lines = o3d.utility.Vector2iVector(line_lines)
    line.colors = o3d.utility.Vector3dVector(line_colour)
    return line
```

Η δεύτερη συνάρτηση χρησιμοποιεί την πρώτη για να δημιουργήσει μια κατακόρυφη γραμμή, και την περιστρέφει ως προς το σημείο προβολής της στο επίπεδο XY.

```
def rotate_line(xk, yk, zmin, zmax, rot_matrix, colour=[0, 0, 1]):
    """
    Rotates a vertical line (parallel to z-axis)
    around her projection (a point on the XY plane)
    ------------------------------------
    xk, yk : coordinates of the line's projection on the xy plane
    zmin, zmax : lower, upper coordinates
    rot_matrix : euler rotation matrix (3x3 numpy array)
```

```
        """

        line = create_line([xk, yk, zmin], [xk, yk, zmax], colour)
        line.rotate(rot_matrix, center=[xk, yk, 0])
        return line
```

Η τρίτη και τελευταία συνάρτηση παίρνει τις παραμέτρους ενός προσαρμοσμένου κυλίνδρου με κέντρο στο επίπεδο XY, όπως και το νέφος στο οποίο προσαρμόστηκε, και τον οπτικοποιεί.

Δίνονται δύο επιπλέον προεραιτικά ορίσματα. Το “extra roof” αυξάνει περαιτέρω το ύψος του κυλίνδρου (ώστε να κοπεί από τα όρια του νέφους η σύγκλιση ακτινών που απαρτίζει την επίπεδη πλευρά του, χάριν ευκρίνειας. Το “thresh_mesh” είναι μια λογική (boolean) μεταβλητή που κρίνει αν ο κύλινδρος είναι το αποτέλεσμα μιας προσαρμογής ή η ένδειξη των ορίων της περιοχής κατωφλίωσης.

```
    def cyl_mesh(pcd, cyl_p, extra_roof=0, thresh_mesh=False):
        """
        Visualizes a cylinder mesh around a corresponding cylindrical point cloud
        --------------------------------
        pcd : point cloud (.ply)
        cyl_p : 6-element list, estimated cylinder parametres
            cyl_p[0] = xk, x coordinate of the cylinder centre (m)
            cyl_p[1] = xk, y coordinate of the cylinder centre (m)
            cyl_p[2] = omega, rotation angle about the x-axis (rad)
            cyl_p[3] = phi, rotation angle about the y-axis (rad)
            cyl_p[4] = r, radius of the cylinder (m)
            cyl_p[5] = RyRx, auxiliary rotation matrix (3x3 numpy array)

        extra_roof : by default the cylinder roof is set at the cloud's highest z
                so it may be visually checked that the independently shaped axis
                indeed passes through the roof center where the wires converge
            increase this to set the roof higher so that the cbox crops it off

        thresh_mesh : if <False>, the function returns a standard visualization
                if <True>, the function is used to convey the threshold cutoff,
                    it takes an already modified cyl_p[4] = r +/- threshold
                    and returns a red guide cylinder wireframe
                    beyond which no points will survive the Thresholding
        """

        xyz_max = pcd.get_max_bound()
        xyz_min = pcd.get_min_bound()
```

Ο κύλινδρος αρχικά δημιουργείται στην αρχή των αξόνων και εκτείνεται μέχρι το υψηλότερο σημείο του δεδομένου νέφους επί του θετικού z, και άλλο τόσο συμμετρικά επί του αρνητικού άξονα z.

Η μεταβλητή “resolution” καθορίζει πόσο πυκνό είναι το τριγωνικό πλέγμα που τον οπτικοποιεί.

```
# create cylinder at (0, 0, 0); it extends to both z-directions from there
mesh_cylinder = o3d.geometry.TriangleMesh.create_cylinder(
    radius=cyl_p[4], height=((2*xyz_max[2] + extra_roof)),
    resolution=32, split=round(4*xyz_max[2]))
# ^ adjust resolution if the radial mesh lines are too thick or sparse
mesh_cylinder.paint_uniform_color([0.1, 0.9, 0.1])
```

Κατόπιν μετασχηματίζεται βάσει των παραμέτρων προσαρμογής ώστε να ταυτιστεί με τον προσαρμοσμένο στο νέφος κύλινδρο.

```
# move cylinder to (xk, yk, 0)
mesh_cylinder.translate((cyl_p[0], cyl_p[1], 0), relative=False)

# rotate cylinder
mesh_cylinder.rotate(
    mesh_cylinder.get_rotation_matrix_from_xyz((cyl_p[2], cyl_p[3], 0)))
```

Προκειμένου να μην εμφανίζεται ολόκληρος ο κύλινδρος στο παράθυρο οπτικοποίησης (που για μέγιστο z του νέφους π.χ. 103.63, θα εκτείνεται από το 104 έως το -104), περιορίζεται εντός ενός μέτρου από τα άκρα του νέφους.

```
# a bounding box (slightly larger than the cloud) to crop the cylinder to
cbox = o3d.geometry.AxisAlignedBoundingBox(
    min_bound=xyz_min-[1, 1, 1], max_bound=xyz_max+[1, 1, 1])

# crop the cylinder so that visualization pans directly to work area
mesh_cropped = copy.deepcopy(mesh_cylinder).crop(cbox)

# create wireframe from mesh and visualize (or return thresh mesh)
wire_cylinder = o3d.geometry.LineSet.create_from_triangle_mesh(
    mesh_cropped)
```

Αναλόγως της χρήσης, αν δηλαδή προορίζεται για οπτικοποίηση του προσαρμοσμένου κυλίνδρου ή των άκρων του περιοχής της κατωφλίωσης, χρωματίζεται διαφορετικά.

