ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΑΓΡΟΝΟΜΩΝ & ΤΟΠΟΓΡΑΦΩΝ ΜΗΧΑΝΙΚΩΝ
Δ.Π.Μ.Σ. ΓΕΩΠΛΗΡΟΦΟΡΙΚΗ

ΣΥΓΚΡΙΣΗ ΠΡΟΣΕΓΓΙΣΕΩΝ ΔΙΑΧΕΙΡΙΣΗΣ ΚΑΙ ΔΙΑΔΙΚΤΥΑΚΗΣ ΟΠΤΙΚΟΠΟΙΗΣΗΣ ΤΗΛΕΠΙΣΚΟΠΙΚΩΝ ΓΕΩΧΩΡΙΚΩΝ ΠΡΟΪΟΝΤΩΝ

Μεταπτυχιακή Εργασία

Ανδρέας Χαλκιαδάκης
Αθήνα, Φεβρουάριος 2016

NATIONAL TECHNICAL UNIVERSITY OF ATHENS
SCHOOL OF RURAL AND SURVEYING ENGINEERING
GEOINFORMATICS, post-graduate programme

HANDLING AND ONLINE VISUALIZATION APPROACHES FOR REMOTE SENSING GEOSPATIAL PRODUCTS

Master's Thesis

Andreas Chalkiadakis
Athens, February 2016

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΑΓΡΟΝΟΜΩΝ & ΤΟΠΟΓΡΑΦΩΝ ΜΗΧΑΝΙΚΩΝ
Δ.Π.Μ.Σ. ΓΕΩΠΛΗΡΟΦΟΡΙΚΗ

Σύγκριση προσεγγίσεων διαχείρισης και διαδικτυακής οπτικοποίησης τηλεπισκοπικών γεωχωρικών προϊόντων

Μεταπτυχιακή Εργασία
Ανδρέας Χαλκιαδάκης

Τριμελής εξεταστική επιτροπή:

Κ. Καράντζαλος	Δ. Αργιαλάς	Β. Καραθανάση
.....	.....	.....
Επ. Καθηγητής Ε.Μ.Π. Επιβλέπων	Καθηγητής Ε.Μ.Π.	Αν. Καθηγήτρια Ε.Μ.Π.

Αθήνα, Φεβρουάριος 2016

Ανδρέας Χαλκιαδάκης
Πτυχιούχος Πληροφορικής και Τηλεπικοινωνιών Ε.Κ.Π.Α.

ΠΕΡΙΛΗΨΗ

Βασική επιδίωξη με την παρούσα εργασία είναι να συγκριθούν διαφορετικές προσεγγίσεις διαχείρισης και διαδικτυακής οπτικοποίησης γεωχωρικών προϊόντων μετά την επεξεργασία τηλεπισκοπικών δεδομένων. Πιο συγκεκριμένα, αρχικά γίνεται η συγχώνευση παγχρωματικών και πολυφασματικών εικόνων με αλγορίθμους στην γλώσσα προγραμματισμού C++. Συγκεκριμένα, με τη χρήση της διαδικασίας High-Pass Filtering (HPF) παράγονται pansharpened ψηφιδωτά (raster) δεδομένα με μεγαλύτερη χωρική ανάλυση σε όλα τα πολυφασματικά κανάλια. Ύστερα, υπολογίζεται ο κανονικοποιημένος δείκτης βλάστησης NDVI (ή νερού NDWI για κάποιες περιπτώσεις) και αφού γίνει ομαδοποίηση των εικονοστοιχείων (pixel) σε 5 κλάσεις με βάση την τιμή των δεικτών, τα ψηφιδωτά (raster) μετατρέπονται σε διανυσματικά(vector) δεδομένα όπου τα γειτονικά εικονοστοιχεία ίδιας κλάσης ομαδοποιούνται σε κοινά πολύγωνα. Σαν δεύτερο μέρος της εργασίας έγινε οπτικοποίηση των παραπάνω διανυσματικών δεδομένων με τη βοήθεια του παροχέα ψηφιακών διανυσματικών οπτικοποιήσεων CartoDB, σε λογαριασμό του οποίου ανέβηκαν τα διανυσματικά δεδομένα. Με τη χρήση html, css, sql, javascript και της παρεχόμενης βιβλιοθήκης CartoDB.js έγινε οπτικοποίησή τους η οποία παρέχει στον χρήστη real-time πρόσβαση στις vector εικόνες ανά περιοχή και μέρα, οι οποίες προβάλλονται με διαφορετικά χρώματα ανά κατηγορία/ομάδα για πιο εύκολη διάκριση.

ABSTRACT

The main objective of this thesis is the comparison of different handling and online visualization approaches of geospatial products after the processing of remote sensing data. Firstly, after the geometric and radiometric corrections, the fusion of the panchromatic and the multispectral bands is taking place. In particular, through the High-Pass Filtering (HPF) algorithm, based on a C++ implementation, the spatial resolution on the multispectral bands is increased. During the next processing step, the Normalized Difference Vegetation Index (or the Normalized Difference Water Index for same cases) is calculated, and after the classification of all pixels into 5 classes the calculated raster images are turned into vector. The vectorization process enforces all neighboring pixels with the same class to be fused into the same polygons. In the second part of this thesis, appropriate manners for the visualization of geospatial products are examined. In particular, the aforementioned vector data through are inserted and stored into the online web mapping service of CartoDB. Based on HTML, JAVASCRIPT, CSS, SQL scripts and given the CartoDB.js API the visualization provides real-time access to the stored on the server-side vector data. Special tools can control the area, the date and other parameters, while data and products can be are projected with different color maps for efficient visual understanding.

ΠΕΡΙΕΧΟΜΕΝΑ

1 Εισαγωγή

Με δεδομένη την ύπαρξη τεράστιου όγκου τηλεπισκοπικών δεδομένων, πριν και μετά την ανάλυσή τους, καθώς και πληθώρας επιχειρησιακών εφαρμογών που τα οδηγούν σε ποικίλες ακολουθίες επεξεργασιών, είναι επιτακτική η ανάγκη να γίνει μια ανάδειξη τέτοιων ακολουθιών, όπως και αξιολόγηση διαφορετικών μορφών αναπαράστασης των προϊόντων τους, όπως τα ψηφιδωτά (raster) και τα διανυσματικά (vector) δεδομένα. Με αυτόν τον τρόπο μπορούν να προταθούν διαφορετικές μορφές γεωχωρικών προϊόντων και υλοποιήσεις ανάλογα με τις ανάγκες του κάθε χρήστη και τις απαιτήσεις καθε εφαρμογής.

Προς την κατεύθυνση αυτή, σε αυτή την εργασία επιδιώκεται μια σύγκριση διαφορετικών μεθόδων διαχείρισης γεωχωρικών προϊόντων μετά από επεξεργασία τηλεπισκοπικών δεδομένων βάση ταχύτητας και όγκου αποτελεσμάτων καθώς και οπτικοποίησή τους σε διαδικτυακό χώρο, από όπου μπορούν να παρουσιαστούν σε προγράμματα-πελάτες (web-client).

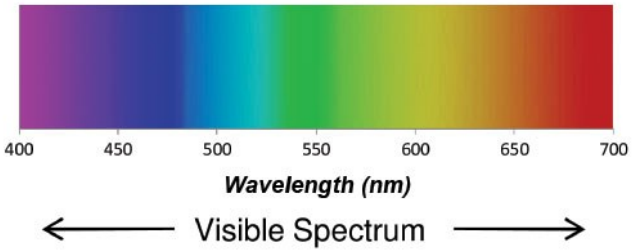
Η επεξεργασία των αρχικών δεδομένων έγινε με τη χρήση της γλώσσας C++, ενώ η οπτικοποίηση με τη βοήθεια της διαδικτυακής βιβλιοθήκης CartoDB.js και συγκεκριμένα με τις γλώσσες HTML, CSS, SQL και Javascript.

Ακολουθεί μια εισαγωγή στις βασικές έννοιες δορυφορικών δεδομένων, στη συνέχεια παρουσιάζεται η μεθοδολογία που ακολουθήθηκε για την υλοποίηση των τηλεπισκοπικών τεχνικών. Τέλος, παρουσιάζονται και σχολιάζονται τρόποι οπτικοποίησης γεωχωρικών προϊόντων και εξετάζονται αποτελέσματα από συγκρίσεις διαφορετικών προσεγγίσεων.

1.1 Δορυφορικά Δεδομένα

Παρακάτω παρουσιάζονται οι βασικές κατηγορίες δορυφορικών δεδομένων που χρησιμοποιήθηκαν για τις ανάγκες της εργασίας.

Παγχρωματικές εικόνες: ασπρόμαυρες εικόνες που είναι ευαίσθητες στο φάσμα του ορατού φωτός.

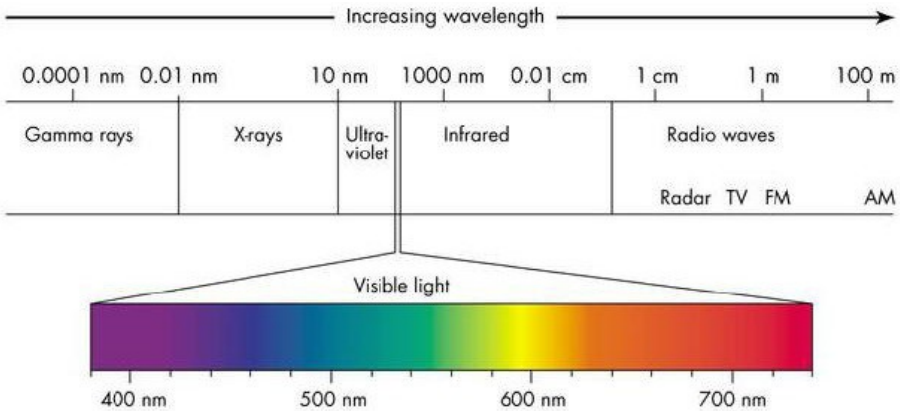


Εικόνα 1: Φάσμα Ορατού Φωτός

Στην αναλογική φωτογραφία, οι παγχρωματικές ήταν πολύ πιο ακριβές από τις ορθοχρωματικές και άργησαν να χρησιμοποιηθούν ευρέως. Χαρακτηριστικό παράδειγμα της ελλιπής αποτύπωσης στις ορθοχρωματικές που δεν μπορούσαν να απορροφήσουν όλο το ορατό φάσμα ήταν η εμφάνιση του έντονου κόκκινου(πχ. βαφές χειλιών) ως μαύρο και του ανοιχτού γαλάζιου (πχ. χρώμα ματιών) ως άσπρο. Από το 1926 και μετά υπήρξε μεγάλη διάδοση του παγχρωματικού φιλμ ακόμα και στις κινηματογραφήσεις.

Σήμερα, χρησιμοποιείται ψηφιακή παγχρωματική φωτογραφία από δορυφόρους (πχ. Quickbird και IKONOS) για να απεικονίσουν όσο πιο αναλυτικά γίνεται την επιφάνεια της Γης.

Πολυφασματικές εικόνες: συνδυάζοντας φωτογραφία με φασματοσκοπία, η συγκεκριμένη τεχνολογία επιτρέπει την απεικόνιση εκτός από του φάσματος ορατού φωτός (400-700nm) και του υπέρυθρου (700-12000nm).



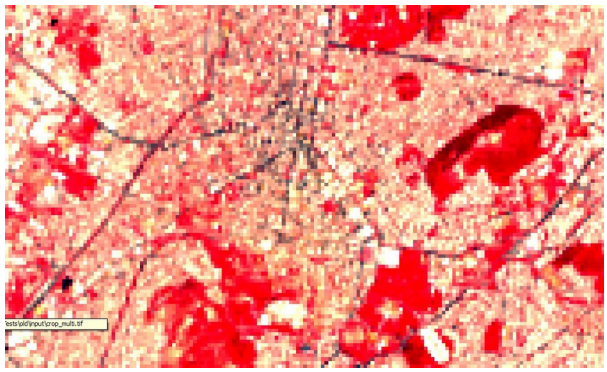
Εικόνα 2: Ηλεκτρομαγνητικό Φάσμα

Οι δορυφόροι για να παράγουν πολυφασματικές εικόνες χρησιμοποιούν από τρία και πάνω ραδιόμετρα/σένσορες. Το κάθε ένα δημιουργεί απεικονίσεις βασισμένο σε συγκεκριμένα μήκη κύματος. Πχ. Ο δορυφόρος Landsat 8 παράγει 7 πολυφασματικές εικόνες με τη χρήση του πολυ-αισθητήρα OLI (Operational Land Imager)[2].

Σε σύγκριση με τις πολυφασματικές αποτυπώσεις του ίδιου δορυφόρου, οι παγχρωματικές έχουν σχεδόν πάντα πολύ μεγαλύτερη ανάλυση. Για παράδειγμα, στον Quickbird, οι παγχρωματικές έχουν μέγεθος pixel $65cm^2$ ενώ οι πολυφασματικές $2,62m^2$[1].

Στις εικόνες 3 και 4 μπορεί να διαπιστωθεί η διαφορά στις αναλύσεις μεταξύ παγχρωματικής και πολυφασματικής της ίδιας περιοχής από τον ίδιο δορυφόρο.

Εικόνα 3: Παγχρωματική Απεικόνιση Κέντρου Αθήνας (Landsat 8)



Εικόνα 4: Πολυφασματική (NIR-R-G) Απεικόνιση Κέντρου Αθήνας (Landsat 8)

Pansharpening: Συνδυασμός παγχρωματικής και πολυφασματικής εικόνας της ίδιας περιοχής για να δημιουργηθεί μια εικόνα με ίδιο αριθμό ζωνών με την πολυφασματική αλλά με την ανάλυση της παγχρωματικής.

Παραδείγματα:

Εικόνα 6: (από QuickBird)

Εικόνα 5: (από IKONOS)

Στη συγκεκριμένη εργασία για το pan-sharpening χρησιμοποιήθηκε η μέθοδος συγχώνευσης HPF (High Pass Filtering), η οποία πρωτοεμφανίστηκε το 1980 από τον Showengerdt.

Κατά την HPF, αφού εφαρμοστεί φίλτρο υψηλών συχνοτήτων στην υψηλής ανάλυσης παγχρωματική εικόνα εξάγεται η πληροφορία μιας περιοχής και προστίθεται στην αντίστοιχη πολυφασματική (σε όλες τις ζώνες που αυτή διαθέτει) με διεργασία σε επίπεδο pixel. Το τελικό αποτέλεσμα έχει την πληροφορία υψηλή ανάλυσης της παγχρωματικής αλλά και τις κατά προσέγγιση φασματικές τιμές των ζωνών της πολυφασματικής.

Ως φίλτρο υψηλών συχνοτήτων αναφέρεται η αποκοπή τιμών της παγχρωματικής εικόνας οι οποίες βρίσκονται κάτω από τον τοπικό μέσο όρο[3].

Για να πραγματοποιηθεί αυτή η μέθοδος σωστά προϋποθέτει πως η παγχρωματική και η πολυφασματική έχουν ακριβώς τον ίδιο αριθμό pixel. Γιαυτό το λόγο πρέπει πρώτα η πολυφασματική η οποία είναι χαμηλότερης ανάλυσης (L) από τη παγχρωματική (H) να έρθει στον ίδιο ακριβώς αριθμό pixel. Αυτό υλοποιείται μέσω αναδόμησής της (resampling).

Επίσης, για να εφαρμοστεί το φίλτρο υψηλών συχνοτήτων πρέπει να βρεθούν οι τοπικοί μέσοι όροι για κάθε pixel εντός δεδομένου παραθύρου.

Τελικά, για να βρεθεί το ζητούμενο αποτέλεσμα (HPF) γίνεται για κάθε pixel (θέση i,j) η πρόσθεση της αναδομημένης πολυφασματικής (Lw) με τη διαφορά (φίλτρο) της αρχικής παγχρωματικής (H) με τη παγχρωματική τοπικών μέσων (Hm)[4].

$$HPF_{i,j} = Lw_{i,j} + \left(H_{i,j} - Hm_{i,j}\right)$$

1.2 Σχετικές εργασίες στη βιβλιογραφία

- Fusion of Multispectral and Panchromatic Images by Local Mean and Variance Matching Filtering Techniques, de Bethune/Muller/Donnay, Universite de Liege, Ιανουάριος 1998
- Διερεύνηση και αξιολόγηση μεθόδων συγχώνευσης πολυφασματικών και παγχρωματικών δορυφορικών δεδομένων, Αριστείδης Δ. Βαϊόπουλος, ΣΑΤΜ – ΕΜΠ, 2013

Παρουσίαση και χρήση αντίστοιχα μεθόδων συγχώνευσης πολυφασματικών και παγχρωματικών εικόνων. Συγκεκριμένα παρουσιάζονται οι HPF, LMM, LMVM και HCS. Από αυτές επιλέχθηκε η HPF για τις ανάγκες της παρούσας εργασίας.

- Ανάλυση Μεγάλων Γεωχωρικών Δεδομένων με Τεχνολογίες Πινάκων για Αγροτικές Εφαρμογές, Αθανάσιος Κ. Κάρμας, ΣΗΜΜΥ – ΕΜΠ, Ιούλιος 2014
- Online Analysis of Remote Sensing Data for Agricultural Applications, Athanasios Karmas/Konstantinos Karatzalos/Spiros Athanasiou, Ιούλιος 2014

Υπολογισμός NDVI και ομαδοποίηση τιμών εικονοστοιχείων σε κλάσεις, οι οποίες χρησιμοποιήθηκαν και για την παρούσα εργασία.

- Εκτίμηση και χαρτογράφηση ποιοτικών χαρακτηριστικών σε υδάτινους αποδέκτες με τεχνικές τηλεπισκόπησης: Η Περίπτωση της Λίμνης Κάρλα, Θεολόγου Ιωάννα, ΣΑΤΜ – ΕΜΠ, Μάιος 2014
- Εκτίμηση και χαρτογράφηση ποιοτικών χαρακτηριστικών σε υδάτινους αποδέκτες από διαχρονικά τηλεπισκοπικά δεδομένα υψηλής χωρικής ανάλυσης, Πατελάκη Μαριάτζελα, ΣΑΤΜ - ΕΜΠ, Ιούλιος 2015

Ανάλυση ποιοτικών χαρακτηριστικών της λίμνης Κάρλα, η οποία χρησιμοποιήθηκε και στην παρούσα εργασία για ανάδειξη του δείκτη NDWI, με εξέταση πολυφασματικών εικόνων. Επίσης παρουσίαση τρόπου εύρεσης και ομαδοποίησης της χλωροφύλλης (chl-a), ο οποίος χρησιμοποιήθηκε και εδώ.

- Διάδοση της Ακτινοβολίας μέσα από την Ατμόσφαιρα. Εφαρμογή & Αξιολόγηση Απόλυτων Ατμοσφαιρικών Αλγόριθμων Διόρθωσης Τηλεπισκοπικών Απεικονίσεων, Παναγιώτης Ι. Σισμανίδης, ΣΑΤΜ-ΕΜΠ, Οκτώβριος 2012

Παρουσίαση και αξιολόγηση αλγορίθμων ατμοσφαιρικών διορθώσεων. Η παρούσα εργασία εκτός από αρχικές (raw) δορυφορικές, χρησιμοποιεί και εικόνες-αποτελέσματα τέτοιων διορθώσεων σαν πηγαίες.

2 Μεθοδολογία και πειραματικό πλαίσιο αξιολόγησης

2.1 Προεπεξεργασίες

Πριν την εφαρμογή της μεθόδου συγχώνευσης HPF-Fusion, χρειάστηκε να γίνει η ακόλουθη προετοιμασία.

Τα δορυφορικά δεδομένα από τον Landsat 8, δόθηκαν σε πακέτα των 11 ζωνών (παράδειγμα ονομασίας πακέτου: '*landsat_L8_184_032_LC81840322013255LGN00*'), οι οποίες καταλάμβαναν ξεχωριστά αρχεία .tif:

B1 – Coastal/Aerosol
B2 - Blue
B3 - Green
B4 - Red
B5 - Near Infrared
B6 - Short Wavelength Infrared
B7 - Short Wavelength Infrared
B8 - Panchromatic
B9 - Cirrus
B10 - Long Wavelength Infrared
B11 - Long Wavelength Infrared

Οι πολυφασματικές εικόνες αποτελούνται μόνο από τις 7 πρώτες, οπότε χρειάστηκε να συμπτυχθούν σε μια, κάτι που πραγματοποιήθηκε με τη βοήθεια του προγράμματος QGIS, συγκεκριμένα με το εργαλείο *Raster->Miscellaneous->Merge.* Επιλέχθηκαν ως “input” οι 7 πρώτες ζώνες με τη σειρά, δόθηκε το επιθυμητό όνομα στο αποτέλεσμα (output) και ενεργοποιήθηκε η επιλογή “Layer Stack” ώστε να είναι πολυζωνικό το αποτέλεσμα.

Η όγδοη ζώνη χρησιμοποιήθηκε ως η αντίστοιχη παγχρωματική.

Για την υλοποίηση της HPF-Fusion, της εύρεσης, κατωφλίωσης και ομαδοποίησης των δεικτών NDVI/NDWI/CHLORO, της μετατροπής των αποτελεσμάτων σε διανυσματικές εικόνες και της σύμπτυξής τους, έγινε χρήση της **βιβλιοθήκης GDAL/OGR** στη γλώσσα προγραμματισμού **C++**.

Η εγκατάσταση της GDAL/OGR πραγματοποιήθηκε ακολουθώντας τα παρακάτω βήματα:

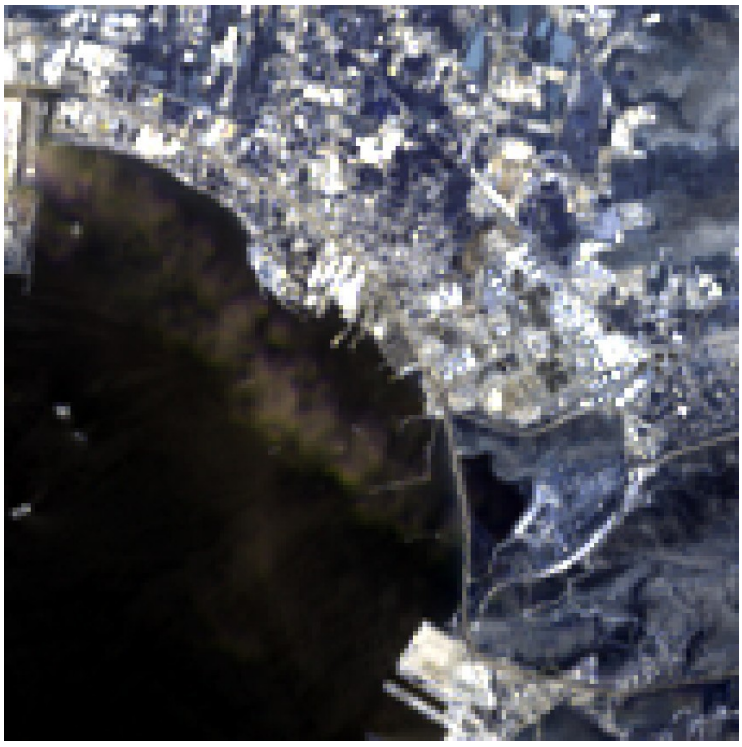
- Console:
  sudo apt-get install binutils libproj-dev gdal-bin;
  sudo apt-get update;
  sudo apt-get install build-essential;
  cd USERHOME;
  pico .profile;
- στο τέλος του αρχείου πρέπει να προστεθεί η εντολή
  export LD_LIBRARY_PATH="/usr/local/lib”

Τέλος, πρέπει να σημειωθεί πως κατά τη μετατροπή κώδικα, που χρειάζεται βιβλιοθήκες GDAL/OGR, σε εκτελέσιμο χρειάζεται μετά τα ονόματα των πηγαίων αρχείων να ακολουθεί και η παράμετρος '*-lgdal*'.

2.2 Pansharpening με τη Μέθοδο HPF-Fusion

Στα πλαίσια δοκιμών και παρουσίασης του pan-sharpening για την παρούσα εργασία χρησιμοποιήθηκαν οι παρακάτω εικόνες (λιμάνι του Πειραιά) από Landsat 8 :

Εικόνα 7: Παγχρωματική Απεικόνιση Λιμανιού Πειραιά

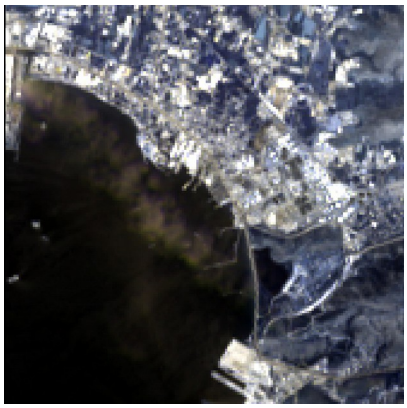


Εικόνα 8: Πολυφασματική Απεικόνιση Λιμανιού Πειραιά

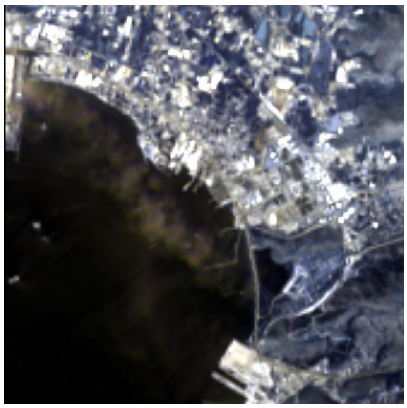
2.2.1 Αναδόμηση πολυφασματικής

Η βιβλιοθήκη GDAL μέσω της διαδικασίας επιτρέπει αναδόμηση εικόνας με τους παρακάτω αλγορίθμους:
Nearest Neighbour, Bilinear, Cubic Convolution Approximation, Cubic B-Spline Approximation, Lanczos windowed sinc interpolation, Average, Mode, Max, Min, Med, Q1, Q3.

Για επιλογή της πιο ικανοποιητικής δοκιμάστηκαν και οι 12:

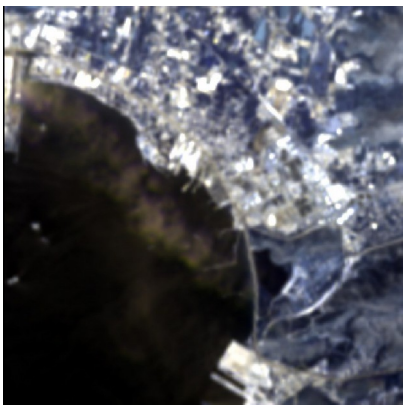


Εικόνα 9: Nearest Neighbour



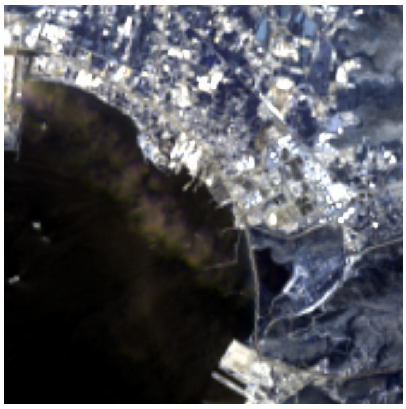
Εικόνα 10: Bilinear

Εικόνα 11: Cubic

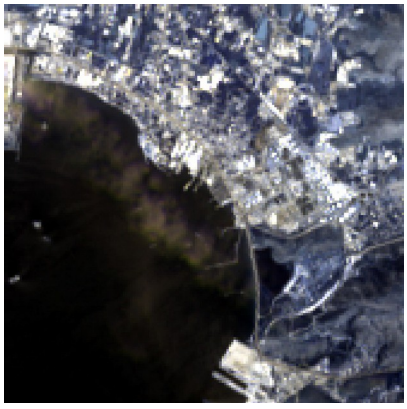


Εικόνα 14: Cubic B-Spline

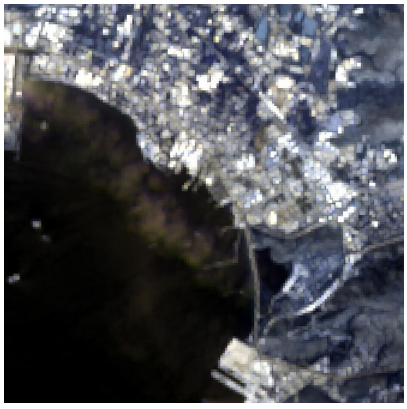
Εικόνα 12: Lanczos



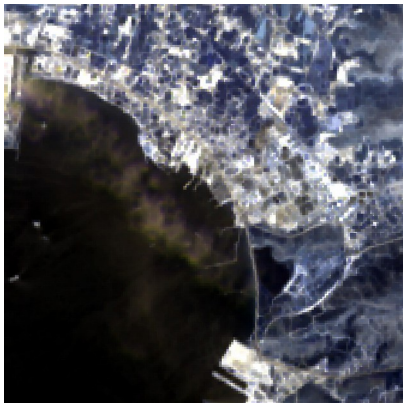
Εικόνα 13: Average



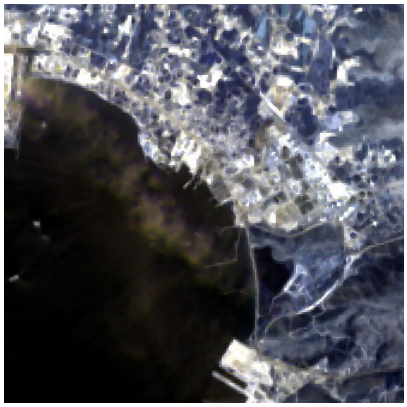
Εικόνα 15: Mode



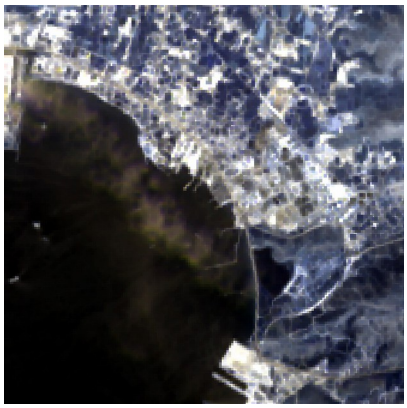
Εικόνα 16: Max



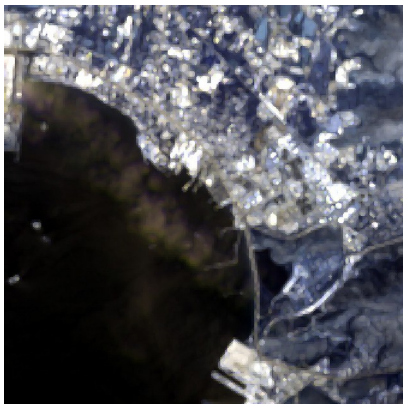
Εικόνα 17: Min



Εικόνα 18: Med



Εικόνα 19: Q1



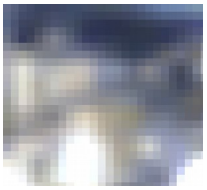
Εικόνα 20: Q3

Ύστερα έγινε παρατήρηση σε ίδια σημεία των απεικονίσεων για επιλογή της πιο αντιπροσωπευτικής αναδόμησης του πρωτότυπου:

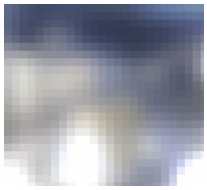
Εικόνα 21: Πρωτότυπο

Εικόνα 27: Nearest Neighbour

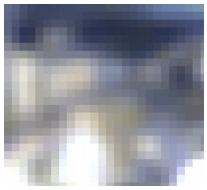
Εικόνα 22: Bilinear



Εικόνα 23: Cubic



Εικόνα 24: Cubic B-Spline



Εικόνα 25: Lanczos

Εικόνα 26: Average

Εικόνα 33: Mode

Εικόνα 28: Max

Εικόνα 29: Min

Εικόνα 30: Med

Εικόνα 31: Q1

Εικόνα 32: Q3

Ως περισσότερο αντιπροσωπευτική κρίθηκε αυτή που εφάρμοσε τον αλγόριθμο Average. Οι άλλες απορρίφθηκαν καθώς κάποιες μετατόπιζαν σημεία της εικόνας (NearestNeighbour, Mode), άλλες δεν την ανέλυσαν σε ομοιόμορφο αριθμό pixel (Max, Min, Med, Q1, Q3) και άλλες δεν είχαν ικανοποιητικό καταμερισμό πεδίου τιμών (Cibic B-Spline, Lanczos).

Περιγραφή Κώδικα Αναδόμησης:

Η προετοιμασία και υλοποίηση της αναδόμησης γίνεται στη συνάρτηση **my_resampling (GDALDataset dataset_pan,GDALDataset dataset_multi,char *fname_multi,double memLimit)** (Παράρτημα 1.6) όπου ως παραμέτρους δέχεται: dataset_pan, dataset_multi οι δείκτες στις εικόνες παγχρωματική (pan) και πολυφασμτική (multi), fname_multi το όνομα της πολυφασματικής και memLimit το όριο (σε bytes) μεγέθους κομματιού (chunk) στα οποία θα χωριστεί η εικόνα ώστε να χωρέσει στη μνήμη κατά τη διάρκεια της αναδόμησης.
Η ίδια η αναδόμηση γίνεται με τη συνάρτηση της GDAL GDALWarpOperation.ChunkAndWarpImage(), αφού στις ρυθμίσεις της GDALWarpOptions οριστούν:

- ο αλγόριθμος eResampleAlg (στην υλοποίηση GRA_Average)
- ο τύπος δεδομένων της εικόνας eWorkingDataType (οι Landsat8 είναι GDT_Int16)
- το μέγιστο επιτρεπόμενο μέγεθος κομματιού dfWarpMemoryLimit (το ελάχιστο 131072 bytes)
- οι πηγαίες ζώνες panSrcBands
- οι ζώνες προορισμού panDstBands (αντιστοιχούν στις panSrcBands μια προς μια)
- ένα ακριβές παράδειγμα γεωαναφοράς pTransformerArg (ορίστηκε η διαδικασία GDALCreateGenImgProjTransformer2 με υπόδειγμα τη μεταφορά από τη παγχρωματική στη πολυφασματική, αφού το αποτέλεσμα θέλουμε να μιμείται τη παγχρωματική).

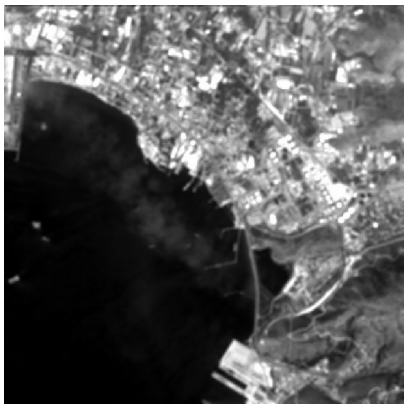
Σχετικός κώδικας:

```
  //Set Warp Options
  GDALResampleAlg resampleAlg=GRA_Average;
  GDALWarpOptions *psWarpOptions = GDALCreateWarpOptions();
  psWarpOptions->hSrcDS = dataset_multi;
  psWarpOptions->hDstDS = dataset_multires;
  psWarpOptions->nBandCount = dataset_multi->GetRasterCount();
  psWarpOptions->dfWarpMemoryLimit = memLimit;
  psWarpOptions->eResampleAlg = resampleAlg;
  psWarpOptions->eWorkingDataType = band_pan->GetRasterDataType();
  psWarpOptions->panSrcBands =
(int*)CPLMalloc(sizeof(int)*psWarpOptions->nBandCount);
  psWarpOptions->panDstBands =
(int*)CPLMalloc(sizeof(int)*psWarpOptions->nBandCount);
  for(int i=0;i<psWarpOptions->nBandCount;i++){
    psWarpOptions->panSrcBands[i]=i+1;
    psWarpOptions->panDstBands[i]=i+1;
  }
  psWarpOptions->pTransformerArg = GDALCreateGenImgProjTransformer2(
dataset_multi,dataset_pan,NULL);
  psWarpOptions->pfnTransformer = GDALGenImgProjTransform;
  //Warp Operation
  GDALWarpOperation oOperation;
  oOperation.Initialize( psWarpOptions );
  oOperation.ChunkAndWarpImage( 0,0,dataset_multires-
>GetRasterXSize(),dataset_multires->GetRasterYSize() );
```

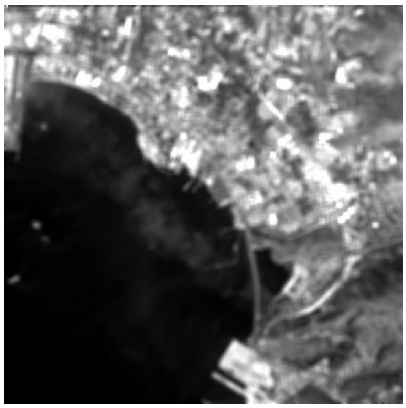
2.2.2 Αλγόριθμος Υπολογισμού Τοπικών Μέσων Όρων στην Παγχρωματική

Για την εύρεση τοπικών μέσων δοκιμάστηκαν παράθυρα 1x1, 3x3, 5x5, 7x7, 9x9:

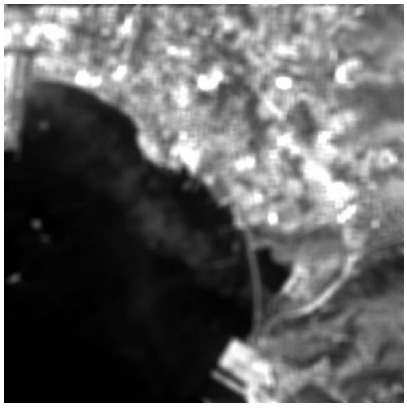
Εικόνα 36: Τοπικοί Μέσοι 1x1



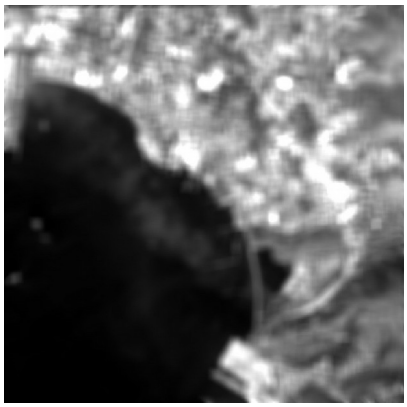
Εικόνα 34: Τοπικοί Μέσοι 3x3



Εικόνα 35: Τοπικοί Μέσοι 5x5



Εικόνα 37: Τοπικοί Μέσοι 7x7



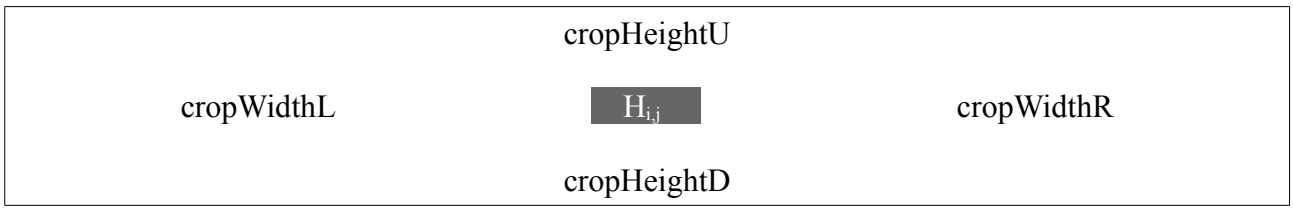
Εικόνα 38: Τοπικοί Μέσοι 9x9

Περιγραφή Κώδικα Υπολογισμού Τοπικών Μέσων:

Ο υπολογισμός των τοπικών μέσων γίνεται στη συνάρτηση **hpf_means (GDALDataset *dataset_pan,char *fname_pan,int winWidth,int winHeight)** (Παράρτημα 1.5), όπου dataset_pan και fname_pan δείκτης και όνομα παγχρωματικής εικόνας, winWidth και winHeight πλάτος και ύψος παραθύρου για τον υπολογισμό των τοπικών μέσων.

