Προγραμματισμός Υπολογιστών με C++

3η Εργασία – Ακαδημαϊκό Έτος 2015-16

Ημερομηνία Παράδοσης Εργασίας: **31 Ιανουαρίου 2016**

1. Εκφώνηση

Να χρησιμοποιηθεί ο κώδικας που αναπτύξατε στις 2 προηγούμενες εργασίες για τη δημιουργία μιας νέας εφαρμογής η οποία αφού φορτώσουμε μια εικόνα ppm, θα εφαρμόζει πάνω σε αυτή μια σειρά από φίλτρα επεξεργασίας εικόνας σύμφωνα με τις προδιαγραφές που περιγράφονται στη συνέχεια. Ο τύπος, ο αριθμός και η σειρά των φίλτρων εικόνας που θα επενεργούν πάνω στην εικόνα προσδιορίζονται από τα ορίσματα της εφαρμογής. Η τροποποιημένη εικόνα θα πρέπει να αποθηκεύεται σε αρχείο ppm κι αυτή, αφού πρώτα αλλαχθεί το όνομα αρχείου παρεμβάλλοντας τη συμβολοσειρά ".filtered" μεταξύ της κατάληξης του αρχείου και του υπόλοιπου ονόματός του.

Το πρόγραμμά σας θα πρέπει να έχει το όνομα "filter.exe" και η εκτέλεσή του από τη γραμμή εντολών γίνεται σύμφωνα με το παράδειγμα που ακολουθεί:

> filter -f blur -f diff -f gray image01.ppm

Στην παραπάνω γραμμή εντολών, μετά το όνομα του εκτελέσιμου, ακολουθούν ζεύγη ορισμάτων -f [όνομα φίλτρου], τόσα όσα τα φίλτρα που θέλουμε να εφαρμόσουμε πάνω στην εικόνα image01.ppm. Η σειρά εφαρμογής των φίλτρων είναι blur, μετά diff και τέλος gray. Η εφαρμογή γράφει στο δίσκο την τελική εικόνα με όνομα image01.filtered.ppm. Γενικά, ο τρόπος εκτέλεσης του προγράμματος ακολουθεί το παρακάτω μοτίβο:

filter -f [φίλτρο1] -f [φιλτρο2] … -f [φίλτρο *k*] [όνομα αρχείου εικόνας]

Ο αριθμός των φίλτρων που θα εφαρμόσουμε δεν είναι καθορισμένος και ένα φίλτρο μπορεί να επαναλαμβάνεται, π.χ:

> filter -f blur -f blur -f blur –f gray image01.ppm

Το αρχείο εικόνας εισόδου δίνεται πάντα τελευταίο. Αν δε δοθεί φίλτρο, η εφαρμογή πρέπει να τερματίζει χωρίς να κάνει τίποτα. Αν δεν προσδιοριστεί αρχείο ή δε δοθεί το όνομα κάποιου φίλτρου σωστά (ή καθόλου), η εφαρμογή πρέπει να βγάζει σχετικό μήνυμα λάθους και να τερματίζει.

Τα διαθέσιμα φίλτρα είναι: gray, color, blur, diff, median. Ειδικά για το φίλτρο color, ακολουθούν 3 επιπλέον ορίσματα που προσδιορίζουν τις 3 συνιστώσες χρώματος (RGB). Επομένως, αν συμπεριλαμβάνεται ένα color φίλτρο στα ορίσματα, αυτά έχουν τη μορφή του ακόλουθου παραδείγματος:

filter -f gray -f color 0.7 0.5 0.4 -f blur image02.ppm

Εδώ, μετά τη λέξη color, περιμένουμε να διαβάσουμε την κόκκινη, πράσινη και μπλε συνιστώσα του χρώματος που χρειάζεται το φίλτρο για να δουλέψει.

Ακολουθεί εξήγηση για το τι είναι και πως δουλεύουν τα φίλτρα που χρειαζόμαστε.

