ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΗΛΕΚΤΡΟΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ, ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ & ΔΙΚΤΥΩΝ

Απόκτηση εικόνων υπερανάλυσης με φορητή συσκευή σε σχεδόν πραγματικό χρόνο

Μενέλαος Γιαννόπουλος
Επιβλέπων καθηγητής: Ιωάννης Κατσαβουνίδης
Βόλος, 2013

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΗΛΕΚΤΡΟΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ, ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ & ΔΙΚΤΥΩΝ

Απόκτηση εικόνων υπερανάλυσης με φορητή συσκευή σε σχεδόν πραγματικό χρόνο

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Του

Μενέλαου Γιαννόπουλου

Επιβλέποντες :

Ιωάννης Κατσαβουνίδης
Αναπληρωτής Καθηγητής Π.Θ.

Γεώργιος Σταμούλης
Καθηγητής Π.Θ.

Εγκρίθηκε από την διμελή εξεταστική επιτροπή την ΗΜΕΡΟΜΗΝΙΑ ΕΞΕΤΑΣΗΣ

(Υπογραφή)	*(Υπογραφή)*
.....	.....
Ιωάννης Κατσαβουνίδης	Γεώργιος Σταμούλης
Αναπληρωτής Καθηγητής Π.Θ.	Καθηγητής Π.Θ.

Βόλος, Μάρτιος 2013

(Υπογραφή)

.....

Μενέλαος Γιαννόπουλος

Διπλωματούχος Μηχανικός Ηλεκτρονικών Υπολογιστών, Τηλεπικοινωνιών και Δικτύων Πανεπιστημίου Θεσσαλίας

Ευχαριστίες

Πρωτίστως θα ήθελα να ευχαριστήσω τον κύριο Ιωάννη Κατσαβουνίδη, για την ιδέα αυτής της διπλωματικής, την πολύτιμη βοήθειά του και την καθοδήγησή του καθ'όλη την διάρκεια της πραγματοποίησης της.

Επίσης ευχαριστώ πολύ τον κύριο Ιωάννη Χάντα, τον Στέργιο Πουλαράκη και τα υπόλοιπα μέλη της ερευνητικής ομάδας για την συμβολή τους.

Με την ολοκλήρωση των σπουδών μου, θα ήταν παράλειψη να μην ευχαριστήσω όλους τους καθηγητές του τμήματος για τις γνώσεις που απέκτησα.

Τέλος, ευχαριστώ την οικογένεια μου, τους συμφοιτητές και φίλους, για την στήριξη τους όλα αυτά τα χρόνια.

Περίληψη

Σκοπός αυτής της διπλωματικής είναι η υλοποιήση μιας εφαρμογής για την απόκτηση φωτογραφιών υπερανάλυσης(Super Resolution) με φορητές συσκευές Android σε σχεδόν πραγματικό χρόνο.

Με την τεχνική Super Resolution, παράγεται μια εικόνα υψηλής ανάλυσης από πολλές εικόνες χαμηλής ανάλυσης με την ίδια σκηνή.

Υλοποιήθηκε ένας αλγόριθμος Super Resolution στην γλώσσα προγραμματισμού C, και έγινε επιτάχυνση του αλγορίθμου με χρήση των εντολών Neon σε επεξεργαστές ARM της σειράς Cortex-A.

Περιεχόμενα

Εισαγωγή

Super Resolution (SR) είναι μία τεχνική για την αύξηση της ανάλυσης σε συστήματα εικόνας ξεπερνώντας τους περιορισμούς που θέτουν η τεχνολογία και οι αισθητήρες. Χρησιμοποιούνται τεχνικές επεξεργασίας σημάτων για την απόκτηση μιας εικόνας υψηλής ανάλυσης από πολλαπλές εικόνες χαμηλότερης ανάλυσης. Το βασικό πλεονέκτημα της συγκεκριμένης προσέγγισης είναι το χαμηλό κόστος καθώς και το γεγονός ότι τα υπάρχοντα συστήματα μπορούν να εξακολουθούν να χρησιμοποιούνται.

Το Super Resolution έχει αποδειχθεί ότι είναι ιδιαίτερα χρήσιμο σε περιπτώσεις στις οποίες πολλές φωτογραφίες με την ίδια σκηνή μπορούν να ληφθούν, όπως για παράδειγμα στην ιατρική, σε δορυφόρους και σε εφαρμογές video.

Στα πλαίσια αυτής της διπλωματικής:

- Μελετήθηκε το πρόβλημα του Super Resolution
- Υλοποιήθηκε μία εφαρμογή η οποία λαμβάνει ως είσοδο K εικόνες με το ίδιο σκηνικό, οποιασδήποτε ανάλυσης και παράγει μια εικόνα με 4 φορές μεγαλύτερη ανάλυση. Η υλοποίηση έγινε κυρίως σε γλώσσα C, αρχικά για PC και στην συνέχεια για Android συσκευές.