Αρχικά γίνεται ενημέρωση των παραμέτρων μετακίνησης του παραθύρου στην εικόνα:

- cropOffsetX, cropOffsetY συντεταγμένες του pixel τέρμα πάνω αριστερά στο παράθυρο
- cropWidthL, cropWidthR πλάτη δεξιά και αριστερά από το pixel του οποίου υπολογίζεται ο τοπικός μέσος όρος
- cropHeightU, cropHeightD ύψη πάνω και κάτω από το pixel του οποίου υπολογίζεται ο τοπικός μέσος όρος



Πίνακας 1: Παράθυρο τοπικού μέσου

Σχετικός κώδικας:

```
//for all width
  for(int i=0;i<dataset_pan->GetRasterXSize();i++){
    //update crop width parameters
    if(i-winWidth<0){
      cropOffsetX=0;
      cropWidthL=i;
    }
    else{
      cropOffsetX=i-winWidth;
      cropWidthL=winWidth;
    }
    if(i+winWidth>=dataset_pan->GetRasterXSize())
      cropWidthR=dataset_pan->GetRasterXSize()-1-i;
    else
      cropWidthR=winWidth;
    //for all height
    for(int j=0;j<dataset_pan->GetRasterYSize();j++){
      //update crop height parameters
      if(j-winHeight<0){
        cropOffsetY=0;
        cropHeightU=j;
      }
      else{
        cropOffsetY=j-winHeight;
        cropHeightU=winHeight;
      }
      if(j+winHeight>=dataset_pan->GetRasterYSize())
        cropHeightD=dataset_pan->GetRasterYSize()-1-j;
      else
        cropHeightD=winHeight;
      //read window
      band_pan->RasterIO (
GF_Read,cropOffsetX,cropOffsetY,cropWidthL+cropWidthR+1,cropHeight
U+cropHeightD+1,buffer,cropWidthL+cropWidthR+1,cropHeightU+cropHei
ghtD+1,bufDataType,0,0 );
      //calculate local mean of window
      local_mean = mean_calc(
buffer,cropWidthL+cropWidthR+1,cropHeightU+cropHeightD+1,bufDataTy
pe );
      //write resulting pixel value
      band_means->RasterIO (
GF_Write,i,j,1,1,local_mean,1,1,bufDataType,0,0 );
    }
  }
```

Ύστερα γίνεται ο υπολογισμός του τοπικού μέσου όρου με κέντρο το pixel $H_{i,j}$ στη συνάρτηση **mean_calc (void *buffer, int bufsizeX, int bufsizeY, GDALDataType bufDataType)** (Παράρτημα 1.5) όπου buffer το περιεχόμενου του παραθύρου, bufsizeX το συνολικό πλάτος (cropWidthL+1+croWidthR) του παραθύρου, bufsizeY το συνολικό ύψος (cropHeightU+1+cropHeightD) του παραθύρου και bufDataType ο τύπος δεδομένων που περιλαμβάνει η εικόνα. Μέσα στη συνάρτηση, αφού αναγνωριστεί ο τύπος δεδομένων και αθροιστούν οι τιμές όλων των pixel του παραθύρου, διαιρούνται με τον συνολικό αριθμό τους για να υπολογιστεί ο μέσος όρος.

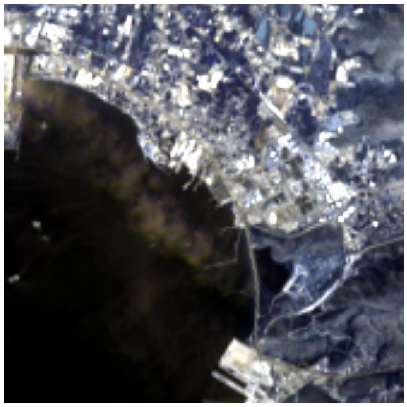
Σχετικός κώδικας:

```
  if(bufDataType==GDT_Int16)
    *(GInt16*)local_mean = *(GUInt32*)sum/(GInt16)
(bufsizeX*bufsizeY);
  else if(bufDataType==GDT_UInt16)
    *(GUInt16*)local_mean = *(GUInt32*)sum/(GUInt16)
(bufsizeX*bufsizeY);
```

Πλέον είναι έτοιμες οι δυο εικόνες που χρειάζονται για την υλοποίηση της συγχώνευσης HPF και εφαρμόζεται με διαδικασία pixel-pixel ο τύπος $HPF_{i,j} = Lw_{i,j} + (H_{i,j} - Hm_{i,j})$ που περιγράφηκε παραπάνω ώστε να βρεθεί το τελικό αποτέλεσμα. Η υλοποίηση του τύπου γίνεται στη συνάρτηση **hpf_fusion (GDALDataset *dataset_pan, char *fname_pan, GDALDataset *dataset_multi, char *fname_multi, int winWidth, int winHeight)** (Παράρτημα 1.4) όπου dataset_pan και dataset_multi δείκτες στις εικόνες (παγχρωματική-πολυφασματική) που αποτελούν πηγή για την pansharpened που θα παραχθεί και fname_multi το όνομα της πολυφασματικής (για την ονοματοποιία του αποτελέσματος). WinWidth και winHeight είναι οι παράμετροι που θα περαστούν όταν η hpf_fusion καλέσει την **hpf_means** και αναφέρονται στο μεγέθους παραθύρου για τον υπολογισμό των τοπικών μέσων.

Σχετικός κώδικας:

```
       if(bufDataType==GDT_Int16)
         if(*(GInt16*)buffer_multires==0||
*(GInt16*)buffer_multires==band_multires->GetNoDataValue(NULL))
           *(GInt16*)buffer_hpf=0;
         else
           *(GInt16*)buffer_hpf=*(GInt16*)buffer_multires+
(*(GInt16*)buffer_pan-*(GInt16*)buffer_panmean);
       else if(bufDataType==GDT_UInt16)
         if(*(GUInt16*)buffer_multires==0||
*(GUInt16*)buffer_multires==band_multires->GetNoDataValue(NULL))
           *(GUInt16*)buffer_hpf=0;
         else
           *(GUInt16*)buffer_hpf=*(GUInt16*)buffer_multires+
(*(GUInt16*)buffer_pan-*(GUInt16*)buffer_panmean);
       //write resulting pixel value on hpf image
       band_hpf->RasterIO (
GF_Write,i,j,1,1,buffer_hpf,1,1,bufDataType,0,0 );
```



Εικόνα 39:
HPF με Παράθυρο 1x1

Εικόνα 40:
HPF με Παράθυρο 3x3

Εικόνα 41:
HPF με Παράθυρο 5x5

Εικόνα 43:
HPF με Παράθυρο 7x7

Εικόνα 42:
HPF με Παράθυρο 9x9

Πιο αναλυτική και καθαρή απεικόνιση φαίνεται να έχει το αποτέλεσμα με Παράθυρο 5x5.

Τελικά, είναι εμφανής η διαφορά ανάμεσα στις αρχικές και τελικές εικόνες:

Εικόνα 46:
Αρχική Παγχρωματική

Εικόνα 44:
Αρχική Πολυφασματική RGB

Εικόνα 45:
Τελική Συγχωνευμένη RGB με Μέθοδο High Pass Filtering

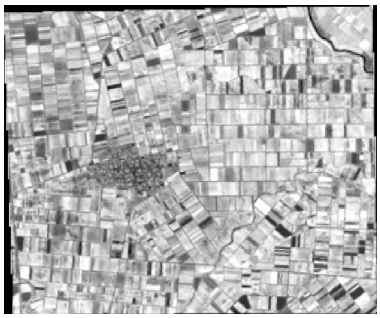
2.3 Υπολογισμός Κανονικοποιημένου Δείκτη Βλάστησης (NDVI)

Ο Κανονικοποιημένος Δείκτης Βλάστησης (Normalized Difference Vegetation Index – NDVI) δείχνει τα επίπεδα βλάστησης σε μια δορυφορική πολυφασματική εικόνα μέσω του τύπου: $NDVI = \frac{NIR - R}{NIR + R}$, όπου NIR το εγγύς υπέρυθρο (ζώνη 5) και R το κόκκινο (ζώνη 4).

Παράδειγμα εύρεσης NDVI:

Εικόνα 47:
Αξιός1 (pansharpened)
2013-09-12
R-G-B

Εικόνα 48:
Αξιός1 (pansharpened)
2013-09-12
NIR-R-G



Εικόνα 49:
Αξιός1 (pansharpened)
2013-09-12
NDVI

Η αριστερή εικόνα είναι μια εικόνα πραγματικών χρωμάτων (το κόκκινο είναι το κόκκινο φάσμα (R), το πράσινο είναι το πράσινο φάσμα (G) και το μπλε είναι το μπλε φάσμα (B)). Τα αγροτικά τεμάχια που έχουν καλλιέργειες εμφανίζονται ως πράσινα. Οι διαφορετικές αποχρώσεις του πράσινου σε διάφορα χωράφια αντιπροσωπεύουν διαφορετικά επίπεδα βλάστησης και συνεπώς διαφορετικές καλλιέργειες ή τις ίδιες, αλλά σε διαφορετικό στάδιο ανάπτυξης. Το γυμνό έδαφος είναι καστανό.

Η μεσαία εικόνα είναι μια εικόνα ψευδών χρωμάτων (το κόκκινο είναι το κοντινό υπέρυθρο (NIR) φάσμα, το πράσινο είναι το κόκκινο φάσμα (R) και το μπλε είναι το πράσινο φάσμα (G)). Εδώ, οι υψηλές τιμές στο κοντινό υπέρυθρο αντιπροσωπεύουν τη βλάστηση, η οποία εμφανίζεται κόκκινη. Το γυμνό έδαφος φαίνεται κυανό, επειδή δεν υπάρχει βλάστηση για να αυξηθεί το κόκκινο (κοντινό υπέρυθρο) στην εικόνα.

Η εικόνα στα δεξιά είναι το NDVI για κάθε pixel της εικόνας. Τα σκούρα pixel έχουν χαμηλό NDVI, τα λευκά έχουν υψηλό και τα γκρι pixel έχουν ενδιάμεσες τιμές.

Περιγραφή Κώδικα Υπολογισμού NDVI:

Η συνάρτηση **calc_ndvi(char * fname_multi)** (Παράρτημα 1.8) με παράμετρο το όνομα της πολυφασματικής της οποίας αναζητούμε το NDVI υπολογίζει το NDVI καλώντας με τη σειρά της τη συνάρτηση **calc_ndi (char *fname_multi, char *fname_ndvi ,int b_minuend,int b_subtrahend)** (Παράρτημα 1.7) με παραμέτρους το όνομα της αρχικής πολυφασματικής, το όνομα της εικόνας που θα δημιουργηθεί και θα περιέχει τις τιμές NDVI καθώς και τους αριθμούς ζωνών για την εφαρμογή του παραπάνω τύπου. Δηλαδή b_minuend ο αφαιρέτης (5-εγγύς υπέρυθρο) και b_subtrahend ο αφαιρετέος (4-κόκκινο).

Σχετικός κώδικας:

```
return calc_ndi(fname_multi,fname_ndvi,5,4);
```

Αφού ανοιχθεί η αρχική εικόνα, δημιουργείται η τελική εικόνα, η οποία θα περιέχει τις τιμές NDVI, με τύπο δεδομένων Float32 αφού το αποτέλεσμα της διαίρεσης έχει τιμές στο διάστημα [-1,1]. Στη συνέχεια αφού δεσμευτούν οι κατάλληλες μεταβλητές με τη συνάρτηση , υπολογίζεται η τιμή NDVI για κάθε pixel ξεχωριστά, τα οποία διαβάζονται από τις πηγαίες ζώνες (5 και 4) με τη βοήθεια της συνάρτησης RasterIO της GDAL, αφαιρούνται και το αποτέλεσμα γράφεται στο αρχείο προορισμού πάλι μέσω της RasterIO.

Σχετικός κώδικας:

```
dataset_ndi = poDriver->Create(fname_ndi,dataset_multi-
>GetRasterXSize(),dataset_multi-
>GetRasterYSize(),1,GDT_Float32,createOptions);

for(int i=0;i<dataset_multi->GetRasterXSize();i++){//for all band
width
    for(int j=0;j<dataset_multi->GetRasterYSize();j++){//for all band
height
     band_min-
>RasterIO(GF_Read,i,j,1,1,min_buf,1,1,bufDataType,0,0);
     band_sub-
>RasterIO(GF_Read,i,j,1,1,sub_buf,1,1,bufDataType,0,0);
     //compute ndi
     if(bufDataType==GDT_UInt16){
       ndi_lower=(GInt32)(*(GUInt16*)min_buf+*(GUInt16*)sub_buf);
       ndi_upper=(GInt16)(*(GUInt16*)min_buf-*(GUInt16*)sub_buf);
       if (ndi_lower!=0)
         ndi=ndi_upper/ (float)ndi_lower;
       else
          ndi=0;
     }
     else if(bufDataType==GDT_Int16){
       ndi_lower=(GInt32)(*(GInt16*)min_buf+*(GInt16*)sub_buf);
       ndi_upper=(*(GInt16*)min_buf-*(GInt16*)sub_buf);
       if (ndi_lower!=0)
         ndi=ndi_upper/ (float)ndi_lower;
       else
          ndi=0;
     }
     else ndi=0;
     //write result
     band_ndi-
>RasterIO(GF_Write,i,j,1,1,&ndi,1,1,GDT_Float32,0,0);
    }
  }
```

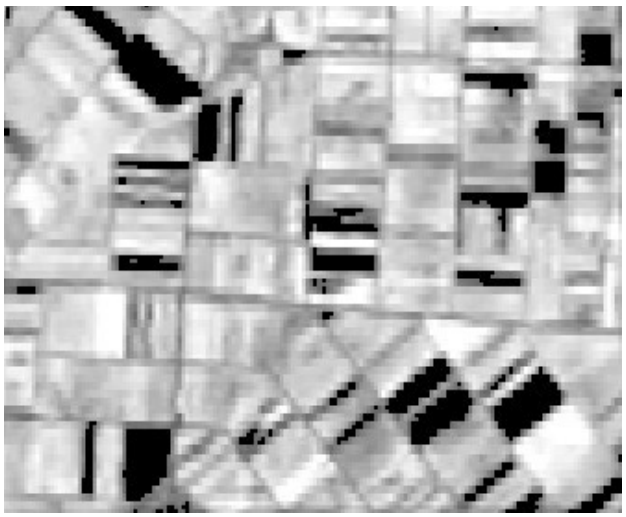
2.4 Κατωφλίωση και Ομαδοποίηση NDVI

2.4.1 Κατωφλίωση NDVI

Πριν ομαδοποιηθούν οι τιμές pixel στην εικόνα με τους δείκτες NDVI, πρέπει να βρεθεί μια τιμή κατωφλίωσης η οποία ορίζει από ποια τιμή και πάνω βρίσκεται η βλάστηση.

Στις εικόνες που πάρθηκαν κατευθείαν από δορυφόρο μετά από δοκιμές, αυτή η τιμή ορίστηκε στο 0.2.

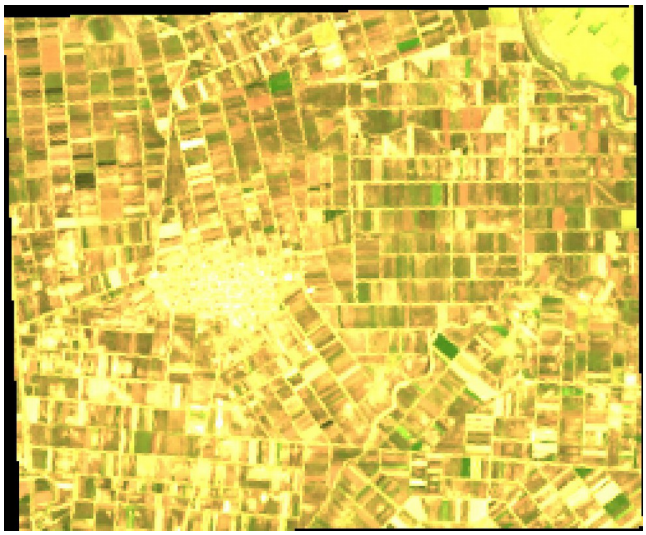
Εικόνα 50: Αξιός1 2013-09-12 τεμάχιο (pansharpened) R-G-B



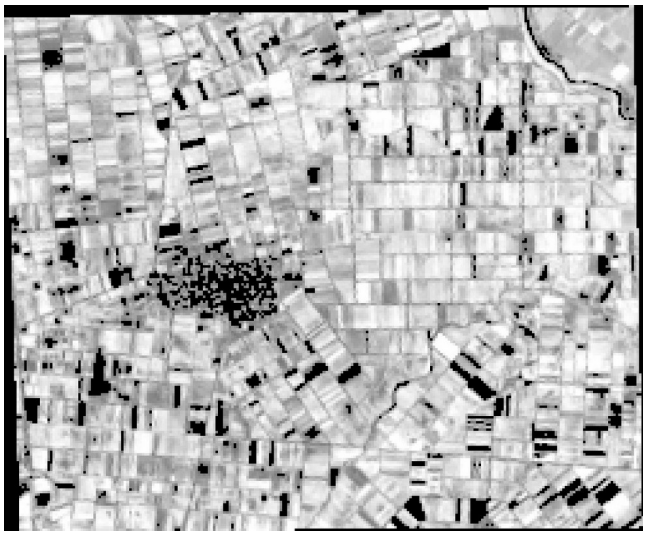
Εικόνα 51: Αξιός1 2013-09-12 τεμάχιο (pansharpened) NDVI-thres0.2

Μπορεί να παρατηρηθεί πως στις περιοχές της RGB με έντονο πράσινο (καλλιέργειες) όταν δούμε τις αντίστοιχες στην NDVI, είναι σε αποχρώσεις γκρι κοντά στο άσπρο, ενώ σε περιοχές που δεν υπάρχει καθόλου πράσινο (κτίρια, δρόμοι) και συνεπώς με τιμές NDVI κάτω του 0.2, υπάρχει απόλυτο μαύρο.

Επίσης υπολογίστηκε και η κατωφλίωση για εικόνες που είχαν υποστεί ατμοσφαιρικές διορθώσεις[5]. Ως τιμή κατωφλίωσης ορίστηκε το 0.4 μετά από δοκιμές.



Εικόνα 52: Αξιός1 2013-09-12 (διορθωμένη) RGB



Εικόνα 53: Αξιός1 2013-09-12 (διορθωμένη) NDVI-thres0.4

Περιγραφή κώδικα Κατωφλίωσης NDVI:

Η κατωφλίωση NDVI εικόνων υλοποιείται στη συνάρτηση **ndvi_threshold(char *fname_src, float threshold)** (Παράρτημα 1.8), όπου fname_src το όνομα του αρχείου που περιλαμβάνει τους δείκτες NDVI και threshold την τιμή κατωφλίωσης (0.2 για αρχικές, 0.4 για διορθωμένες).

Μέσα στη συνάρτηση γίνεται έλεγχος ανά pixel της αρχικής εικόνας και αν η τιμή τους είναι χαμηλότερη της τιμής κατωφλίωσης, μηδενίζονται.

Σχετικός κώδικας:

```
float pixel;
  //apply threshold
  for(int i=0;i<dataset_src->GetRasterXSize();i++){//for all band
width
    for(int j=0;j<dataset_src->GetRasterYSize();j++){//for all band
height
      band_src-
>RasterIO(GF_Read,i,j,1,1,&pixel,1,1,GDT_Float32,0,0);//read source
(i,j) value
      //if outside the thresholds, nullify; max_threshold must be
of positive value (negative for only min_threshold)
      if(pixel<threshold)
          pixel=0;
      band_thres-
>RasterIO(GF_Write,i,j,1,1,&pixel,1,1,GDT_Float32,0,0);//write
pixel value to resulting band
    }
  }
```

2.4.2 Ομαδοποίηση NDVI

Για την ομαδοποίηση τιμών NDVI του αποτελέσματος της κατωφλίωσης χρησιμοποιήθηκαν οι παρακάτω κλάσεις[6]:

Αρχικές (Raw)		Διορθωμένες	
Αριθμός Κλάσης	Διάστημα	Αριθμός Κλάσης	Διάστημα
1	(0.2, 0.3]	1	(0.4 , 0.5]
2	(0.3 , 0.4]	2	(0.5 , 0.6]
3	(0.4 , 0.5]	3	(0.6 , 0.7]
4	(0.5 , 0.6]	4	(0.7 , 0.8]
5	(0.6 , 0.7]	5	(0.8 , 0.9]

Πίνακας 2: Κλάσεις δείκτη βλάστησης

Οι τιμές εκτός των διαστημάτων μπήκαν σε κατηγορίες 0 (για κάτω του χαμηλότερου) και 6 (για άνω του υψηλότερου) αλλά δεν λαμβάνονται υπόψη στη διάκριση βλάστησης.

Το αποτέλεσμα είναι μια εικόνα της οποία τα pixel έχουν μόνο τις ακέραιες τιμές 0-6, ανάλογα με την κλάση στην οποία ανήκουν.

Περιγραφή κώδικα Ομαδοποίησης NDVI:

Η ομαδοποίησης των τιμών NDVI γίνεται στη συνάρτηση **ndvi**_classification**(char *fname_src,float threshold,float step)** (Παράρτημα 1.9), όπου fname_src το όνομα του αρχείου με τις τιμές NDVI μετά την κατωφλίωση, threshold η τιμή κατωφλίωσης και step το βήμα μεταξύ των κλάσεων (δεδομένου ότι και στις δυο περιπτώσεις το βήμα είναι 0.1).

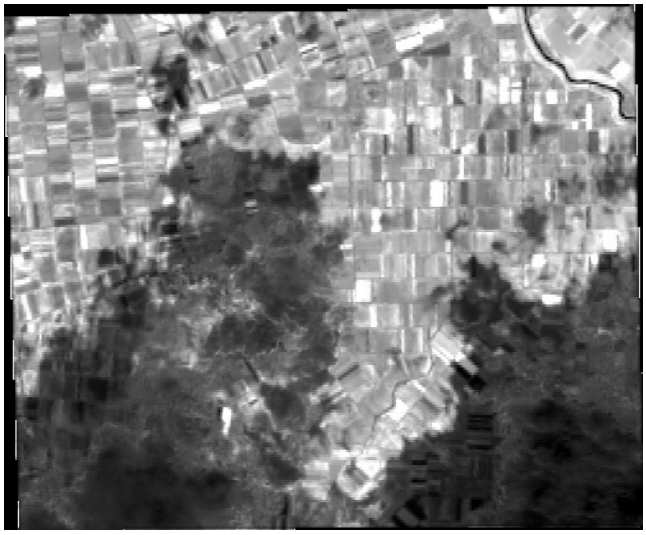
Για τις τιμές κάτω της κατωφλίωσης δίνεται η τιμή 0. Για τις υπόλοιπες υπολογίζεται μέσω της παρακάτω διαδικασίας:

κλάση=τιμή_κατωφλίωσης
όσο τιμή_pixel_εισόδου > κλάση
κλάση = κλάση + βήμα
*τιμή_pixel_εξόδου = (κλάση – βήμα - τιμή_κατωφλίωσης) * 10 + 1.1*

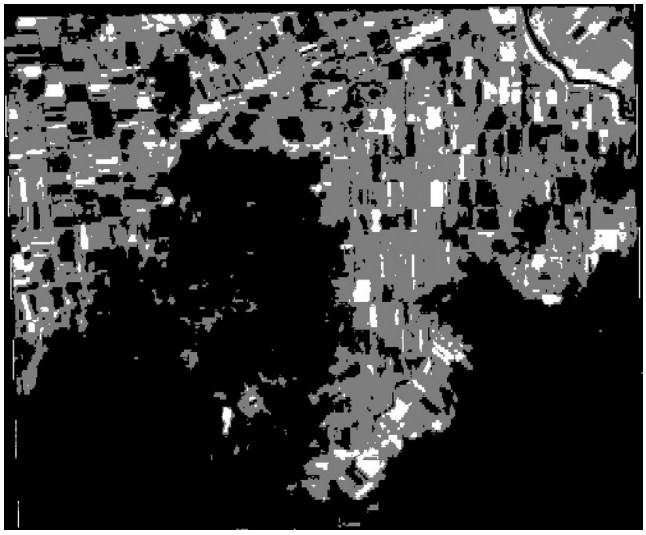
με αυτό τον τρόπο βρίσκεται το διάστημα με κάτω άκρο 'κλάση' στο προηγούμενο απ'το οποίο ανήκει το 'pixel_εισόδου' και όταν βρεθεί υπολογίζεται η ακέραια 'τιμή_pixel_εξόδου' αφαιρώντας από την 'κλάση' το 'βήμα' (βρίσκεται η τιμή του κάτω άκρου του σωστού διαστήματος) μετά την 'τιμή_κατωφλίωσης' (βρίσκεται ο αριθμός διαστήματος σε δεκαδικό με πρώτο το 0) και αφού πολλαπλασιαστεί το αποτέλεσμα με το 10 για να γίνει ακέραιος, του προστίθεται 1.1 ώστε να βρεθεί στη σωστή τιμή κλάσης (το 0 είναι εκτός διαστημάτων!). Το 0.1 δίνεται για να ξεπεραστεί κάποια πιθανή ανακρίβεια στην τιμή του δεκαδικού και η τελική τιμή να είναι σίγουρα πάνω αλλά και ταυτόχρονα κοντά στην ακέραια τιμή που ζητείται.

Σχετικός κώδικας:

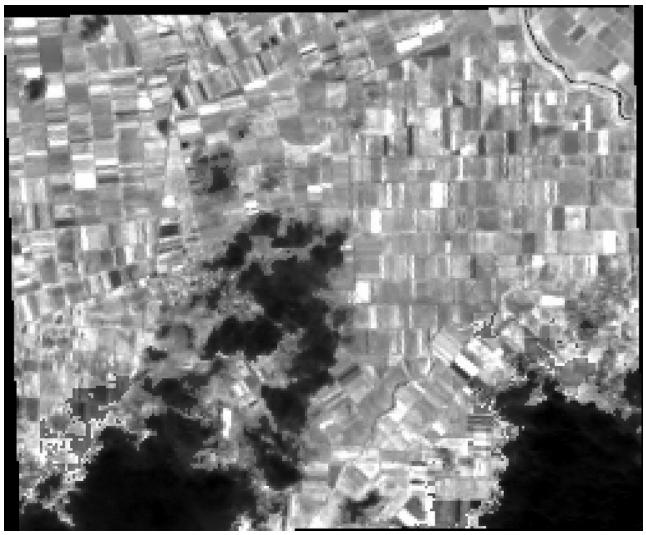
```
//apply threshold
  for(int i=0;i<dataset_src->GetRasterXSize();i++){//for all band
width
    for(int j=0;j<dataset_src->GetRasterYSize();j++){//for all band
height
     band_src-
>RasterIO(GF_Read,i,j,1,1,&pixelI,1,1,GDT_Float32,0,0);//read
source (i,j) value
     //if outside the thresholds, nullify; max_threshold must be
of positive value (negative for only min_threshold)
     pixelO=0;
     if(pixelI>threshold){
       clas=threshold;
       while(pixelI>=clas)
         clas+=step;
       pixelO=(clas-step-threshold)*10+1.1;
     }
     band_thres-
>RasterIO(GF_Write,i,j,1,1,&pixelO,1,1,GDT_UInt16,0,0);//write
pixel value to resulting band
    }
  }
```



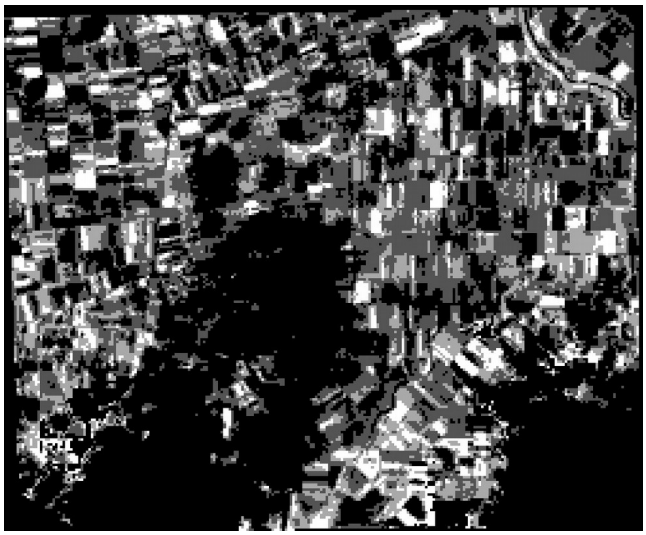
Εικόνα 54: Αξιός1 2013-09-28 (αρχική) NDVI



Εικόνα 55: Αξιός1 2013-09-28 (αρχική) Ομαδοποίηση NDVI



Εικόνα 56: Αξιός1 2013-09-28 (διορθωμένη) NDVI



Εικόνα 57: Αξιός1 2013-09-28 (διορθωμένη) Ομαδοποίηση NDVI

Η ομαδοποίηση των pixel σε τιμές από 0 μέχρι 6 είναι εμφανής και από τη μεγάλη μείωση των αποχρώσεων του γκρι σε σχέση με την αντίστοιχη NDVI.

Στη συγκεκριμένη εικόνα, υπάρχει μια ανωμαλία σε μεγάλο κομμάτι του κάτω μέρους καθώς τη στιγμή της φωτογράφησης μεσολαβούσαν σύννεφα στη ατμόσφαιρα.

2.5 Υπολογισμός Κανονικοποιημένων Δεικτών Νερού NDWI και Χλωροφύλλης

2.5.1 Κανονικοποιημένος Δείκτης Νερού NDWI

Ο Κανονικοποιημένος Δείκτης Νερού (NDWI) βοηθά στη διάκριση υδάτινων και μη υδάτινων χαρακτηριστικών και μπορεί να υπολογιστεί από διάφορους συνδυασμούς διαφορετικών ζευγών καναλιών: πράσινο (G) και εγγύς υπέρυθρο (NIR), NIR και σύντομο κύμα IR (SWIR), κόκκινο (R) και μέσο υπέρυθρο (MIR). Στην παρούσα εργασία χρησιμοποιήθηκε ο πρώτος.

Ο δείκτης $NDWI = \frac{G - NIR}{G + NIR}$, όπου G πράσινο (ζώνη 3) και NIR εγγύς υπέρυθρο (ζώνη 5), αποσκοπεί στη μεγέθυνση της υψηλής τιμής ανακλαστικότητας του νερού στο πράσινο κανάλι, στη μείωση της χαμηλής τιμής ανακλαστικότητας του νερού στο NIR και στη χρήση της διακεκριμένης αντίθεσης μεταξύ του νερού και γήινης επιφάνειας στο NIR.

Εικόνα 58: Λίμνη Κάρλα 2013-09-28 (pansharpened) R-G-B

Εικόνα 59: Λίμνη Κάρλα 2013-09-28 (pansharpened) NDWI

Μπορεί να παρατηρηθεί πως όσο πιο κοντά στο άσπρο είναι οι αποχρώσεις, τόσο περισσότερη υγρασία υπονοείται (η λίμνη φαίνεται κάτασπρη), ενώ κοντά στο μαύρο υπάρχει έλλειψη.

Περιγραφή κώδικα Υπολογισμού NDWI:

Για τον υπολογισμό του δείκτη NDWI χρησιμοποιείται η συνάρτηση **calc_ndwi(char *fname_multi)** (Παράρτημα 1.10), όπου fname_multi το όνομα της εικόνας της οποίας θέλουμε να τον υπολογίσουμε. Αυτή με τη σειρά της καλεί όπως και η calc_ndvi που περιγράφηκε παραπάνω την **calc_ndi** (Παράρτημα 1.7), αλλά με τιμές αφαιρέτη 3 (G) και αφαιρετέου 5 (NIR).

Ακριβώς όπως και στην περίπτωση του NDVI, το τελικό αρχείο θα έχει τιμές pixel στο διάστημα [-1, 1], οπότε και πάλι θα είναι τύπου Float32.

Σχετικός κώδικας:

```
return calc_ndi(fname_multi,fname_ndwi,3,5);
```

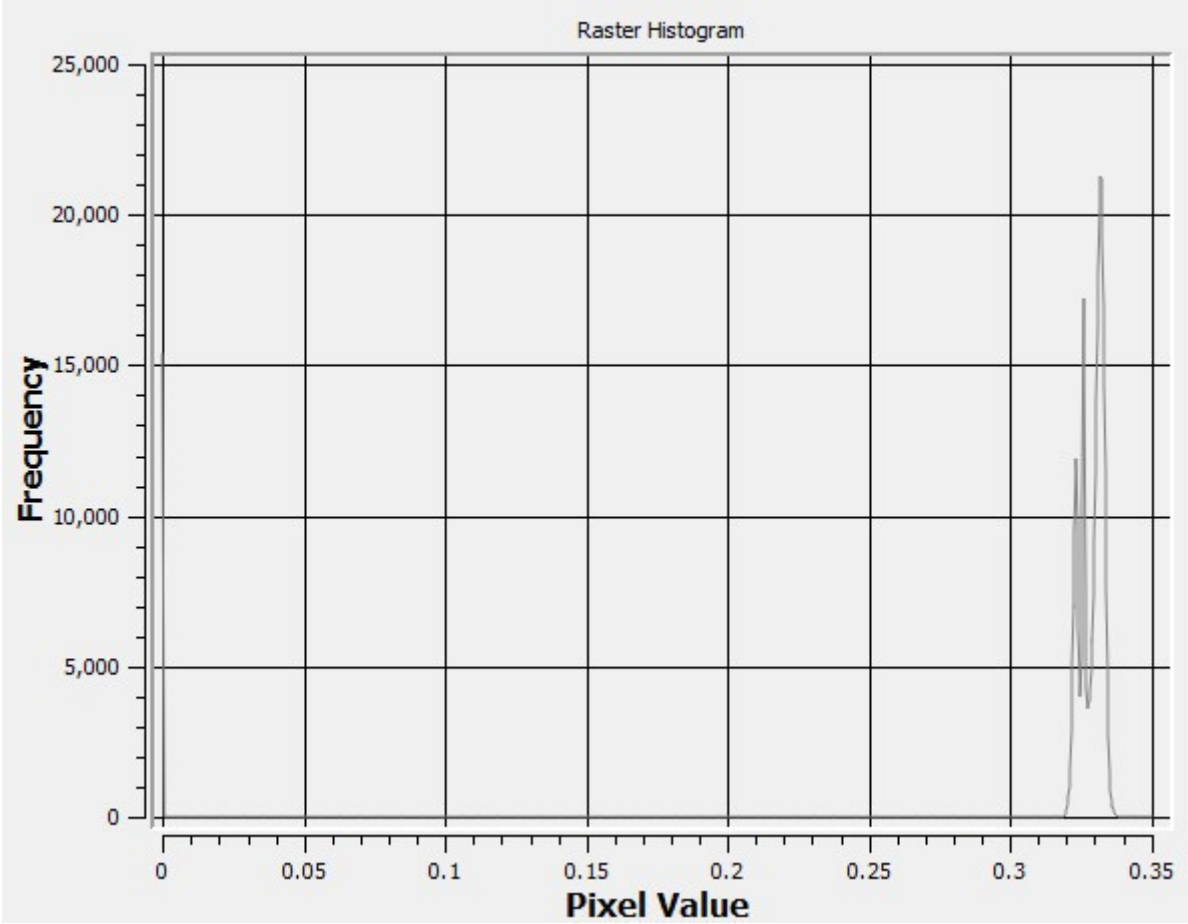
2.5.2 Δείκτης Χλωροφύλλης

Ο υπολογισμός της χλωροφύλλης γίνεται από τον τύπο $CHLORO = \frac{B}{C+B+G}$ [7], όπου B μπλε (ζώνη 2), C παράκτιο (ζώνη 1) και G πράσινο (ζώνη 3).

Εικόνα 60: Λίμνη Κάρλα 2013-09-28 (pansharpened) R-G-B

Εικόνα 61: Λίμνη Κάρλα 2013-09-28 (pansharpened) CHLORO

Οι τιμές μπορούν να είναι εντός του διαστήματος [0,1], και όπως φαίνεται από το ιστόγραμμα συμπυκνώνονται γύρω από την τιμή 0.34, οπότε υπάρχει μικρή διαφορά μεταξύ των αποχρώσεων, οι οποίες κυμαίνονται γύρω από το άσπρο-μέγιστη τιμή (0.35 στη συγκεκριμένη εικόνα), και έτσι διακρίνεται δύσκολα το επίπεδο χλωροφύλλης.



Εικόνα 62: Λίμνη Κάρλα 2013-09-28 (pansharpened) CHLORO Ιστόγραμμα

Περιγραφή κώδικα Υπολογισμού Χλωροφύλλης:

Ο υπολογισμός γίνεται με βάση τον παραπάνω τύπο στη συνάρτηση **calc_chloro(char *fname_multi)** (Παράρτημα 1.10), όπου fname_multi η πολυφασματική εικόνα της οποίας θα υπολογιστεί η χλωροφύλλη. Ανά pixel, υπολογίζεται πρώτα ο παρονομαστής C+B+G και αν δεν είναι 0 διαιρείται από την τιμή του μπλε (B), αλλιώς (αν είναι 0) μηδενίζεται κατευθείαν. Η ανάγνωση και εγγραφή κάθε pixel γίνεται όπως και στις υπόλοιπες συναρτήσεις με τη συνάρτηση RasterIO της GDAL.

Σχετικός κώδικας:

```
for(int i=0;i<dataset_multi->GetRasterXSize();i++){//for all band
width
        for(int j=0;j<dataset_multi->GetRasterYSize();j++){//for
all band height
            //read (i,j) values from minuend and subtrahend
            band_coastal-
>RasterIO(GF_Read,i,j,1,1,p_coastal,1,1,bufDataType,0,0);
            band_blue-
>RasterIO(GF_Read,i,j,1,1,p_blue,1,1,bufDataType,0,0);
            band_green-
>RasterIO(GF_Read,i,j,1,1,p_green,1,1,bufDataType,0,0);
            //compute ndi
            if(bufDataType==GDT_UInt16){

subtrahend=*(GUInt16*)p_coastal+*(GUInt16*)p_blue+*(GUInt16*)p_gre
en;
                if (subtrahend!=0)
                    p_chloro=*(GUInt16*)p_blue/(float)subtrahend;
                else
                    p_chloro=0;
            }
            else if(bufDataType==GDT_Int16){

subtrahend=*(GInt16*)p_coastal+*(GInt16*)p_blue+*(GInt16*)p_green;
                if (subtrahend!=0)
                    p_chloro=*(GInt16*)p_blue/(float)subtrahend;
                else
                    p_chloro=0;
            }
            else p_chloro=0;
            //write result
            band_chloro-
>RasterIO(GF_Write,i,j,1,1,&p_chloro,1,1,GDT_Float32,0,0);
        }
    }
```

2.6 NDWI Φιλτράρισμα και Ομαδοποίηση Χλωροφύλλης

2.6.1 NDWI Φιλτράρισμα Χλωροφύλλης

Για την αναζήτηση της χλωροφύλλης εντός υδάτινων στοιχείων της εικόνας χρειάζεται να φιλτραριστεί η CHLORO με βάση την κατωφλίωση της NDWI ανά pixel. Μετά από δοκιμές σε αρχικές (raw) εικόνες, εντοπίστηκε ως κατώφλι της NDWI, δηλαδή ως κατώτερη ένδειξη παρουσίας υδάτινου στοιχείου, η τιμή 0.

Αποτέλεσμα είναι μια εικόνα με τιμές χλωροφύλλης μόνο εντός υδάτινων στοιχείων.