2. Φίλτρα εικόνας

Για να επεξεργαστούμε μια εικόνα και να αλλάξουμε τα χρωματικά και μορφολογικά της χαρακτηριστικά, εφαρμόζουμε συχνά σε κάθε εικονοστοιχείο της (pixel) έναν τελεστή, που μετασχηματίζει τη χρωματική πληροφορία του σε μια νέα. Αυτός ο μετασχηματισμός μπορεί να είναι τοπικός, δηλαδή να παράγει μια νέα τιμή χρώματος με βάση μόνο την παλιά, ή να εξαρτάται κι από άλλα pixels, συνήθως γειτονικά.

Στην εφαρμογή μας, θα δούμε 2 φίλτρα τοπικά (gray, color) και 3 φίλτρα που επηρεάζουν ένα pixel με βάση μια 3X3 γειτονιά με κέντρο το pixel αυτό (blur, diff, median).

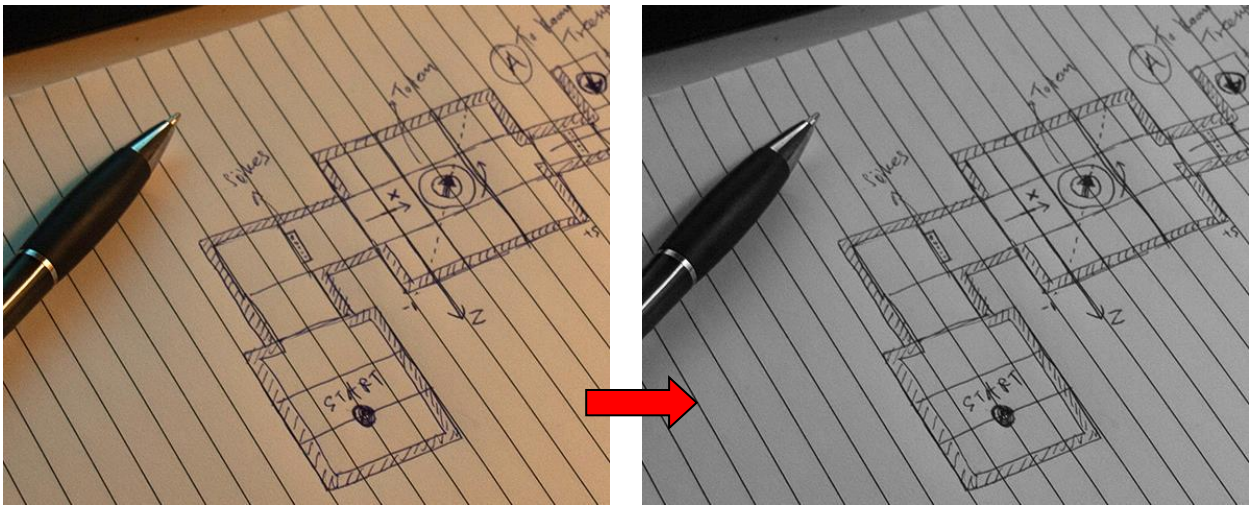
2.1 Τοπικά Φίλτρα

gray. Όταν το φίλτρο που συναντήσουμε στη γραμμή εντολών έχει όνομα "gray", τότε δημιουργούμε μια νέα έκδοση της εικόνας που κάθε pixel της $\mathbf{p}'(i,j) = R'(i,j), G'(i,j), B'(i,j)$ γίνεται:

$$\mathbf{p}'(i,j) = (\mu, \mu, \mu)$$

$$\mu = \frac{R(i,j) + G(i,j) + B(i,j)}{3}$$

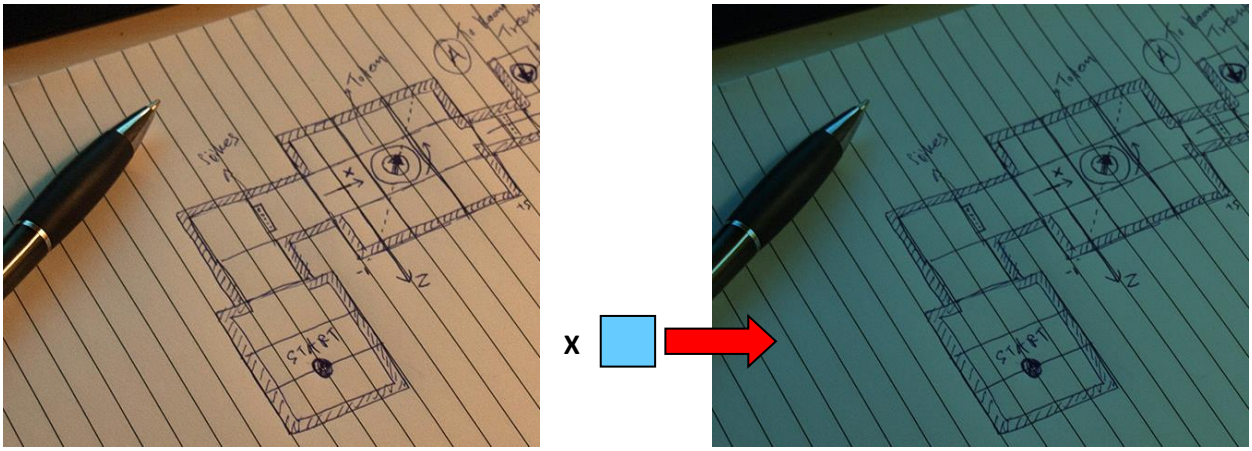
Όπου $R(i,j), G(i,j), B(i,j)$ το αρχικό χρώμα $\mathbf{p}$ του pixel της εικόνας εισόδου στο φίλτρο συντεταγμένες (i,j). Ο υπολογισμός αυτός επαναλαμβάνεται για κάθε pixel της εικόνας.



Παράδειγμα εφαρμογής του φίλτρου gray.

color. Το δεύτερο φίλτρο που θα πρέπει να υλοποιηθεί, δημιουργείται και επενεργεί πάνω στην εικόνα όταν συναντήσουμε στα ορίσματα το όνομα φίλτρου "color". Όπως ειπώθηκε, πάντα μετά το όρισμα color πρέπει να ακολουθούν 3 ακόμα παράμετροι που προσδιορίζουν το χρώμα $\mathbf{c} = (R_c, G_c, B_c)$ που θα εφαρμόσει το φίλτρο μας, πολλαπλασιαστικά πάνω στο αντίστοιχο χρώμα κάθε pixel:

$$\mathbf{p}'(i,j) = \mathbf{c} \cdot \mathbf{p}(i,j) = (R_c\ R(i,j),\ G_c G(i,j),\ B_c B(i,j))$$



Παράδειγμα εφαρμογής του φίλτρου color.