Παρακάτω γίνεται περιγραφή του προβλήματος, του αλγορίθμου, των τεχνικών και των εργαλειών που χρησιμοποιήθηκαν.

Σημείωση:
Για το Super Resolution υπάρχουν πολλές και διαφορετικές προσεγγίσεις. Σκοπός αυτής της διπλωματικής είναι να τις παρουσιάσει εν συντομία. Ο ενδιαφερόμενος αναγνώστης καλείται να ανατρέξει στην αναφορά [1] για περισσότερες λεπτομέρειες.

1.Μοντελοποίηση του προβλήματος

Ως πρώτο βήμα για να κατανοήσουμε το πρόβλημα του Super Resolution ας θεωρήσουμε ότι παίρνουμε μία εικόνα υψηλής ανάλυσης(HR-High Resolution) και από αυτήν θα δημιουργήσουμε έναν αριθμό από k διαφορετικές εικόνες χαμηλής ανάλυσης(LR-Low Resolution).

Έστω μια HR εικόνα μεγέθους $P1N1 \times P2N2$ που μπορεί να γραφτεί σαν ένα διάνυσμα τιμών $a = [a1, a2, \dots, a_N]^T$,

όπου $N = P1N1 \times P2N2$ και $P1, P2$ οι συντελεστές σμίκρυνσης(down-sampling) για τον οριζόντιο και τον κάθετο άξονα. Έτσι με επαναδειγματοληψία, μπορούν να προκύψουν οι (LR) εικόνες μεγέθους $N1 \times N2$

Ας θεωρήσουμε την i-στη (LR) εικόνα ως $y_i = [y_{i1}, y_{i2}, \dots, y_{iM}]^T$, $M = N1 \times N2$

Για τις εικόνες αυτές μπορούμε να θεωρήσουμε το παρακάτω μοντέλο:

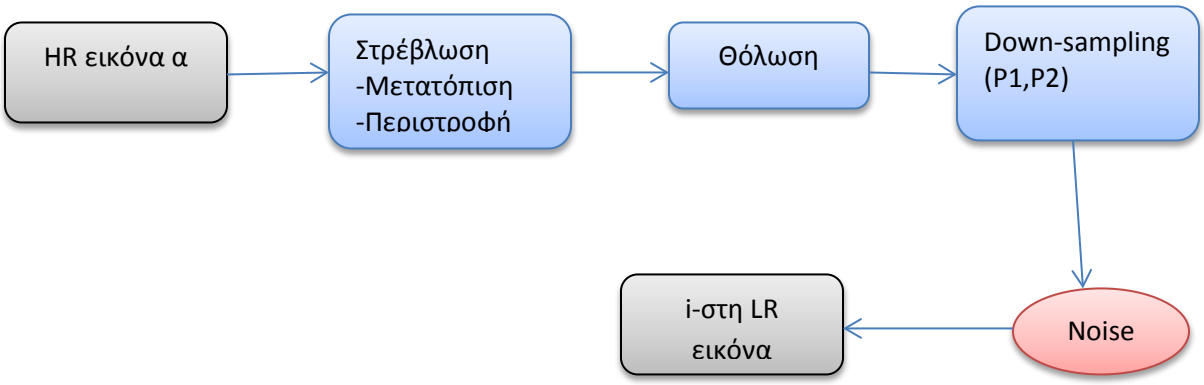
$y_i = DB_iW_ia + n_i$, $1 \le i \le k$

D: πίνακας σμίκρυνσης(down-sampling) μεγέθους $N1N2 \times P1N1P2N2$

B_i*: πίνακας θόλωσης μεγέθους* $P1N1P2N2 \times P1N1P2N2$

W_x*: πίνακας στρέβλωσης μεγέθους* $P1N1P2N2 \times P1N1P2N2$

n_i*: θόρυβος*



Έστω πίνακας Q_i τέτοιος ώστε $Q_i = DB_iW_i$ μεγέθους $N1N2 \times P1N1P2N2$

Τότε, $y_i = Q_i a + n_i$, $1 \leq i \leq k$

Ως συμπέρασμα μπορούμε εύκολα να δούμε ότι το πρόβλημα Super Resolution τελικά είναι ένα μεγάλο γραμμικό σύστημα:

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_k \end{bmatrix} = \begin{bmatrix} Q_1 \\ Q_2 \\ \vdots \\ Q_k \end{bmatrix} a + \begin{bmatrix} n_1 \\ n_2 \\ \vdots \\ n_k \end{bmatrix}$$

Συνεπώς, μπορούμε να εφαρμόσουμε μια από τις πολλές διαθέσιμες μεθόδους αντιστροφής για να ανακτήσουμε την επιθυμητή εικόνα υψηλής ανάλυσης (a) από τις εικόνες χαμηλής ανάλυσης. Η πιο γνωστή μέθοδος είναι αυτή του ψευδο-αντίστροφου πίνακα, υπό συνθήκες.