Περιγραφή κώδικα Φιλτραρίσματος Χλωροφύλλης βάσει Κατωφλιωμένης NDWI:

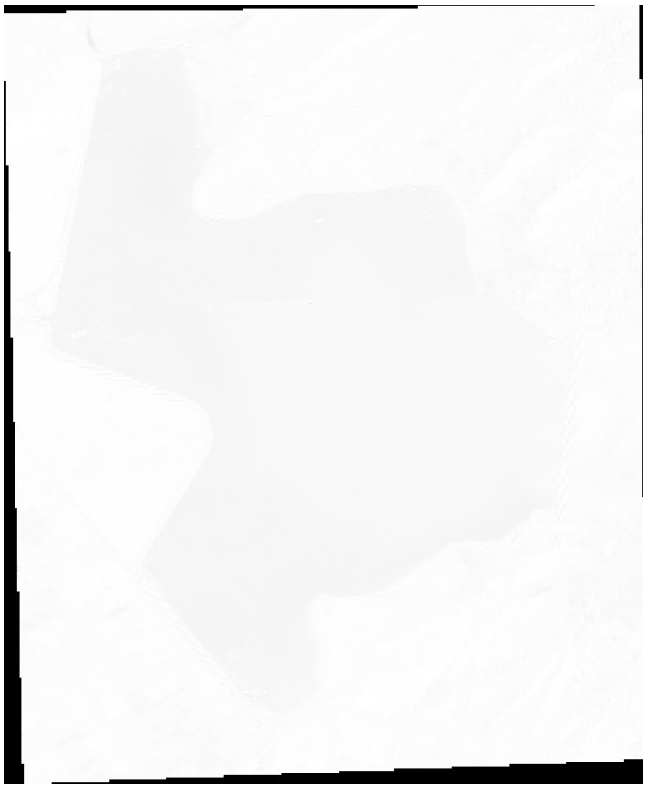
Το φιλτράρισμα χλωροφύλλης βάσει κατωφλιωμένης NDWI γίνεται στη συνάρτηση apply_**ndwi**_filter**(char *fname_chloro, char *fname_ndwi,** float water_**thres)** (Παράρτημα 1.11), όπου fname_chloro το όνομα της εικόνας με το Δείκτη Χλωροφύλλης, fname_ndwi το όνομα της εικόνας με το NDWI και water_thres η τιμή κατωφλίωσης (εδώ μόνο για αρχικές εικόνες).

Εφόσον η τιμή του pixel στο σημείο (x,y) είναι άνω του κατωφλιού, τότε το pixel (x,y) της τελικής εικόνας παίρνει την τιμή του pixel (x,y) της CHLORO, ειδάλλως μηδενίζεται.

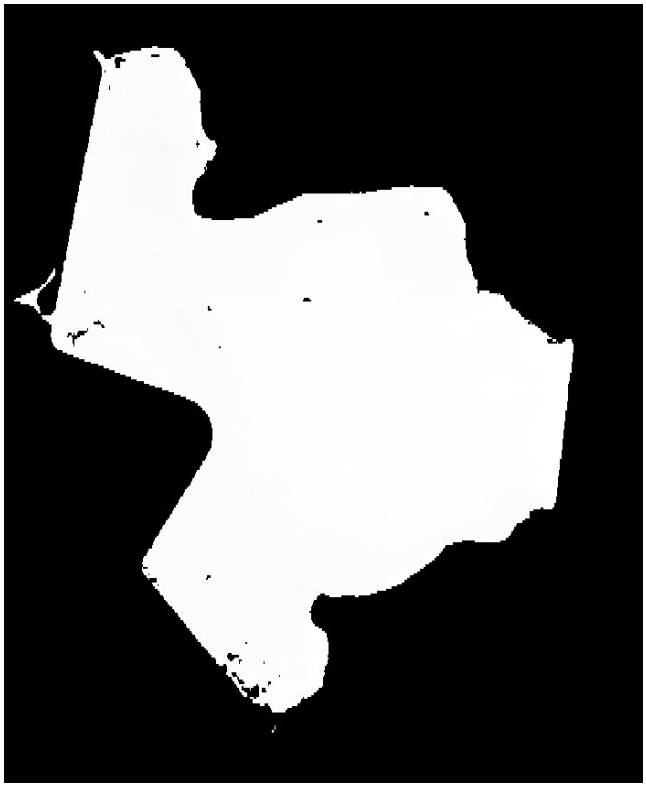
Σχετικός κώδικας:

```
  //apply filter (if ndwi no zero, copy chloro value)
  for(int i=0;i<dataset_ndwi->GetRasterXSize();i++){//for all band
width
    for(int j=0;j<dataset_ndwi->GetRasterYSize();j++){//for all band
height
     band_ndwi-
>RasterIO(GF_Read,i,j,1,1,&pixel_ndwi,1,1,GDT_Float32,0,0);
     if(pixel_ndwi<water_thres)//if ndwi is under the water_thres,
filtered will be zero
       pixel_chloro=0;
     else//copy chloro values to filtered
       band_chloro-
>RasterIO(GF_Read,i,j,1,1,&pixel_chloro,1,1,GDT_Float32,0,0);//rea
d source (i,j) value
     band_filtered-
>RasterIO(GF_Write,i,j,1,1,&pixel_chloro,1,1,GDT_Float32,0,0);//wr
ite pixel value to resulting band
    }
  }
```

Εικόνα 63: Λίμνη Κάρλα 2013-09-28 (pansharpened) NDWI



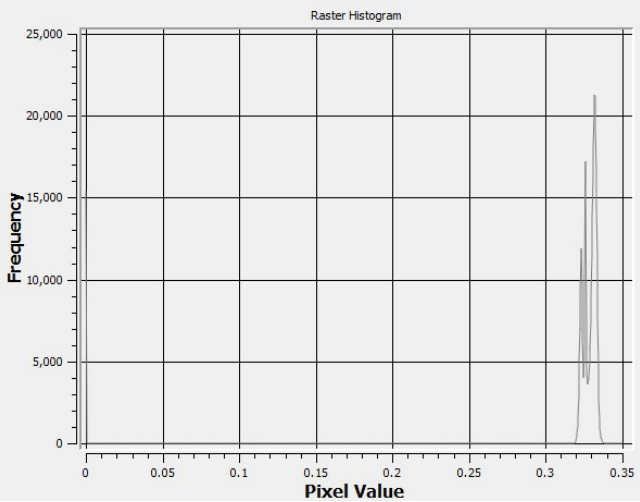
Εικόνα 64: Λίμνη Κάρλα 2013-09-28 (pansharpened) CHLORO



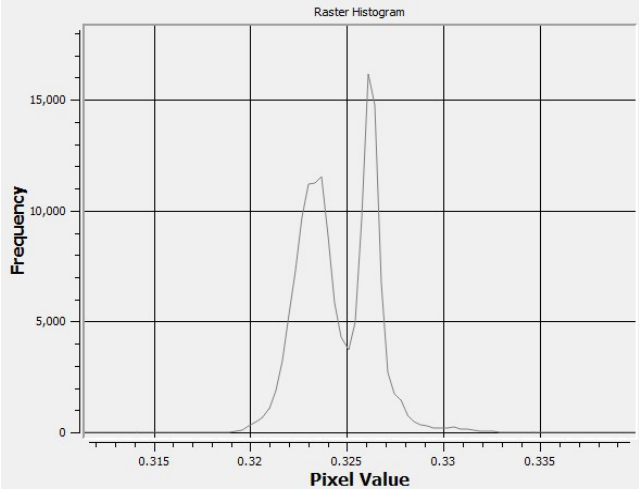
Εικόνα 65: Λίμνη Κάρλα 2013-09-28 (pansharpened)
CHLORO φιλτραρισμένη με βάση το NDWI

Εικόνα 66: Λίμνη Κάρλα 2013-09-28 (pansharpened)
CHLORO φιλτραρισμένη βάση NDWI MIN:0.31

Καθώς το αποτέλεσμα έχει αποχρώσεις με διαφορές σε επίπεδο τιμής 0.01, (κάτω αριστερά ιστόγραμμα), για τη δεξιά απεικόνιση έγινε μια αλλαγή στο ελάχιστο (κατώτερη τιμή μαύρου) στο 0.31 ώστε να γίνει πιο κατανοητή η εικόνα-αποτέλεσμα του φιλτραρίσματος.



Εικόνα 67: Λίμνη Κάρλα 2013-09-28 (pansharpened)
CHLORO φιλτραρισμένη με βάση το NDWI
Ιστόγραμμα



Εικόνα 68: Λίμνη Κάρλα 2013-09-28 (pansharpened)
CHLORO φιλτραρισμένη με βάση το NDWI
MIN:0.31
Ιστόγραμμα

2.6.2 Ομαδοποίηση Φιλτραρισμένης Χλωροφύλλης

Για την ομαδοποίηση τιμών Χλωροφύλλης του αποτελέσματος του NDWI φιλτραρίσματος χρησιμοποιήθηκαν οι παρακάτω κλάσεις[7]:

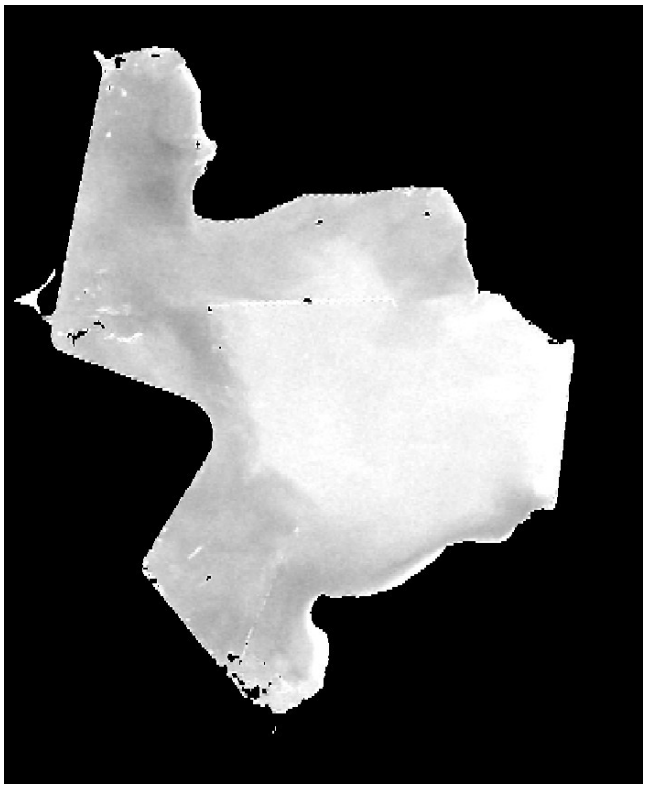
Αρχικές (Raw)	
Αριθμός Κλάσης	Διάστημα
1	[0.315 , 0.317)
2	[0.317 , 0.32)
3	[0.32 , 0.323)
4	[0 .323 , 0.335)
5	[0.335 , 0.34)

Πίνακας 3: Κλάσεις δείκτη χλωροφύλλης

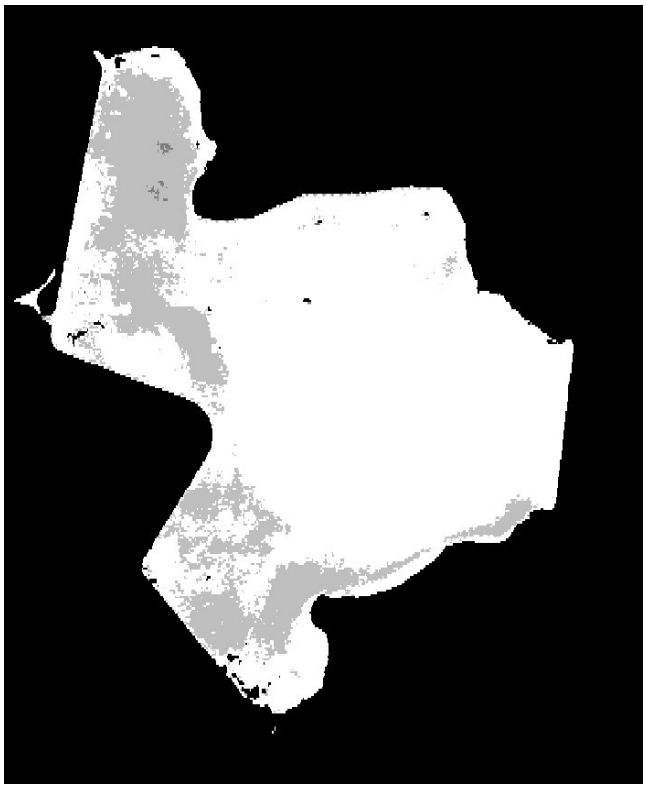
Οι τιμές κάτω του 0.315 συμπεριλήφθηκαν στην κατηγορία 0, ενώ ενώ του 0.34 στην κατηγορία 6. Αλλά δεν λαμβάνονται υπόψη για τον εντοπισμό χλωροφύλλης.

Το αποτέλεσμα, όπως και στην ομαδοποίηση του NDVI, είναι μια εικόνα της οποία τα pixel έχουν μόνο τις ακέραιες τιμές 0-6, ανάλογα με την κλάση στην οποία ανήκουν.

Δεν υλοποιήθηκε ομαδοποίηση Χλωροφύλλης για διορθωμένες εικόνες, αντίθετα με την περίπτωση του NDVI.



Εικόνα 69: Λίμνη Κάρλα 2013-09-28 (pansharpened)
CHLORO φιλτραρισμένη με βάση το NDWI
MIN:0.31



Εικόνα 70: Λίμνη Κάρλα 2013-09-28 (pansharpened)
CHLORO φιλτραρισμένη με βάση το NDWI
Ομαδοποιημένη

Περιγραφή κώδικα Ομαδοποίησης NDWI Φιλτραρισμένης Χλωροφύλλης:

Η ομαδοποίηση γίνεται στη συνάρτηση **ndwichlorofiltered_classification(char *fname_src)** (Παράρτημα 1.12), όπου fname_src το όνομα της εικόνας με τη φιλτραρισμένη χλωροφύλλη της οποίας θα γίνει η ομαδοποίηση. Επιστρέφει δείκτη σε αρχείο τύπου unsigned int 16 που περιλαμβάνει pixel με τιμές από 0 μέχρι 6.

Για την ομαδοποίηση γίνεται απλός έλεγχος ανά pixel για το διάστημα στο οποίο ανήκει με βάση τον παραπάνω πίνακα.

Σχετικός κώδικας:

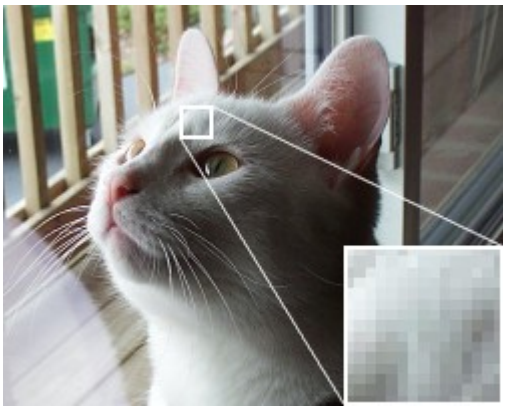
```
//apply chloro classification
  for(int i=0;i<dataset_src->GetRasterXSize();i++){//for all band
width
    for(int j=0;j<dataset_src->GetRasterYSize();j++){//for all band
height
      band_src->RasterIO(GF_Read,i,j,1,1,&pixel_src,1,1,band_src-
>GetRasterDataType(),0,0);//read source (i,j) value
      //classify
      if(pixel_src<0.315)
        pixel_dst=0;
      else if(pixel_src<0.317)
        pixel_dst=1;
      else if(pixel_src<0.32)
        pixel_dst=2;
      else if(pixel_src<0.323)
        pixel_dst=3;
      else if(pixel_src<0.335)
        pixel_dst=4;
      else if(pixel_src<0.34)
        pixel_dst=5;
      else
        pixel_dst=6;
      band_class-
>RasterIO(GF_Write,i,j,1,1,&pixel_dst,1,1,band_class-
>GetRasterDataType(),0,0);//write pixel value to resulting band
    }
  }
```

2.7 Αλγόριθμος Διανυσματοποίησης

2.7.1 Διανυσματοποίηση Εικόνας

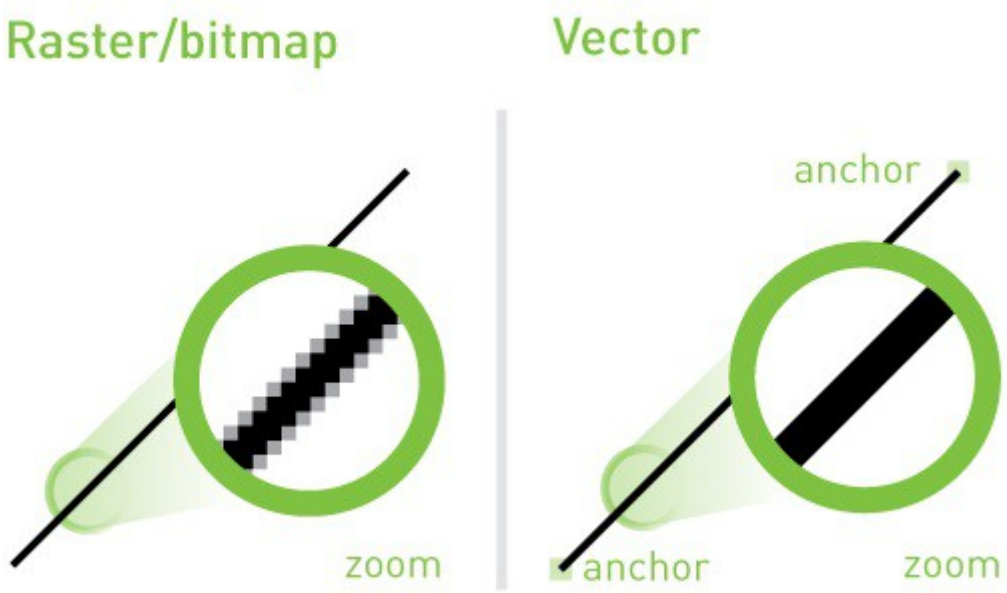
Διανυσματοποίηση (vectorization) ή ιχνογράφηση εικόνας (image tracing) είναι η μετατροπή των γραφικών πλαισίου (raster) σε διανυσματικά (vector).

Οι raster εικόνες είναι ουσιαστικά μια συλλογή από σημεία με τιμές χρώματος. Όταν μεγεθυνθούν όμως, παρατηρείται μια παραμόρφωση που ονομάζεται pixelation, όπου το κάθε σημείο φαίνεται να έχει τετραγωνισμένες άκρες με αποτέλεσμα η εικόνα να αποκτά οδοντωτές γραμμές αντί για συνεχόμενες.



Εικόνα 71: Παράδειγμα pixelation

Αυτό διορθώνεται με τη μετατροπή σε vector εικόνα. Αντί για συλλογή από σημεία, τα γραφικά δημιουργούνται σαν πολύγωνα ίδιου χρώματος και οι άκρες τους σαν συνεχόμενες γραμμές μεταξύ άκρων (anchors-άγκυρες). Με αυτό τον τρόπο, όση μεγέθυνση και να γίνει, η εικόνα δεν παραμορφώνεται/πιξελιάζει και οι άκρες διατηρούν τη συνεχή ροή τους.



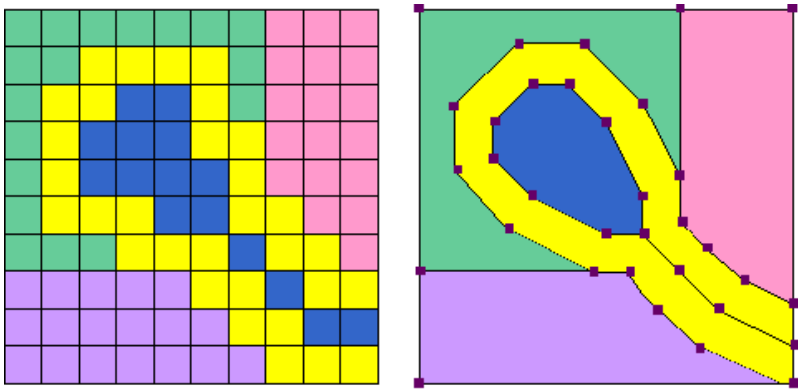
Εικόνα 72: Σύγκριση raster και vector γραφικών

Εικόνες με συνεχείς χρωματικούς τόνους δεν διανυσματοποιούνται σωστά, αντίθετα με σύνθετες εικόνες όπως χάρτες, σκίτσα και λογότυπα που αποτελούνται κυρίως από γεωμετρικά σχήματα και καμπύλες.

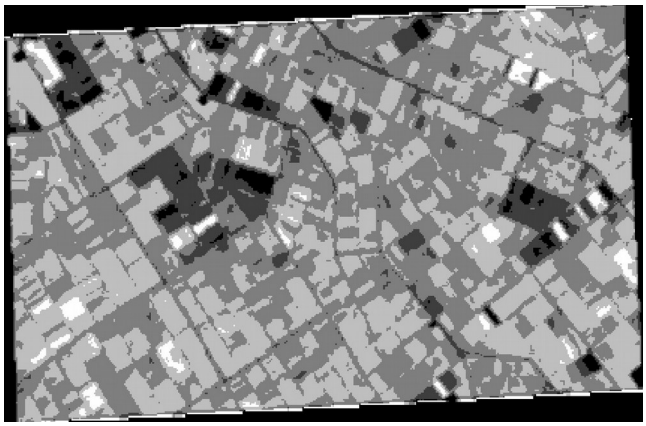
Εικόνα 73: Παράδειγμα διανυσματοποίησης φωτογραφίας με συνεχείς τόνους

Εικόνα 74: Παράδειγμα διανυσματοποίησης σκίτσου με ασυνεχή χρώματα

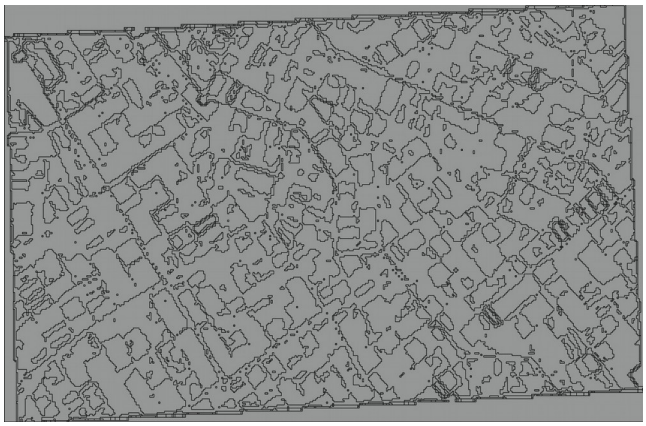
Σαν είσοδος στη διανυσματοποίηση μπορούν να είναι raster είδη αρχείων όπως bmp, png και tiff, ενώ σαν έξοδος είναι ήδη vector αρχείων όπως svg, kml,geoJSON και shapefile. Η διαδικασία που ακολουθείται είναι: αρχικά η εύρεση των άκρων, μετά η σύνδεσή τους με γραμμές και τέλος η αποθήκευσή των πολυγώνων που σχηματίζονται από αυτές.



Εικόνα 75: Σύγκριση raster και λεπτομερούς vector



Εικόνα 76: Αξιός2 (αρχική) 2013-09-12 Ομαδοποιημένη



Εικόνα 77: Αξιός2 (αρχική) 2013-09-12 Ομαδοποιημένη Διανυσματοποιημένη

Περιγραφή Κώδικα Διανυσματοποίησης:

Η διανυσματοποίηση γίνεται στη συνάρτηση single**_polygonize(char *fname_src)** (Παράρτημα 1.13), όπου fname_src το όνομα της εικόνας που θέλουμε να μετατρέψουμε σε vector.

Για την υλοποίηση της διανυσματοποίησης στα πλαίσια της εργασίας χρησιμοποιήθηκαν σαν είσοδος γεωαναφερμένες εικόνες tiff και σαν έξοδος γεωαναφερμένες διανυσματικές geoJSON.

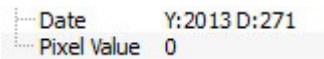
Αφού γίνει έλεγχος για το αν υπάρχει ήδη το αρχείο, γίνεται μεταφορά της γεωαναφοράς από το αρχείο tiff στο geoJSON μέσω της συνάρτησης **OSRNewSpatialReference** της GDAL. Ύστερα ανά ζώνη του tiff (τα συγκεκριμένα tiff με τις ομαδοποιήσεις αποτελούνται μόνο από μία) προστίθενται δυο νέα πεδία "Date" και "Pixel Value" τύπου string με τη χρήση της **OGR_L_CreateField** και στη συνέχεια δημιουργούνται τα πολύγωνα. Το κάθε πολύγωνο περιλαμβάνει όλη την περιοχή του tiff όπου υπήρχαν γειτονικά pixel με ίδια τιμή. Αυτό υλοποιείται με τη συνάρτηση **GDALPolygonize**, στην οποία τίθεται στην παράμετρο 3 (iPixValField) ως πεδίο τιμής pixel το πεδίο "Pixel Value", που δημιουργήθηκε αμέσως πριν, το οποίο είναι το πεδίο υπ' αριθμόν 1 (πεδίο 0 είναι το Date), οπότε σαν παράμετρος μπαίνει ο ακέραιος '1'.

Σχετικός κώδικας:

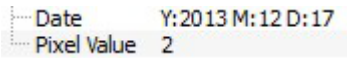
```
OGRSpatialReferenceH
hSpatialRef=OSRNewSpatialReference(dataset_src-
>GetProjectionRef());
OGRLayerH hLayer;
  for(int b=1;b<=dataset_src->GetRasterCount();b++){
poBand = dataset_src->GetRasterBand(b);
    hLayer =
GDALDatasetCreateLayer(hDS,lname,hSpatialRef,wkbPolygon,NULL)

OGR_L_CreateField(hLayer,OGR_Fld_Create("Date",OFTString),FALSE);/
/id=0 for geojson, 2 for kml
    OGR_L_CreateField(hLayer,OGR_Fld_Create("Pixel
Value",OFTString),FALSE);//id=1 for geojson, 3 for kml
    GDALPolygonize (poBand, NULL, hLayer, 1, NULL, NULL,
NULL)==CE_Failure)
}
```

Εφόσον στη συγκεκριμένη υλοποίηση το πηγαίο αρχείο είναι μια ομαδοποίηση (σε 7 κατηγορίες 0-6) εικόνας, με δείκτες NDVI ή NDWI/χλωροφύλλης αντίστοιχα, το αποτέλεσμα της διανυσματοποίησης είναι μια συλλογή από γεωαναφερμένα πολύγωνα με καινούριο πεδίο "Pixel Value" να περιέχει τον αριθμό κατηγορίας και πεδίο "Date" που προς το παρόν είναι κενό.

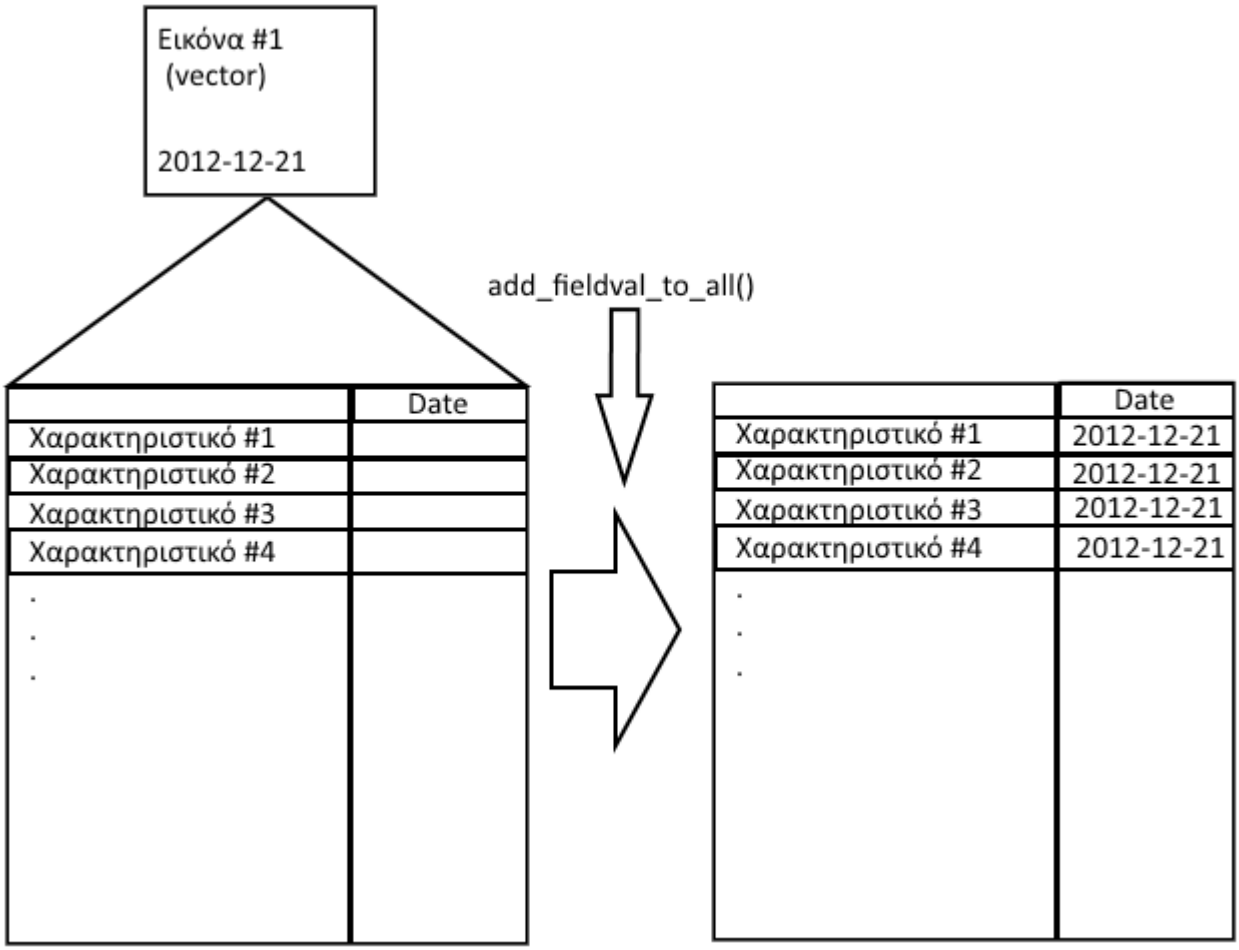
Στη συνέχεια για να συμπληρωθεί στο πεδίο "Date" κάθε πολυγώνου η ημερομηνία κατά την οποία αποτυπώθηκε η συγκεκριμένη φωτογραφία, καλείται η συνάρτηση add**_fieldval**_to**_all(char *fname**_cur**, const char ***field_name**, const **char *value)** (Παράρτημα 1.14), όπου fname_cur το όνομα της vector εικόνας, field_name το πεδίο που πρέπει να συμπληρωθεί και value η τιμή που θα αποδοθεί. Η ημερομηνία εξάγεται από τον τίτλο της εικόνας με διαφορετικό τρόπο ανάλογα αν είναι αρχική (raw) ή διορθωμένη (reflectance). Για τις αρχικές χρησιμοποιείται η συνάρτηση get_date_from**_fname**_raw (Παράρτημα 1.16), η οποία δέχεται σαν παράμετρο το όνομα του αρχείου πχ. *'LC81840322013255LGN00' και εξάγει το χρόνο '2013' και τη μέρα '255' τα οποία επιστρέφει σε μορφή string με τη μορφή 'Y:2013 D:255', ενώ για τις διορθωμένες η* get_date_from**_fname_refl** (Παράρτημα 1.16), η οποία αντίστοιχα δέχεται το όνομα με τη μορφή '2013-09-13' και επιστρέφει την ημερομηνία στη μορφή 'Y:2013 M:09 D:13'.



Εικόνα 78: Παράδειγμα- νέα πεδία vector αρχικής



Εικόνα 79: Παράδειγμα- νέα πεδία vector διορθωμένης



Εικόνα 80: Σχηματική αναπαράσταση συνάρτησης add_fieldval_to_all()

Σχετικός κώδικας:

(add_fieldval_to_all)

```
    //scan all features
    for(int i=0;i<OGR_L_GetFeatureCount(srcLayer,TRUE);i++){
      //get feature from source layer
      srcFeat=OGR_L_GetFeature(srcLayer,i);
      //clone feature
      dstFeat=OGR_F_Clone(srcFeat);
      //find field index for change
      field_date_index=OGR_F_GetFieldIndex (dstFeat,field_name);
      //change field's value
      OGR_F_SetFieldString(dstFeat,field_date_index,value);
      //add changed feature to new layer
      OGR_L_CreateFeature(dstLayer,dstFeat);
      }
      //destroy feature structs
      OGR_F_Destroy(srcFeat);
      OGR_F_Destroy(dstFeat);
    }
```

(get_date_from_fname_raw)

```
  //TEMPLATE: LC81840322013255LGN00
  date[0]='Y';date[1]=':';
  for(int i=year_i;i<year_i+year_len;i++)date[2+i-
year_i]=*(fname_cur+i);
  date[2+year_len]='
';date[2+year_len+1]='D';date[2+year_len+2]=':';
  for(int i=day_i;i<day_i+day_len;i++)date[2+year_len+3+i-
day_i]=*(fname_cur+i);
  date[2+year_len+3+day_len]='\0';
  //TEMPLATE: Y:2013 D:244
```

(get_date_from_fname_refl)

```
  //TEMPLATE: 2013-09-12
  int year_i=0,year_len=4,month_i=5,month_len=2,day_i=8,day_len=2;
  date[0]='Y';date[1]=':';
  for(int i=year_i;i<year_i+year_len;i++)date[2+i-
year_i]=*(fname_cur+i);
  date[2+year_len]='
';date[2+year_len+1]='M';date[2+year_len+2]=':';
  for(int i=month_i;i<month_i+month_len;i++)date[2+year_len+3+i-
month_i]=*(fname_cur+i);
  date[2+year_len+3+month_len]='
';date[2+year_len+3+month_len+1]='D';date[2+year_len+3+month_len+2
]=':';
  for(int i=day_i;i<day_i+day_len;i+
+)date[2+year_len+3+month_len+3+i-day_i]=*(fname_cur+i);
  date[2+year_len+3+month_len+3+day_len]='\0';
  //TEMPLATE: Y:2013 M:09 D:12
```

2.7.2 Σύμπτυξη Πολλαπλών Διανυσματικών Εικόνων

Η συγκεκριμένη λειτουργία προστέθηκε για να παρακάμψει τους περιορισμούς της οπτικοποίησης μέσω CartoDB που ακολουθεί σε επόμενο κεφάλαιο. Το CartoDB επιτρέπει την προσθήκη μέχρι 4 επιπέδων στη δωρεάν έκδοση και μέχρι το πολύ 10 στην ανώτατη επί-πληρωμή. Αυτό δεν θα επέτρεπε την οπτικοποίηση πάνω από 10 εικόνων geoJSON στην καλύτερη περίπτωση. Με τη σύμπτυξη πολλαπλών διανυσματικών εικόνων σε μια, μπορεί να γίνει ταυτόχρονη οπτικοποίηση, σε ένα επίπεδο, όσων εικόνων μπορούν να χωρέσουν στα 250MB που διαθέτει το CartoDB ανά λογαριασμό χρήστη. Ο διαχωρισμός τους ξανά στις πολλαπλές μπορεί να πραγματοποιηθεί κατά την οπτικοποίηση με τη σωστή χρήση SQL ερωτημάτων στη βάση δεδομένων του λογαριασμού CartoDB η οποία περιέχει το μοναδικό (τελικά) επίπεδο.

Η λογική της σύμπτυξης είναι απλή: δημιουργείται νέο αρχείο προορισμού geoJSON και για κάθε υπάρχον αρχείο εικόνας (geoJSON που δημιουργήθηκαν κατά τη διανυσματοποίηση) γίνεται αντιγραφή/μεταφορά/προσθήκη όλων των χαρακτηριστικών-πολυγώνων τους μαζί με όλα τα πεδία τους στο αρχείο προορισμού.

Δεδομένου ότι όλα τα χαρακτηριστικά-πολύγωνα του κάθε geoJSON έχουν κοινή τιμή στο πεδίο ημερομηνία, μπορούν να διαχωριστούν με βάση αυτή όταν συμπτυχθούν σε ένα αρχείο, αφού το κάθε αρχικό αρχείο περιέχει εικόνα από διαφορετική μέρα.

Με άλλα λόγια το σύνολο των χαρακτηριστικών του σύνθετου αρχείου που έχουν μια συγκεκριμένη ημερομηνία, αποτελούν το σύνολο των χαρακτηριστικών της ξεχωριστής αρχικής εικόνας της ίδιας ημερομηνίας.

Περιγραφή κώδικα Σύμπτυξης Πολλαπλών Διανυσματικών Εικόνων σε Μια:

Η σύμπτυξη πραγματοποιείται στη συνάρτηση **appendage(char *path_dir)** (Παράρτημα 1.15), όπου path_dir η διαδρομή για το φάκελο όπου βρίσκονται οι πολλαπλές geoJSON εικόνες με τη βοήθεια της append**_geojson(OGRDataSourceH srcDs, OGRDataSourceH curDs)** (Παράρτημα 1.15), όπου srcDS και curDs δείκτες σε μια αρχική και στην εικόνα προορισμού αντίστοιχα.

Η appendage() δημιουργεί, στο φάκελο που βρίσκεται η εφαρμογή, την εικόνα προορισμού με τίτλο βασισμένο στο όνομα του φακέλου και στην ημερομηνία που τρέχει η εφαρμογή και μετά καλεί επαναληπτικά την append_geojson() για κάθε μια από τις εικόνες εντός του φακέλου αφού τις ανοίξει με GDALOpenEx() (με flags GDAL_OF_VECTOR|GDAL_OF_READONLY), όπου διαβάζει ένα ένα τα πολύγωνά τους και τα προσθέτει στην εικόνα προορισμού.

Επίσης γίνεται έλεγχος για αριθμό layer και αναλόγως, το κάθε πολύγωνο γράφεται στο αντίστοιχο layer της εικόνας προορισμού, αλλά αφού οι εικόνες που γίνονται append έχουν μόνο ένα (προέρχονται από ομαδοποιήσεις ndvi και ndwi) εγγραφές γίνονται μόνο σε ένα layer.