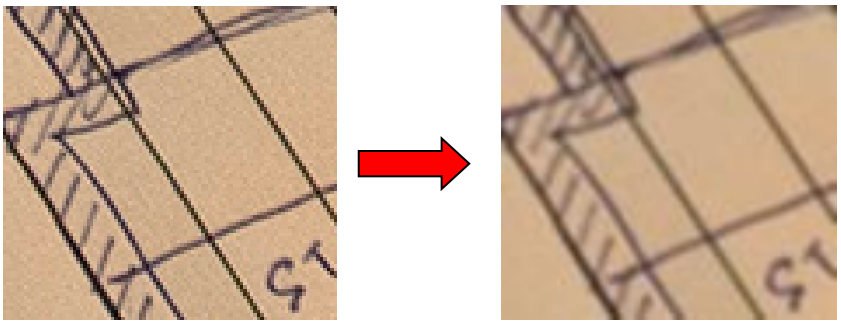
2.2 Φίλτρα Γειτονιάς

Για κάθε pixel (i, j) της τελικής εικόνας, τα φίλτρα γειτονιάς χρησιμοποιούν μια περιοχή NXM pixels γύρω από τη θέση (i, j) της αρχικής εικόνας για να υπολογίσουν το χρώμα. Έτσι, μπορούν να «δουν» τι συμβαίνει στην ευρύτερη γειτονιά ενός σημείου μέσα στην εικόνα και όχι απομονωμένα σε ένα pixel. Πρόκειται για μια μεγάλη κατηγορία φίλτρων επεξεργασίας εικόνας με πολλούς χρήσιμους τελεστές. Στη συνέχεια θα δούμε αναλυτικά τις 2 οικογένειες φίλτρων που θα χρησιμοποιήσουμε στην εργασία.

blur. Όταν στη γραμμή εντολών συναντήσουμε ως όρισμα φίλτρου τη λέξη "blur", δημιουργούμε ένα φίλτρο το οποίο με κέντρο κάθε pixel (i, j) της εικόνας προορισμού, διαβάζει μια περιοχή 3X3 pixels της εικόνας εισόδου του φίλτρου και θέτει ως χρώμα του pixel (i, j) το μέσο όρο των 9 χρωμάτων του 3X3 block. Δηλαδή, υλοποιεί τον τελεστή:

$$\mathbf{p}'(i,j) = \frac{1}{9}\sum_{m=-1}^{1}\sum_{n=-1}^{1}\mathbf{p}(i+m, j+n)$$

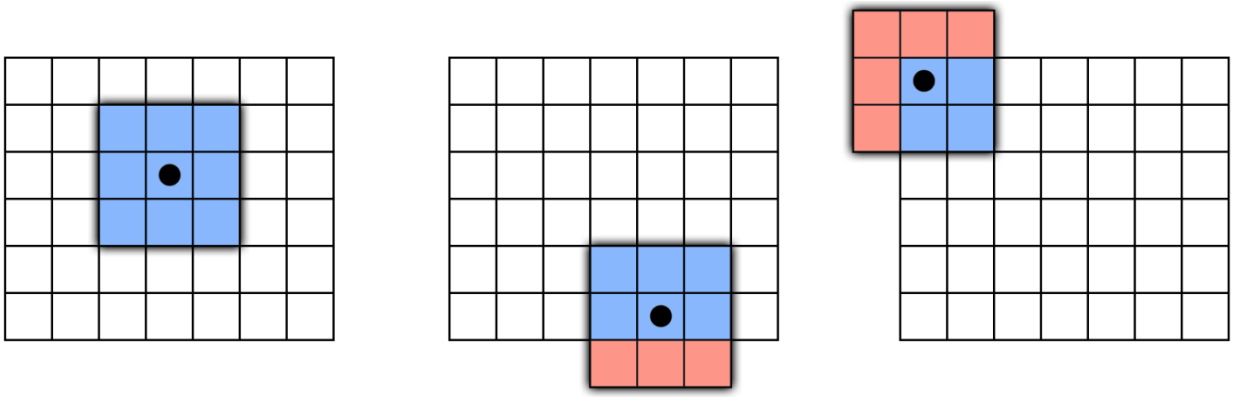
Για παράδειγμα, μπορείτε να δείτε στην παρακάτω εικόνα ότι αντικαθιστώντας στην εικόνα – αποτέλεσμα κάθε pixel με το μέσο όρο της γειτονιάς του, «αμβλύνεται» (θολώνεται - blur) η χρωματική πληροφορία:



Στα φίλτρα γειτονιάς, όπως το φίλτρο μέσου όρου που είδαμε, τα άκρα της εικόνας χρειάζονται ιδιαίτερο χειρισμό, καθώς οι τιμές $(i+m, j+n)$ μπορούν να ξεπεράσουν τα όρια της εικόνας, αφού τα i, j ανήκουν στο διάστημα [0, width-1], [0, height-1] και τα m, n στο [-1,1]. Γι αυτό το λόγο, πρέπει να ελέγχουμε αν μια συντεταγμένη $(i+m, j+n)$ είναι έγκυρη και μόνο τότε να προσμετράμε στο μέσο όρο το αντίστοιχο pixel. Αυτό σημαίνει αντίστοιχα ότι ο πληθυσμός με τον οποίο διαιρούμε το άθροισμα δεν είναι σταθερός (9), όπως στον παραπάνω απλό τύπο, αλλά ισούται με τον αριθμό των έγκυρων pixels. Έτσι, ο τελεστής του blur που πρέπει να χρησιμοποιήσετε γίνεται:

$$\mathbf{p}'(i,j) = \frac{1}{\sum V(i+m, j+n)} \sum_{m=-1}^{1} \sum_{n=-1}^{1} \mathbf{p}(i+m, j+n) V(i+m, j+n)$$

Όπου $V(i+m, j+n) = 1$ όταν το pixel $(i+m, j+n)$ είναι έγκυρο (εντός ορίων), διαφορετικά είναι 0. Παραστατικά, στην ακόλουθη εικόνα φαίνεται ποια pixels προσμετρώνται σε διαφορετικές περιπτώσεις.

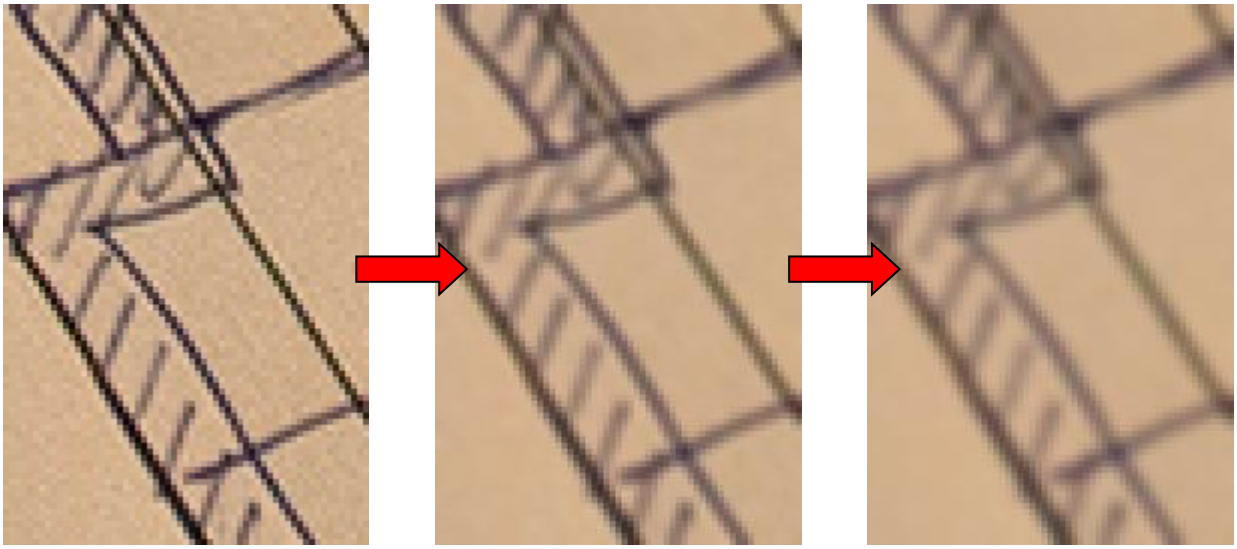


Έγκυρα pixels: 9 Έγκυρα pixels: 6 Έγκυρα pixels: 4

Σημείωση: Επειδή τα φίλτρα γειτονιάς για να υπολογίσουν την κεντρική τιμή χρησιμοποιούν τα γειτονικά pixels, **πρέπει η εικόνα στην οποία γράφουμε να είναι διαφορετική από αυτή που διαβάζουμε**, έστω και προσωρινό αντίγραφο. Διαφορετικά, θα επαναχρησιμοποιούμε καινούρια υπολογισμένα δεδομένα από τους γείτονες αντί για τα αρχικά.