Συνήθως, οι μέθοδοι αυτές είναι υπολογιστικά πολύπλοκες, κάτι που τις κάνει σχεδόν απαγορευτικές για χρήση σε φορητές συσκευές με περιορισμένους πόρους επεξεργαστή και μνήμης, όπως και περιορισμένα ενεργειακά αποθέματα (μπαταρία).

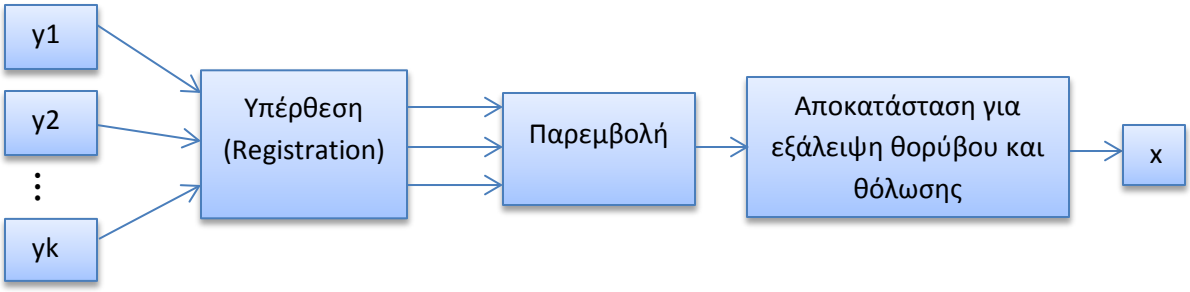
Για το λόγο αυτό αποφασίσαμε να στραφούμε σε μεθόδους πιο απλές (υπολογιστικά), με την ελπίδα να μπορούν να τρέξουν σε σχεδόν πραγματικό χρόνο σε μια φορητή συσκευή.

Οι περισσότερες μέθοδοι που έχουν προταθεί αποτελούνται από 3 στάδια, την υπέρθεση, την παρεμβολή και την αποκατάσταση για την εξάλειψη θόλωσης και θορύβου.

Στην υπέρθεση, μία από τις εικόνες χαμηλής ανάλυσης θεωρείται ως εικόνα αναφοράς και στη συνέχεια εκτιμώνται οι παράμετροι περιστροφής και μετατόπισης που παρουσιάζει κάθε μια από τις υπόλοιπες εικόνες χαμηλής ανάλυσης ως προς αυτή την εικόνα αναφοράς.

Στο δεύτερο στάδιο γίνεται αντιστοίχιση των pixels σε ένα πλέγμα super resolution. Δηλαδή δημιουργείται μια εικόνα μεγαλύτερης ανάλυσης.

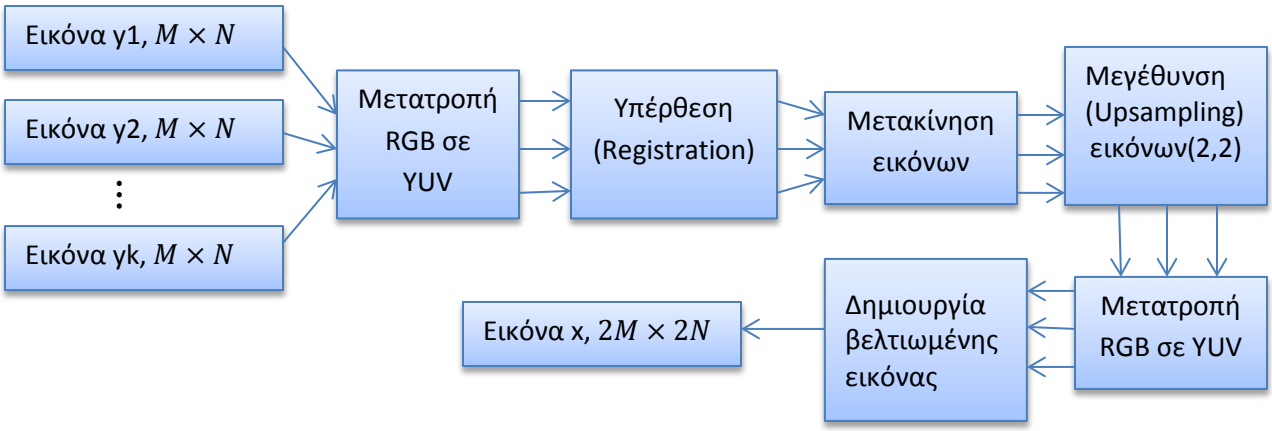
Τέλος στο τρίτο στάδιο, εφαρμόζεται η αποκατάσταση αυτής της εικόνας για να εξαλειφθεί ο θόρυβος καθώς και η θόλωση που προκαλεί ένας αισθητήρας.