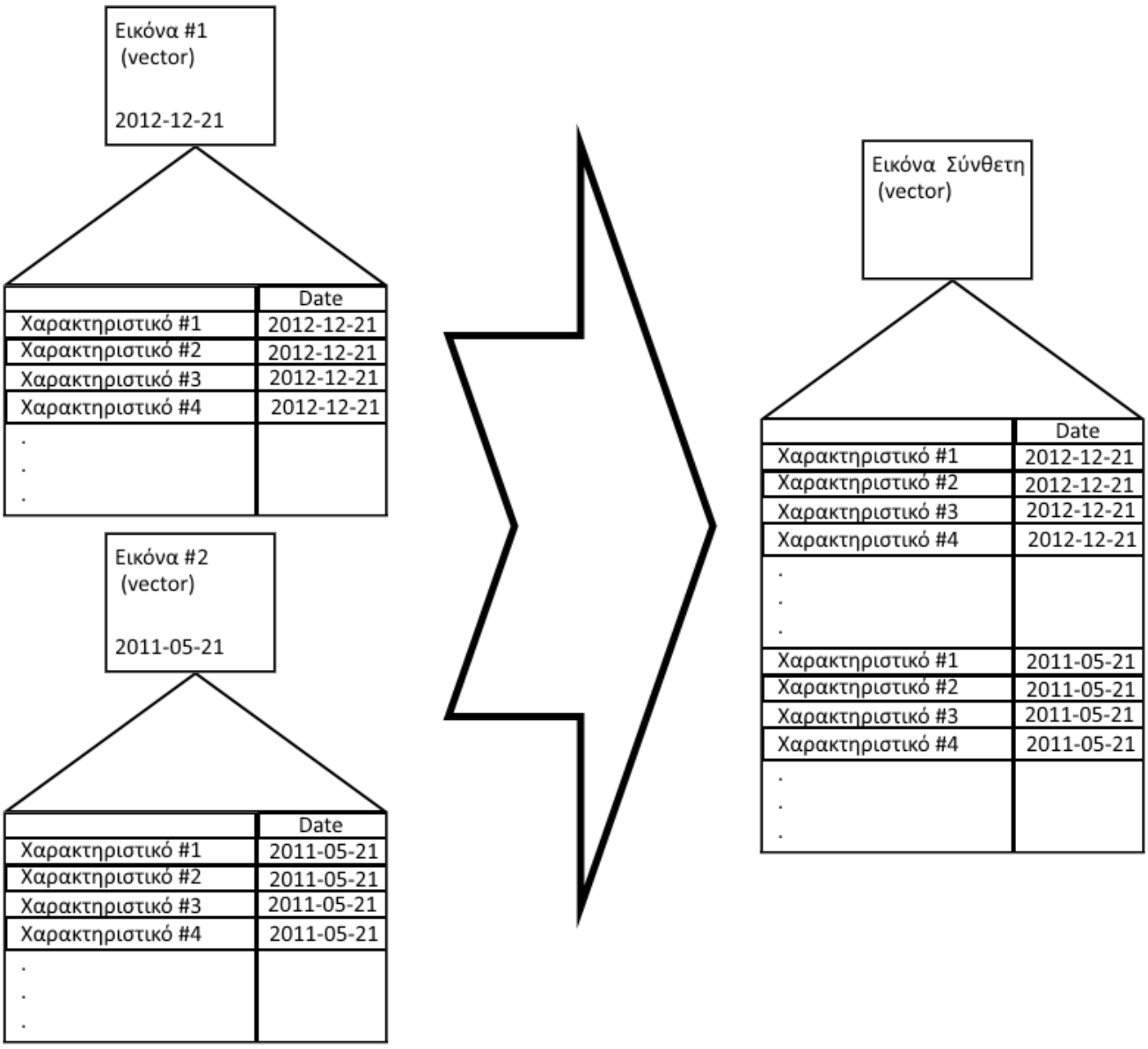
Σχετικός κώδικας:

(appendage)

```
  //create destination dataset
  dstDs = (OGRDataSourceH *)
GDALCreate(poDriver,fname_dst,0,0,0,GDT_Unknown,NULL)
  if ((dir = opendir (path_dir)) != NULL) {//open dir
    chdir(path_dir);
    while ((ent = readdir (dir)) != NULL) {//for each entry
      if(strlen(ent->d_name)>vnum && strcmp(ent->d_name+strlen(ent-
>d_name)-vnum,vexte) == 0){//if has file extension ".geoJSON"
        curDs=(OGRDataSourceH *)GDALOpenEx(ent-
>d_name,GDAL_OF_VECTOR|GDAL_OF_READONLY,NULL,NULL,NULL);//open
vector file
        append_geojson(dstDs,curDs);//append it to source file
        GDALClose(curDs);
        }
      }
    }
```

(append_geojson)

```
for(int b=0;b<GDALDatasetGetLayerCount(curDs);b++){
  //get current layer for adding
  curLayer=GDALDatasetGetLayer(curDs,b);
  if(srcLayerCount==0){
    //new srcDs so copy entire current layer
    char lname[10];
    sprintf(lname,"Layer_%d",b);
    srcLayer=GDALDatasetCopyLayer(srcDs,curLayer,lname,NULL);
  }
  else{
    //get source layer
    srcLayer=GDALDatasetGetLayer(srcDs,b);
    //scan all current layer's features
    curFeat=OGR_L_GetFeature(curLayer,0);
    while((curFeat=OGR_L_GetNextFeature(curLayer))!=NULL){
      if(OGR_L_CreateFeature(srcLayer,curFeat)!=OGRERR_NONE){
        cout<<"COULDN'T SET CURRENT LAYER #"<<b<<"'S FEATURE"<<endl;
        OGR_F_Destroy(curFeat);
        return;
      }
      //destroy feature structs
      OGR_F_Destroy(curFeat);
    }
    OGR_L_SyncToDisk(srcLayer);
}
```



Εικόνα 81: Σχηματική αναπαράσταση συνάρτησης appendage()

2.8 Οπτικοποίηση Διανυσματικών Αρχείων (CartoDB.js)

2.8.1 Εισαγωγή

Γεννημένο από μια ανεξάρτητη ανερχόμενη ισπανική εταιρία, το CartoDB είναι μια μηχανή οπτικοποίησης χαρτών σε προγράμματα περιήγησης (πχ. Mozilla Firefox, Google Chrome) η οποία μπορεί να δημιουργήσει ή να φορτώσει ήδη υπάρχοντα γεωαναφερμένα δεδομένα (τύποι: σημεία, γραμμές και πολύγωνα, δηλαδή διανυσματικά) και αφού τα αναλύσει έξυπνα, να τα οπτικοποιήσει με ποικίλους τρόπους.

Το CartoDB είναι φτιαγμένο ώστε να δουλεύει με vector δεδομένα και πιο συγκεκριμένα είναι συμβατό με τους τύπους αρχείων CSV, Shapefile, OpenStreeMap, KML, GeoJSON και SVG. Επίσης, στη δωρεάν έκδοσή του επιτρέπει την ταυτόχρονη παρουσία μόνο μέχρι τεσσάρων διαφορετικών επιπέδων (layers) και συνολικά 250 MB δεδομένων ανά λογαριασμό. Στην ακριβότερη επί-πληρωμή έκδοση για εταιρίες, τα επιτρεπόμενα επίπεδα φτάνουν τα δέκα και ο διαθέσιμος χώρος μπορεί αν είναι πάνω από 5 GB.

Για τους παραπάνω λόγους, προτιμήθηκε ο τύπος αρχείου GeoJSON για τη διανυσματοποίηση των ομαδοποιημένων εικόνων στα πλαίσια της εργασίας, καθώς και η σύμπτυξη των διαφορετικών vector εικόνων σε μια, αφού μπορούν να είναι αρκετά περισσότερα ακόμα και από δέκα και θα ήταν αδύνατο να βρίσκονται η κάθε μια σε διαφορετικό επίπεδο.

Αφού φορτωθούν τα επιθυμητά δεδομένα online σαν σύνολα δεδομένων (datasets) σε λογαριασμό CartoDB (λογική νέφους-μπορεί να αποκτηθεί πρόσβαση σε αυτά από οπουδήποτε, αρκεί να υπάρχει σύνδεση με το Διαδίκτυο), μπορούν να συνδεθούν με έναν ή παραπάνω χάρτες εντός του λογαριασμού (ή εκτός εφόσον είναι γνωστό το όνομα του λογαριασμού και των dataset) και για την οπτικοποίησή τους χρησιμοποιούνται συνθήκες επιλογής τους με τη χρήση της γλώσσας επερωτημάτων SQL, ενώ για τη διακόσμησή τους (χρωματισμός, πλαίσια κτλ) γίνεται χρήση του κώδικα scripting CSS με όλες τις δυνατότητες που αυτός προσφέρει (πχ. διαχωρισμός διακοσμήσεων με βάση class, id κτλ).

Όταν ζητηθεί αν οπτικοποιηθούν τα επιθυμητά δεδομένα, το CartoDB ελέγχει ποια ζητήθηκαν από τα επερωτήματα SQL, τα επιλέγει, τα διακοσμεί με βάση το δοσμένο κώδικα CSS και τα συνδέει σε έναν εικονικό χάρτη. Αυτό τον χάρτη τον μετατρέπει σε εικονικό raster, τον χωρίζει σε πλακάκια (tiles) και τέλος στέλνει αυτά τα πλακάκια στον χάρτη (εντός ή εκτός του λογαριασμού που περιέχει τα δεδομένα) που έστειλε το επερώτημα.

2.8.2 Ερωτήματα με χρήση της γλώσσας SQL

Το CartoDB, πέρα από τις online δυνατότητές του, προσφέρει ένα API μέσω javascript για customized οπτικοποιήσεις. Με τη χρήση αυτού του API υλοποιήθηκε οπτικοποίηση για τα αποτελέσματα των παραπάνω διαδικασιών (Παράρτημα 2).

Αρχικά, ανέβηκαν τα τελικά συγκεντρωτικά geoJSON (που περιλαμβάνουν ndvi ή ndwi πολλών ημερομηνιών για συγκεκριμένη περιοχή) σε λογαριασμό του CartoDB. Εκεί αποθηκεύονται σε κοινή βάση δεδομένων σαν ξεχωριστά datasets/tables και με χρήση sql μπορούν να γίνουν επιθυμητά ερωτήματα.

Τα dataset μπορούν να έχουν μορφή σημείων, γραμμών, πολυγώνων και γενικά όλων των γνωστών γεωμετριών που μπορούν να αναπαρασταθούν με διανύσματα (vectors). Τα συγκεκριμένα geoJSON που παράχθηκαν για την εργασία είναι πολύγωνα και το κάθε πολύγωνο αποτελεί entry στο dataset. Δηλαδή με τα κατάλληλα sql ερωτήματα μπορεί να ζητηθεί από τη βάση ένας αριθμός από πολύγωνα.

Εφόσον χρειάζεται να παρουσιαστούν μόνο τα επίπεδα βλάστησης και υγρασίας, αντίστοιχα για ndvi και ndwi, πρέπει να ζητηθούν όλα τα entries εκτός από αυτά με pixel_value 0 και 6. Συνεπώς το βασικό ερώτημα προς τη βάση θα είναι **"SELECT * FROM "+dataset[i]+" WHERE "+pixel_value_col+" !='0' AND "+pixel_value_col+"!='6' "**, όπου dataset[i] το αντίστοιχο dataset και pixel_value_col το όνομα της στήλης με την τιμή pixel/αριθμό επιπέδου (ορίστηκε ως “Pixel Value” στο βήμα της διανυσματοποίησης 2.7). Αυτό θα επιστρέψει όλα τα πολύγωνα του πίνακα dataset[i] που ανήκουν στις κατηγορίες από 1 έως 5.

Για να περιοριστούν και στις αντίστοιχες ημερομηνίες κάθε φορά, πρέπει να προστεθεί και η συνθήκη “ AND Date= X”, όπου X η ημερομηνία προς απεικόνιση.

Όμως το κάθε geoJSON έχει διαφορετικές ημερομηνίες σε όνομα και αριθμό. Οπότε θα πρέπει πρώτα να μαθευτεί ο αριθμός και το όνομα όλων των διαφορετικών ημερομηνιών και μετά να ζητηθούν τα αντίστοιχα πολύγωνα. Αυτό επιτυγχάνεται με το sql ερώτημα **"SELECT DISTINCT "+date_col+" FROM "+dataset[datIn]+" ORDER BY "+date_col+" ASC"**, όπου date_col το όνομα της στήλης με την ημερομηνία του πολύγωνου (ορίστηκε ως “Date” στη διανυσματοποίηση) και dataset[datIn] το dataset που ζητείται. Αυτό επιστρέφει όλες τις διαφορετικές ημερομηνίες που υπάρχουν στο dataset[datIn] σε αύξουσα σειρά.

Μετά, για κάθε διαφορετική ημερομηνία μπορούν να ζητηθούν όλα τα πολύγωνα με τις τιμές pixel που χρειάζονται (δηλαδή εκτός 0 και 6).

Τα ερωτήματα γίνονται στη βάση με τη βοήθεια του CartoDB API που δίνει τις εξής λειτουργίες:

την κλάση **cartodb.SQL(account_name)** με παράμετρο το όνομα του λογαριασμού και συνεπώς της βάσης η οποία επικοινωνεί με τη βάση και εκτελεί ερωτήματα μέσω της συνάρτησης-μέλους της **execute(sql_quarry)**,

και τη συνάρτηση-μέλος **setSQL(sql_quarry)** της κλάσης layer, η οποία κάνει ερωτήματα στη βάση και εμφανίζει τα πολύγωνα που ζητούνται στο αντίστοιχο layer.

2.8.3 Παράμετροι Οπτικοποίησης (CSS & HTML)

Για το χρωματισμό των πολυγώνων χρησιμοποιήθηκαν τα παρακάτω χρώματα (σε rgb)[6][7]:

Τιμή Pixel	Χρώμα	
	NDVI	NDWI
1	rgb(255,53,0)	rgb(255,0,0)
2	rgb(255,102,0)	rgb(255,255,0)
3	rgb(255,154,0)	rgb(0,255,0)
4	rgb(255,220,0)	rgb(0, 153, 0)
5	rgb(0,255,102)	rgb(0, 102, 0)

Πίνακας 4: Χρωματισμός πολυγώνων με βάση κατηγορίες κλάσεων

Οι τιμές των χρωμάτων αποθηκεύτηκαν στους πίνακες css_ndvi και css_ndwi.

Η δομή html της σελίδας είναι απλή: υπάρχει το div **#map** στο οποίο θα εμφανίζεται ο χάρτης, το dropdown menu **#area_select** όπου θα εμφανίζονται τα ονόματα των περιοχών που μπορούν να εμφανιστούν και το div **#date_select** στο οποίο θα δημιουργηθεί η λίστα με τις διαφορετικές ημερομηνίες των geoJSON ανά περιοχή / συγκεντρωτική geoJSON.

2.8.4 Υλοποίηση Πλατφόρμας για τη Διαχείριση και Οπτικοποίηση (JavaScript)

Για την ομαλή λειτουργία του CartoDB API χρειάζεται μια βασική συνάρτηση η οποία καλείται όταν ανοίξει η σελίδα. Σε αυτή ορίζεται ο αριθμός των layer, μπορεί να της ανατεθεί χάρτης-υπόβαθρο (basemap) και ενεργοποιούνται όσα listeners χρειάζονται για άμεσες αλλαγές όταν γίνει κάποιο γεγονός στη σελίδα όπως το πάτημα ενός κουμπιού.

Στην οπτικοποίηση που υλοποιήθηκε για την ανάδειξη των geoJSON της εργασίας, η συνάρτηση αυτή ονομάστηκε **init()** και καλείται μέσω της λειτουργίας της javascript **window.onload**. Το πρώτο πράγμα που κάνει είναι να δημιουργήσει ένα αντικείμενο δομής χάρτη μέσω της συνάρτησης κατασκευής **L.Map(map_id , {center:my_center,zoom:myZoom})**, όπου map_id το id του div στο οποίο θα εμφανίζεται ο χάρτης μέσα στη σελίδα, myCenter ένα ζευγάρι συντεταγμένων latlong (πχ. 40.599297, 22.568977) το οποίο θα είναι το σημείο που θα εμφανιστεί ως κέντρο του παραθύρου όταν φορτώσει η σελίδα και myZoom η τιμή της αντίστοιχης μεγέθυνσης.

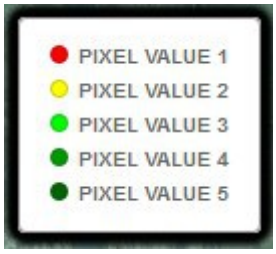
Έπειτα, ορίζεται το υπόβαθρο ως αντικείμενο δομής **L.tileLayer(basemap_url)** και προστίθεται στον χάρτη. Εδώ προτιμήθηκε ο χάρτης **World_Imagery της ESRI**, ο οποίος προσομοιάζει ανάγλυφη δορυφορική απεικόνιση.

Τα L υποδεικνύουν πως τα Map και tilelayer είναι δομές της βιβλιοθήκης leaflet.js, η οποία συνεργάζεται με το CartoDB για τη δημιουργία διαδραστικών χαρτών[8].

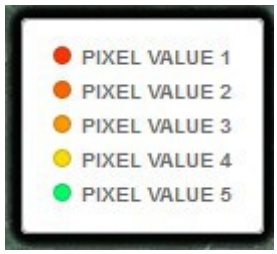
Εικόνα 82: Υπόβαθρο World_Imagery της ESRI

Στη συνέχεια δημιουργείται το υπόμνημα μέσω της συνάρτησης **my_legend()** με βάση τον αριθμό και τα χρώματα που έχουν οριστεί στη θέση datIn (id της εικόνας που οπτικοποιείται) του πίνακα css_val. Εφόσον παρουσιάζονται 5 διαφορετικές εικόνες διανυσματικών γραφικών που χρησιμοποιούν ndvi και ndwi, ο **css_val** ορίστηκε ως [css_ndvi, css_ndwi, css_ndwi, css_ndwi, css_ndwi] (οι υποπίνακες css_ndvi και css_ndwi παρουσιάζονται στο κεφάλαιο 2.8.3).

Αυτή η σειρά αντιστοιχεί στην αντίστοιχη του πίνακα **dataset** όπου ορίζονται τα ονόματα των πέντε dataset που έχουν ανέβει στο λογαριασμό CartoDB. Ο πίνακας dataset έχει οριστεί ως ["karla_raw", "axios1_raw", "axios2_raw", "axios1_refl", "axios2_refl"], όπου karla, axios1, axios2 οι τρεις περιοχές στις οποίες εφαρμόστηκαν ndvi και ndwi και raw/refl υποδεικνύουν αν οι πηγαίες εικόνες ήταν αρχικές (raw) ή διορθωμένες (reflectance). Με την αντίστοιχη σειρά ορίζεται ο πίνακας **area**_names με συμβατικά ονόματα των περιοχών ["Karla Raw", "Axios 1 Raw", "Axios 2 Raw", "Axios 1 Refl", "Axios 2 Refl"], ο πίνακας **myCenter** με τα σημεία-κέντρα των περιοχών αυτών [[39.48385,22.81487], [40.599297, 22.568977], [40.590846, 22.766869], [40.599297, 22.568977], [40.590846, 22.766869]] καθώς και ο **myZoom** με τις απαραίτητες μεγεθύνσεις για την καλύτερη οπτικοποίηση [13, 13, 14, 13, 14].



Εικόνα 84: Υπόμνημα NDVI



Εικόνα 83: Υπόμνημα NDWI

Ύστερα καλούνται οι συναρτήσεις

area_select(), η οποία δημιουργεί στο div #area_select ένα dropdown μενού με όλα τα ονόματα των περιοχών που μπορούν να απεικονιστούν αν επιλεγούν (τα διαβάζει από τον πίνακα area_names). Στην εργασία προκαθορίζεται σαν selected το πρώτο dataset (karla_raw).

date_select(), η οποία δημιουργεί τη λίστα με τις διαφορετικές ημερομηνίες κάνοντας το αντίστοιχο SQL ερώτημα, όπως υποδείχθηκε στο 2.8.2. Η κάθε ημερομηνία είναι κουμπί τύπου radio και όταν επιλεγεί καλεί με τη σειρά της τη συνάρτηση **check()** η οποία στέλνει το ανανεωμένο με τη νέα ημερομηνία SQL ερώτημα στη βάση και ζητά τα αντίστοιχα πολύγωνα.

Εικόνα 85: area_select

Εικόνα 86: date_select με δυο ημερομηνίες

Πίσω στην init(), αφού έχει δημιουργήσει τα menu #area_select και #date_select, δημιουργεί μια λίστα με layers πάνω στα οποία μπορούν να σταθούν τα πολύγωνα που επιστρέφονται από τα ερωτήματα. Εφόσον στην παρούσα εργασία δεν χρειάζεται να εμφανίζεται πάνω από ένα επίπεδο κάθε στιγμή (οι περιοχές είναι μακρυά μεταξύ τους για πιθανή ταυτόχρονη απεικόνιση) δημιουργείται μόνο ένα layer με τη συνάρτηση **cartodb.createLayer(map, layerSource)** η οποία παίρνει ως παραμέτρους το αντικείμενο χάρτη map και τα στοιχεία του dataset που είναι επιθυμητό να παρουσιαστεί πρώτο:

1. όνομα λογαριασμού CartoDB,
2. τύπος layer- εδώ 'cartodb',
3. λίστα με layer (εδώ μόνο ένα) που αναπαρίστανται από ζεύγη {sql_quarry, css_style}:

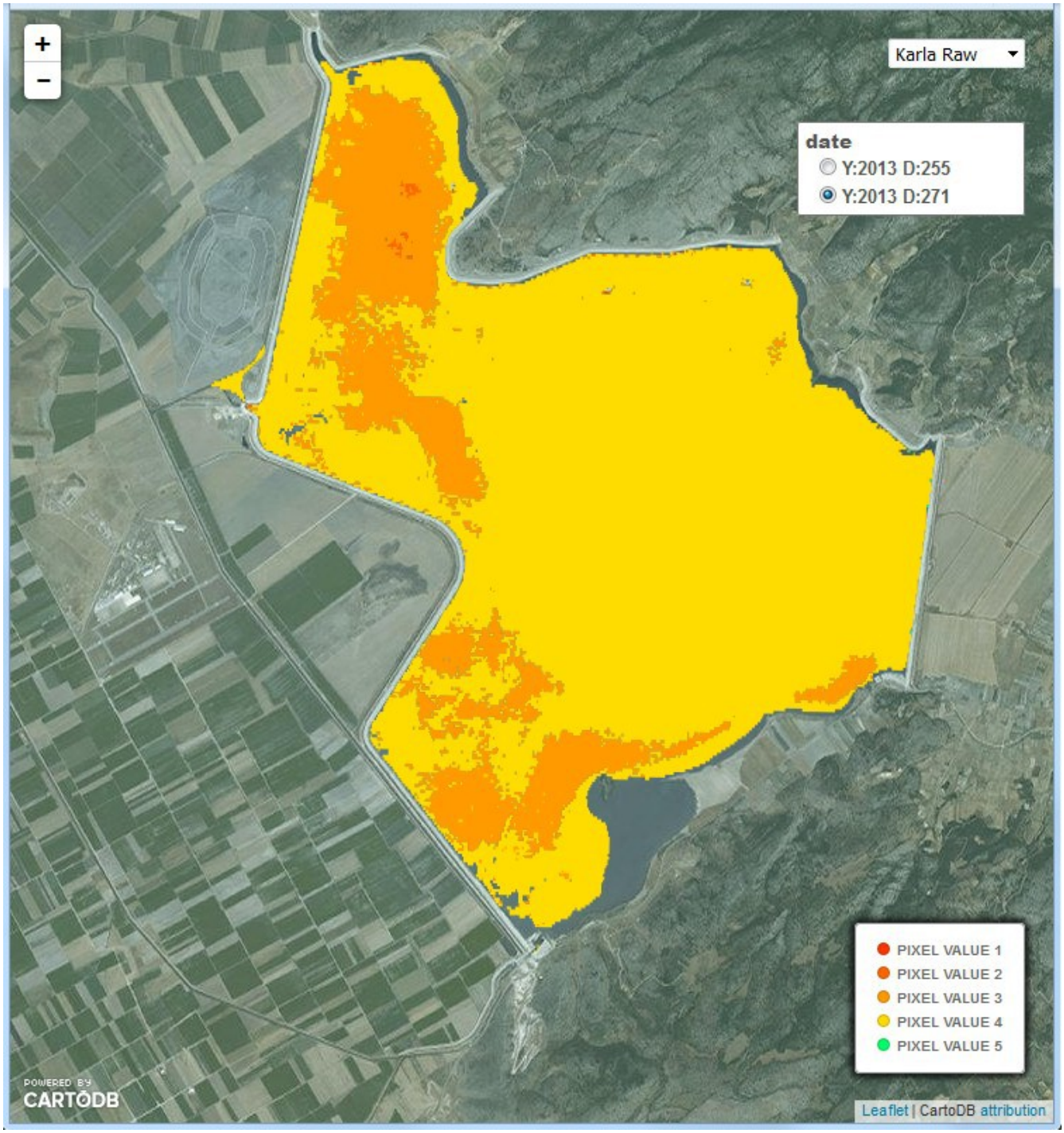
- ερώτημα SQL – το βασικό όπως παρουσιάστηκε στο 2.8.2 αλλά μαζί με τη συνθήκη μέρας:
  "SELECT * FROM "+dataset[datIn]+" WHERE "+pixel_value_col+" !='0' AND "
  +pixel_value_col+"!='6' AND "+date_col+"="
  +document.getElementById("myRadio"+i).value+"\'",
  όπου myRadio το id των μελών της λίστας #date_select και i ο αριθμός κουμπιού της ημερομηνίας που έχει επιλεχθεί και
- CSS style – χρωματισμοί πολυγώνων με βάση τον πίνακα css_val.

Αφού δημιουργηθεί το layer, προστίθεται στο χάρτη map και η global μεταβλητή **mySublayer** γίνεται δείκτης σε αυτό με τη συνάρτηση-μέλος **.getSubLayer(0)** της δομής layer για χρήση εκτός της init().

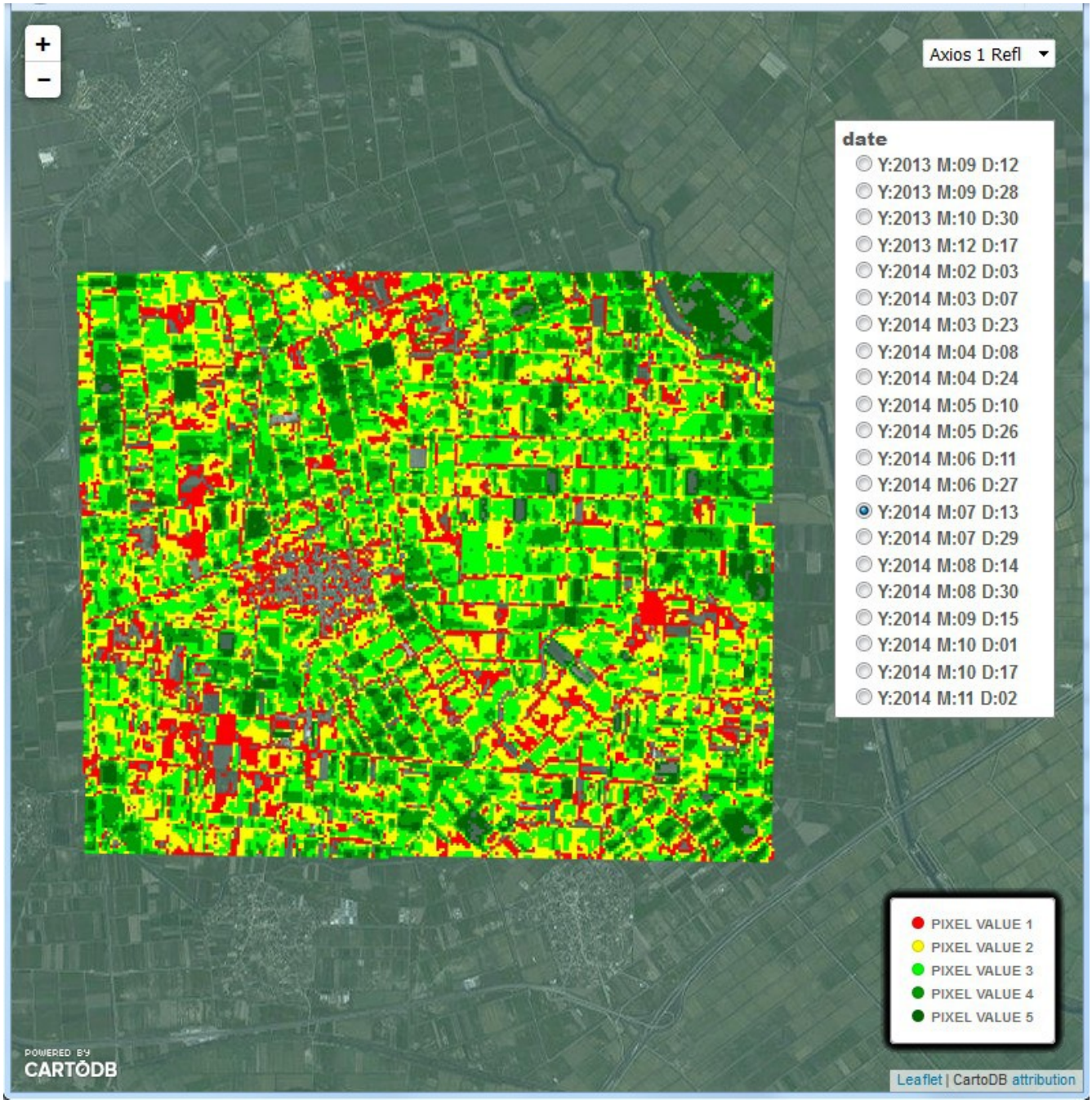
Τέλος, εντός της init() ενεργοποιούνται και δυο διαχειριστές γεγονότων (event handlers) με jQuery για τις περιπτώσεις αλλαγής περιοχής μέσω του #area_select και για αυτόματη αλλαγή ημερομηνιών από τη στιγμή που φορτώσει η σελίδα.

Ο πρώτος ενεργοποιείται με τη συνάρτηση .change() στο αντικείμενο #area_select και αφού βρει ποια περιοχή έχει επιλεχθεί, ενημερώνει το χάρτη με τη συνάρτηση μέλος **.setView(myCenter[i],myZoom[i])**, όπου i το index της περιοχής, επομένως το κέντρο της σελίδας μετατοπίζεται στην αντίστοιχη περιοχή και στέλνει το νέο ερώτημα στη βάση ώστε να εμφανιστούν τα ζητούμενα πολύγωνα στο layer. Η ανανέωση του layer επιτυγχάνεται με τη χρήση των συναρτήσεων **.setSQL(sql_quarry)**, .**setCartoCSS(myCSS(datIn))** της δομής layer που προσφέρει το CartoDB API, όπου datIn δείκτης στο χρωματισμό του αντίστοιχου dataset (datasetIndex). Αφού ενημερωθεί το layer, δημιουργείται το καινούριο υπόμνημα μέσω της my_legend() και ολοκληρώνεται η αποστολή του συγκεκριμένου διαχειριστή.

Ο δεύτερος όταν αναγνωρίσει πως ο αρχικός χάρτης έχει φορτωθεί στη σελίδα, θέτει ένα διάστημα (3.5 δευτερόλεπτα) και με το πέρας του καλεί επαναλαμβανόμενα την τοπική (στα πλαίσια του handler) συνάρτηση **next_date()**, στην οποία επιλέγεται η επόμενη ημερομηνία και ενημερώνεται το layer με τα αντίστοιχα πολύγωνα μέσω της check(). Η αυξητική μεταβλητή i σε συνδυασμό με τη χρήση του τελεστή υπολοίπου % αναλαμβάνουν την αλλαγή ημερομηνίας και την επανάληψη από το 0 όταν το i έχει φτάσει στην τελευταία ημερομηνία της λίστας.



Εικόνα 87: Στιγμιότυπο Οπτικοποίησης NDWI της Κάρλα (αρχική πηγή)



Εικόνα 88: Στιγμιότυπο Οπτικοποίησης NDVI του Αξιός1 (διορθωμένη πηγή)

3. Αποτελέσματα και Αξιολόγηση

Για τις συγκρίσεις και την οπτικοποίηση χρησιμοποιήθηκαν τρεις περιοχές βάση των εργασιών του Αθ. Κάρμα[6] και της Ι. Θεολόγου[7]. Το πρόγραμμα Matlab που χρησιμοποιήθηκε για τις συγκρίσεις του pansharpening είναι του Α. Βαϊόπουλου[4] με ελάχιστες μετατροπές (Παράρτημα 3).

Εισαγωγικά, ακολουθεί μια σύντομη παρουσίαση της υλοποίησης με C++ σαν συνολικό πρόγραμμα.

Όταν ξεκινήσει να τρέχει η main δίνει την επιλογή (Y/N: yes/no) να γίνει pansharpening ή όχι. Αν ναι, σημαίνει πως υπάρχουν δυο κατάλογοι, στο σημείο που βρίσκεται και το πρόγραμμα, με τα ονόματα 'panchromatic' και 'multispectral' οι οποίοι περιέχουν παγχρωματικές και πολυφασματικές εικόνες με τα ίδια ονόματα στη μορφή 'LC81840322013255LGN00.tif'. Το αποτέλεσμα του pansharpening, αποθηκεύεται στο φάκελο 'input' για περαιτέρω υπολογισμούς.
Αν όχι, τότε σημαίνει πως σε κατάλογο 'input' στο σημείο που βρίσκεται και το πρόγραμμα, υπάρχει διορθωμένη πολυφασματική ή pansharpened με όνομα πάλι στη μορφή 'LC81840322013255LGN00.tif.' Αμέσως μετά ερωτάται με το ερώτημα 'Raw?' για το αν είναι αρχική (Y), δηλαδή απλή πολυφασματική ή pansharpened, ή διορθωμένη (N).

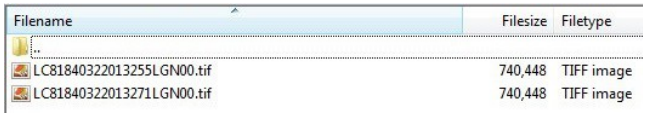
Ύστερα, ο χρήστης μπορεί να επιλέξει αν θέλει να εφαρμόσει το δείκτη NDVI ή το NDWI/CHLORO ώστε να χρησιμοποιηθεί η αντίστοιχη ομαδοποίηση, όπως παρουσιάζονται στα κεφάλαια 2.4.2 και 2.6.2.

Τέλος, μπορεί να επιλεχθεί αν μετά το πέρας όλων των υπολογισμών (pansharpening, ndvi/ndwi, ομαδοποίηση, διανυσματοποίηση και σύμπτυξη vector εικόνων) ο χρήστης θέλει να κρατήσει τα ενδιάμεσα αρχεία ή μόνο το τελικό αποτέλεσμα – ένα GeoJSON αρχείο με όλες τις διαφορετικές εικόνες που δόθηκαν σαν πηγή.

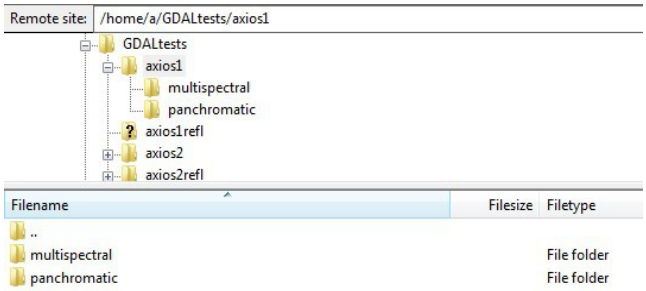
Τα τελικά αποτελέσματα (πολλαπλά GeoJSON ή το τελευταίο, ανάλογα αν επιλέχθηκε να διαγραφούν τα πολλά) τοποθετούνται στον κατάλογο 'output.'

```
a@a-M1110M:~/GDALtests/axios1refl$ ./out
Pansharpen? Y/N: n

You'd better have files (with name pattern 'LC81840322013255LGN00.tif') in the input directory for this
to work!


Raw? ('N' for reflectance) Y/N: n
Compute NDVI or NDWI? V/W: v
Delete middle files after exit? Y/N: n
```

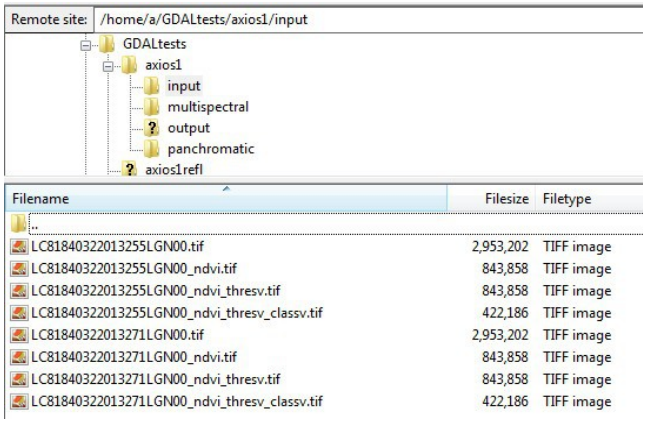
Εικόνα 89: Παράδειγμα τρεξίματος main για υπολογισμό NDVI διορθωμένων πολυφασματικών



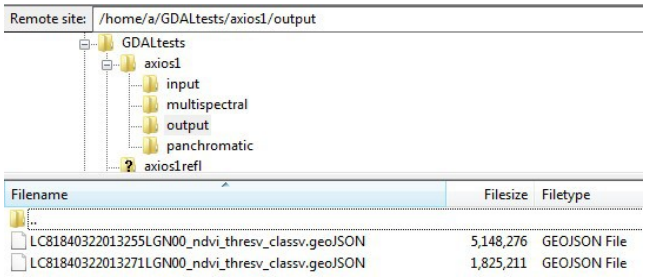
Εικόνα 90: Παράδειγμα πηγαίων καταλόγων 'panchromatic' και 'multispectral'



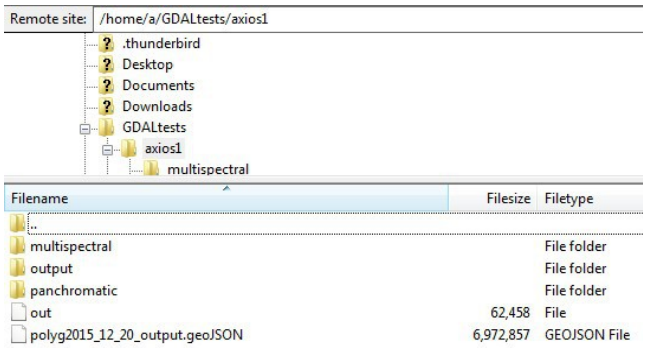
Εικόνα 91: Αρχικός κατάλογος πριν τρέξει το πρόγραμμα (χωρίς το ίδιο το πρόγραμμα) για pansharpening αρχικών



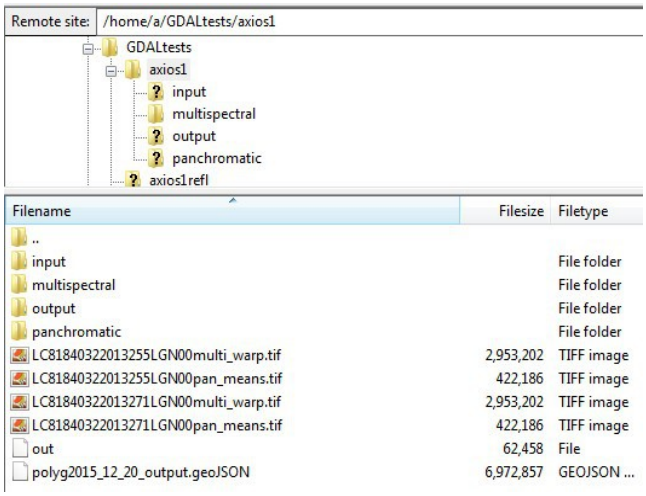
Εικόνα 92: Περιεχόμενο καταλόγου 'input' μετά το πέρας τρεξίματος του προγράμματος για NDVI



Εικόνα 93: Περιεχόμενο καταλόγου 'output' μετά το πέρας τρεξίματος του προγράμματος για NDVI



Εικόνα 94: Περιεχόμενου καταλόγου όπου βρίσκεται το πρόγραμμα με διαγραφή των ενδιάμεσων εικόνων (δεν υπάρχει ούτε ο κατάλογος 'input')



Εικόνα 95: Περιεχόμενου καταλόγου όπου βρίσκεται το πρόγραμμα χωρίς διαγραφή των ενδιάμεσων εικόνων

3.1 Σύγκριση Αλγορίθμων Pansharpening με Υλοποίηση σε MatLab και C++

Για την αύξηση χωρικής ανάλυσης στα πολυφασματικά κανάλια, χρησιμοποιήθηκαν οι ίδιες πηγαίες παγχρωματικές και πολυφασματικές εικόνες.