Για να θολώσουμε περισσότερο την εικόνα, μπορούμε να δηλώσουμε από τη γραμμή εντολών 2 ή περισσότερες φορές το φίλτρο blur:

filter -f blur -f blur test.ppm



Παράδειγμα διαδοχικής εφαρμογής του φίλτρου blur, 2 φορές.

diff και **median**. Και τα 2 αυτά φίλτρα είναι φίλτρα «ταξινόμησης»: Διαβάζουμε τις κόκκινες, πράσινες και μπλε συνιστώσες των χρωμάτων στην 3Χ3 γειτονιά και, χωριστά για κάθε κανάλι χρώματος, ταξινομούμε τις τιμές που διαβάσαμε. Το φίλτρο median επιλέγει ως χρωματική συνιστώσα την τιμή του μεσαίου στοιχείου της ταξινομημένης σειράς χρωματικών συνιστωσών. Το φίλτρο diff, υπολογίζει την τελική χρωματική συνιστώσα ως τη διαφορά της ελάχιστης από τη μέγιστη τιμή.

Παράδειγμα. Έστω η παρακάτω 3Χ3 περιοχή με κέντρο ένα pixel:

20, 50, 5	10, 60, 7	10, 66, 6
50, 180, 3	72, 165, 4	60, 170, 7
50, 130, 2	75, 140, 9	66, 165, 5

Οι πίνακες των αταξινόμητων συνιστωσών είναι:

20, 10, 10, 50, 72, 60, 50, 75, 66

50, 60, 66, 180, 165, 170, 130, 140, 165

5, 7, 6, 3, 4, 7, 2, 9, 5

Και μετά την ταξινόμηση:

10, 10, 20, 50, **50**, 60, 66, 72, 75

50, 60, 66, 130, **140**, 165, 165, 170, 180

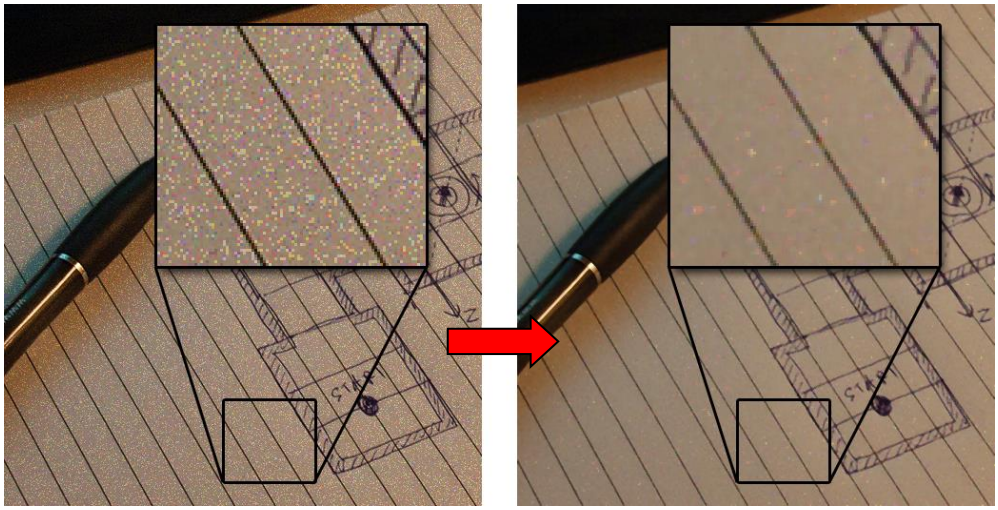
2, 3, 4, 5, **5**, 6, 7, 7, 9

Οπότε, στο median φίλτρο το αποτέλεσμα είναι ένα χρώμα με το μεσαίο στοιχείο (5^{ο} εδώ) από κάθε ταξινομημένο πίνακα: (50, 140, 5)