```
# colour cylinder differently based on whether it will depict
if thresh_mesh == False:  # green for fitted cylinder
    wire_cylinder.paint_uniform_color([0, 0.8, 0.2])
else:  # red, the blood of angry thresholds
    wire_cylinder.paint_uniform_color([1, 0, 0])

return wire_cylinder
```

Βιβλιογραφία

Albert A. A. (1949, 2016), *Solid Analytic Geometry*, Dover, ISBN 978-0-486-81026-3

Baltsavias E.P., 1999. A comparison between photogrammetry and laser scanning, ISPRS Journal of Photogrammetry & Remote Sensing, 54, 83-94

Chan T.O and Lichti D.D., 2012. Cylinder-based self calibration of a panoramic terrestrial laser scanner, International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Volume XXXIX-B5, 2012 XXII. ISPRS Congress, 25 August – 01 September 2012, Melbourne, Australia.

Fischler M., Bolles R., 1981. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography, Communications of the ACM 24 (6), 381-395.

Georgopoulos, A., Skamantzari, M., Tapinaki, S., 2020. Digitally Developing Medieval Fortifications. In Navarro Palazon, Garcia Pulido (eds.) Defensive Architecture of the Mediterranean, Vol X, pp. 317-324., doi: https://dx.doi.org/10.4995/FORTMED2020.2020.11468

Karras et al, 1996. Digital monoplotting and photo-unwrapping of developable surfaces in architectural photogrammetry, International Archives of Photogrammetry & Remote Sensing, 31(5), 290-294

Kraus K. (1982, 2003, 2010) *Photogrammetrie*, *Φωτογραμμετρία*, Τόμος 1: Βασικές Έννοιες και μέθοδοι, Γ' Έκδοση. Αθήνα: Τεχνικό Επιμελητήριο Ελλάδας, ISBN13 978-9-607-01890-8

Marshall, D., Lukacs, G., and Martin, R., 2001. Robust segmentation of primitives from range data in presence of geometric degeneracy. IEEE Trans.PAMI., 23(3), pp. 304–314

Muja M. and Lowe D., 2009. Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration., VISAPP 2009, Proceedings of the Fourth International Conference on Computer Vision Theory and Applications, Lisboa, Portugal, January 2009

Nurunnabi A., Sadahiro Y., Lindenbergh R., 2017. Robust Cylinder Fitting in Three-Dimensional Point Cloud Data. ISPRS, Volume XLII-1, 42(1W1), 63-70 http://dxdoi:10.5194/isprs-archives-XLII-1-W1-63-2017

Rabbani et al., 2007. An integrated approach for modelling and global registration of point clouds, ISPRS Journal of Photogrammetry & Remote Sensing, 61(6): 355 – 370.

Zhou Q.Y., Park J., Koltun V., 2018. Open3D: A Modern Library for 3D Data Processing. arXiv:1801.09847

Αγατζά-Μπαλοδήμου Α.Μ. (2009) *Θεωρία Σφαλμάτων και Συνορθώσεις*, Αθήνα: ΕΜΠ.

Βασιλείου Θ., 2018. Αναπτύγματα Κυλινδρικών Επιφανειών, Διπλωματική Εργασία ΣΑΤΜ ΕΜΠ (επιβλέπων: Α. Γεωργόπουλος). http://dx.doi.org/10.26240/heal.ntua.15436

Γαλαζούλα, Π., 2021. Αναπτύγματα μη αναπτυκτών επιφανειών με χρήση σύμμορφων προβολών. Διπλωματική Εργασία, Εργαστήριο Φωτογραμμετρίας, ΣΑΤΜ, ΕΜΠ

Δολαψάκη Μ.Μ., 2020. Ανάπτυξη αλγορίθμου εντοπισμού ακμών σε νέφος σημείων από ψηφιακές εικόνες, Διπλωματική Εργασία ΣΑΤΜ ΕΜΠ (επιβλέπων: Α. Γεωργόπουλος). http://dx.doi.org/10.26240/heal.ntua.19715

Κουρνιάτη Α.Μ., Κουρνιάτης Ν., (2016). Γεωμετρικές Απεικονίσεις ΙΙ: Γεωμετρικές Απεικονίσεις και Πληροφορική, σημειώσεις διαλέξεων. NTUA Open Academic Courses, Αθήνα.

Μαγκούτης, Κ., Νικολάου, Χ., (2015) Εισαγωγή στον αντικειμενοστραφή προγραμματισμό με Python, Αθήνα: Σύνδεσμος Ελληνικών Ακαδημαϊκών Βιβλιοθηκών. (http://hdl.handle.net/11419/1708)

Μάρας Ι., 2011. Θέματα ελαχίστων τετραγώνων και μέθοδοι επίλυσης, Διπλωματική εργασία ΣΕΜΦΕ ΕΜΠ (επιβλέπων: Κ. Χρυσαφίνος). http://dx.doi.org/10.26240/heal.ntua.7021

Μητροπούλου Κ., 2017. Ανάπτυξη διαδικασίας εντοπισμού επιπέδων και ακμών σε μη οργανωμένα νέφη σημείων, Διπλωματική Εργασία ΣΑΤΜ ΕΜΠ (επιβλέπων: Α. Γεωργόπουλος). http://dx.doi.org/10.26240/heal.ntua.14094

Εγχειρίδια Βιβλιοθηκών Python

- NumPy Documentation (https://numpy.org/doc/stable/user/index.html)
- SciPy Documentation (https://docs.scipy.org/doc/scipy/reference/)
- Open3D Documentation (http://www.open3d.org/docs/release/)