Αξιός1 2013-255	Παγχρωματική	Πολυφασματική	HPF - Matlab	HPF - C++
Χρόνος			0.6s	17s
Μέγεθος Αρχείου	412KB	723KB	1685KB	2953KB

Πίνακας 5: Μεγέθη Παγχρωματικής, Πολυφασματικής και Pansharpened Αξιός1-2013-255

Αξιός1 2013-271	Παγχρωματική	Πολυφασματική	HPF - Matlab	HPF - C++
Χρόνος			0.7s	17s
Μέγεθος Αρχείου	412KB	723KB	1685KB	2953KB

Πίνακας 6: Μεγέθη Παγχρωματικής, Πολυφασματικής και Pansharpened Αξιός1-2013-271

Αξιός2 2013-255	Παγχρωματική	Πολυφασματική	HPF - Matlab	HPF - C++
Χρόνος			0.5s	9s
Μέγεθος Αρχείου	227KB	397KB	904KB	1625KB

Πίνακας 7: Μεγέθη Παγχρωματικής, Πολυφασματικής και Pansharpened Αξιός2-2013-255

Αξιός2 2013-271	Παγχρωματική	Πολυφασματική	HPF - Matlab	HPF - C++
Χρόνος			0.4s	9s
Μέγεθος Αρχείου	227KB	397KB	904KB	1625KB

Πίνακας 8: Μεγέθη Παγχρωματικής, Πολυφασματικής και Pansharpened Αξιός2-2013-271

Κάρλα 2013-255	Παγχρωματική	Πολυφασματική	HPF - Matlab	HPF - C++
Χρόνος			1.4s	33.5s
Μέγεθος Αρχείου	823KB	1433KB	3280KB	5735KB

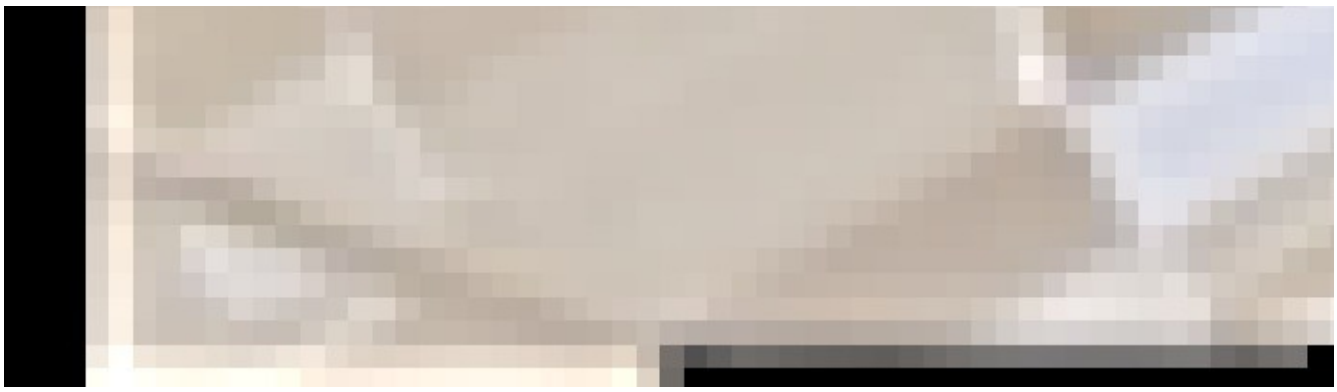
Πίνακας 9: Μεγέθη Παγχρωματικής, Πολυφασματικής και Pansharpened Κάρλα-2013-255

Κάρλα 2013-271	Παγχρωματική	Πολυφασματική	HPF - Matlab	HPF - C++
Χρόνος			1.7s	34s
Μέγεθος Αρχείου	823KB	1433KB	3280KB	5735KB

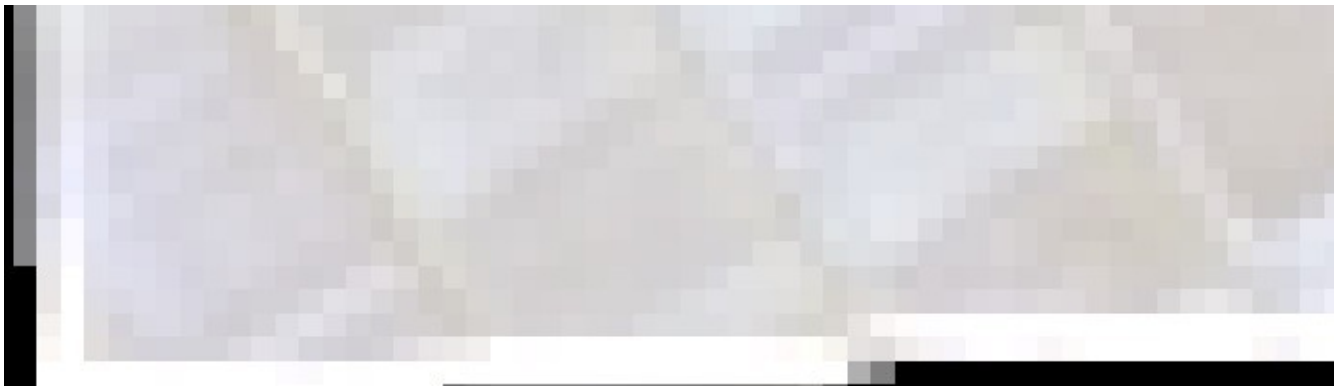
Πίνακας 10: Μεγέθη Παγχρωματικής, Πολυφασματικής και Pansharpened Κάρλα-2013-271

Είναι προφανές πως τα μεγέθη και οι χρόνοι είναι σχεδόν ίδιοι για τις ίδιες περιοχές - εικόνες με ίδιο μέγεθος. Δηλαδή, η ταχύτητα και το αποτέλεσμα των αλγορίθμων είναι ανάλογο του μεγέθους (πλάτος, μήκος) των πηγαίων εικόνων και όχι των τιμών που επεξεργάζονται.

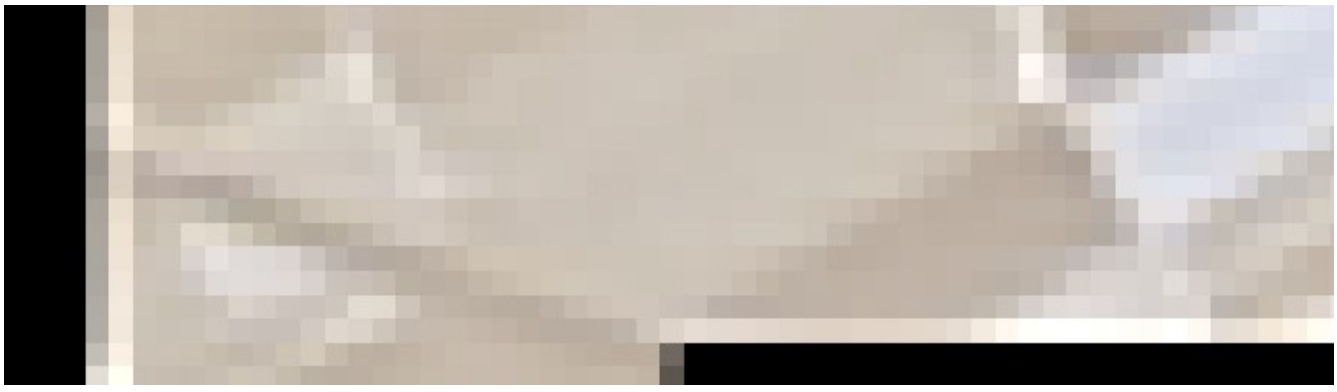
Εικόνα 96: Αξιός2 2013-255 HPF C++



Εικόνα 97: Κάρλα 2013-271 HPF MatLab



Εικόνα 98: Αξιός2 2013-255 HPF MatLab



Εικόνα 99: Κάρλα 2013-271 HPF C++

Από τις μεγενθυμένες εικόνες φαίνεται πως αν και ελάχιστα η υλοποίηση σε C++ έχει αποτελέσματα με περισσότερη αντίθεση. Επίσης, δεν λαμβάνει υπόψη σχεδόν καθόλου τα μαυρισμένα άκρα εκτός περιοχής με αποτέλεσμα να μην επηρεάζονται οι τιμές pixel της κυρίως περιοχής, σε αντίθεση με τα αποτελέσματα του MatLab.

Κατά τα άλλα υλοποίηση σε MatLab έχει πολύ καλύτερους χρόνους και μεγέθη αποτελέσματος, κάνοντας την υλοποίηση σε C+ περιττά αργή και τα αποτελέσματά της ~1.75 φορές μεγαλύτερα.

Ένα λογικό συμπέρασμα θα ήταν να προτιμάται η υλοποίηση MatLab όταν υπάρχει ανάγκη ταχύτητας και οικονομίας χώρου - όπως για μεγάλες ή πολλές εικόνες - και η C++ όταν χρειάζεται υψηλότερη ακρίβεια.

Μια πιθανή εξήγηση για αυτή τη διαφορά ταχύτητας είναι πως η C++ διαβάζει και γράφει κάθε pixel ξεχωριστά και ως γνωστόν η λειτουργίες IO είναι πολύ αργές. Αν γίνει αναλυτικός έλεγχος του μεγέθους της μνήμης και ανέβει η εικόνα σε μεγάλα κομμάτια, θα μειωθεί εξαιρετικά ο χρόνος. Το MatLab είναι πιθανό να διαβάζει όλη την εικόνα μαζί και έτσι να γλιτώνει πολύ χρόνο στο IO.

3.2 Κανονικοποιημένοι Δείκτες: Πολυφασματικές και Εικόνες Δεικτών

Ακολουθούν πίνακες με μεγέθη εικόνων πριν και μετά την εφαρμογή κανονικοποιημένων δεικτών καθώς και των ομαδοποιήσεων.

αρχικές	Πολυφασματική (HPF)	NDVI/NDWI	Ομαδοποιημένη
Αξιός1 - Y:2013 D:255	2.81 MB	824 KB	412 KB
Αξιός1 - Y:2013 D:271	2.81 MB	824 KB	412 KB
Αξιός2 - Y:2013 D:255	1.54 MB	453 KB	226 KB
Αξιός2 - Y:2013 D:271	1.54 MB	453 KB	226 KB
Κάρλα - Y:2013 D:255	5.46 MB	1.56 MB	800 KB
Κάρλα - Y:2013 D:271	5.46 MB	1.56 MB	800 KB

Πίνακας 11: Μεγέθη Πολυφασματικών και Κανονικοποιημένων Δεικτών Αρχικών Εικόνων

Οι διαφορές στα μεγέθη μπορούν να εξηγηθούν από τους τύπους δεδομένων των τιμών pixel ανά εικόνα. Δηλαδή, στις πολυφασματικές πρόκειται για ακέραιους μεγέθους 16bit * αριθμός ζωνών * μήκος ζώνης * πλάτος ζώνης. Οπότε είναι λογικό να είναι ~7πλάσιο από το μέγεθος μιας ζώνης με τιμές pixel ακέραιους 16bit.

Ύστερα, στην εφαρμογή των δεικτών, οι τιμές pixel μετατρέπονται σε δεκαδικούς 32bit οπότε θα έχουν το διπλάσιο μέγεθος από αυτό μιας ζώνης με τιμές pixel 16bit.

Τέλος, κατά την ομαδοποίηση, εφόσον οι τιμές pixel κυμαίνονται μεταξύ 0 και 6, μετατρέπονται ξανά σε ακέραιους 16bit, με αποτέλεσμα τα μεγέθη τους να υποδιπλασιάζονται.

Το ίδιο ακριβώς συμβαίνει και με τις διορθωμένες εικόνες. Ακολουθούν παραδείγματα.

διορθωμένες	Πολυφασματική	NDVI/NDWI	Ομαδοποιημένη
Αξιός1 2013-09-12	722 KB	207 KB	104 KB
Αξιός1 2014-03-07	722 KB	207 KB	104 KB
Αξιός2 2013-10-30	396 KB	113 KB	57.1 KB
Αξιός2 2014-11-18	396 KB	113 KB	57.1 KB

Πίνακας 12: Μεγέθη Πολυφασματικών και Κανονικοποιημένων Δεικτών Διορθωμένων Εικόνων

Όπως αναφέρθηκε και παραπάνω, ανά περιοχή, όλες οι raster εικόνες έχουν το ίδιο μήκος και πλάτος, οπότε ο αριθμός pixel και συνεπώς τα μεγέθη των αρχείων δεν αλλάζουν.

Με άλλα λόγια, όλες οι υπόλοιπες εικόνες/ημερομηνίες των ίδιων δυο περιοχών ακολουθούν το ίδιο ακριβώς μοτίβο μεγεθών που παρατηρείται στα παραδείγματα του πάνω πίνακα.

3.3 Διανυσματοποίηση: GeoTIFF και GeoJSON

Όπως περιγράφηκε στο κεφάλαιο 2.7.1, στην υλοποίηση με C++ πραγματοποιείται η μετατροπή των raster εικόνων σε vector με τη χρήση του **GDALPolygonize** της βιβλιοθήκης GDAL/OGR, η οποία καλείται από τη συνάρτηση single**_polygonize** που γράφτηκε για τις ανάγκες της παρούσας εργασίας.

Η εικόνα που πρόκειται να μετατραπεί σε vector, είναι προϊόν ομαδοποίησης των pixel εικόνας που περιέχει τους δείκτες NDVI ή NDWI, ανάλογα με την περιοχή που απεικονίζει (Αξιός1-NDVI, Αξιός2-NDVI, Κάρλα-NDWI), εφαρμοσμένους σε διορθωμένη ή αρχική πολυφασματική δορυφορική εικόνα.

Ακολουθεί πίνακας με τα μεγέθη πριν και μετά τη διανυσματοποίηση των ομαδοποιημένων αρχικών πολυφασματικών που χρησιμοποιήθηκαν.

	Raster	Vector
Axios1 - ομαδοποιημένη - Y:2013 D:255	413 KB	5,028 KB
Axios1 - ομαδοποιημένη - Y:2013 D:271	413 KB	1,783 KB
Axios2 - ομαδοποιημένη - Y:2013 D:255	227 KB	2,485 KB
Axios2 - ομαδοποιημένη - Y:2013 D:271	227 KB	1,702 KB
Karla - ομαδοποιημένη - Y:2013 D:255	801 KB	206 KB
Karla - ομαδοποιημένη - Y:2013 D:271	801 KB	898 KB

Πίνακας 13: Μεγέθη Raster Και Vector Ομαδοποιημένων Αρχικών Εικόνων

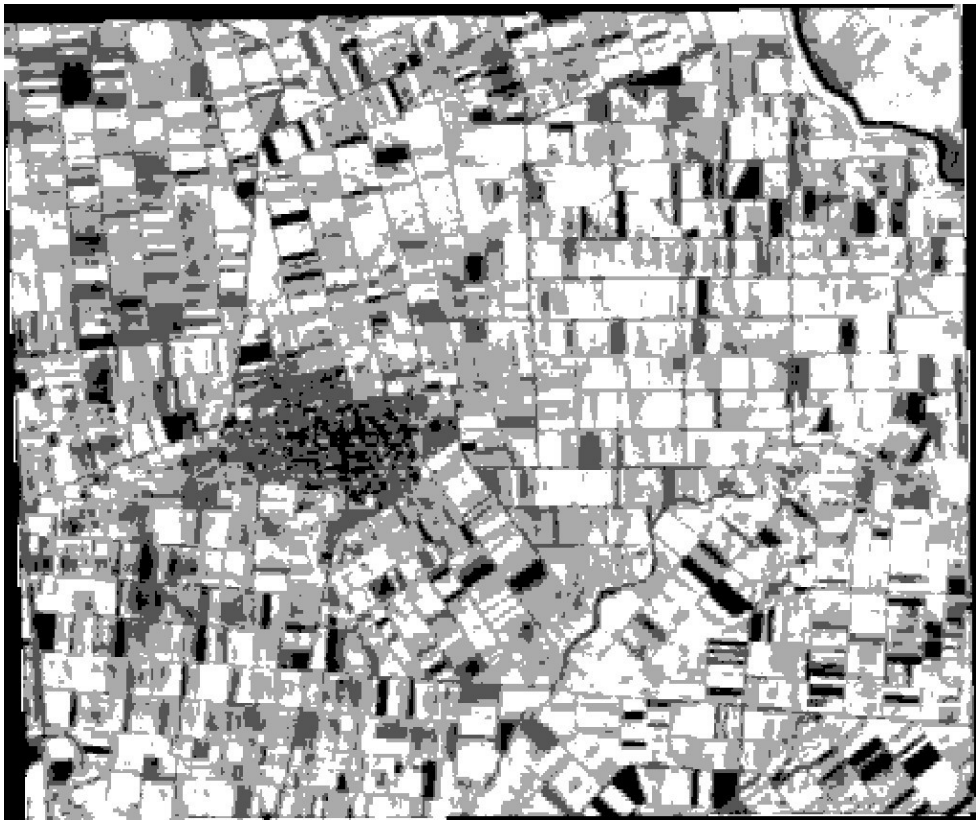
Οι raster έχουν ίδιο ακριβώς μέγεθος ανά περιοχή ανεξαρτήτως ημερομηνίας, αφού πρόκειται για εικόνες με ακριβώς ίδιο αριθμό εικονοστοιχείων. Στις vector είναι λογικό να υπάρχουν διαφορές, αφού ο αριθμός των χαρακτηριστικών/πολυγώνων εξαρτάται από τις ομάδες γειτονικών pixel με ίδια τιμή στην αντίστοιχη raster. Μια διανυσματοποιημένη εικόνα με πολλά πολύγωνα θα έχει μεγαλύτερο μέγεθος από μια με λίγα.

Στις επόμενες σελίδες ακολουθούν οι αντίστοιχες εικόνες.

Στις Axios1, είναι προφανής η διαφορά στους αριθμούς των πολυγώνων της Y:255 (πολλά) με της Y:271 (αρκετά λιγότερα). Έτσι εξηγείται και η μεγάλη διαφορά (λόγος ~0.33) στα μεγέθη 5,028KB και 1,783KB αντίστοιχα. Ενώ, οι raster αντίστοιχες εικόνες έχουν ίδιο ακριβώς μέγεθος ανεξάρτητα από τις διαφορές στις τιμές pixel τους.

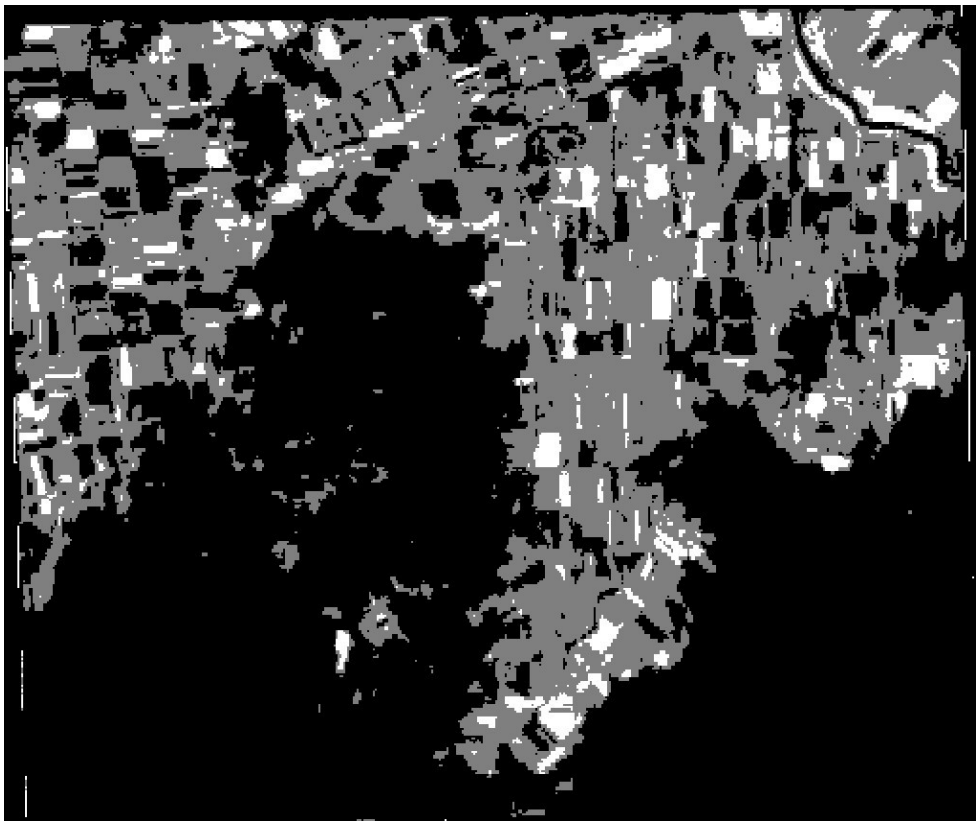
Στις Αξιός2 και οι δυο εικόνες έχουν σχετικά μεγάλο αριθμό πολυγώνων, οπότε η διαφορά τους είναι αρκετά μικρότερη 2,485KB η Y:2013 D:255 και 1,702KB η Y:2013 D:271, δηλαδή έχουν λόγο ~0.7.

Στις Κάρλα, η Y:2013 D:271 έχει αρκετά περισσότερα πολύγωνα και συνεπώς το μέγεθός της (898KB) είναι αρκετά μεγαλύτερο από της Y:2013 D:255 (206KB) και συγκεκριμένα έχουν λόγο ~0.2.

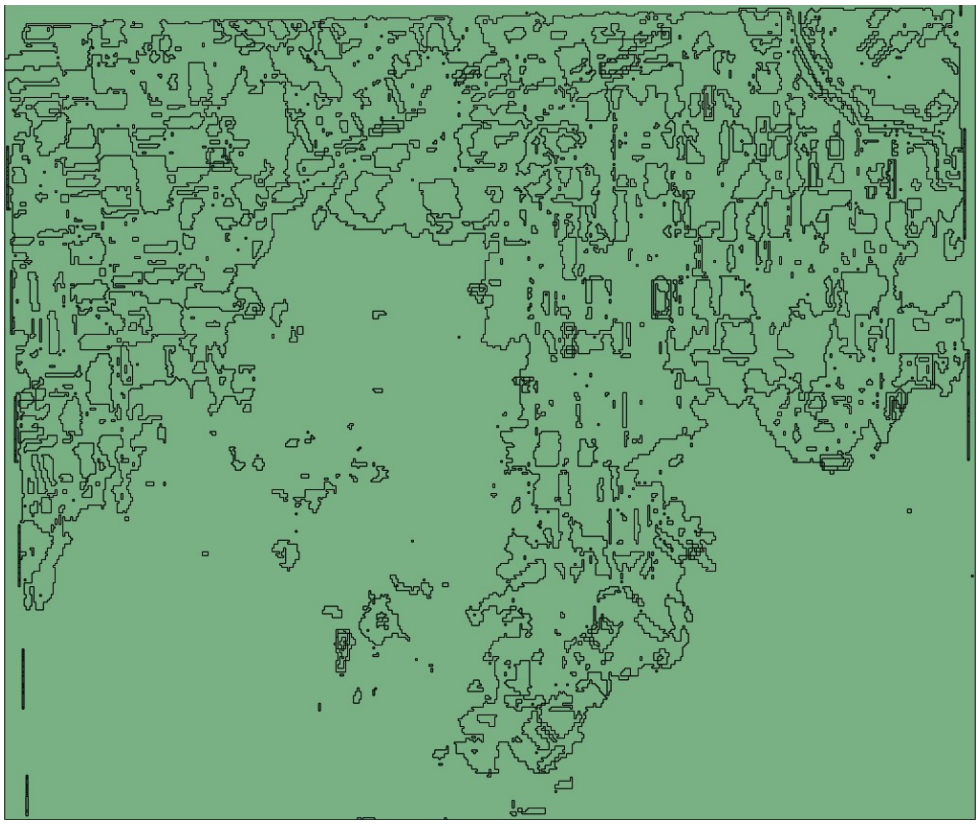


Εικόνα 100: Axios1 Y:2013 D:255 ομαδοποιημένο NDVI (raster) 413KB

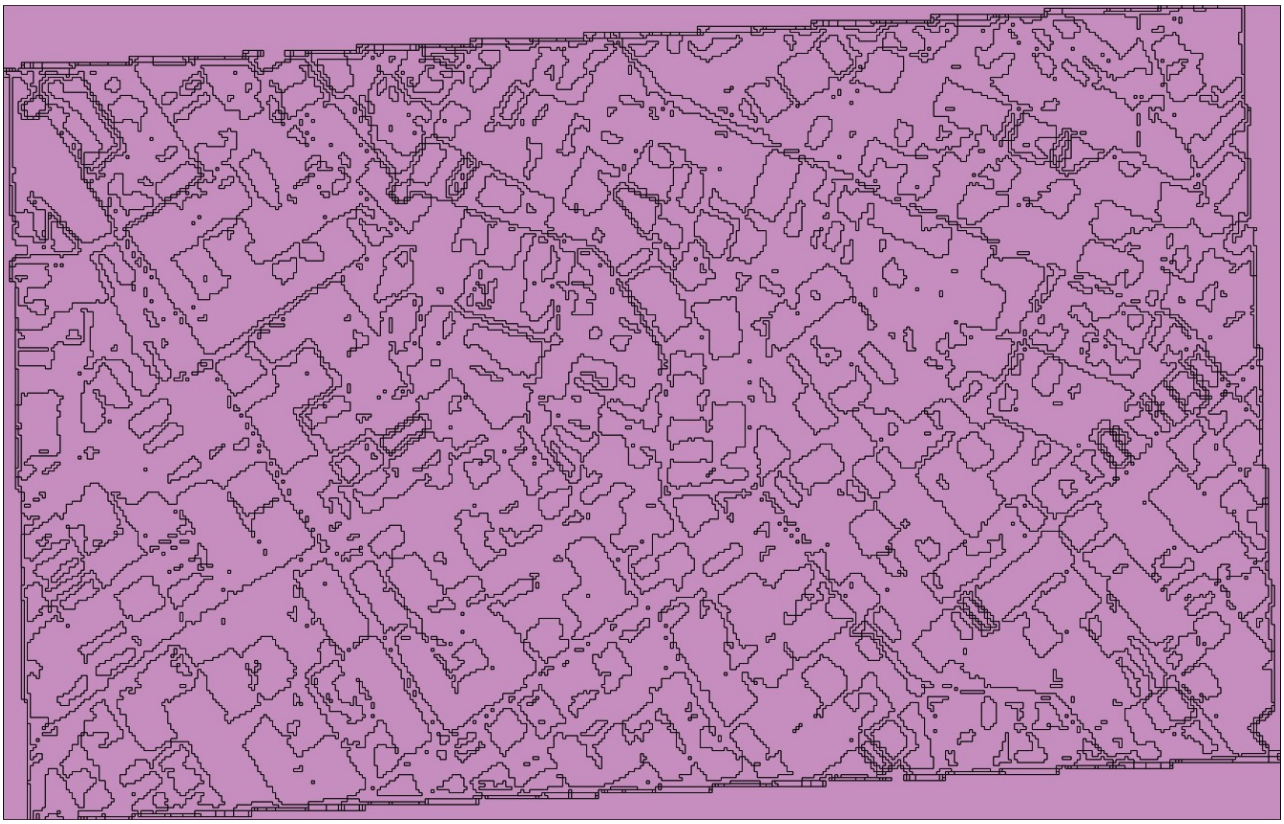
Εικόνα 101: Axios1 Y:2013 D:255 ομαδοποιημένο NDVI (vector) 5,028KB



Εικόνα 102: Axios1 Y:2013 D:271 ομαδοποιημένο NDVI (raster) 413KB



Εικόνα 103: Axios1 Y:2013 D:271 ομαδοποιημένο NDVI (vector) 1,783KB



Εικόνα 104: Axios2 Y:2013 D:255 ομαδοποιημένο NDVI (vector) 2,485KB

Εικόνα 105: Axios2 Y:2013 D:271 ομαδοποιημένο NDVI (vector) 1,702KB

Εικόνα 106: Κάρλα Y:2013 D:255
ομαδοποιημένο NDWI (vector) 206KB

Εικόνα 107: Κάρλα Y:2013 D:271
ομαδοποιημένο NDWI (vector) 898KB

Αντίστοιχα, στις διορθωμένες υπάρχουν πάλι διαφορές στα μεγέθη βασισμένες, ομοίως με τις αρχικές, στον αριθμό των πολυγώνων. Ακολουθεί πίνακας με τα μεγέθη των ομαδοποιημένων raster και στη συνέχεια vector που παρήχθησαν. Δόθηκαν διορθωμένες μόνο από τις περιοχές Αξιός1 και Αξιός2, οπότε χρησιμοποιήθηκε μόνο ο δείκτης NDVI.

διορθωμένες	Axios1 Raster	Axios1 Vector	Axios2 Raster	Axios2 Vector
2013-09-12	103 KB	4,884 KB	57,1 KB	1,832 KB
2013-09-28	103 KB	2,884 KB	57,1 KB	1,213 KB
2013-10-14	103 KB	1 KB	57,1 KB	1 KB
2013-10-30	103 KB	824 KB	57,1 KB	19 KB
2013-11-15	103 KB	1 KB	57,1 KB	1 KB
2013-12-17	103 KB	581 KB	57,1 KB	144 KB
2014-01-02	103 KB	1 KB	57,1 KB	1 KB
2014-02-03	103 KB	1,652 KB	57,1 KB	798 KB
2014-02-19	103 KB	1 KB	57,1 KB	1 KB
2014-03-07	103 KB	1 KB	57,1 KB	1 KB
2014-03-23	103 KB	2,609 KB	57,1 KB	1,225 KB
2014-04-08	103 KB	2,096 KB	57,1 KB	1,565 KB
2014-04-24	103 KB	557 KB	57,1 KB	1 KB
2014-05-10	103 KB	1,451 KB	57,1 KB	556 KB
2014-05-26	103 KB	1,111 KB	57,1 KB	1,025 KB
2014-06-11	103 KB	1,852 KB	57,1 KB	2,307 KB
2014-06-27	103 KB	3,335 KB	57,1 KB	2,784 KB
2014-07-13	103 KB	5,254 KB	57,1 KB	3 KB
2014-07-29	103 KB	5,089 KB	57,1 KB	2,221 KB
2014-08-14	103 KB	4,506 KB	57,1 KB	2,339 KB
2014-08-30	103 KB	1,368 KB	57,1 KB	1,818 KB
2014-09-15	103 KB	3,458 KB	57,1 KB	383 KB
2014-10-01	103 KB	4,624 KB	57,1 KB	1,591 KB
2014-10-17	103 KB	144 KB	57,1 KB	13 KB
2014-11-02	103 KB	2,605 KB	57,1 KB	358 KB
2014-11-18	103 KB	1 KB	57,1 KB	1 KB

Πίνακας 14: Μεγέθη Raster Και Vector Ομαδοποιημένων Διορθωμένων Εικόνων

Οι raster εικόνες πάλι είναι ίδιες σε μέγεθος ανά περιοχή, όπως αναμενόταν, αφού εξαρτώνται από τον αριθμό των pixel, δηλαδή μήκος επί πλάτος της εικόνας.

Στις διανυσματικές, υπάρχουν κάποιες ημερομηνίες κατά τις οποίες παρατηρείται μέγεθος 1KB, δηλαδή καθόλου βλάστηση (πχ. 2014-01-02, 2014-02-19, 2014-03-07).

Εικόνα 108: Παράδειγμα vector εικόνας με μόνο ένα πολύγωνο

Όλες οι vector εικόνες με μέγεθος 1KB έχουν την ίδια μορφή αφού αποτελούνται από μόνο ένα πολύγωνο το οποίο αποτελεί και το όριο της εικόνας.

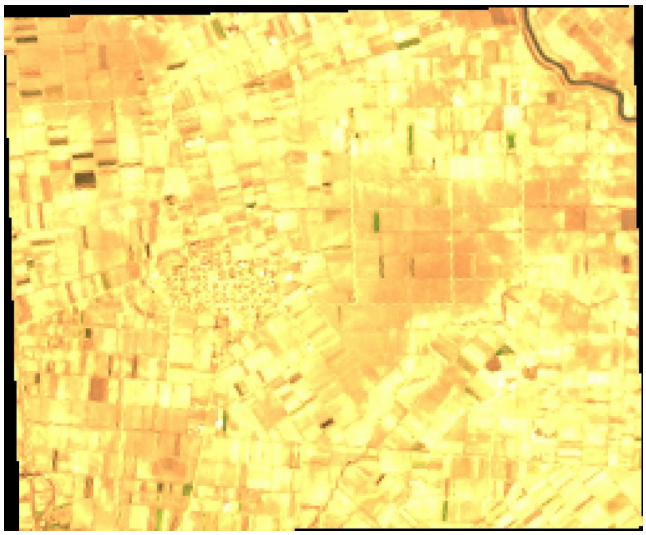
Αυτές οι περιπτώσεις εξηγούνται από τη μεσολάβηση νεφών μεταξύ δορυφόρου και περιοχής απεικόνισης.

Ακολουθούν δυο παραδείγματα διαφορετικών ημερομηνιών, ένα από κάθε περιοχή.

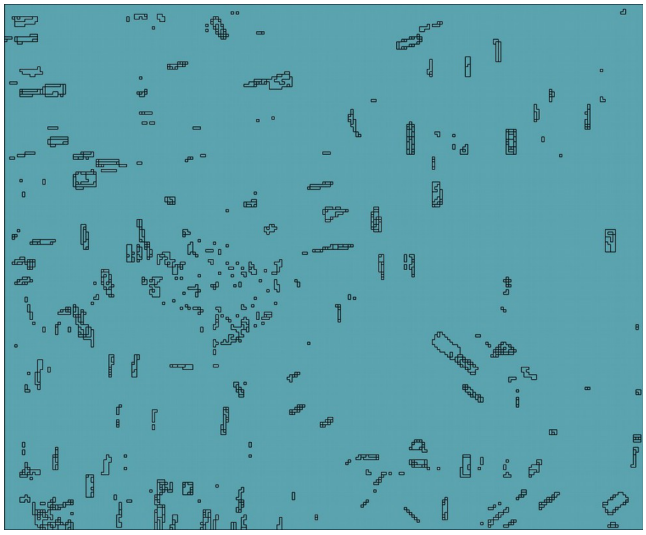
Εικόνα 109: Axios1 2014-02-19 διορθωμένη RGB (raster)

Εικόνα 110: Axios2 2014-03-07 διορθωμένη RGB (raster)

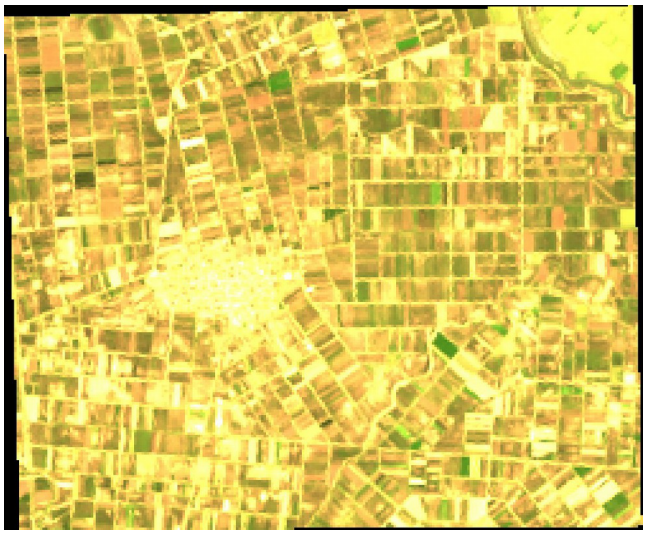
Στις υπόλοιπες, παρατηρούνται μεγάλες διαφορές ανά περιόδους και αυτό είναι λογικό, αφού για παράδειγμα το χειμώνα (πχ. 2013-12-17, 2014-02-03, 2014-11-02), η βλάστηση είναι ελάχιστη και άρα υπάρχουν λίγες διαφορές στις τιμές των pixel, ενώ την άνοιξη (πχ. 2014-06-11, 2014-06-27) και το φθινόπωρο (πχ. 2013-09-12) υπάρχει άνθιση με συνέπεια την ποικιλία των τιμών pixel ανά αγροτεμάχιο.



Εικόνα 111: Αξιός1 (διορθωμένη) 2013-12-17 (χειμώνας=>χαμηλό NDVI) RGB



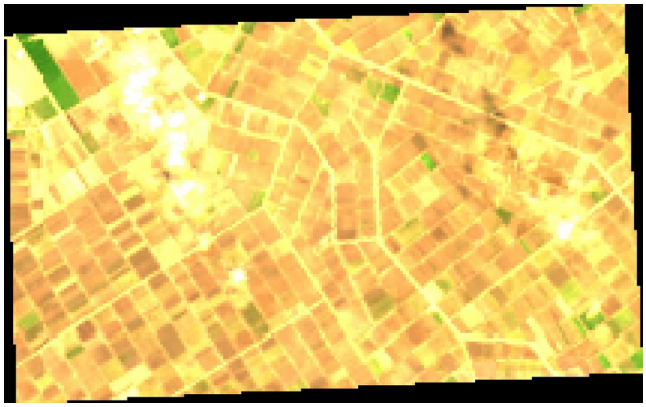
Εικόνα 112: Αξιός1 (διορθωμένη) 2013-12-17 (χειμώνας=>χαμηλό NDVI) vector



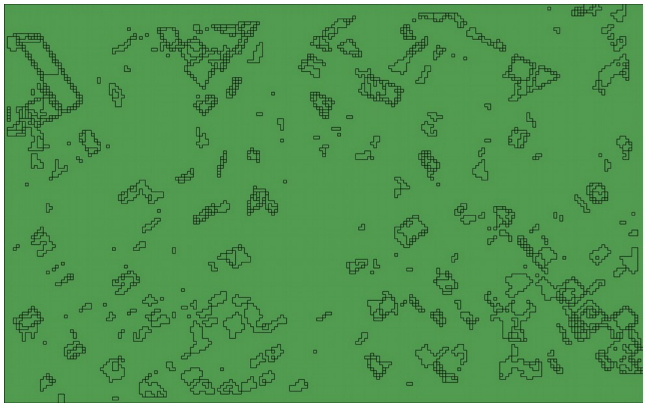
Εικόνα 113: Αξιός1 (διορθωμένη) 2013-09-12 (φθινόπωρο=>υψηλό NDVI) RGB

Εικόνα 114: Αξιός1 (διορθωμένη) 2013-09-12 (φθινόπωρο=>υψηλό NDVI) vector

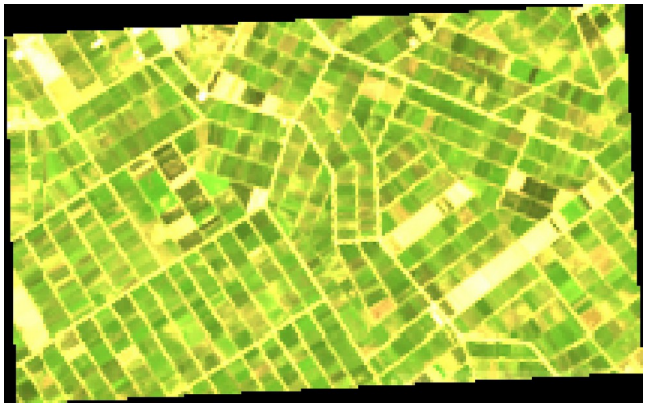
Η διαφορά στη βλάστηση είναι προφανής μεταξύ των χειμερινών και των υπόλοιπων εικόνων. Στην Αξιός1 2013-09-12 παρά το γεγονός πως δεν είναι έντονη, υπάρχει βλάστηση, και μάλιστα με ποικιλία στις τιμές.



Εικόνα 115: Αξιός2 (διορθωμένη) 2014-02-03 (χειμώνας=>χαμηλό NDVI) RGB



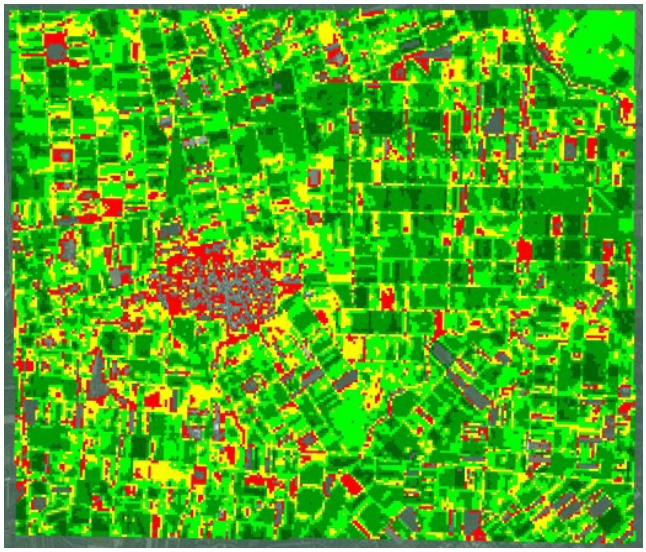
Εικόνα 116: Αξιός2 (διορθωμένη) 2014-02-03 (χειμώνας=>χαμηλό NDVI) vector



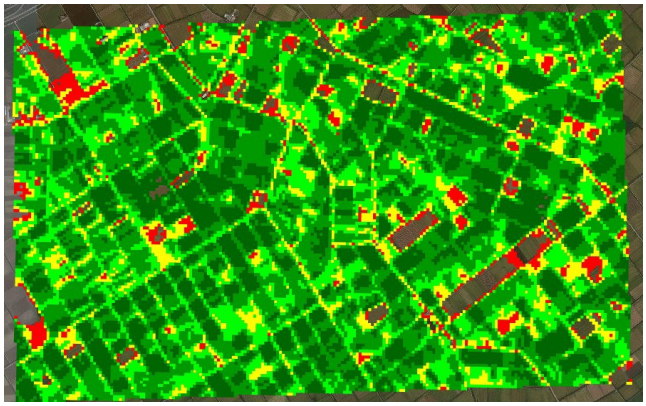
Εικόνα 117: Αξιός2 (διορθωμένη) 2014-06-27 (καλοκαίρι=>υψηλό NDVI) RGB

Εικόνα 118: Αξιός2 (διορθωμένη) 2014-06-27 (καλοκαίρι=>υψηλό NDVI) vector

Για καλύτερη κατανόηση, ακολουθούν οι αντίστοιχες οπτικοποιήσεις τους μέσω του CartoDB.