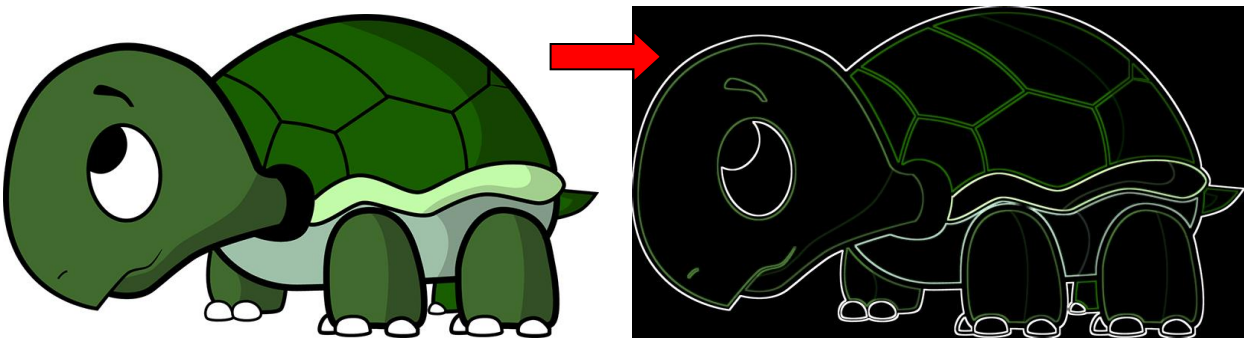
Αντίστοιχα, στο diff φίλτρο, το αποτέλεσμα είναι: (75-10, 180-50, 9-2) → (65, 130, 7)

Δεδομένου ότι όπως και στο φίλτρο blur, έτσι κι εδώ η περιοχή εφαρμογής του φίλτρου είναι μεταβλητή για να μην επιχειρήσουμε να διαβάσουμε άκυρα pixels εκτός εικόνας, η λίστα των στοιχείων που ταξινομούμε δεν έχει απαραίτητα ακριβώς 9 στοιχεία αλλά μέχρι 9 και το μεσαίο στοιχείο είναι το $\lfloor N/2 \rfloor$ (ακέραια διαίρεση) με *N* το πλήθος των στοιχείων.

Το αποτέλεσμα της εφαρμογής των 2 φίλτρων φαίνεται στο παρακάτω παράδειγμα. Το φίλτρο median βοηθάει στην απαλλαγή της εικόνας από θόρυβο, αφού κρατώντας τη μεσαία τιμή από ένα block εικόνας, ακραίες διακυμάνσεις απορρίπτονται. Αντίστοιχα, το φίλτρο diff κρατάει τις ακραίες διαφορές, αναδεικνύοντας έτσι ακμές και έντονες μεταβάσεις στην εικόνα.



Φίλτρο median



Φίλτρο diff

3. Οδηγίες Προγραμματισμού

Είστε ελεύθεροι να χρησιμοποιήσετε όποιες έτοιμες βιβλιοθήκες κρίνετε σκόπιμο (π.χ. STL) και οποιαδήποτε κομμάτια κώδικα έχετε αναπτύξει στις προηγούμενες εργασίες. Σίγουρα θα χρειαστείτε τη λειτουργικότητα για την ανάγνωση και εγγραφή από/προς PPM και τις δομές Array και Image.

Δεδομένου ότι όλα τα φίλτρα επενεργούν πάνω σε εικόνες με ενιαίο τρόπο, σας δίνεται μια βασική abstract class "Filter" (στο αρχείο filter.h) για να αποτελέσει κοινό κορμό για τα πιο εξειδικευμένα φίλτρα. Σχεδιάστε τις κλάσεις σας με τον πιο γενικό και επεκτάσιμο τρόπο. Μπορείτε να δημιουργήσετε sub-classes της Filter και μπορείτε - και πρέπει - να προσθέσετε πεδία και μεθόδους και στην ίδια την κλάση Filter.