Για την επίλυση του προβλήματος Super Resolution έχουν αναπτυχθεί πολλές τεχνικές. Μερικές κατηγορίες αυτών των τεχνικών είναι:

- SR μη ομοιόμορφης παρεμβολής
- Μέθοδος συχνοτικού χώρου
- Στοχαστικές τεχνικές regularized
- Προβολή σε κυρτά σύνολα (POCS)
- Υβριδικές τεχνικές ML-POCS
- Επαναληπτική μέθοδος οπισθοπροβολής

2.ΥΛΟΠΟΙΗΣΗ



Στο διάγραμμα αυτό φένεται η υλοποίηση που κάναμε. Τα βασικά – και με μεγαλύτερες υπολογιστικές απαιτήσεις – μέρη της υλοποιήσης είναι δύο: το πρώτο είναι η υπέρθεση, και το δεύτερο συμπεριλαμβάνει την μεγέθυνση των εικόνων και την δημιουργία βελτιωμένης εικόνας.

3.ΜΕΤΑΤΡΟΠΗ ΑΠΟ RGB ΣΕ YUV ΚΑΙ ΑΠΟ YUV ΣΕ RGB

Το πρότυπο RGB χρησιμοποιεί τις συντεταγμένες (R,G,B) (κόκκινο, πράσινο, μπλε) για την απεικόνιση του χρώματος. Επειδή όμως υπάρχει μεγάλη συσχέτιση αυτών των συντεταγμένων, για την επεξεργασία των εικόνων θα χρησιμοποιήσουμε τις συντεταγμένες (Y,U,V). Οι παρακάτω τύποι δίνουν μια υπολογιστικά «φιλική» υλοποίηση για τη μετατροπή από το ένα σύστημα συντεταγμένων στο άλλο, καθώς χρησιμοποιούν μόνο προσθέσεις/αφαιρέσεις, πολλαπλασιασμούς και ολισθήσεις (αντί για διαιρέσεις).

RGB ΣΕ YUV

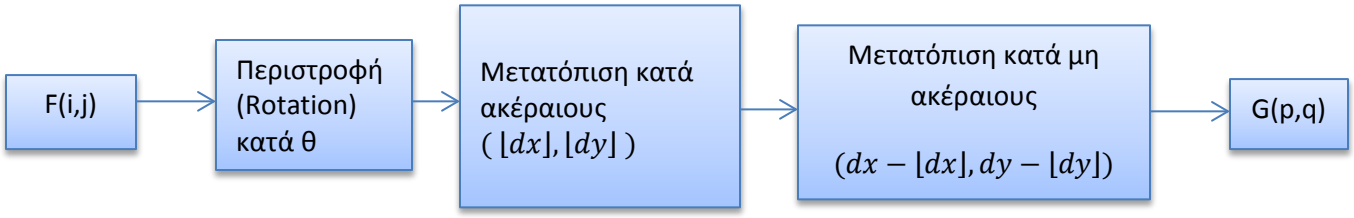
```
Y = ((  66 * R + 129 * G +  25 * B + 128) >> 8) +  16
U = (( -38 * R -  74 * G + 112 * B + 128) >> 8) + 128
V = (( 112 * R -  94 * G -  18 * B + 128) >> 8) + 128
```