Εικόνα 119: Αξιός1 (διορθωμένη) 2013-09-12 (φθινόπωρο=>υψηλό NDVI) CartoDB



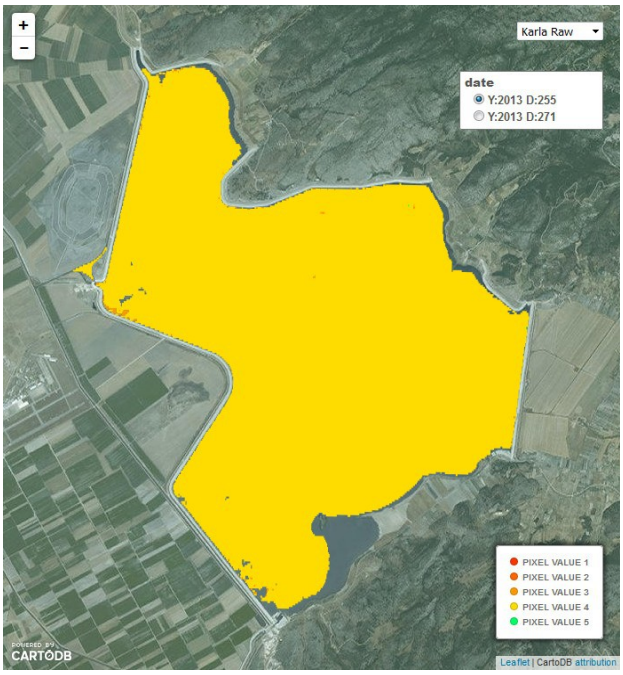
Εικόνα 120: Αξιός2 (διορθωμένη) 2014-06-27 (καλοκαίρι=>υψηλό NDVI) CartoDB

3.4 Αξιολόγηση Παραμέτρων Οπτικοποίησης: Χρόνος Φόρτωσης και Μεταβολές Δεικτών

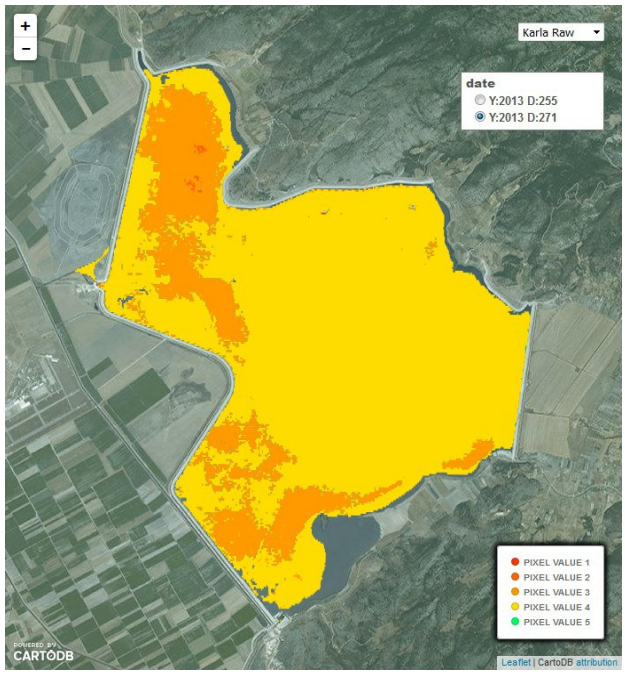
Μετά από μέτρηση, ο μέσος χρόνος που χρειάζεται για το κατέβασμα ενός πλήρους GeoJSON σε μορφή πλακιδίων από το νέφος του CartoDB στον περιηγητή που το καλεί μέσω της οπτικοποίησης είναι ~1.5-2 δευτερόλεπτα σε σύνδεση ταχύτητας 16mbps. Είναι ανάλογο της σχέσης του συνολικού μεγέθους της εικόνας με τη μεγέθυνση παρατήρησης και δεν εξαρτάται από τον αριθμό των πολυγώνων όταν ζητείται ολόκληρη. Ένα πιο συγκεκριμένο επερώτημα όμως θα καθυστερήσει το κατέβασμα των πλακιδίων αφού θα χρειαστεί να κάνει το αντίστοιχο φιλτράρισμα των πολυγώνων πρώτα.

Πρέπει να αναφερθεί πως ένας διαφορετικός τύπος απεικόνισης πχ. με απεικόνιση κάθε pixel σαν διαφορετικό πολύγωνο αντί για σύμπτυξη όλων των γειτονικών με ίδιες τιμές, ίσως να βελτίωνε την ταχύτητα φόρτωσης των εικόνων, αφού θα παρέκαμπτε τον χρονοβόρο υπολογισμό όλων των ορίων του κάθε πολυγώνου και τα πλακάκια θα σχηματίζονταν πολύ πιο γρήγορα.

Μέσω του CartoDB, εφόσον έχουν χρησιμοποιηθεί οι κατάλληλοι χρωματισμοί για τα διαφορετικά πέντε επίπεδα βλάστησης, μπορεί να παρατηρηθεί η μεταβολή κατά μήκος του χρόνου για το δείκτη μιας συγκεκριμένης περιοχής.

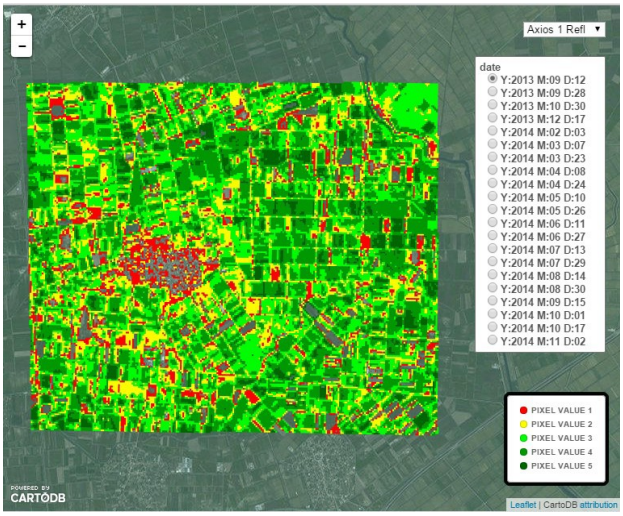


Εικόνα 121: Κάρλα Y:2013 D:255 (2013-09-12)

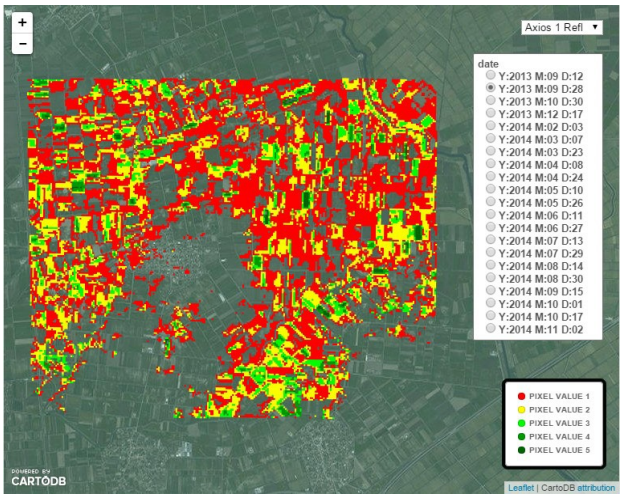


Εικόνα 122: Κάρλα Y:2013 D:271 (2013-09-28)

Στις απεικονίσεις (χλωροφύλλης-α φιλτραρισμένης με NDWI σε διορθωμένες δορυφορικές εικόνες) των δυο ημερομηνιών της λίμνης Κάρλα παρατηρείται πως τη μέρα 271 του 2013 η χλωροφύλλη έχει μειωθεί αρκετά σε σχέση με τη μέρα 255. Πιο συγκεκριμένα υπάρχουν αισθητά πολλές περιοχές που από τιμή pixel 4 έχουν πέσει στην τιμή 3. Αυτό μπορεί να εξηγηθεί από μειωμένη ηλιοφάνεια στις μέρες που μεσολάβησαν καθώς και από συνολική μεταβολή του καιρού αφού οι μέρες πλησιάζουν προς το φθινόπωρο, η Γη απομακρύνεται από τον Ήλιο και η θερμοκρασία σιγά σιγά πέφτει.

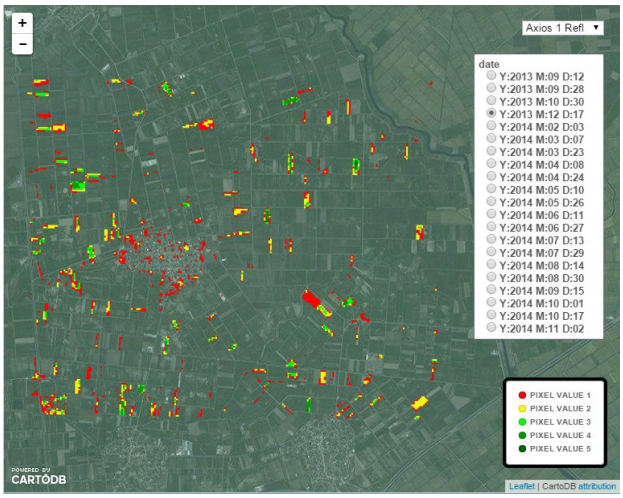


Εικόνα 123: Αξιός1 2013-09-12

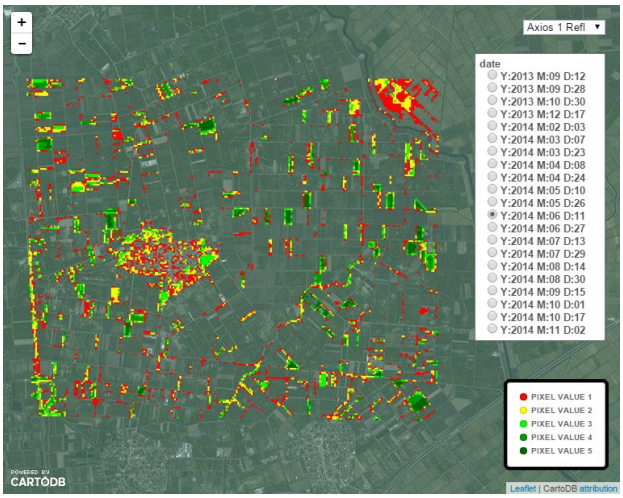


Εικόνα 124: Αξιός1 2013-09-28

Στην περιοχή Αξιός1, παρατηρείται πως όσο πλησιάζει προς το χειμώνα (αρχές προς τέλος Σεπτέμβρη), η βλάστηση μειώνεται.

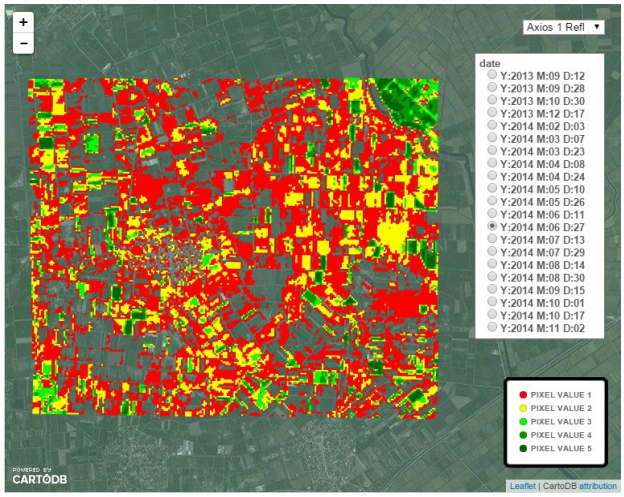


Εικόνα 125: Αξιός1 2013-12-17

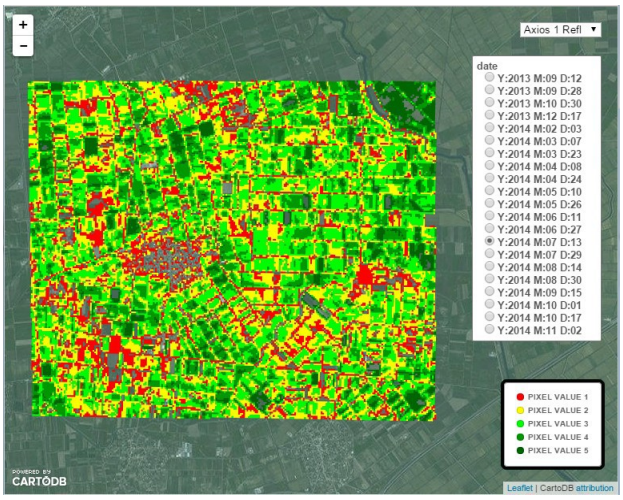


Εικόνα 126: Αξιός1 2014-06-11

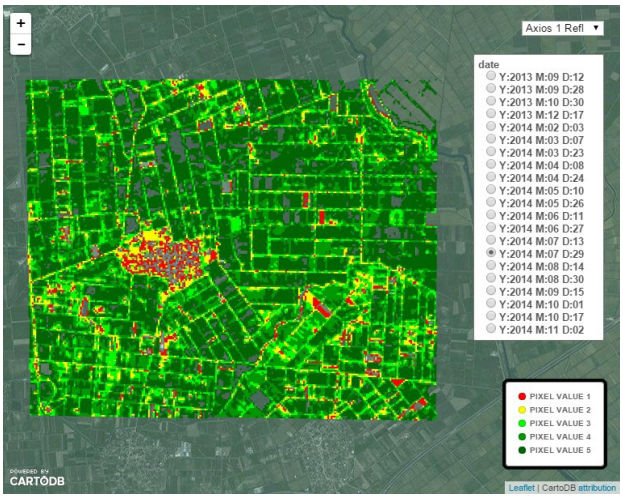
Μέχρι που μπαίνει ο Δεκέμβρης και το πράσινο σχεδόν εξαφανίζεται. Αυτό συνεχίζεται μέχρι τα μέσα Ιουνίου, όπου αρχίζει πάλι να εμφανίζεται και όταν πλέον μπει ο Ιούλιος, η βλάστηση έχει εξαπλωθεί σχεδόν παντού.



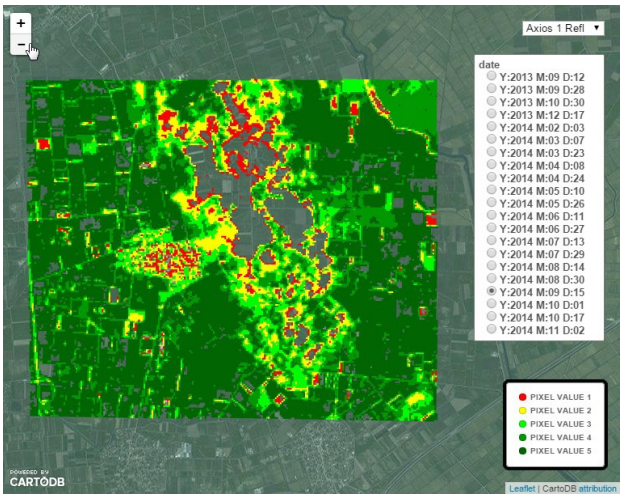
Εικόνα 127: Αξιός1 2014-06-27



Εικόνα 128: Αξιός1 2014-07-13

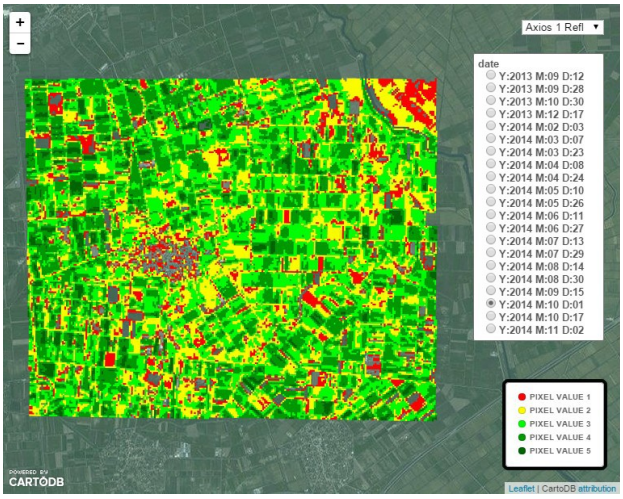


Εικόνα 129: Αξιός1 2014-07-29

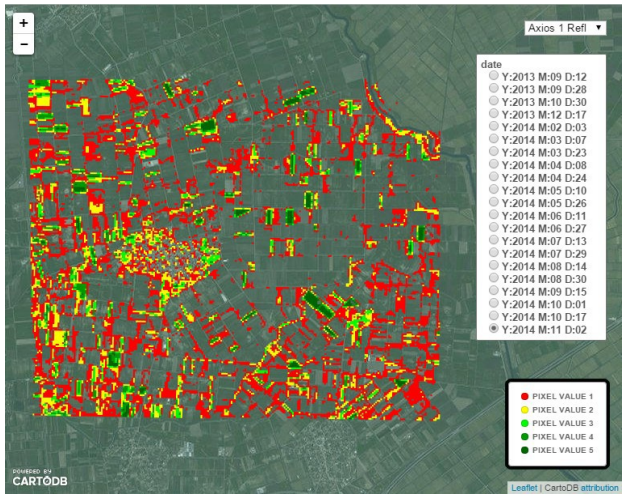


Εικόνα 130: Αξιός1 2014-09-15

Παρατηρείται έντονη βλάστηση και μάλιστα σε υψηλά επίπεδα (τιμή pixel/ομάδα 5) σχεδόν σε όλη την περιοχή (εκτός φυσικά από την πόλη και σημεία που καλύπτονται από σύννεφα – Εικόνα 130) μέχρι και τα μέσα Σεπτέμβρη, όπου την προηγούμενη χρονιά (Εικόνες 123-124) είχε αρχίσει να φθίνει. Αυτό μπορεί να εξηγηθεί από την ανώμαλη αλλαγή του καιρού των τελευταίων χρόνων και τη μετατόπιση των εποχών που και το 2016 όταν γράφεται η εργασία, κυριαρχούν θερμοκρασίες φθινοπώρου ακόμα και τον Ιανουάριο.



Εικόνα 131: Αξιός1 2014-10-01

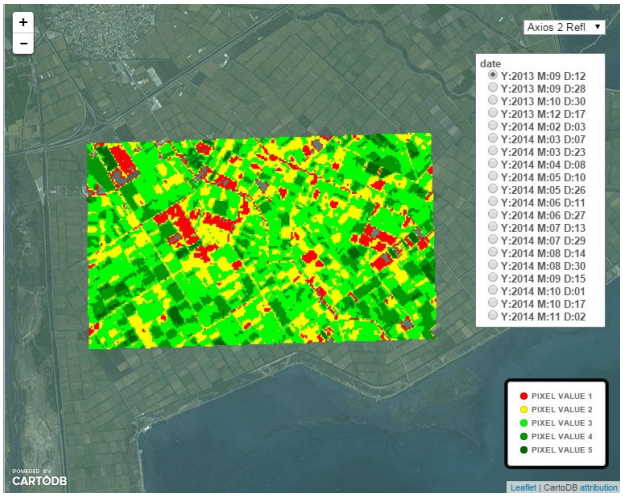


Εικόνα 132: Αξιός1 2014-11-02

Η μείωση της βλάστησης αρχίζει και γίνεται αισθητή μέσα στον Οκτώβρη (Εικόνα 131) και όταν μπει ο Νοέμβρης, το πράσινο έχει σχεδόν εξαφανιστεί και η βλάστηση κινείται σε πολύ χαμηλά επίπεδα (κυρίως ομάδα 1).

Μπορεί να παρατηρηθεί μια μετατόπιση σχεδόν πάνω από ενός μήνα στις διακυμάνσεις της βλάστησης. Συγκεκριμένα, η φθινοπωρινή μείωση του πράσινου που το 2013 παρατηρήθηκε στα τέλη Σεπτέμβρη, το 2014 έγινε στις αρχές Νοέμβρη.

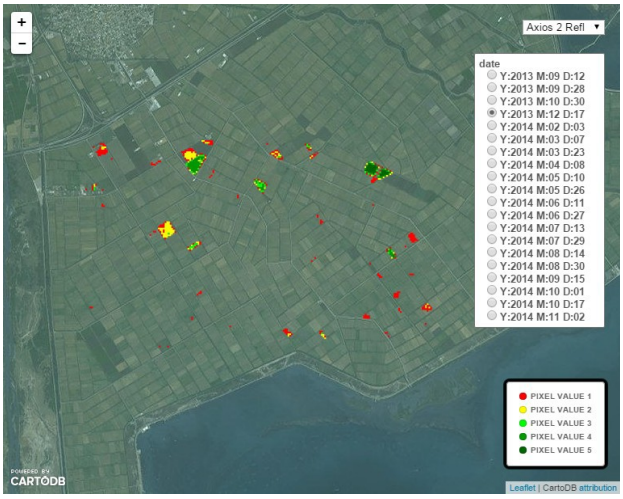
Την ίδια πορεία ακολουθεί και η βλάστηση στην περιοχή Αξιός2.



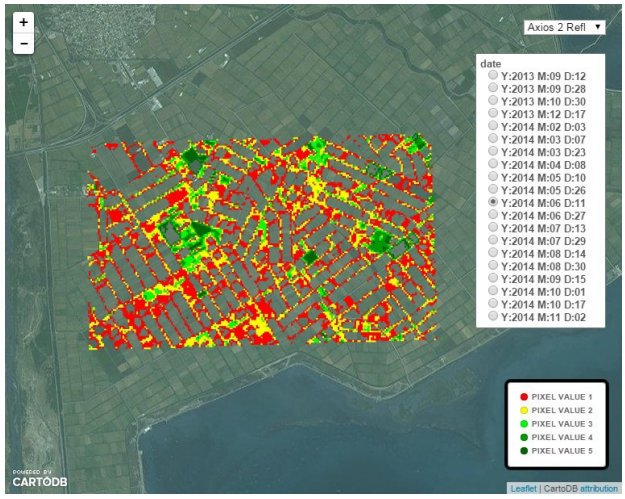
Εικόνα 133: Αξιός2 2013-09-12

Εικόνα 134: Αξιός2 2013-09-28

Μείωση το Σεπτέμβριο του 2013.



Εικόνα 135: Αξιός2 2013-12-17



Εικόνα 136: Αξιός2 2014-06-11

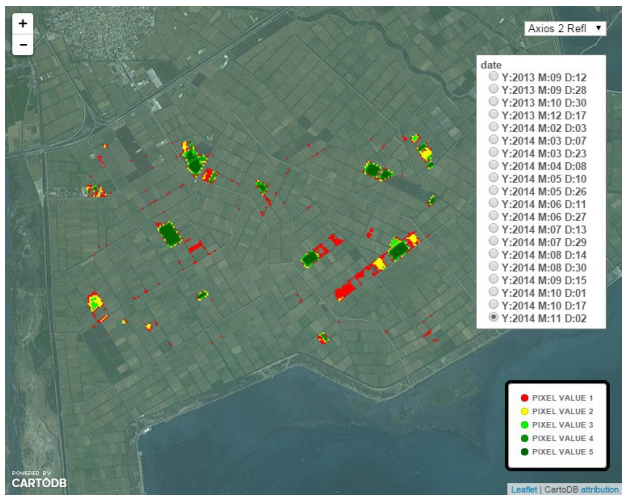
Καθόλου βλάστηση από το χειμώνα μέχρι τον Ιούνιο του 2014.

Εικόνα 137: Αξιός2 2014-07-13

Εικόνα 138: Αξιός2 2014-08-14

Έντονη βλάστηση το καλοκαίρι του 2014.

Εικόνα 139: Αξιός2 2014-10-01



Εικόνα 140: Αξιός2 2014-11-02

Μείωση της βλάστησης μέσα στον Οκτώβρη και σχεδόν εκμηδενισμός του πράσινου πάλι από το Νοέμβρη του 2014.

4. Συμπεράσματα

Σχετικά με το **pansharpening**, συνίσταται να προτιμάται η υλοποίηση σε Matlab όταν υπάρχει ανάγκη ταχύτητας και οικονομίας χώρου - όπως για μεγάλες ή πολλές εικόνες - ενώ η C++ μπορεί να χρησιμοποιείται όταν χρειάζεται υψηλότερη ακρίβεια.

Στη **διανυσματοποίηση**, για μικρό αριθμό κλάσεων, οι vector εικόνες έχουν μικρότερο μέγεθος (λίγα πολύγωνα) από τις raster, ενώ το αντίθετο ισχύει για περιπτώσεις με μεγάλη διασπορά κλάσεων στην εικόνα (πολλά πολύγωνα).

Για την **οπτικοποίηση με CartoDB**, βασικό σημείο ενδιαφέροντος είναι ο χρόνος φόρτωσης κάθε εικόνας:

- ~1.5s (για ταχύτητα internet 16mbps)
- ανάλογος της σχέσης του μεγέθους της εικόνας με τη μεγέθυνση παρατήρησης
- δεν εξαρτάται από τον αριθμό των πολυγώνων όταν ζητείται ολόκληρη
- ένα συγκεκριμένο επερώτημα θα καθυστερήσει περισσότερο αφού θα κάνει φιλτράρισμα των πολυγώνων πρώτα

Μια διαφορετική απεικόνιση π.χ. κάθε pixel σαν διαφορετικό πολύγωνο, ίσως βελτίωνε τους χρόνους, αφού θα παρέκαμπτε τον υπολογισμό των ορίων κάθε πολυγώνου.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] QuickBird Satellite Sensor Characteristics (http://satimagingcorp.com/satellite-sensors/quickbird/)
[2] Landsat Missions (http://landsat.usgs.gov/landsat8.php)
[3] Fusion of Multispectral and Panchromatic Images by Local Mean and Variance Matching Filtering Techniques, de Bethune/Muller/Donnay, Universite de Liege, Ιανουάριος 1998
[4] Διερεύνηση και αξιολόγηση μεθόδων συγχώνευσης πολυφασματικών και παγχρωματικών δορυφορικών δεδομένων, Αριστείδης Δ. Βαϊόπουλος, ΣΑΤΜ – ΕΜΠ, 2013
[5] Διάδοση της Ακτινοβολίας μέσα από την Ατμόσφαιρα. Εφαρμογή & Αξιολόγηση Απόλυτων Ατμοσφαιρικών Αλγόριθμων Διόρθωσης Τηλεπισκοπικών Απεικονίσεων, Παναγιώτης Ι. Σισμανίδης, ΣΑΤΜ-ΕΜΠ, Οκτώβριος 2012
[6] Ανάλυση Μεγάλων Γεωχωρικών Δεδομένων με Τεχνολογίες Πινάκων για Αγροτικές Εφαρμογές, Αθανάσιος Κ. Κάρμας, ΣΗΜΜΥ – ΕΜΠ, Ιούλιος 2014
[7] Εκτίμηση και χαρτογράφηση ποιοτικών χαρακτηριστικών σε υδάτινους αποδέκτες με τεχνικές τηλεπισκόπησης: Η Περίπτωση της Λίμνης Κάρλα, Θεολόγου Ιωάννα, ΣΑΤΜ – ΕΜΠ, Μάιος 2014
[8] Leaflet, an open-source JavaScript library for mobile-friendly interactive maps (http://leafletjs.com)

ΠΑΡΑΡΤΗΜΑ

1. Κώδικας C++

1.1 Περιλήψεις Βιβλιοθηκών και Δηλώσεις Μεταβλητών

```
#include <iostream>
#include <fstream>
#include <string>
#include <unistd.h>//UNIX
using namespace std;//UNIX
#include <sys/stat.h>//UNIX
#include "gdal_priv.h"
#include "gdal_alg.h"
#include "gdalwarper.h" //for my_resampling
#include "cpl_conv.h"//for band scanline
#include "cpl_string.h"//for CREATE()
#include <ogr_spatialref.h>//for CREATE()
#include "ogr_core.h"//for polygonize
#include "ogrsf_frmts.h"//for ogr open
#include <ogr_feature.h>//for polygonize fields and features
#include <ctime>//for polygon name
#include "dirent.h"//for getting all files

#ifdef MAX_PATH
static int const maxpath=MAX_PATH;
#else
static int const maxpath=150;
#endif
#define dname_src "sources"//for raw sources
#define dname_pan "panchromatic"//for panchromatic
#define dname_multi "multispectral"//for multispectral
#define dname_in "input"//for hpf-fused
#define dname_out "output"//for vector files
#define fexte ".tif"
#define fexte2 ".TIF"
#define ftype "GTiff"
#define vexte ".geoJSON"//".kml" ".shp"
#define vtype "geoJSON"//"KML" "ESRI Shapefile"
#define vnum 8//4 for kml&shp
```

1.2 Δηλώσεις Συναρτήσεων

```
//pansharpening
GDALDataset *my_pansharpen(char *fname_pan,char *fname_multi,int
winWidth,int winHeight);
GDALDataset *hpf_fusion (GDALDataset *dataset_pan,char
*fname_pan,GDALDataset *dataset_multi,char *fname_multi,int
winWidth,int winHeight);

//pansharpening helpers
GDALDataset *my_resampling(GDALDataset *dataset_pan,GDALDataset
*dataset_multi,char* fname_multi,double memLimit);
void *mean_calc(void *buffer,int bufsizeX,int bufsizeY,GDALDataType
bufDataType);
GDALDataset *hpf_means (GDALDataset *dataset_pan,char *fname_pan,int
winWidth,int winHeight);

//ndvi
GDALDataset *calc_ndvi(char *fname_multi);
GDALDataset *ndvi_threshold(char *fname_src,float threshold);
GDALDataset *ndvi_classification(char *fname_src,float
threshold,float step);

//ndwi & chloro
GDALDataset *calc_ndwi(char *fname_multi);
GDALDataset *calc_chloro(char *fname_multi);
GDALDataset *apply_ndwi_filter(char *fname_chloro,char
*fname_ndwi,float water_thres);
GDALDataset *apply_ndwi_filter(char *fname_chloro,GDALDataset
*dataset_chloro,GDALDataset *dataset_ndwi,float water_thres);
GDALDataset *ndwichlorofiltered_classification(char *fname_src);
GDALDataset *ndwichlorofiltered_classification(GDALDataset
*dataset_src,char *fname_src);

//ndvi & ndwi helper
GDALDataset *calc_ndi(char *fname_multi,char *fname_ndi,int
b_minuend,int b_subtrahend);

//vectorize
OGRDataSourceH single_polygonize(char *fname_src);
int add_fieldval_to_all(char *fname_cur,const char *field_name,const
char *value);
void appendage(char *path_dir);
void append_geojson(OGRDataSourceH dataset_src,OGRDataSourceH
dataset_new);

//overall helpers
int GeoRefTransfer(GDALDataset *dataset_src,GDALDataset
*dataset_dst);
void* bufAlloc(int bufsizeX,int bufsizeY,GDALDataType bufDataType);
```

```
void replaceChar(char *str, char ch1, char ch2);
string get_curdate_str();
string get_date_from_fname_raw(char *fname_cur);
string get_date_from_fpath_raw(char *path_cur);
string get_date_from_fname_refl(char *fname_cur);
string get_date_from_fpath_refl(char *path_cur);
string get_nopath_name(char *dir_fpath);
string get_noext_name(char *full_fname);
```

1.3 Main

```
int main( int argc, char* argv[] ){

  short int flag_pansharpen=1,
flag_chloro=0,flag_ndwi=0,flag_ndwi_filter=0,flag_ndwichloro_class
=0,//mono tis karla
  flag_ndvi=0,flag_ndvithres=0,flag_ndviclass=0,//mono tis
axios1,axios2
  flag_svector=1,flag_mvector=0,
  flag_appendAll=1,
  flag_delete_all_useless=0,
  raw=1;//0 for reflectance

  char answer=' ';
  do{
    if(answer!='\n')
    printf("Pansharpen? Y/N: ");
    answer=getchar();
  }while(answer!='Y'&&answer!='y'&&answer!='N'&&answer!='n');
  if(answer=='Y'||answer=='y'){flag_pansharpen=1;printf("\nPut
same-named {.tif}s in directories '%s' '%s' accordingly.\nTheir
names should follow the pattern
'LC81840322013255LGN00'\n\n",dname_pan,dname_multi);}
  else {
    flag_pansharpen=0;
    printf("\nYou'd better have files (with name pattern
'LC81840322013255LGN00.tif') in the %s directory for this to
work!\n\n",dname_in);
    do{
      if(answer!='\n')
        printf("\nRaw? ('N' for reflectance) Y/N: ");
      answer=getchar();
    }while(answer!='Y'&&answer!='y'&&answer!='N'&&answer!='n');
    if(answer=='N'||answer=='n')raw=0;
  }

  do{
    if(answer!='\n')
      printf("Compute NDVI or NDWI? V/W/N for no: ");
    answer=getchar();
  }while(answer!='V'&&answer!='v'&&answer!='W'&&answer!
='w'&&answer!='N'&&answer!='n');
  if(answer=='V'||answer=='v')
{flag_ndviclass=1;flag_ndwichloro_class=0;}
  else if(answer=='W'||answer=='w')
{flag_ndviclass=0;flag_ndwichloro_class=1;}
  else {flag_ndviclass=0;flag_ndwichloro_class=0;}

  do{
```

```
    if(answer!='\n')
      printf("Delete middle files after exit? Y/N: ");
    answer=getchar();
  }while(answer!='Y'&&answer!='y'&&answer!='N'&&answer!='n');
  if(answer=='Y'||answer=='y')flag_delete_all_useless=1;
  else flag_delete_all_useless=0;

  if(flag_ndwichloro_class)
    flag_ndwi_filter=1;
  if(flag_ndwi_filter){
    flag_ndwi=1;
    flag_chloro=1;
  }

  if(flag_ndviclass)
    flag_ndvithres=1;
  if(flag_ndvithres)
    flag_ndvi=1;

  char
fname_cur[maxpath],fname_temp[maxpath],fname_tempcur[maxpath];
  int winWidth=5,winHeight=5;//for hpf-fusion
  float threshold,step,water_thres;
  if(raw){//for ndvi raw
    threshold=0.2;
    step=0.1;
  }
  else{//for ndvi reflectance
    threshold=0.4;
    step=0.1;
  }
  water_thres=0;//for ndwi raw

  //set directory names
  char path_cur[maxpath];
  getcwd(path_cur, sizeof(path_cur));
  //SetCurrentDirectory(path_cur); //WINDOWS
  if(chdir(path_cur)<0){
    cout<<"WRONG CURRENT DIRECTORY";//UNIX
    return -1;
  }
  char path_dirI[maxpath],path_dirO[maxpath];
  sprintf(path_dirI,"%s/%s",path_cur,dname_in);
  sprintf(path_dirO,"%s/%s",path_cur,dname_out);
  //Register all GDAL drivers
  GDALAllRegister();
  //load driver for tif deletion
  GDALDriver *poDriver = GetGDALDriverManager()-
>GetDriverByName(ftype);
```

```
  //PAN-SHARPENING
  DIR *dir,*dir_multi;
  struct dirent *ent,*ent_multi;

  if(flag_pansharpen){
    char path_dirP[maxpath],path_dirM[maxpath];
    sprintf(path_dirP,"%s/%s",path_cur,dname_pan);
    sprintf(path_dirM,"%s/%s",path_cur,dname_multi);
    if ((dir = opendir (path_dirP)) != NULL) {
      char fname_pan[maxpath],fname_multi[maxpath];
      char fname_pan_path[maxpath],fname_multi_path[maxpath];
      sprintf(fname_multi_path,"fakename");
      char
fname_pan_path_temp[maxpath],fname_multi_path_temp[maxpath];
      while ((ent = readdir (dir)) != NULL) {
        if(strlen(ent->d_name) > 4 && strcmp(ent->d_name + strlen(ent-
>d_name) - 4, fexte) == 0){//if has file extension ".tif"
          //find multi with same name as pan
          string fname = get_noext_name(ent->d_name);
          sprintf(fname_pan_path,"%s/%s
%s",path_dirP,fname.c_str(),fexte);
          sprintf(fname_pan_path_temp,"%s/%span
%s",path_cur,fname.c_str(),fexte);
          if ((dir_multi = opendir (path_dirM)) != NULL) {
            while ((ent_multi = readdir (dir_multi)) != NULL) {
              if(strlen(ent_multi->d_name) > 4 && strcmp(ent_multi-
>d_name + strlen(ent_multi->d_name) - 4, fexte) == 0){//if has file
extension ".tif"
                if(!strcmp(ent->d_name,ent_multi->d_name)){
                  sprintf(fname_multi_path,"%s/%s
%s",path_dirM,fname.c_str(),fexte);
                  sprintf(fname_multi_path_temp,"%s/%smulti
%s",path_cur,fname.c_str(),fexte);
                  break;
                }
              }
            }
            if(!strcmp(fname_multi_path,"fakename")){
              cout<<"MULTISPECTRAL "<<ent->d_name<<" NOT FOUND IN
DIR"<<path_dirM<<endl;
              return 1;
            }
          }
          //move pan&multi to cwd for fusion
          rename(fname_pan_path,fname_pan_path_temp);
          rename(fname_multi_path,fname_multi_path_temp);

strcpy(fname_pan,get_nopath_name(fname_pan_path_temp).c_str());

strcpy(fname_multi,get_nopath_name(fname_multi_path_temp).c_str())
```

```
;
          //pansharpen pan and multi
          GDALDataset *dataset_fusion=NULL;

dataset_fusion=my_pansharpen(fname_pan,fname_multi,winWidth,winHei
ght);
          if(flag_delete_all_useless){
            char fname_multires[maxpath];
            sprintf(fname_multires,"%s_warp%s",
(get_noext_name(fname_multi)).c_str(),fexte);
            GDALDeleteDataset(poDriver,fname_multires);
            char fname_panmean[maxpath];
            sprintf(fname_panmean,"%s_means%s",
(get_noext_name(fname_pan)).c_str(),fexte);
            GDALDeleteDataset(poDriver,fname_panmean);
          }
          if(dataset_fusion==NULL){
            cout<<"PANSHARPENING FAILED"<<endl;
            rename(fname_pan_path_temp,fname_pan_path);
            rename(fname_multi_path_temp,fname_multi_path);
            return 2;
          }
          if( dataset_fusion != NULL )
            GDALClose(dataset_fusion);
          sprintf(fname_cur,"%s_hpf%s",
(get_noext_name(fname_pan)).c_str(),fexte);
          if(flag_ndvi||flag_ndwi){
            if(opendir(path_dirI)==NULL)
              mkdir(path_dirI,S_IRWXU|S_IRWXG|S_IROTH|S_IXOTH);
            sprintf(fname_temp,"%s/%s
%s",path_dirI,fname.c_str(),fexte);//move hpf into "input" directory
with same name as pan&multi
          }
          else{
            if(opendir(path_dirO)==NULL)
              mkdir(path_dirO,S_IRWXU|S_IRWXG|S_IROTH|S_IXOTH);
            sprintf(fname_temp,"%s/%s
%s",path_dirO,fname.c_str(),fexte);//move hpf into "output" directory
with same name as pan&multi
          }
          //send hpf to input (for further manip) or output
          rename(fname_cur,fname_temp);
          //send pan&multi back to their directories
          rename(fname_pan_path_temp,fname_pan_path);
          rename(fname_multi_path_temp,fname_multi_path);
        }
      }
    }
  }
  if(flag_ndvi||flag_ndwi||flag_svector){
```

```
  if ((dir = opendir (path_dirI)) != NULL) {
    chdir(path_dirI);
    char fname_chloro[maxpath];
    GDALDataset *dataset_chloro=NULL;
    GDALDataset *dataset_ndwi=NULL;
    GDALDataset *dataset_filtered=NULL;
    //get all the files and directories within directory
    while ((ent = readdir (dir)) != NULL) {
      if(strlen(ent->d_name) > 4 && strcmp(ent->d_name + strlen(ent-
>d_name) - 4, fexte) == 0){//if has file extension ".tif"
        strcpy(fname_cur,ent->d_name);
        if(flag_chloro){//compute chlorophyl
          dataset_chloro=NULL;
          dataset_chloro=calc_chloro(fname_cur);
          if(dataset_chloro==NULL){
            cout<<"CHLORO CALCULATION FAILED"<<endl;
            closedir (dir);
            return 3;
          }
          if( !flag_ndwi_filter ){//won't need for filtering
            sprintf(fname_tempcur,"%s_chloro%s",
(get_noext_name(fname_cur)).c_str(),fexte);
            GDALClose(dataset_chloro);
            sprintf(fname_temp,"%s/%s",path_dirO,fname_tempcur);
            rename(fname_tempcur,fname_temp);//send to output
          }
        }
        if(flag_ndwi){//compute ndwi
          dataset_ndwi=NULL;
          dataset_ndwi=calc_ndwi(fname_cur);
          if(dataset_ndwi==NULL){
            cout<<"NDWI CALCULATION FAILED"<<endl;
            closedir (dir);
            return 3;
          }
          if( !flag_ndwi_filter ){//won't need for filtering
            sprintf(fname_tempcur,"%s_ndwi%s",
(get_noext_name(fname_cur)).c_str(),fexte);
            GDALClose(dataset_ndwi);
            sprintf(fname_temp,"%s/%s",path_dirO,fname_tempcur);
            rename(fname_tempcur,fname_temp);//send to output
          }
        }
        if(flag_ndwi_filter){//compute ndwi-filter
          sprintf(fname_chloro,"%s_chloro%s",
(get_noext_name(fname_cur)).c_str(),fexte);
          dataset_filtered=NULL;

dataset_filtered=apply_ndwi_filter(fname_chloro,dataset_chloro,dat
```

```
aset_ndwi,water_thres);
         GDALClose(dataset_chloro);
         GDALClose(dataset_ndwi);
         if(dataset_filtered==NULL){
           cout<<"NDWI FILTERING FAILED"<<endl;
           closedir (dir);
           return 3;
         }
         if(flag_delete_all_useless){
           GDALDeleteDataset(poDriver,fname_chloro);
           sprintf(fname_temp,"%s_ndwi%s",
(get_noext_name(fname_cur)).c_str(),fexte);
           GDALDeleteDataset(poDriver,fname_temp);
         }
         sprintf(fname_tempcur,"%s_ndwifilter%s",
(get_noext_name(fname_chloro)).c_str(),fexte);
         if( !flag_ndwichloro_class ){//won't need for
classification
           GDALClose(dataset_filtered);
           sprintf(fname_temp,"%s/%s",path_dirO,fname_tempcur);
           rename(fname_tempcur,fname_temp);//send to output
         }
         else{//make current
           if(flag_delete_all_useless)
             rename(fname_tempcur,fname_cur);
           else{
             memset(fname_cur,0,sizeof(fname_cur));
             strcpy(fname_cur,fname_tempcur);
           }
         }
       }
       if(flag_ndwichloro_class){//classify ndwi-chloro filtered
         GDALDataset *dataset_ndwiclass=NULL;
         if(flag_ndwi_filter){//ndwi-filter is open

dataset_ndwiclass=ndwichlorofiltered_classification(dataset_filter
ed,fname_cur);
           GDALClose(dataset_filtered);
         }
         else{//must open ndwi-filter

dataset_ndwiclass=ndwichlorofiltered_classification(fname_cur);
         }
         if(dataset_ndwiclass==NULL){
           cout<<"NDWI-FILTER CLASSIFICATION FAILED"<<endl;
           closedir (dir);
           return 3;
         }
         GDALClose(dataset_ndwiclass);
         if(flag_delete_all_useless)
```

```
          GDALDeleteDataset(poDriver,fname_cur);
        sprintf(fname_tempcur,"%s_classw%s",
(get_noext_name(fname_cur)).c_str(),fexte);
        if(!flag_svector){//send to output
          sprintf(fname_temp,"%s/%s",path_dir0,fname_tempcur);
          rename(fname_tempcur,fname_temp);
        }
        else{//make current
          if(flag_delete_all_useless)
            rename(fname_tempcur,fname_cur);
          else{
            memset(fname_cur,0,sizeof(fname_cur));
            strcpy(fname_cur,fname_tempcur);
          }
        }
      }
      if(flag_ndvi){//compute ndvi
        strcpy(fname_cur,ent->d_name);
        GDALDataset *dataset_ndvi=NULL;
        dataset_ndvi=calc_ndvi(fname_cur);
        if(dataset_ndvi==NULL){
          cout<<"NDVI CALCULATION FAILED"<<endl;
          closedir (dir);
          return 3;
        }
        if( dataset_ndvi != NULL )
          GDALClose(dataset_ndvi);
        //reset name for consequent thresholding
        sprintf(fname_tempcur,"%s_ndvi%s",
(get_noext_name(fname_cur)).c_str(),fexte);
        if(!flag_ndvithres){//send to output
          sprintf(fname_temp,"%s/%s",path_dir0,fname_tempcur);
          rename(fname_tempcur,fname_temp);
        }
        else{//make current
          if(flag_delete_all_useless)
            rename(fname_tempcur,fname_cur);
          else{
            memset(fname_cur,0,sizeof(fname_cur));
            strcpy(fname_cur,fname_tempcur);
          }
        }
      }
      if(flag_ndvithres){//thresholding
        GDALDataset *dataset_thres=NULL;
        dataset_thres=ndvi_threshold(fname_cur,threshold);
        if ( dataset_thres == NULL ) {
          cout<<"THRESHOLDING FAILED"<<endl;
          closedir (dir);
```

```
          return 4;
        }
        if( dataset_thres != NULL )
          GDALClose(dataset_thres);
        //reset name for consequent classification
        if(flag_delete_all_useless)
          GDALDeleteDataset(poDriver,fname_cur);
        sprintf(fname_tempcur,"%s_thresv%s",
(get_noext_name(fname_cur)).c_str(),fexte);
        if(!flag_ndviclass){//send to output
          sprintf(fname_temp,"%s/%s",path_dir0,fname_tempcur);
          rename(fname_tempcur,fname_temp);
        }
        else{//make current
          if(flag_delete_all_useless)
            rename(fname_tempcur,fname_cur);
          else{
            memset(fname_cur,0,sizeof(fname_cur));
            strcpy(fname_cur,fname_tempcur);
          }
        }
      }
      if(flag_ndviclass){//classification
        GDALDataset *dataset_class=NULL;

dataset_class=ndvi_classification(fname_cur,threshold,step);
        if ( dataset_class == NULL ) {
          cout<<"CLASSIFICATION FAILED"<<endl;
          closedir (dir);
          return 5;
        }
        if( dataset_class != NULL )
          GDALClose(dataset_class);
        //reset name for consequent vectorization
        if(flag_delete_all_useless)
          GDALDeleteDataset(poDriver,fname_cur);
        sprintf(fname_tempcur,"%s_classv%s",
(get_noext_name(fname_cur)).c_str(),fexte);
        if(!flag_svector){//send to output
          sprintf(fname_temp,"%s/%s",path_dir0,fname_tempcur);
          rename(fname_tempcur,fname_temp);
        }
        else{//make current
          if(flag_delete_all_useless)
            rename(fname_tempcur,fname_cur);
          else{
            memset(fname_cur,0,sizeof(fname_cur));
            strcpy(fname_cur,fname_tempcur);
          }
```

```
        }
      }
      if(flag_svector){//vectorization
        //check if "output" directory exists
        if(opendir(path_dirO)==NULL)
          mkdir(path_dirO,S_IRWXU|S_IRWXG|S_IROTH|S_IXOTH);
        //get vector file's name
        sprintf(fname_tempcur,"%s
%s",get_noext_name(fname_cur).c_str(),vexte);
        sprintf(fname_temp,"%s/%s",path_dirO,fname_tempcur);
        //check if file already exists
        ifstream infile(fname_temp,ifstream::in);
        if (infile.good()){
          cout<<fname_tempcur<<" EXISTS"<<endl;
          infile.close();
        }
        else{
          OGRDataSourceH hDS=NULL;
          hDS=single_polygonize(fname_cur);
          if( hDS == NULL ){
            cout<<fname_cur<<" VECTORIZATION FAILED"<<endl;
            closedir (dir);
            return 6;
          }
          OGR_DS_Destroy(hDS);
          rename(fname_tempcur,fname_temp);
          chdir(path_dirO);
          if(raw)

add_fieldval_to_all(fname_tempcur,"Date",get_date_from_fname_raw(f
name_tempcur).c_str());//raw
          else

add_fieldval_to_all(fname_tempcur,"Date",get_date_from_fname_refl(
fname_tempcur).c_str());//refl
          chdir(path_dirI);
        }
        if(flag_delete_all_useless)
          GDALDeleteDataset(poDriver,fname_cur);
      }
    }
  }
  closedir (dir);
 }
 else {
  /* could not open directory */
  cerr<<"NO INPUT DIRECTORY "<<path_dirI<<endl;
  return -1;
 }
 }
```

```
  if(flag_delete_all_useless)
    rmdir(path_dirI);
  if(flag_appendAll){
    chdir(path_cur);
    appendage(path_dirO);
  }
  return 0;
}
```