Για να εφαρμόσετε ένα φίλτρο πάνω σε μια εικόνα, σίγουρα θα χρειαστεί να φτιάξετε μια μέθοδο ή τελεστή (π.χ. <<) για να περνάτε στο φίλτρο μια εικόνα και να παίρνετε πίσω ένα αποτέλεσμα. Καλό είναι, η μέθοδος ή ο τελεστής που εφαρμόζει το φίλτρο πάνω στην εικόνα να επενεργεί πάνω σε μια αναφορά της εικόνας εισόδου, ανεξάρτητα από το αν εσωτερικά δημιουργείται κάποιο προσωρινό αντίγραφο της εικόνας για να δουλέψει σωστά το φίλτρο.

Για τις ταξινομήσεις, μπορείτε να εκμεταλλευτείτε τις έτοιμες συναρτήσεις της STL (βλ. κεφαλίδα algorithm). Δείτε ότι η ταξινόμηση δουλεύει τόσο για απλούς πίνακες, όσο και για συλλογές. Χρησιμοποιήστε ο,τι κρίνετε καλύτερο.

Δεδομένου ότι ο αριθμός των φίλτρων που εφαρμόζουμε σε μια εικόνα δεν είναι γνωστός εκ των προτέρων, αλλά καθορίζεται από τα ορίσματα που έχουμε δώσει στην εφαρμογή, πρέπει να μπορείτε να φτιάξετε στη main σας τις απαραίτητες δομές ελέγχου και συλλογές για να υλοποιείται τα κατάλληλα στιγμιότυπα φίλτρων και να τα εφαρμόζετε διαδοχικά πάνω στην εικόνα σας.

Στην εργασία αυτή, πέρα από την αρτιότητα του αποτελέσματος, θα συνεκτιμηθεί και ο καλός σχεδιασμός και η πληρότητα του κώδικα.

Κριτήριο	Βαθμολογία
Σωστός πολυμορφικός σχεδιασμός και επαναχρησιμοποίηση κώδικα (με bonus +0.3)	0.2
Υλοποίηση τοπικών φίλτρων	0.2
Υλοποίηση φίλτρων γειτονιάς	0.3
Υλοποίηση της κύριας εφαρμογής σύμφωνα με τις προδιαγραφές εκτέλεσης	0.2
Εκτέλεση του κώδικα με σωστή δέσμευση και αποδέσμευση μνήμης	0.1

4. Γενικές οδηγίες

Για την υλοποίηση των προγραμμάτων, μπορείτε να χρησιμοποιήσετε όποιο περιβάλλον θέλετε αλλά ο κώδικας που θα παραδώσετε θα πρέπει να μπορεί να γίνει compile και link είτε με τον gcc σε περιβάλλον Windows, είτε με το Visual Studio 2012, πάλι προφανώς σε περιβάλλον Windows. Παραδίδετε έτοιμο Visual Studio project ή αντίστοιχα makefile (για τον gcc) και όχι μόνο τα .cpp και .h αρχεία και έχετε επιβεβαιώσει ότι γίνονται σωστά build.

Για τον έλεγχο του αποτελέσματος που βγάλατε, χρησιμοποιείστε την εφαρμογή (viewer) που σας παρέχεται.

Εργάζεστε σε ομάδες των 2 ατόμων. Ειδικά για τους φοιτητές 5+ έτους, επιτρέπεται να παραδώσουν τις εργασίες και μόνοι τους αν το επιθυμούν.

Προσοχή: Σε οποιαδήποτε περίπτωση διαπιστωθεί **αντιγραφή** κώδικα ή αδυναμία του εξεταζομένου φοιτητή να **εξηγήσει την υλοποίησή του**, η εργασία **μηδενίζεται** αυτομάτως. Επίσης, το εκτελέσιμο της εργασίας **πρέπει να δουλεύει** για να βαθμολογηθεί η τελευταία.