YUV ΣΕ RGB

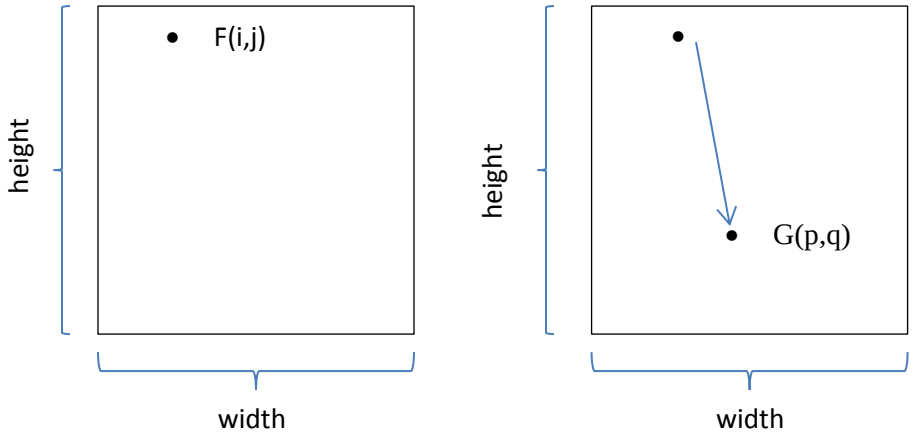
```
C = Y - 16
D = U - 128
E = V - 128
R = (298 * C + 409 * E + 128) >> 8   , 0<=R<=255
G = (298 * C - 100 * D - 208 * E + 128) >> 8   ,  0<=G<=255
B = (298 * C + 516 * D           + 128) >> 8   ,  0<=B<=255
```

4.ΜΕΤΑΚΙΝΗΣΗ ΜΙΑΣ ΕΙΚΟΝΑΣ

Η μετακίνηση/περιστροφή μιας εικόνας κατά μια γωνία, θ, και κατά διάνυσμα (dx,dy), το οποίο σε συντομία το συμβολίζουμε ως (θ,dx,dy), έγινε με την παρακάτω διαδικασία:



Πρώτα γίνεται η περιστροφή της εικόνας. Στην συνέχεια μετατοπίζουμε την περιστραμένη εικόνα κατά τους ακέραιους αριθμούς $(\lfloor dx \rfloor, \lfloor dy \rfloor)$ οριζοντίως και καθέτως (το σύμβολο [] υποδηλώνει το κάτω ακέραιο μέρος του ορίσματος), και τέλος αυτήν την εικόνα την μετατοπίζουμε κατά τους δεκαδικούς αριθμούς $(dx - \lfloor dx \rfloor, dy - \lfloor dy \rfloor)$. Η διαδικασία χωρίστηκε σε αυτά τα 3 μέρη για λόγους βελτιστοποίησης (βλέπε παρεμβολή - διαχωρισιμότητα).

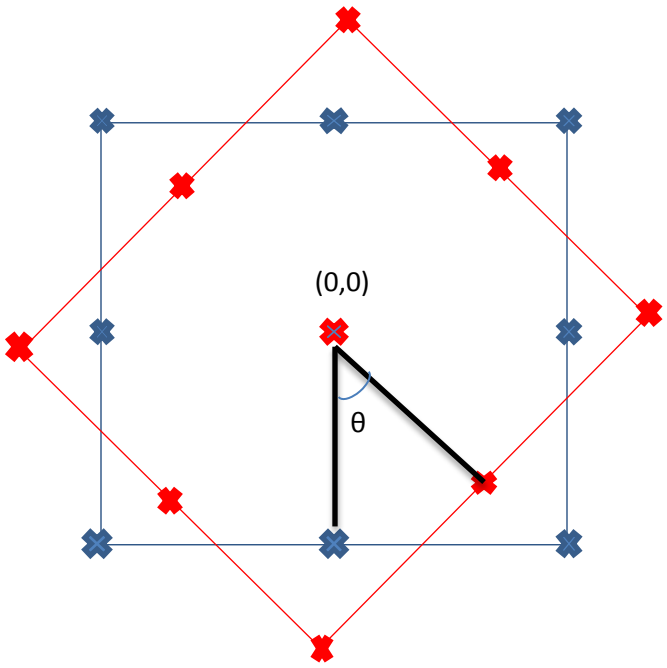


4.1.ΠΕΡΙΣΤΡΟΦΗ (ROTATION)

Έστω height το ύψος της εικόνας, και width το μήκος της. Για την περιστροφή θεωρούμε σημείο (0,0) το κέντρο της εικόνας. Οπότε υπολογίζουμε τις συντεταγμένες της νέας εικόνας με τις παρακάτω σχέσεις:

$$p = \left(i - \frac{heigth}{2}\right) * cos\theta - \left(j - \frac{width}{2}\right) * sin\theta + \frac{height}{2}, \qquad 0 \le p, i \le height - 1$$

$$q = \left(i - \frac{heigth}{2}\right) * sin\theta + \left(j - \frac{width}{2}\right) * cos\theta + \frac{width}{2}, \qquad 0 \le p, i \le height - 1$$



4.2.ΜΕΤΑΤΟΠΙΣΗ (TRANSLATION)

$$p = i + dy, \qquad 0 \le p, i \le height - 1$$

$$q = j + dx, \qquad 0 \le q, j \le width - 1$$

5.ΠΑΡΕΜΒΟΛΗ (INTERPOLATION)

Στην περιστροφή και στην μετατόπιση κατά δεκαδικούς αριθμούς απαιτείται να υπολογίσουμε τις τιμές της αρχικής εικόνας σε μη ακέραιες συντεταγμένες. Για αυτό το πρόβλημα χρησιμοποιήσαμε το φίλτρο Lanczos [7] το οποίο έχει θεωρηθεί ως μια πολύ καλή συμβιβαστική λύση για αυτό τον σκοπό μεταξύ πολλών απλών φίλτρων. Στην μετατόπιση κατά ακεραίους δεν χρειάζεται η χρήση του φίλτρου, οπότε συνετά την αποφεύγουμε.