1.4 my_pansharpen & hpf-fusion

```
GDALDataset *my_pansharpen(char *fname_pan,char *fname_multi,int
winWidth,int winHeight){
  //open panchromatic and multispectral
  GDALDataset *dataset_pan=NULL, *dataset_multi=NULL;
  dataset_pan = (GDALDataset*)GDALOpen(fname_pan, GA_ReadOnly);
  if ( dataset_pan == NULL ) {
    cout<<"NO PANCHROMATIC IMAGE"<<endl;
    return NULL;
  }
  dataset_multi = (GDALDataset*)GDALOpen(fname_multi, GA_ReadOnly);
  if ( dataset_multi == NULL ) {
    cout<<"NO MULTISPECTRAL IMAGE"<<endl;
    if( dataset_pan != NULL )
      GDALClose(dataset_pan);
    return NULL;
  }
  //hpf-fusion
  GDALDataset *dataset_fusion = hpf_fusion
(dataset_pan,fname_pan,dataset_multi,fname_multi,winWidth,winHeigh
t);
  if(dataset_fusion==NULL){
    cout<<"HPF-FUSION FAILED"<<endl;
  }
  if( dataset_pan != NULL )
    GDALClose(dataset_pan);
  if( dataset_multi != NULL )
    GDALClose(dataset_multi);
  return dataset_fusion;
}
```

```
GDALDataset *hpf_fusion (GDALDataset *dataset_pan,char
*fname_pan,GDALDataset *dataset_multi,char *fname_multi,int
winWidth,int winHeight){
  GDALDataset *dataset_hpf=NULL;
  char fname_hpf[maxpath];
  sprintf(fname_hpf,"%s_hpf%s",
(get_noext_name(fname_pan)).c_str(),fexte);
  ifstream infile(fname_hpf,ifstream::in);
  if (infile.good()){
    dataset_hpf=(GDALDataset*)GDALOpen(fname_hpf, GA_ReadOnly);
    return dataset_hpf;
  }
  //get multispectrals resample with panchromatic's dimensions
  double memLimit = 131072;
  GDALDataset *dataset_multires=NULL;

dataset_multires=my_resampling(dataset_pan,dataset_multi,fname_mul
ti,memLimit);
  if(dataset_multires==NULL){
    cout<<fname_multi<<" NOT RESAMPLED"<<endl;
    return NULL;
  }
  //get panchromatic's local means image
  GDALDataset *dataset_panmean=NULL;

dataset_panmean=hpf_means(dataset_pan,fname_pan,winWidth,winHeight
);
  if(dataset_panmean==NULL){
    cout<<fname_pan<<" LOCAL MEANS NOT COMPUTED"<<endl;
    return NULL;
  }
  //prepare for hpf fusion
  char **hpfOptions = NULL;
  GDALDriver *poDriver;
  poDriver = GetGDALDriverManager()->GetDriverByName(ftype);
  if( poDriver == NULL ){
    cout<<"UNSUPPORTED DRIVER"<<endl;
    return NULL;
  }
  char **hpfMetadata = poDriver->GetMetadata();
  if( !CSLFetchBoolean( hpfMetadata, GDAL_DCAP_CREATE, FALSE ) ){
     cout<<"Driver "<<ftype<<" doesn't support Create()
method."<<endl;
    return NULL;
  }
  GDALRasterBand *band_pan;
  band_pan=dataset_pan->GetRasterBand(1);
  GDALDataType bufDataType = band_pan->GetRasterDataType();
  dataset_hpf = poDriver->Create(fname_hpf,dataset_pan-
>GetRasterXSize(),dataset_pan->GetRasterYSize(),dataset_multi-
```

```
>GetRasterCount(),bufDataType,hpfOptions);
  if(dataset_hpf==NULL){
    cout<<"COULD NOT CREATE "<<fname_hpf<<endl;
    return NULL;
  }
  //make one buffer for each image's pixel value
  void *buffer_pan,*buffer_panmean,*buffer_multires,*buffer_hpf;
  buffer_pan=bufAlloc(1,1,bufDataType);
  buffer_panmean=bufAlloc(1,1,bufDataType);
  buffer_multires=bufAlloc(1,1,bufDataType);
  buffer_hpf=bufAlloc(1,1,bufDataType);
  //fusion
  GDALRasterBand *band_panmean,*band_multires,*band_hpf;
  band_panmean=dataset_panmean->GetRasterBand(1);
  for(int b=1;b<=dataset_multires->GetRasterCount();b++){//for all
bands
    band_multires=dataset_multires->GetRasterBand(b);
    band_hpf=dataset_hpf->GetRasterBand(b);
    for(int i=0;i<dataset_pan->GetRasterXSize();i++){//for all width
      //cout<<b<<":"<<i<<endl;
      for(int j=0;j<dataset_pan->GetRasterYSize();j++){//for all
height
       //read pixel values from panchromatic, panchromatic's local
means and resampled multispectral
       band_pan-
>RasterIO(GF_Read,i,j,1,1,buffer_pan,1,1,bufDataType,0,0);
       band_panmean-
>RasterIO(GF_Read,i,j,1,1,buffer_panmean,1,1,bufDataType,0,0);
       band_multires-
>RasterIO(GF_Read,i,j,1,1,buffer_multires,1,1,bufDataType,0,0);
       //buffer_hpf=buffer_multires+(buffer_pan-buffer_panmean)
       if(bufDataType==GDT_Byte)
         if(*(GByte*)buffer_multires==0||
*(GByte*)buffer_multires==band_multires->GetNoDataValue(NULL))
           *(GByte*)buffer_hpf=0;
         else
           *(GByte*)buffer_hpf=*(GByte*)buffer_multires+
(*(GByte*)buffer_pan-*(GByte*)buffer_panmean);
       else if(bufDataType==GDT_Int16)
         if(*(GInt16*)buffer_multires==0||
*(GInt16*)buffer_multires==band_multires->GetNoDataValue(NULL))
           *(GInt16*)buffer_hpf=0;
         else
           *(GInt16*)buffer_hpf=*(GInt16*)buffer_multires+
(*(GInt16*)buffer_pan-*(GInt16*)buffer_panmean);
       else if(bufDataType==GDT_UInt16)
         if(*(GUInt16*)buffer_multires==0||
*(GUInt16*)buffer_multires==band_multires->GetNoDataValue(NULL))
           *(GUInt16*)buffer_hpf=0;
         else
```

```
          *(GUInt16*)buffer_hpf=*(GUInt16*)buffer_multires+
(*(GUInt16*)buffer_pan-*(GUInt16*)buffer_panmean);
       else if(bufDataType==GDT_Int32)
         if(*(GInt32*)buffer_multires==0||
*(GInt32*)buffer_multires==band_multires->GetNoDataValue(NULL))
          *(GInt32*)buffer_hpf=0;
         else
          *(GInt32*)buffer_hpf=*(GInt32*)buffer_multires+
(*(GInt32*)buffer_pan-*(GInt32*)buffer_panmean);
       else if (bufDataType==GDT_UInt32)
         if(*(GUInt32*)buffer_multires==0||
*(GUInt32*)buffer_multires==band_multires->GetNoDataValue(NULL))
          *(GUInt32*)buffer_hpf=0;
         else
          *(GUInt32*)buffer_hpf=*(GUInt32*)buffer_multires+
(*(GUInt32*)buffer_pan-*(GUInt32*)buffer_panmean);
       else
         if(*(GIntBig*)buffer_multires==0||
*(GIntBig*)buffer_multires==band_multires->GetNoDataValue(NULL))
          *(GIntBig*)buffer_hpf=0;
         else
          *(GIntBig*)buffer_hpf=*(GIntBig*)buffer_multires+
(*(GIntBig*)buffer_pan-*(GIntBig*)buffer_panmean);
       //write resulting pixel value on hpf image
       band_hpf-
>RasterIO(GF_Write,i,j,1,1,buffer_hpf,1,1,bufDataType,0,0);
      }
    }
  }
  //transfer georeference
  if(GeoRefTransfer(dataset_pan,dataset_hpf)>0){
    cout<<"GEOTRANSFER ERROR"<<endl; return NULL;
  }
  //close resample and local_means datasets
  if( dataset_multires != NULL )
    GDALClose(dataset_multires);
  if( dataset_panmean != NULL )
    GDALClose(dataset_panmean);
  return dataset_hpf;
}
```

1.5 hpf_means & mean_calc (υπολογισμός τοπικών μέσων)

```
GDALDataset *hpf_means (GDALDataset *dataset_pan,char *fname_pan,int
winWidth,int winHeight){
  GDALDataset *dataset_panmean=NULL;

  //prepare for resampling
  char **meansOptions = NULL,fname_panmean[maxpath];
  sprintf(fname_panmean,"%s_means%s",
(get_noext_name(fname_pan)).c_str(),fexte);
  ifstream infile(fname_panmean,ifstream::in);
  if (infile.good()){
    dataset_panmean=(GDALDataset*)GDALOpen(fname_panmean,
GA_ReadOnly);
    return dataset_panmean;
  }

  GDALDriver *poDriver;
  poDriver = GetGDALDriverManager()->GetDriverByName(ftype);
  if( poDriver == NULL ){
    cout<<"UNSUPPORTED DRIVER"<<endl;
    return NULL;
  }
  char **meansMetadata = poDriver->GetMetadata();
  if( !CSLFetchBoolean( meansMetadata, GDAL_DCAP_CREATE, FALSE ) ){
    cout<<"Driver "<<ftype<<" doesn't support Create()
method."<<endl;
    return NULL;
  }
  GDALRasterBand *band_pan, *band_means;
  band_pan=dataset_pan->GetRasterBand(1);
  GDALDataType bufDataType = band_pan->GetRasterDataType();
  dataset_panmean = poDriver->Create(fname_panmean,dataset_pan-
>GetRasterXSize(),dataset_pan-
>GetRasterYSize(),1,bufDataType,meansOptions);
  if(dataset_panmean==NULL){
    cout<<"COULD NOT CREATE "<<fname_panmean<<endl;
    return NULL;
  }

  //buffers for calculating window and resulting pixel value
  void *buffer, *local_mean;
  buffer=bufAlloc(winWidth,winHeight,bufDataType);
  local_mean=bufAlloc(1,1,bufDataType);
  band_means=dataset_panmean->GetRasterBand(1);

  //initialize crop parameters
  int
cropOffsetX=0,cropOffsetY=0,cropWidthL,cropWidthR,cropHeightU,crop
```

```
HeightD;
  winWidth/=2;winHeight/=2;
  //for all width
  for(int i=0;i<dataset_pan->GetRasterXSize();i++){
    //update crop width parameters
    if(i-winWidth<0){
      cropOffsetX=0;
      cropWidthL=i;
    }
    else{
      cropOffsetX=i-winWidth;
      cropWidthL=winWidth;
    }
    if(i+winWidth>=dataset_pan->GetRasterXSize())
      cropWidthR=dataset_pan->GetRasterXSize()-1-i;
    else
      cropWidthR=winWidth;
    //for all height
    for(int j=0;j<dataset_pan->GetRasterYSize();j++){
      //update crop height parameters
      if(j-winHeight<0){
        cropOffsetY=0;
        cropHeightU=j;
      }
      else{
        cropOffsetY=j-winHeight;
        cropHeightU=winHeight;
      }
      if(j+winHeight>=dataset_pan->GetRasterYSize())
        cropHeightD=dataset_pan->GetRasterYSize()-1-j;
      else
        cropHeightD=winHeight;
      //read window
      band_pan-
>RasterIO(GF_Read,cropOffsetX,cropOffsetY,cropWidthL+cropWidthR+1,
cropHeightU+cropHeightD+1,buffer,cropWidthL+cropWidthR+1,cropHeigh
tU+cropHeightD+1,bufDataType,0,0);
      //calculate local mean of window

local_mean=mean_calc(buffer,cropWidthL+cropWidthR+1,cropHeightU+cr
opHeightD+1,bufDataType);
      //write resulting pixel value
      band_means-
>RasterIO(GF_Write,i,j,1,1,local_mean,1,1,bufDataType,0,0);
      //nullify buffers
      memset(buffer,0,sizeof(buffer));
      memset(local_mean,0,sizeof(local_mean));
    }
  }
  //transfer georeference
```

```
  if(GeoRefTransfer(dataset_pan,dataset_panmean)>0){
    cout<<"GEOTRANSFER ERROR"<<endl;
    return NULL;
  }
  //free buffer memories
  CPLFree(buffer);
  CPLFree(local_mean);
  return dataset_panmean;
}
```

```
void *mean_calc(void *buffer,int bufsizeX,int bufsizeY,GDALDataType
bufDataType){
  //buffer for advancing sum and initialize as 0
  void *sum;
  sum=bufAlloc(1,1,GDT_UInt32);
  memset(sum,0,sizeof(sum));
  for(int i=0;i<bufsizeX;i++){//for all width
    for(int j=0;j<bufsizeY;j++){//for all height
      //sum+=buffer[i][j]
      if(bufDataType==GDT_Byte)
        *(GUInt32*)sum+=((GByte*)buffer)[i*bufsizeX+j];
      else if(bufDataType==GDT_Int16)
        *(GUInt32*)sum+=((GInt16*)buffer)[i*bufsizeX+j];
      else if(bufDataType==GDT_UInt16)
        *(GUInt32*)sum+=((GUInt16*)buffer)[i*bufsizeX+j];
      else if(bufDataType==GDT_Int32)
        *(GUInt32*)sum+=((GInt32*)buffer)[i*bufsizeX+j];
      else if (bufDataType==GDT_UInt32)
        *(GUInt32*)sum+=((GUInt32*)buffer)[i*bufsizeX+j];
      else
        *(GUInt32*)sum+=((GIntBig*)buffer)[i*bufsizeX+j];
    }
  }
  //buffer for local mean pixel value
  void *local_mean;
  local_mean=bufAlloc(1,1,bufDataType);
  //local_mean=sum/(windowX*windowY)
  if(bufDataType==GDT_Byte)
    *(GByte*)local_mean = *(GUInt32*)sum/(GByte)(bufsizeX*bufsizeY);
  else if(bufDataType==GDT_Int16)
    *(GInt16*)local_mean = *(GUInt32*)sum/(GInt16)
(bufsizeX*bufsizeY);
  else if(bufDataType==GDT_UInt16)
    *(GUInt16*)local_mean = *(GUInt32*)sum/(GUInt16)
(bufsizeX*bufsizeY);
  else if(bufDataType==GDT_Int32)
    *(GInt32*)local_mean = *(GUInt32*)sum/(GInt32)
(bufsizeX*bufsizeY);
  else if (bufDataType==GDT_UInt32)
     *(GUInt32*)local_mean = *(GUInt32*)sum/(GUInt32)
(bufsizeX*bufsizeY);
  else
    *(GIntBig*)local_mean = *(GUInt32*)sum/(GIntBig)
(bufsizeX*bufsizeY);
  //free sum buffer memory
  CPLFree(sum);
  return local_mean;
}
```

1.6 my_resampling

```
GDALDataset *my_resampling(GDALDataset *dataset_pan,GDALDataset
*dataset_multi, char* fname_multi,double memLimit){
  GDALDataset *dataset_multires=NULL;
  //prepare for resampling
  char **createOptions = NULL;
  char fname_multires[maxpath];
  sprintf(fname_multires,"%s_warp%s",
(get_noext_name(fname_multi)).c_str(),fexte);
  ifstream infile(fname_multires,ifstream::in);
  if (infile.good()){
    dataset_multires=(GDALDataset*)GDALOpen(fname_multires,
GA_ReadOnly);
    return dataset_multires;
  }
  GDALDriver *poDriver = GetGDALDriverManager()-
>GetDriverByName(ftype);
  char **metadata = poDriver->GetMetadata();
  if( !CSLFetchBoolean( metadata, GDAL_DCAP_CREATE, FALSE ) ){
    cout<<"Driver "<<ftype<<" doesn't support Create()
method."<<endl;
    return NULL;
  }
  GDALRasterBand *band_pan = dataset_pan->GetRasterBand(1);
  dataset_multires = poDriver->Create(fname_multires,dataset_pan-
>GetRasterXSize(),dataset_pan->GetRasterYSize(),dataset_multi-
>GetRasterCount(),band_pan->GetRasterDataType(),createOptions);
  if(dataset_multires==NULL){
    cout<<"COULD NOT CREATE "<<fname_multires<<endl;
    return NULL;
  }
  //Set Warp Options
  GDALResampleAlg resampleAlg=GRA_Average;
  GDALWarpOptions *psWarpOptions = GDALCreateWarpOptions();
  psWarpOptions->hSrcDS = dataset_multi;
  psWarpOptions->hDstDS = dataset_multires;
  psWarpOptions->nBandCount = dataset_multi->GetRasterCount();
  psWarpOptions->dfWarpMemoryLimit = memLimit;
  psWarpOptions->eResampleAlg = resampleAlg;
  psWarpOptions->eWorkingDataType = band_pan->GetRasterDataType();
  psWarpOptions->panSrcBands =
(int*)CPLMalloc(sizeof(int)*psWarpOptions->nBandCount);
  psWarpOptions->panDstBands =
(int*)CPLMalloc(sizeof(int)*psWarpOptions->nBandCount);
  for(int i=0;i<psWarpOptions->nBandCount;i++){
    psWarpOptions->panSrcBands[i]=i+1;
    psWarpOptions->panDstBands[i]=i+1;
  }
```

```
  psWarpOptions->pTransformerArg = GDALCreateGenImgProjTransformer2(
dataset_multi,dataset_pan,NULL );
  psWarpOptions->pfnTransformer = GDALGenImgProjTransform;
  //Warp Operation
  GDALWarpOperation oOperation;
  oOperation.Initialize( psWarpOptions );
  oOperation.ChunkAndWarpImage( 0,0,dataset_multires-
>GetRasterXSize(),dataset_multires->GetRasterYSize() );
  //transfer georeference
  if(GeoRefTransfer( dataset_pan,dataset_multires )>0){
    cout<<"GEOTRANSFER ERROR"<<endl;
  }
  //close operation
  GDALDestroyGenImgProjTransformer( psWarpOptions->pTransformerArg
);
  GDALDestroyWarpOptions( psWarpOptions );
  return dataset_multires;
}
```

1.7 calc_ndi

```
GDALDataset *calc_ndi(char *fname_multi,char *fname_ndi,int
b_minuend,int b_subtrahend){
  GDALDataset *dataset_ndi=NULL;
  ifstream infile(fname_ndi,ifstream::in);
  if (infile.good()){
    cout<<fname_ndi<<" EXISTS"<<endl;
    dataset_ndi=(GDALDataset*)GDALOpen(fname_ndi, GA_ReadOnly);
    return dataset_ndi;
  }
  GDALDataset *dataset_multi = (GDALDataset*)GDALOpen(fname_multi,
GA_ReadOnly);
  if ( dataset_multi == NULL ) {
    cout<<"NO MULTISPECTRAL IMAGE"<<endl;return NULL;}
  //get minuend and subtrahend bands
  GDALRasterBand *band_sub,*band_min;
  band_sub=dataset_multi->GetRasterBand(b_subtrahend);
  band_min=dataset_multi->GetRasterBand(b_minuend);
  if(band_sub->GetRasterDataType()!=GDT_UInt16 && band_sub-
>GetRasterDataType()!=GDT_Int16){
    cout<<"INVALID DATASET DATATYPE ("<<band_sub-
>GetRasterDataType()<<")"<<endl;
    if( dataset_multi != NULL )
      GDALClose(dataset_multi);
    return NULL;
  }
  //Create New NDI File with Float32 dataType
  GDALDataType bufDataType=band_sub->GetRasterDataType();
  char **createOptions = NULL;
  GDALDriver *poDriver = GetGDALDriverManager()-
>GetDriverByName(ftype);
  char **metadata = poDriver->GetMetadata();
  if( !CSLFetchBoolean( metadata, GDAL_DCAP_CREATE, FALSE ) ){
    cout<<"Driver "<<ftype<<" doesn't support Create()
method."<<endl;
    if( dataset_multi != NULL )
      GDALClose(dataset_multi);
    return NULL;
  }
  dataset_ndi = poDriver->Create(fname_ndi,dataset_multi-
>GetRasterXSize(),dataset_multi-
>GetRasterYSize(),1,GDT_Float32,createOptions);
  if(dataset_ndi==NULL){
    cout<<"COULD NOT CREATE "<<fname_ndi<<endl;
    if( dataset_multi != NULL )GDALClose(dataset_multi);
    return NULL;
  }
  //allocate buffers based on dataType
```

```
  GDALRasterBand *band_ndi;
  band_ndi=dataset_ndi->GetRasterBand(1);
  GInt32 ndi_lower;//has to be bigger because it's used as sum
  void *min_buf,*sub_buf;
  min_buf=bufAlloc(1,1,bufDataType);
  sub_buf=bufAlloc(1,1,bufDataType);
  GInt16 ndi_upper;
  float ndi;
  for(int i=0;i<dataset_multi->GetRasterXSize();i++){//for all band
width
    for(int j=0;j<dataset_multi->GetRasterYSize();j++){//for all band
height
      //read (i,j) values from minuend and subtrahend
      band_min-
>RasterIO(GF_Read,i,j,1,1,min_buf,1,1,bufDataType,0,0);
      band_sub-
>RasterIO(GF_Read,i,j,1,1,sub_buf,1,1,bufDataType,0,0);
      //compute ndi
      if(bufDataType==GDT_UInt16){
        ndi_lower=(GInt32)(*(GUInt16*)min_buf+*(GUInt16*)sub_buf);
        ndi_upper=(GInt16)(*(GUInt16*)min_buf-*(GUInt16*)sub_buf);
        if (ndi_lower!=0)
          ndi=ndi_upper/ (float)ndi_lower;
        else
           ndi=0;
      }
      else if(bufDataType==GDT_Int16){
        ndi_lower=(GInt32)(*(GInt16*)min_buf+*(GInt16*)sub_buf);
        ndi_upper=(*(GInt16*)min_buf-*(GInt16*)sub_buf);
        if (ndi_lower!=0) ndi=ndi_upper/ (float)ndi_lower;
        else ndi=0;
      }
      else ndi=0;
      //write result
      band_ndi-
>RasterIO(GF_Write,i,j,1,1,&ndi,1,1,GDT_Float32,0,0);
    }
  }
  //transfer georeference
  if(GeoRefTransfer( dataset_multi,dataset_ndi )>0)
    cout<<"GEOTRANSFER ERROR"<<endl;
  if( dataset_multi != NULL )GDALClose(dataset_multi);
  return dataset_ndi;
}
```

1.8 calc_ndvi & ndvi_threshold

```
GDALDataset *calc_ndvi(char *fname_multi){
  char fname_ndvi[maxpath];
  sprintf(fname_ndvi,"%s_ndvi%s",
(get_noext_name(fname_multi)).c_str(),fexte);
  return calc_ndi(fname_multi,fname_ndvi,5,4);
}

GDALDataset *ndvi_threshold(char *fname_src,float threshold){
  //check if thres exists
  GDALDataset *dataset_thres=NULL;
  char fname_thres[maxpath];
  sprintf(fname_thres,"%s_thresv%s",
(get_noext_name(fname_src)).c_str(),fexte);
  ifstream infile(fname_thres,ifstream::in);
  if (infile.good()){
    cout<<fname_thres<<" EXISTS"<<endl;
    dataset_thres=(GDALDataset*)GDALOpen(fname_thres, GA_ReadOnly);
    return dataset_thres;
  }
  //open src dataset
  GDALDataset *dataset_src=NULL;
  dataset_src = (GDALDataset*)GDALOpen(fname_src, GA_ReadOnly);
  if ( dataset_src == NULL ) {
    cout<<fname_src<<" IMAGE NOT FOUND"<<endl;
    return NULL;
  }
  //Create New NDVI File
  GDALRasterBand *band_src=dataset_src->GetRasterBand(1);
  if(band_src->GetRasterDataType()!=GDT_Float32){
    cout<<"INVALID DATASET DATATYPE ("<<band_src-
>GetRasterDataType()<<")"<<endl;
    return NULL;
  }
  char **createOptions = NULL;
  GDALDriver *poDriver = GetGDALDriverManager()-
>GetDriverByName(ftype);
  char **metadata = poDriver->GetMetadata();
  if( !CSLFetchBoolean( metadata, GDAL_DCAP_CREATE, FALSE ) ){
    cout<<"Driver "<<ftype<<" doesn't support Create()
method."<<endl;
    if( dataset_src != NULL )
      GDALClose(dataset_src);
    return NULL;
  }
  dataset_thres = poDriver->Create(fname_thres,dataset_src-
>GetRasterXSize(),dataset_src-
>GetRasterYSize(),1,GDT_Float32,createOptions);
```

```
  if(dataset_thres==NULL){
    cout<<"COULD NOT CREATE "<<fname_thres<<endl;
    if( dataset_src != NULL )
      GDALClose(dataset_src);
    return NULL;
  }
  GDALRasterBand *band_thres=dataset_thres->GetRasterBand(1);
  float pixel;
  //apply threshold
  for(int i=0;i<dataset_src->GetRasterXSize();i++){//for all band
width
    for(int j=0;j<dataset_src->GetRasterYSize();j++){//for all band
height
      band_src-
>RasterIO(GF_Read,i,j,1,1,&pixel,1,1,GDT_Float32,0,0);//read source
(i,j) value
      //if outside the thresholds, nullify; max_threshold must be
of positive value (negative for only min_threshold)
      if(pixel<threshold)
         pixel=0;
      band_thres-
>RasterIO(GF_Write,i,j,1,1,&pixel,1,1,GDT_Float32,0,0);//write
pixel value to resulting band
    }
  }
  //transfer georeference
  if(GeoRefTransfer( dataset_src,dataset_thres )>0){
    cout<<"GEOTRANSFER ERROR"<<endl;
  }
  if( dataset_src != NULL )
    GDALClose(dataset_src);
  return dataset_thres;
}
```

1.9 ndvi_classification

```
GDALDataset *ndvi_classification(char *fname_src,float
threshold,float step){
  char fname_class[maxpath];
  sprintf(fname_class,"%s_classv%s",
(get_noext_name(fname_src)).c_str(),fexte);
  GDALDataset *dataset_class=NULL;
  ifstream infile(fname_class,ifstream::in);
  if (infile.good()){
    cout<<fname_class<<" EXISTS"<<endl;
    dataset_class=(GDALDataset*)GDALOpen(fname_class, GA_ReadOnly);
    return dataset_class;
  }
  GDALDataset *dataset_src=NULL;
  dataset_src = (GDALDataset*)GDALOpen(fname_src, GA_ReadOnly);
  if ( dataset_src == NULL ) {
    cout<<fname_src<<" IMAGE NOT FOUND"<<endl;
    return NULL;
  }
  //Create New NDVI File
  GDALRasterBand *band_src=dataset_src->GetRasterBand(1);
  if(band_src->GetRasterDataType()!=GDT_Float32){
    cout<<"INVALID DATASET DATATYPE ("<<band_src-
>GetRasterDataType()<<")"<<endl;
    return NULL;
  }
  char **createOptions = NULL;
  GDALDriver *poDriver = GetGDALDriverManager()-
>GetDriverByName(ftype);
  char **metadata = poDriver->GetMetadata();
  if( !CSLFetchBoolean( metadata, GDAL_DCAP_CREATE, FALSE ) ){
    cout<<"Driver "<<ftype<<" doesn't support Create()
method."<<endl;
    if( dataset_src != NULL )
      GDALClose(dataset_src);
    return NULL;
  }
  dataset_class = poDriver->Create(fname_class,dataset_src-
>GetRasterXSize(),dataset_src-
>GetRasterYSize(),1,GDT_UInt16,createOptions);
  if(dataset_class==NULL){
    cout<<"COULD NOT CREATE "<<fname_class<<endl;
    if( dataset_src != NULL )
      GDALClose(dataset_src);
    return NULL;
  }
  GDALRasterBand *band_thres=dataset_class->GetRasterBand(1);
  float pixelI,clas=threshold;
```

```
  GUInt16 pixelO;
  //apply threshold
  for(int i=0;i<dataset_src->GetRasterXSize();i++){//for all band
width
    for(int j=0;j<dataset_src->GetRasterYSize();j++){//for all band
height
      band_src-
>RasterIO(GF_Read,i,j,1,1,&pixelI,1,1,GDT_Float32,0,0);//read
source (i,j) value
      //if outside the thresholds, nullify; max_threshold must be
of positive value (negative for only min_threshold)
      pixelO=0;
      if(pixelI>threshold){
        clas=threshold;
        while(pixelI>=clas)
          clas+=step;
        pixelO=(clas-step-threshold)*10+1.1;
      }
      band_thres-
>RasterIO(GF_Write,i,j,1,1,&pixelO,1,1,GDT_UInt16,0,0);//write
pixel value to resulting band
    }
  }
  //transfer georeference
  if(GeoRefTransfer( dataset_src,dataset_class )>0){
    cout<<"GEOTRANSFER ERROR"<<endl;
  }
  if( dataset_src != NULL )
    GDALClose(dataset_src);
  return dataset_class;
}
```

1.10 calc_ndwi & ndwi_chloro

```
GDALDataset *calc_ndwi(char *fname_multi){
  char fname_ndwi[maxpath];
  sprintf(fname_ndwi,"%s_ndwi%s",
(get_noext_name(fname_multi)).c_str(),fexte);
  return calc_ndi(fname_multi,fname_ndwi,3,5);
}

GDALDataset *calc_chloro(char *fname_multi){
  char fname_chloro[maxpath];
  sprintf(fname_chloro,"%s_chloro%s",
(get_noext_name(fname_multi)).c_str(),fexte);
  GDALDataset *dataset_chloro=NULL;
  ifstream infile(fname_chloro,ifstream::in);
  if (infile.good()){
    cout<<fname_chloro<<" EXISTS"<<endl;
    dataset_chloro=(GDALDataset*)GDALOpen(fname_chloro,
GA_ReadOnly);
    return dataset_chloro;
  }
  GDALDataset *dataset_multi = (GDALDataset*)GDALOpen(fname_multi,
GA_ReadOnly);
  if ( dataset_multi == NULL ) {
    cout<<"NO MULTISPECTRAL IMAGE"<<endl;
    return NULL;
  }
  //Create New NDI File with Float32 dataType
  GDALRasterBand *band_multi;
  band_multi=dataset_multi->GetRasterBand(1);
  GDALDataType bufDataType=band_multi->GetRasterDataType();
  char **createOptions = NULL;
  GDALDriver *poDriver = GetGDALDriverManager()-
>GetDriverByName(ftype);
  char **metadata = poDriver->GetMetadata();
  if( !CSLFetchBoolean( metadata, GDAL_DCAP_CREATE, FALSE ) ){
    cout<<"Driver "<<ftype<<" doesn't support Create()
method."<<endl;
    if( dataset_multi != NULL )
      GDALClose(dataset_multi);
    return NULL;
  }
  dataset_chloro = poDriver->Create(fname_chloro,dataset_multi-
>GetRasterXSize(),dataset_multi-
>GetRasterYSize(),1,GDT_Float32,createOptions);
  if(dataset_chloro==NULL){
    cout<<"COULD NOT CREATE "<<fname_chloro<<endl;
    if( dataset_multi != NULL )
      GDALClose(dataset_multi);
```

```
    return NULL;
  }
  //allocate buffers based on dataType
  GDALRasterBand *band_chloro,*band_coastal,*band_blue,*band_green;
  band_chloro=dataset_chloro->GetRasterBand(1);
  band_coastal=dataset_multi->GetRasterBand(1);
  band_blue=dataset_multi->GetRasterBand(2);
  band_green=dataset_multi->GetRasterBand(3);
  float p_chloro;
  int subtrahend;
  void *p_coastal=bufAlloc(1,1,bufDataType);
  void *p_blue=bufAlloc(1,1,bufDataType);
  void *p_green=bufAlloc(1,1,bufDataType);
  for(int i=0;i<dataset_multi->GetRasterXSize();i++){//for all band
width
    for(int j=0;j<dataset_multi->GetRasterYSize();j++){//for all band
height
      //read (i,j) values from minuend and subtrahend
      band_coastal-
>RasterIO(GF_Read,i,j,1,1,p_coastal,1,1,bufDataType,0,0);
      band_blue-
>RasterIO(GF_Read,i,j,1,1,p_blue,1,1,bufDataType,0,0);
      band_green-
>RasterIO(GF_Read,i,j,1,1,p_green,1,1,bufDataType,0,0);
      //compute ndi
      if(bufDataType==GDT_UInt16){

subtrahend=*(GUInt16*)p_coastal+*(GUInt16*)p_blue+*(GUInt16*)p_gre
en;
        if (subtrahend!=0)
          p_chloro=*(GUInt16*)p_blue/(float)subtrahend;
        else
          p_chloro=0;
      }
      else if(bufDataType==GDT_Int16){

subtrahend=*(GInt16*)p_coastal+*(GInt16*)p_blue+*(GInt16*)p_green;
        if (subtrahend!=0)
          p_chloro=*(GInt16*)p_blue/(float)subtrahend;
        else
          p_chloro=0;
      }
      else p_chloro=0;
      //write result
      band_chloro-
>RasterIO(GF_Write,i,j,1,1,&p_chloro,1,1,GDT_Float32,0,0);
    }
  }
  //transfer georeference
  if(GeoRefTransfer( dataset_multi,dataset_chloro )>0)
```

```
    cout<<"GEOTRANSFER ERROR"<<endl;
  if( dataset_multi != NULL )
    GDALClose(dataset_multi);
  return dataset_chloro;
}
```