5.1.LANCZOS ΦΙΛΤΡΟ

Το Lanczos φίλτρο που χρησιμοποιήσαμε είναι ένα γραμμικό φίλτρο(FIR) το οποίο χρησιμοποιεί την συνάρτηση sinc.

Πυρήνας Lanczos:

$$L(x) = \begin{cases} sinc(x)sinc\left(\frac{x}{a}\right), & -a < x < a \\ 0, & \alpha\lambda\lambda\iota\acute{\omega}\varsigma \end{cases}$$

$$L(x) = \begin{cases} 1, & x = 0 \\ \frac{\text{asin}(\pi x)\text{sin}(\frac{\pi x}{a})}{\pi^2 x^2}, & 0 < |x| < a \\ 0, & \alpha\lambda\lambda\iota\acute{\omega}\varsigma \end{cases}$$

Η παράμετρος α είναι ένας θετικός ακέραιος αριθμός συνήθως 2 ή 3 και προσδιορίζει πόσο μεγάλο είναι ένα φίλτρο.

Τύπος για Παρεμβολή σε μία διάσταση με s_i δείγματα:

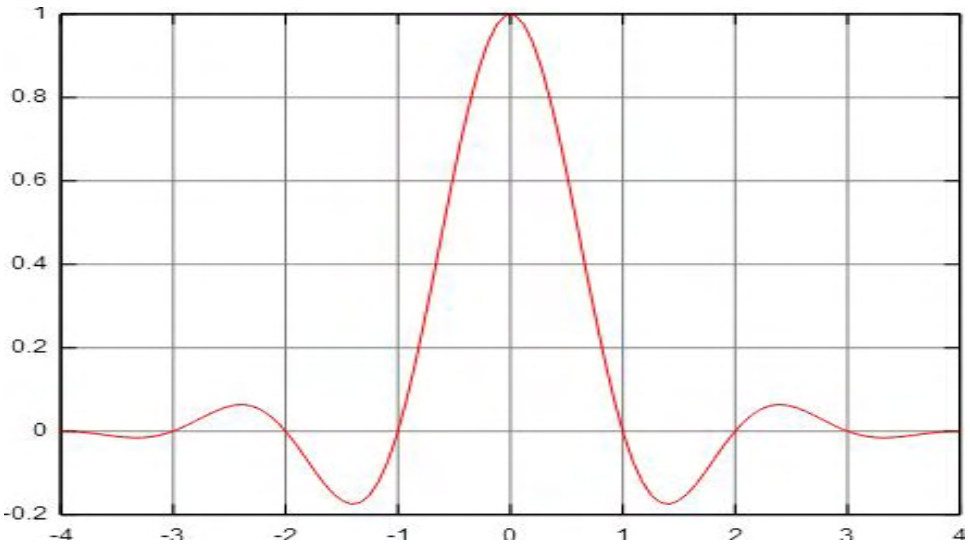
$$S(x) = \sum_{i=\lfloor x \rfloor - a + 1}^{\lfloor x \rfloor + a} s_i L(x - i)$$

Τύπος για παρεμβολή σε δύο διαστάσεις:

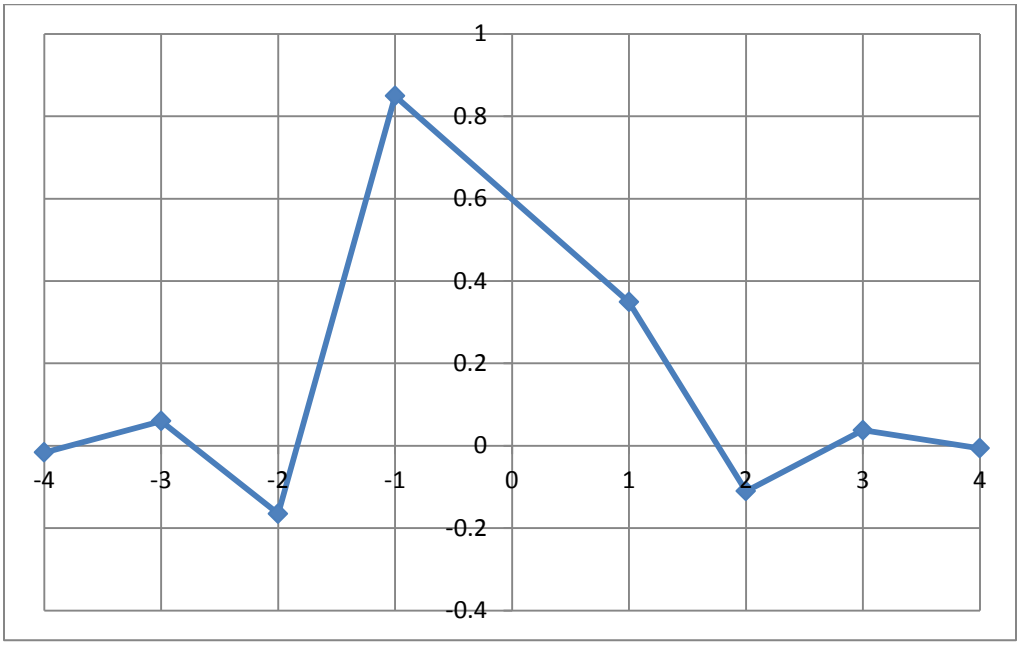
$$L(x, y) = L(x)L(y)$$

$$S(x, y) = \sum_{i=\lfloor x \rfloor - a + 1}^{\lfloor x \rfloor + a} \sum_{j=\lfloor y \rfloor - a + 1}^{\lfloor y \rfloor + a} s_{ij} L(x - i)L(y - j)$$

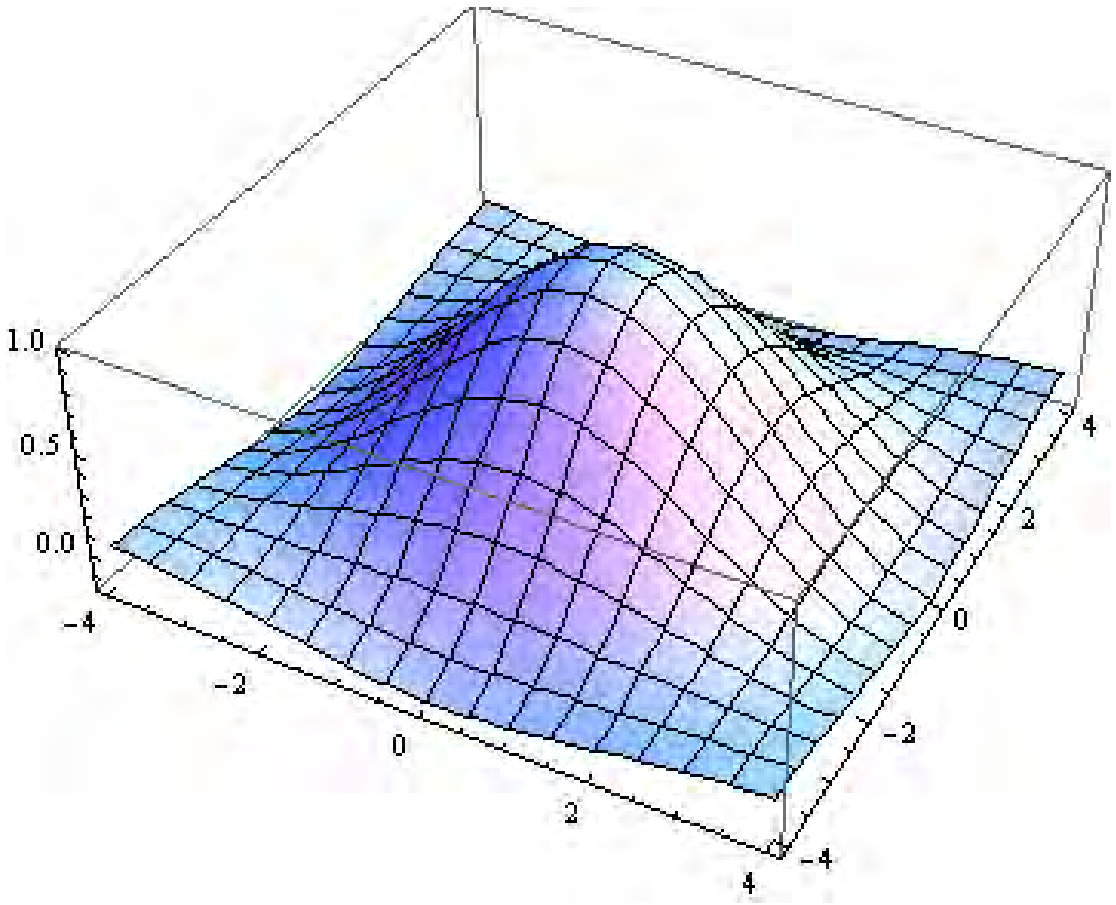
Στην υλοποίηση μας χρησιμοποιήσαμε για την παράμετρο α την τιμή 4. Οι τιμές που παίρνει το φίλτρο μίας διάστασης είναι 8.