1.11 apply_ndwi_filter (υπερφορτωμένη)

```
GDALDataset *apply_ndwi_filter(char *fname_chloro,char
*fname_ndwi,float water_thres){
  //open chloro (input) and ndwi (filter) files
  GDALDataset *dataset_chloro=NULL;
  dataset_chloro = (GDALDataset*)GDALOpen(fname_chloro,
GA_ReadOnly);
  if ( dataset_chloro == NULL ) {
    cout<<fname_chloro<<" IMAGE NOT FOUND"<<endl;
    return NULL;
  }
  GDALDataset *dataset_ndwi=NULL;
  dataset_ndwi = (GDALDataset*)GDALOpen(fname_ndwi, GA_ReadOnly);
  if ( dataset_ndwi == NULL ) {
    cout<<fname_ndwi<<" IMAGE NOT FOUND"<<endl;
    if( dataset_chloro != NULL )
      GDALClose(dataset_chloro);
    return NULL;
  }
  //filter chloro with ndwi
  GDALDataset *dataset_filtered=NULL;

dataset_filtered=apply_ndwi_filter(fname_chloro,dataset_chloro,dat
aset_ndwi,water_thres);
  //close chloro and ndwi
  if( dataset_chloro != NULL )
    GDALClose(dataset_chloro);
  if( dataset_ndwi != NULL )
    GDALClose(dataset_ndwi);
  return dataset_filtered;
}
```

```
GDALDataset *apply_ndwi_filter(char *fname_chloro,GDALDataset
*dataset_chloro,GDALDataset *dataset_ndwi,float water_thres){
  //confirm chloro and ndwi are same-sized
  if(dataset_chloro->GetRasterXSize()!=dataset_ndwi-
>GetRasterXSize()){
    cout<<"UNEQUAL X SIZES: CHLORO("<<dataset_chloro-
>GetRasterXSize()<<") NDWI("<<dataset_ndwi-
>GetRasterXSize()<<")"<<endl;
    return NULL;
  }
  if(dataset_chloro->GetRasterYSize()!=dataset_ndwi-
>GetRasterYSize()){
    cout<<"UNEQUAL Y SIZES: CHLORO("<<dataset_chloro-
>GetRasterYSize()<<") NDWI("<<dataset_ndwi-
>GetRasterYSize()<<")"<<endl;
    return NULL;
  }
  GDALDataset *dataset_filtered=NULL;
  char fname_filtered[maxpath];
  sprintf(fname_filtered,"%s_ndwifilter%s",
(get_noext_name(fname_chloro)).c_str(),fexte);
  //check if ndwi-filtered file exists
  ifstream infile(fname_filtered,ifstream::in);
  if (infile.good()){
    cout<<fname_filtered<<" EXISTS"<<endl;
    dataset_filtered=(GDALDataset*)GDALOpen(fname_filtered,
GA_ReadOnly);
    return dataset_filtered;
  }
  //Create New (Filtered) File
  GDALRasterBand *band_chloro=dataset_chloro->GetRasterBand(1);
  if(band_chloro->GetRasterDataType()!=GDT_Float32){
    cout<<"INVALID DATASET DATATYPE ("<<band_chloro-
>GetRasterDataType()<<") SHOULD BE
FLOAT32("<<GDT_Float32<<")"<<endl;
    if( dataset_chloro != NULL )
      GDALClose(dataset_chloro);
    if( dataset_ndwi != NULL )
      GDALClose(dataset_ndwi);
    return NULL;
  }
  char **createOptions = NULL;
  GDALDriver *poDriver = GetGDALDriverManager()-
>GetDriverByName(ftype);
  char **metadata = poDriver->GetMetadata();
  if( !CSLFetchBoolean( metadata, GDAL_DCAP_CREATE, FALSE ) ){
    cout<<"Driver "<<ftype<<" doesn't support Create() method."
<<endl;
    if( dataset_chloro != NULL )
      GDALClose(dataset_chloro);
```

```
    if( dataset_ndwi != NULL )
      GDALClose(dataset_ndwi);
    return NULL;
  }
  dataset_filtered = poDriver->Create(fname_filtered,dataset_chloro-
>GetRasterXSize(),dataset_chloro->GetRasterYSize(),dataset_chloro-
>GetRasterCount(),GDT_Float32,createOptions);
  if(dataset_filtered==NULL){
    cout<<"COULD NOT CREATE "<<fname_filtered<<endl;
    if( dataset_chloro != NULL )
      GDALClose(dataset_chloro);
    if( dataset_ndwi != NULL )
      GDALClose(dataset_ndwi);
    return NULL;
  }
  GDALRasterBand *band_ndwi=dataset_ndwi->GetRasterBand(1);
  GDALRasterBand *band_filtered=dataset_filtered->GetRasterBand(1);
  float pixel_ndwi=0, pixel_chloro=0;//GDT_Float32
  //apply filter (if ndwi no zero, copy chloro value)
  for(int i=0;i<dataset_ndwi->GetRasterXSize();i++){//for all band
width
    for(int j=0;j<dataset_ndwi->GetRasterYSize();j++){//for all band
height
      band_ndwi-
>RasterIO(GF_Read,i,j,1,1,&pixel_ndwi,1,1,GDT_Float32,0,0);
      if(pixel_ndwi<water_thres)//if ndwi is under the water_thres,
filtered will be zero
        pixel_chloro=0;
      else//copy chloro values to filtered
        band_chloro-
>RasterIO(GF_Read,i,j,1,1,&pixel_chloro,1,1,GDT_Float32,0,0);//rea
d source (i,j) value
      band_filtered-
>RasterIO(GF_Write,i,j,1,1,&pixel_chloro,1,1,GDT_Float32,0,0);//wr
ite pixel value to resulting band
    }
  }
  //transfer georeference
  if(GeoRefTransfer( dataset_chloro,dataset_filtered )>0){
    cout<<"GEOTRANSFER ERROR"<<endl;
  }
  return dataset_filtered;
}
```

1.12 ndwichlorofiltered_classification (υπερφορτωμένη)

```
GDALDataset *ndwichlorofiltered_classification(char *fname_src){
  //open src dataset
  GDALDataset *dataset_src=NULL;
  dataset_src = (GDALDataset*)GDALOpen(fname_src, GA_ReadOnly);
  if ( dataset_src == NULL ) {
    cout<<fname_src<<" IMAGE NOT FOUND"<<endl;
    return NULL;
  }
  GDALDataset *dataset_class=NULL;

dataset_class=ndwichlorofiltered_classification(dataset_src,fname_
src);
  if( dataset_src != NULL )
    GDALClose(dataset_src);
  return dataset_class;
}


GDALDataset *ndwichlorofiltered_classification(GDALDataset
*dataset_src,char *fname_src){
  //check if expected output exists
  GDALDataset *dataset_class=NULL;
  char fname_class[maxpath];
  sprintf(fname_class,"%s_classw%s",
(get_noext_name(fname_src)).c_str(),fexte);
  ifstream infile(fname_class,ifstream::in);
  if (infile.good()){
    cout<<fname_class<<" EXISTS"<<endl;
    dataset_class=(GDALDataset*)GDALOpen(fname_class, GA_ReadOnly);
    return dataset_class;
  }
  //Create New Classification File
  GDALRasterBand *band_src=dataset_src->GetRasterBand(1);
  if(band_src->GetRasterDataType()!=GDT_Float32){
    cout<<"INVALID DATASET DATATYPE ("<<band_src-
>GetRasterDataType()<<")"<<endl;
    return NULL;
  }
  char **createOptions = NULL;
  GDALDriver *poDriver = GetGDALDriverManager()-
>GetDriverByName(ftype);
  char **metadata = poDriver->GetMetadata();
  if( !CSLFetchBoolean( metadata, GDAL_DCAP_CREATE, FALSE ) ){
    cout<<"Driver "<<ftype<<" doesn't support Create()
method."<<endl;
    return NULL;
  }
```

```
  dataset_class = poDriver->Create(fname_class,dataset_src-
>GetRasterXSize(),dataset_src-
>GetRasterYSize(),1,GDT_UInt16,createOptions);
  if(dataset_class==NULL){
    cout<<"COULD NOT CREATE "<<fname_class<<endl;
    return NULL;
  }
  GDALRasterBand *band_class=dataset_class->GetRasterBand(1);
  float pixel_src=0;
  int pixel_dst=0;
  //apply chloro classification
  for(int i=0;i<dataset_src->GetRasterXSize();i++){//for all band
width
    for(int j=0;j<dataset_src->GetRasterYSize();j++){//for all band
height
      band_src->RasterIO(GF_Read,i,j,1,1,&pixel_src,1,1,band_src-
>GetRasterDataType(),0,0);//read source (i,j) value
      //classify
      if(pixel_src<0.315)
        pixel_dst=0;
      else if(pixel_src<0.317)
        pixel_dst=1;
      else if(pixel_src<0.32)
        pixel_dst=2;
      else if(pixel_src<0.323)
        pixel_dst=3;
      else if(pixel_src<0.335)
        pixel_dst=4;
      else if(pixel_src<0.34)
        pixel_dst=5;
      else
        pixel_dst=6;
      band_class-
>RasterIO(GF_Write,i,j,1,1,&pixel_dst,1,1,band_class-
>GetRasterDataType(),0,0);//write pixel value to resulting band
    }
  }
  //transfer georeference
  if(GeoRefTransfer( dataset_src,dataset_class )>0){
    cout<<"GEOTRANSFER ERROR"<<endl;
  }
  return dataset_class;
}
```

1.13 single_polygonize

```
OGRDataSourceH single_polygonize(char *fname_src){
  GDALRasterBand *poBand;
  char lname[10];
  char fname_dst[maxpath];

  sprintf(fname_dst,"%s
%s",get_noext_name(fname_src).c_str(),vexte);
  //check if expected output exists
  ifstream infile(fname_dst,ifstream::in);
  if (infile.good()){
    cout<<fname_dst<<" EXISTS"<<endl;
    return NULL;
  }
  //load driver & create shapefile
  OGRRegisterAll();
  OGRSFDriverH hDriver;
  if((hDriver = OGRGetDriverByName(vtype))==NULL){
    cout<<"COULD NOT FIND "<<vtype<<" DRIVER"<<endl;
    return NULL;
  }
  OGRDataSourceH hDS=NULL;
  //open source image
  GDALDataset *dataset_src=NULL;
  dataset_src = (GDALDataset*)GDALOpen(fname_src, GA_ReadOnly);
  if ( dataset_src == NULL ) {
    cout<<fname_src<<" IMAGE NOT FOUND"<<endl;
    return NULL;
  }
  OGRSpatialReferenceH
hSpatialRef=OSRNewSpatialReference(dataset_src-
>GetProjectionRef());
  if((hDS = (OGRDataSourceH *)
GDALCreate(hDriver,fname_dst,0,0,0,GDT_Unknown,NULL))==NULL){
    cout<<"COULD NOT CREATE OGR DATASOURCE"<<endl;
    if( dataset_src != NULL )GDALClose(dataset_src);
    return NULL;
  }
  //polygonize per band
  OGRLayerH hLayer;
  for(int b=1;b<=dataset_src->GetRasterCount();b++){
    memset(lname,0,sizeof(lname));
    sprintf(lname,"Layer_%d",b);
    replaceChar(lname,'-','_');
    poBand = dataset_src->GetRasterBand(b);
    if((hLayer =
GDALDatasetCreateLayer(hDS,lname,hSpatialRef,wkbPolygon,NULL))==NU
LL){
```

```
      cout<<"COULD NOT CREATE LAYER #"<<b<<endl;
      if( dataset_src != NULL )GDALClose(dataset_src);
      if( hDS != NULL )GDALClose(hDS);
      return NULL;
    }

OGR_L_CreateField(hLayer,OGR_Fld_Create("Date",OFTString),FALSE);//
/id=0 for geojson, 2 for kml
    OGR_L_CreateField(hLayer,OGR_Fld_Create("Pixel
Value",OFTString),FALSE);//id=1 for geojson, 3 for kml
    if(GDALPolygonize (poBand, NULL, hLayer, 1, NULL, NULL,
NULL)==CE_Failure){
      cout<<"POLYGONIZE ERROR"<<endl;
      if( dataset_src != NULL )GDALClose(dataset_src);
      if( hDS != NULL )GDALClose(hDS);
      return NULL;
    }
    OGR_L_SyncToDisk(hLayer);
  }
  if(dataset_src != NULL )GDALClose(dataset_src);
  return hDS;
}
```

1.14 add_fieldval_to_all

```
int add_fieldval_to_all(char *fname_cur,const char *field_name,const
char *value){
  OGRDataSourceH srcDs=NULL,dstDs=NULL;
  //open source dataset
  srcDs = (OGRDataSourceH *) GDALOpenEx(fname_cur, GDAL_OF_VECTOR|
GDAL_OF_READONLY, NULL, NULL, NULL);
  if(srcDs==NULL){
    cout<<"COULDN'T OPEN "<<fname_cur<<" VECTOR"<<endl;
    return 1;
  }
  char fname_temp[maxpath];
  sprintf(fname_temp,"temp_%s",fname_cur);
  //load driver
  GDALDriver *poDriver = GetGDALDriverManager()-
>GetDriverByName(vtype);
  if(poDriver==NULL){
    cout<<"COULD NOT FIND "<<vtype<<" DRIVER"<<endl;
    return 2;
  }
  //create destination dataset
  if((dstDs = (OGRDataSourceH
*)GDALCreate(poDriver,fname_temp,0,0,0,GDT_Unknown,NULL))==NULL){
    cout<<"COULD NOT CREATE OGR DATASOURCE"<<endl;
    if(srcDs!=NULL)GDALClose(srcDs);
    return 3;
  }
  char lname[10];
  int field_date_index,field_pixel_value_index;
  OGRFeatureH srcFeat,dstFeat;
  OGRLayerH srcLayer,dstLayer;
  OGRSpatialReferenceH georef;

  for(int b=0;b<GDALDatasetGetLayerCount(srcDs);b++){
    //create new layer
    if((srcLayer=GDALDatasetGetLayer(srcDs,b))==NULL){
      cout<<"COULDN'T GET LAYER #"<<b<<endl;
      if( srcDs != NULL )GDALClose(srcDs);
      if( dstDs != NULL )GDALClose(dstDs);
      return 4;
    }
    if((georef=OGR_L_GetSpatialRef(srcLayer))==NULL){
      cout<<"COULDN'T GET LAYER #"<<b<<"'s SPATIAL
REFERENCE"<<endl;
      if( srcDs != NULL )GDALClose(srcDs);
      if( dstDs != NULL )GDALClose(dstDs);
      return 5;
    }
```

```
    sprintf(lname,"Layer_%d",b);
    if((dstLayer =
GDALDatasetCreateLayer(dstDs,lname,georef,wkbPolygon,NULL))==NULL)
{
      cout<<"COULD NOT CREATE LAYER #"<<b<<endl;
      if( srcDs != NULL )GDALClose(srcDs);
      if( dstDs != NULL )GDALClose(dstDs);
      return 6;
    }
    //scan all features
    for(int i=0;i<OGR_L_GetFeatureCount(srcLayer,TRUE);i++){
      //get feature from source layer
      if((srcFeat=OGR_L_GetFeature(srcLayer,i))==NULL){
        cout<<"COULDN'T GET FEATURE #"<<i<<endl;
        if( srcDs != NULL )GDALClose(srcDs);
        if( dstDs != NULL )GDALClose(dstDs);
        return 7;
      }
      //clone feature
      dstFeat=OGR_F_Clone(srcFeat);
      //find field index for change
      if((field_date_index=OGR_F_GetFieldIndex
(dstFeat,field_name))==-1){
        cout<<"COULDN'T GET FIELD["<<field_name<<"]'s' INDEX OF
FEATURE "<<i<<endl;
        if( srcDs != NULL )GDALClose(srcDs);
        if( dstDs != NULL )GDALClose(dstDs);
        OGR_F_Destroy(srcFeat);
        OGR_F_Destroy(dstFeat);
        return 8;
      }
      //change field's value
      OGR_F_SetFieldString(dstFeat,field_date_index,value);
      //add changed feature to new layer
      if(OGR_L_CreateFeature(dstLayer,dstFeat)!=OGRERR_NONE){
        cout<<"COULDN'T SET FEATURE "<<i<<endl;
        if( srcDs != NULL )GDALClose(srcDs);
        if( dstDs != NULL )GDALClose(dstDs);
        OGR_F_Destroy(srcFeat);
        OGR_F_Destroy(dstFeat);
        return 9;
      }
      //destroy feature structs
      OGR_F_Destroy(srcFeat);
      OGR_F_Destroy(dstFeat);
    }
    //synchronize layer
    if(OGR_L_SyncToDisk(dstLayer)!=OGRERR_NONE){
      cout<<"COULDN'T SYNC LAYER #"<<b<<endl;
```

```
      if( srcDs != NULL )GDALClose(srcDs);
      if( dstDs != NULL )GDALClose(dstDs);
      return 10;
    }
  }
  GDALClose(srcDs);
  GDALClose(dstDs);
  //overwrite old dataset
  GDALDeleteDataset(poDriver,fname_cur);
  rename(fname_temp,fname_cur);
  return 0;
}
```

1.15 appendage & append_geojson

```
void appendage(char *path_dir){
  OGRDataSourceH curDs=NULL,dstDs=NULL;
  //build sum vector filename
  char fname_dst[maxpath];
  string date;
  date=get_curdate_str();
  string dname=get_nopath_name(path_dir);
  sprintf(fname_dst,"polyg%s_%s
%s",date.c_str(),dname.c_str(),vexte);
  //get driver
  GDALDriver *poDriver = GetGDALDriverManager()-
>GetDriverByName(vtype);
  if(poDriver==NULL){
    cout<<"COULD NOT FIND "<<vtype<<" DRIVER"<<endl;
    return;
  }
  //create destination dataset
  if((dstDs = (OGRDataSourceH *)
GDALCreate(poDriver,fname_dst,0,0,0,GDT_Unknown,NULL))==NULL){
    cout<<"COULD NOT CREATE "<<fname_dst<<" OGR DATASOURCE"<<endl;
    return;
  }
  //scan directory for all vector files
  DIR *dir;
  struct dirent *ent;
  if ((dir = opendir (path_dir)) != NULL) {//open dir
    chdir(path_dir);
    while ((ent = readdir (dir)) != NULL) {//for each entry
      if(strlen(ent->d_name)>vnum && strcmp(ent->d_name+strlen(ent-
>d_name)-vnum,vexte) == 0){//if has file extension ".geoJSON"
        curDs=(OGRDataSourceH *)GDALOpenEx(ent-
>d_name,GDAL_OF_VECTOR|GDAL_OF_READONLY,NULL,NULL,NULL);//open
vector file
        if(curDs==NULL)
          cout<<"COULDN'T OPEN "<<ent->d_name<<" VECTOR"<<endl;
        else{
          append_geojson(dstDs,curDs);//append it to source file
          GDALClose(curDs);
        }
      }
    }
    chdir("..");
  } else {
    //could not open directory
    cerr<<"NO OUTPUT DIRECTORY "<<path_dir<<endl;
    GDALClose(dstDs);
    GDALDeleteDataset(poDriver,ent->d_name);
```

```
    return;
  }
  closedir(dir);
  GDALClose(dstDs);
  return;
}
```

```
void append_geojson(OGRDataSourceH srcDs,OGRDataSourceH curDs){
  //check layer numbers must be same
  int srcLayerCount=GDALDatasetGetLayerCount(srcDs);
  if((srcLayerCount!=GDALDatasetGetLayerCount(curDs)) &&
srcLayerCount>0){
    cout<<"WRONG LAYER NUMBER"<<endl;
    return;
  }
  OGRFeatureH curFeat,srcFeat;
  OGRLayerH curLayer,srcLayer;
  OGRSpatialReferenceH georef;
  for(int b=0;b<GDALDatasetGetLayerCount(curDs);b++){
    //get current layer for adding
    if((curLayer=GDALDatasetGetLayer(curDs,b))==NULL){
      cout<<"COULDN'T GET CURRENT LAYER #"<<b<<endl;
      return;
    }
    if(srcLayerCount==0){
      //new srcDs so copy entire current layer
      char lname[10];
      sprintf(lname,"Layer_%d",b);
      srcLayer=GDALDatasetCopyLayer(srcDs,curLayer,lname,NULL);
    }
    else{
      //get source layer
      if((srcLayer=GDALDatasetGetLayer(srcDs,b))==NULL){
        cout<<"COULDN'T GET SOURCE LAYER #"<<b<<endl;
        return;
      }
      //scan all current layer's features
      curFeat=OGR_L_GetFeature(curLayer,0);
      while((curFeat=OGR_L_GetNextFeature(curLayer))!=NULL){
        if(OGR_L_CreateFeature(srcLayer,curFeat)!=OGRERR_NONE){
          cout<<"COULDN'T SET CURRENT LAYER #"<<b<<"'S
FEATURE"<<endl;
          OGR_F_Destroy(curFeat);
          return;
        }
        //destroy feature structs
        OGR_F_Destroy(curFeat);
      }
    }
    //synchronize layer
    if(OGR_L_SyncToDisk(srcLayer)!=OGRERR_NONE)
      cout<<"COULDN'T SYNC SOURCE LAYER #"<<b<<endl;
  }
}
```

1.16 Βοηθητικές Συναρτήσεις

```
int GeoRefTransfer(GDALDataset *dataset_src,GDALDataset
*dataset_dst){
  //transfer georeference metadata
  double adfGeoTransform[6];
  if( dataset_src->GetGeoTransform( adfGeoTransform ) == CE_Failure ){
    cout<<"NO GEOTRANSFORM ON PANCHROMATIC"<<endl;
    return 1;
  }
  dataset_dst->SetGeoTransform(adfGeoTransform);
  //transfer projection metadata
  if(dataset_dst->SetProjection( dataset_src->GetProjectionRef() )
== CE_Failure){
    cout<<"COULD NOT SET PROJECTION"<<endl;
    return 2;
  }
  return 0;
}

void* bufAlloc(int bufsizeX,int bufsizeY,GDALDataType bufDataType){
  void* buffer;
  if(bufDataType==GDT_Byte)
    buffer = (GByte *) CPLMalloc(sizeof(GByte)*bufsizeX * bufsizeY);
  else if(bufDataType==GDT_Int16)
    buffer = (GInt16 *) CPLMalloc(sizeof(GInt16)*bufsizeX * bufsizeY);
  else if(bufDataType==GDT_UInt16)
    buffer = (GUInt16 *) CPLMalloc(sizeof(GUInt16)*bufsizeX *
bufsizeY);
  else if(bufDataType==GDT_Int32)
    buffer = (GInt32 *) CPLMalloc(sizeof(GInt32)*bufsizeX * bufsizeY);
  else if (bufDataType==GDT_UInt32)
    buffer = (GUInt32 *) CPLMalloc(sizeof(GUInt32)*bufsizeX *
bufsizeY);
  else
    buffer = (GIntBig *) CPLMalloc(sizeof(GIntBig)*bufsizeX *
bufsizeY);
  return buffer;
}

void replaceChar(char *str, char ch1, char ch2){
 for (int i=0;i<strlen(str);i++){
  if (str[i]==ch1)
   str[i]=ch2;
 }
}

string get_curdate_str(){
  time_t t = time(0);  // get time now
```

```
  struct tm * now = localtime( & t );
  char date[10];
  sprintf(date,"%d_%d_%d",(now->tm_year+1900),(now->tm_mon+1),now-
>tm_mday);
  return string(date);
}

string get_date_from_fname_raw(char *fname_cur){
  char date[15];
  int year_i=9,year_len=4,day_i=13,day_len=3;
  //LOGIC: LC81840322013255LGN00
  date[0]='Y';date[1]=':';
  for(int i=year_i;i<year_i+year_len;i++)date[2+i-
year_i]=*(fname_cur+i);
  date[2+year_len]='
';date[2+year_len+1]='D';date[2+year_len+2]=':';
  for(int i=day_i;i<day_i+day_len;i++)date[2+year_len+3+i-
day_i]=*(fname_cur+i);
  date[2+year_len+3+day_len]='\0';
  //LOGIC: Y:2013 D:244
  return string(date);
}

string get_date_from_fpath_raw(char *fname_multi_path){
  char temp_fname[maxpath];
  strcpy(temp_fname,get_nopath_name(fname_multi_path).c_str());
  return get_date_from_fname_raw(temp_fname);
}

string get_date_from_fname_refl(char *fname_cur){
  char date[20];
  //LOGIC: 2013-09-12
  int year_i=0,year_len=4,month_i=5,month_len=2,day_i=8,day_len=2;
  date[0]='Y';date[1]=':';
  for(int i=year_i;i<year_i+year_len;i++)date[2+i-
year_i]=*(fname_cur+i);
  date[2+year_len]='
';date[2+year_len+1]='M';date[2+year_len+2]=':';
  for(int i=month_i;i<month_i+month_len;i++)date[2+year_len+3+i-
month_i]=*(fname_cur+i);
  date[2+year_len+3+month_len]='
';date[2+year_len+3+month_len+1]='D';date[2+year_len+3+month_len+2
]=':';
  for(int i=day_i;i<day_i+day_len;i+
+)date[2+year_len+3+month_len+3+i-day_i]=*(fname_cur+i);
  date[2+year_len+3+month_len+3+day_len]='\0';
  //LOGIC: Y:2013 M:09 D:12
  return string(date);
}
```

```
string get_date_from_fpath_refl(char *fname_multi_path){
  char temp_fname[maxpath];
  strcpy(temp_fname,get_nopath_name(fname_multi_path).c_str());
  return get_date_from_fname_refl(temp_fname);
}

string get_nopath_name(char *dir_fpath){
  string dname(dir_fpath);
  size_t found = dname.find_last_of("/\\");
  dname=dname.substr(found + 1);
  return dname;
}

string get_noext_name(char *full_fname){
  string fname(full_fname);
  size_t found = fname.find_last_of(".");
  fname=fname.substr(0,found);
  return fname;
}
```

2. Κώδικας HTML, CSS και JavaScript για CartoDB

```
<!DOCTYPE html>
<html>
 <head>
  <title>Agri Mash</title>
  <meta name="viewport" content="initial-scale=1.0, user-
scalable=no" />
  <meta http-equiv="content-type" content="text/html; charset=UTF-
8"/>
  <link rel="shortcut icon"
href="http://cartodb.com/assets/favicon.ico" />
  <link rel="stylesheet"
href="http://libs.cartocdn.com/cartodb.js/v3/themes/css/cartodb.cs
s" />
  <style>
   html,body,#map{
    width:100%;height:100%;
    padding:0;margin:0;
    }
   #map{
    width:100%;height:100%;
    background:black;
   }
   #area_select{
    position:absolute;
    margin:20px;
    float:right;
    top:0;
    right:0;
   }
   #date_select{
    width:150px;
    position:absolute;
    top:10%;
    right:20px;
    font:bold 13px "Helvetica",Arial;
    color:#555;
    padding:5px;
    background:#ffffff;
    border:1px solid #777777;
    border-radius:2px;
   }
   #date_select input{margin-left:10px;}
  </style>
 </head>
 <body>
  <!-- include cartodb.js library -->
  <script src =
```

```
"http://libs.cartocdn.com/cartodb.js/v3/cartodb.js"></script>

  <script>
   //datasets
   var myUser = "andyhalk";
   var area_names = ["Karla Raw","Axios 1 Raw","Axios 2 Raw","Axios
1 Refl","Axios 2 Refl"];
   var dataset =
["karla_raw","axios1_raw","axios2_raw","axios1_refl","axios2_refl"
];
   var myCenter = [[39.48385,22.81487],[40.599297, 22.568977],
[40.590846, 22.766869],[40.599297, 22.568977],[40.590846,
22.766869]];
   var myZoom = [13,13,14,13,14];
   //number of raw images
   var raw_num=3;
   //index used for toggling
   var datIn;

   //CSS
   var css_ndvi =
["rgb(255,0,0)","rgb(255,53,0)","rgb(255,102,0)","rgb(255,154,0)",
"rgb(255,220,0)","rgb(0,255,102)","rgb(0,162,255)"];
   var css_ndwi = ["rgb(0,0,0)", "rgb(255,0,0)",
"rgb(255,255,0)","rgb(0,255,0)", "rgb(0, 153, 0)","rgb(0, 102,
0)","rgb(0,0,0)"];
   var css_val = [css_ndvi,css_ndwi,css_ndwi,css_ndwi,css_ndwi];

   //custom columns
   var pixel_value_col="pixel_value";
   var date_col="date";

   //vector CSS
   function myCSS(data_index){
    var custom_css = "#"+dataset[data_index];
    custom_css+="{";
    for(i=0;i<css_val[data_index].length;i+
+)custom_css+="["+pixel_value_col+"=\""+i+"\"]{polygon-fill:
"+css_val[data_index][i]+";}";
    custom_css+="}";
    return custom_css;
   }

   //SQL functions;
   var sql = new cartodb.SQL({ user: myUser })
   function base_sql_q(i){
    return "SELECT * FROM "+dataset[i]+" WHERE "+pixel_value_col+" !
='0' AND "+pixel_value_col+"!='6' ";
   }
```

```
   function check_dates(){
    var
date_count=document.getElementById("date_select").getElementsByTag
Name("input").length;
    for(var i=0;i<date_count;i++)
     if(document.getElementById("myRadio"+i).checked)
      return " AND
"+date_col+"=\'"+document.getElementById("myRadio"+i).value+"\'";
    return "NO DATE CHOSEN";
   }

   //baseMaps
   var baseMaps={
    esri_world_imagery :
'http://server.arcgisonline.com/ArcGIS/rest/services/World_Imagery
/MapServer/tile/{z}/{y}/{x}'
   };

    //legend
   function myLegend(){
    var legend_array=[];
    for(var i=1;i<css_val[datIn].length-1;i++)legend_array.push({
name: "Pixel Value "+i, value: css_val[datIn][i] });
    var legend = new cdb.geo.ui.Legend({type: "custom",data:
legend_array});
    $('#map').append(legend.render().el);
   }
   function subLayerOptions(i){
    sql_q=base_sql_q(i);
    sql_q+=check_dates();
    console.log(sql_q);
    return {sql: sql_q,cartocss: myCSS(datIn)};
   }

   //global vars
   var map;
   var basemap;
   var mySublayer=[];

   function init() {
    // initiate leaflet map
    datIn=0;
    map = new L.Map('map',
{center:myCenter[datIn],zoom:myZoom[datIn]});
    basemap = L.tileLayer(baseMaps.esri_world_imagery)
     .addTo(map);
    //stuff
    myLegend();
    area_select();
    date_select();
```

```
    //subLayers
    var mySubLayers=[];
    mySubLayers.push(subLayerOptions(datIn));
    var layerSource={
     user_name:myUser,
     type:'cartodb',
     sublayers:mySubLayers
    }
    //myLayer
    cartodb.createLayer(map, layerSource)
    .addTo(map)
    .on('done', function(layer) {
     mySublayer=layer.getSubLayer(0);
    })
    .on('error', function(err){console.log(err);});

    $("#area_select")
     .change(function() {
      var changed=0;
      $("select option:selected").each(function() {
       for(var i=0;i<dataset.length;i++)// search all layers
        if($(this).attr('value')=="lay"+i){// to find the current
chosen
         if((datIn<raw_num&&i<raw_num)||
(datIn>=raw_num&&i>=raw_num)) change=0;
         else changed=1;
         datIn=i;// make it current dataset
         map.setView(myCenter[i],myZoom[i]);// move view to that
position
        }
      })
      var sql_q=base_sql_q(datIn);// all polygons of current layer
except for cats 0 and 6
      if(changed==0)sql_q+=check_dates();// of the chosen date
      else date_select();
      if(changed==1)date_select();
      sql_q+=check_dates();
      mySublayer.setSQL(sql_q);// request them
      mySublayer.setCartoCSS(myCSS(datIn));
      myLegend();//show new legend
    });

    $(document).ready(function(){
     setInterval(next_date,3500);
     var i=0;
     function next_date(){
      i=i
%document.getElementById("date_select").getElementsByTagName("inpu
t").length;
```

```
      $('#myRadio'+i).prop('checked',true);
      i++;
      check();
     }
    });
   }

   function check() {
     var sql_q=base_sql_q(datIn);//checklayer
      sql_q+=check_dates();
      mySublayer.setSQL(sql_q);
      myLegend();
   }

   function area_select(){
    var custom_html="";
    custom_html+="<option value=\"lay0\"
selected=\"selected\">"+area_names[0]+"</option>";
    for(var i=1;i<dataset.length;i++)custom_html+="<option
value=\"lay"+i+"\">"+area_names[i]+"</option>";
    document.getElementById("area_select").innerHTML = custom_html;
   }

   function date_select(){
    sql.execute("SELECT DISTINCT "+date_col+" FROM "+dataset[datIn]
+" ORDER BY "+date_col+" ASC")
    .done(function(data){
     myHtml="<b>"+date_col+"</b></br><form>";
     for(var i=0;i<data.total_rows;i++)//needs variable instead of
date
      myHtml+="<input type=\"radio\" name=\""+date_col+"\"
onclick=\"check()\" id=\"myRadio"+i+"\"
value=\""+data.rows[i].date+"\">"+data.rows[i].date+"<br>";
     myHtml+="</form>";
     document.getElementById("date_select").innerHTML = myHtml;
    })
    .error(function(errors){console.log("errors:" + errors);})
   }
   window.onload = function (event){init()};
  </script>

  <div id="map"></div>
  <select id="area_select"></select>
  <div id="date_select"></div>

 </body>
</html>
```

3. Κώδικας MatLab

3.1 Main

```
tic;
% Diavazoume eikones, H:panchromatic, L: multispectral
cd('C:\path_to_images');
[H,RH]=geotiffread('panchromatic_img.tif');
[L,RL]=geotiffread('multispectral_img.tif');

% Kovoume tis tesseris prwtes zwnes/bands, giati parapanw den
yposthrizontai apo palies ekdoseis tou MATLAB/imwrite (ayto borei
kai na exei alla3ei se teleytaies ekdoseis)
L = L(:,:,1:4);

% Ylopoioume to fusion stis tesseris prwtes zwnes
F = hpffusion(H,L,'bicubic');

% Grafoume thn pansharpened eikona sto disko
geotiffclone('pan255.tif','hpf-fused_img.tif',F)

ExecTime=toc;
fprintf ('FINISHED in %d seconds\n\n',ExecTime);
```

3.2 HPF-Fusion (Βαϊόπουλος)

```
function [F, Lr, HPFI, HPKW] = hpffusion(H,L,resizemethod,R)
% HPF Fusion - Spatial Image Enchancement
% Aristidis Vaiopoulos
% 04/07/2010 - Version 1.0
% 19/06/2012 - Version 1.5F - Function version
% 18/12/2012 - Version 2.0F - Flexibility in ratio R.

% Class of panchromatic image
cl = class(H);

Hrows = size(H,1);
Hcols = size(H,2);
Lrows = size(L,1);
Lcols = size(L,2);
bands = size(L,3);
if nargin == 2
  resizemethod = 'bicubic';
  R = Hrows/Lrows;
end
if nargin == 3
  R = Hrows/Lrows;
end
if nargin == 3 && R == 1
  disp('-Improper H/L ratio (R). Using R = 2 and HPK size 5.');
  R = 2;
end
% Resize multispectral image
Lr = imresize(L,[Hrows Hcols],resizemethod);

% HPF algorithm
%HPK size
if 1 < R & R < 2.5
  hpk = 5;     dv = 24;     M  = 0.25;
elseif 2.5 <= R & R < 3.5
  hpk = 7;     dv = 48;     M  = 0.5;
elseif 3.5 <= R & R < 5.5
  hpk = 9;     dv = 80;     M  = 0.5;
elseif 5.5 <= R & R < 7.5
  hpk = 11;     dv = 120;     M  = 0.65;
elseif 7.5 <= R & R < 9.5
  hpk = 13;     dv = 168;     M  = 1;
elseif R >= 9.5
  hpk = 15;     dv = 336;     M  = 1.35;
end
% High pass kernel window (filter)
HPKW = (ones(hpk)).*(-1);
rhpk = round(hpk/2);
HPKW(rhpk,rhpk) = HPKW(rhpk,rhpk)*(-1)*dv;
```

```
% High pass filter the high resolution image
HPFI = single(filter2(HPKW, H));

% Weights
W = zeros(bands,1);
for band = 1:bands
  W(band) = ( std2(Lr(:,:,band)) / std2(HPFI) ) * M;
end

% Preallocate fused image
F = single( zeros(Hrows,Hcols,bands) );
Lr = single(Lr);
% Fused image
for band = 1:bands
  F(:,:,band) = Lr(:,:,band) + (HPFI .* W(band));
  F(:,:,band) = F(:,:,band) - mean2(F(:,:,band)) ...
        .* ( std2(Lr(:,:,band) ./ std2(F(:,:,band))) )...
        + mean2(Lr(:,:,band));
end

% Force class of panchromatic image
Lr= cast(Lr,cl);
F = cast(F,cl);
end
```

3.3 GeoTIFFClone (Βαϊόπουλος)

```
function geotiffclone(georef_file,geocloned_file,image)
% 23/12/2012 - Version 1.0
% 23/12/2012 - Version 1.6 / Forces RasterPixelIsArea to
GTRasterTypeGeoKey

% Reads the georeference from tiff file [georef_file], then writes image
% [image] with the georeference of the [georef_file].

% Consider you have a georeferenced image A and you have created a
% processed image B = f(A). If you want to write the image B and keep the
% geoinfo of A, you may use this function as follows:

% geotiffclone( filename(A), filename(B), B )

% Author: Aristides D. Vaiopoulos

% Read geotiffinfo
gti     = geotiffinfo(georef_file);
R       = gti.RefMatrix;
GTT_GKDT  = gti.GeoTIFFTags.GeoKeyDirectoryTag;

% Force RasterPixelIsArea
GTT_GKDT.GTRasterTypeGeoKey = 1;

geotiffwrite( geocloned_file,image,R,'GeoKeyDirectoryTag',GTT_GKDT)

end
```

ΠΙΝΑΚΑΣ ΕΙΚΟΝΩΝ

ΠΙΝΑΚΑΣ ΕΙΚΟΝΩΝ

ΠΙΝΑΚΑΣ ΕΙΚΟΝΩΝ

ΠΙΝΑΚΑΣ ΠΙΝΑΚΩΝ