Φίλτρο Lanczos με α=4



Παράδειγμα φίλτρου Lanczos με α=4 για μετατόπιση dx=0.3



Δισδιάστατο φίλτρο lanczos με α=4

5.2.ΔΙΑΧΩΡΙΣΙΜΟΤΗΤΑ ΦΙΛΤΡΟΥ

Όπως είπαμε, χρησιμοποιήσαμε την παράμετρο $a = 4$. Αυτό σημαίνει ότι το φίλτρο μας έχει 8 μη-μηδενικές τιμές σε κάθε διάσταση και έτσι προκύπτει ένας δισδιάστατος πίνακας $8 \times 8 = 64$ τιμών και πρέπει να κάνουμε για κάθε pixel της εικόνας 64 προσθέσεις και πολλαπλασιασμούς για να υπολογίσουμε την πράξη της δισδιάστατης συνέλιξης.

Ένα θέμα που πρέπει να εξετάζουμε όταν χρησιμοποιούμε ένα φίλτρο στην επεξεργασία εικόνας για λόγους πολυπλοκότητας, είναι αν το φίλτρο είναι διαχωρίσιμο. Δηλαδή κατά πόσο μπορούμε να χρησιμοποιήσουμε αντί για το αρχικό δισδιάστατο, 2 ή περισσότερα φίλτρα μιας διάστασης.

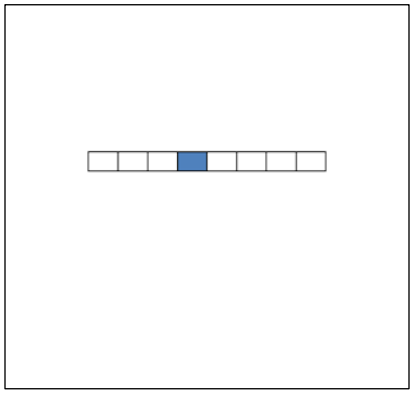
Το φίλτρο Lanczos μπορούμε να δούμε ότι είναι διαχωρίσιμο, κι έτσι μπορούμε να μειώσουμε την πολυπλοκότητα.

$$S(x,y) = \sum_{i=\lfloor x \rfloor - a+1}^{\lfloor x \rfloor + a} \sum_{j=\lfloor y \rfloor - a+1}^{\lfloor y \rfloor + a} s_{ij} L(x-i) L(y-j)$$

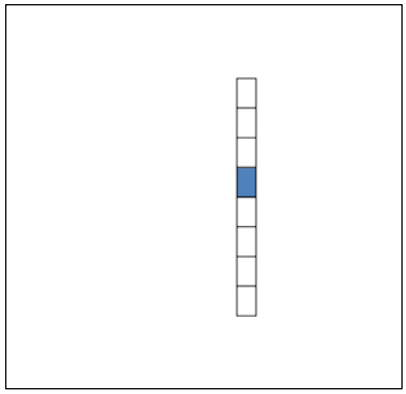
$$S(x,y) = \sum_{i=\lfloor x \rfloor - a+1}^{\lfloor x \rfloor + a} L(x-i) \sum_{j=\lfloor y \rfloor - a+1}^{\lfloor y \rfloor + a} s_{ij} L(y-j)$$

Έτσι, αντί να κάνουμε ένα φιλτράρισμα της εικόνας με ένα δισδιάστατο φίλτρο Lanczos μπορούμε να κάνουμε πρώτα ένα οριζόντιο φιλτράρισμα με ένα μονοδιάστατο φίλτρο Lanczos και μετά σε αυτήν την εικόνα που έχει δημιουργηθεί, να κάνουμε ένα κάθετο φιλτράρισμα με ένα άλλο μονοδιάστατο φίλτρο Lanczos.

οριζόντιο φιλτράρισμα



κάθετο φιλτράρισμα



Με αυτόν τον τρόπο οι πράξεις που γίνονται για ένα pixel είναι 8 πολλαπλασιασμοί και προσθέσεις για το οριζόντιο φιλτράρισμα και 8 πολλαπλασιασμοί και προσθέσεις για το κάθετο φιλτράρισμα. Συνολικά $8 + 8 = 16$ προσθέσεις και πολλαπλασιασμοί για ένα pixel, αντί για 64 που απαιτεί το αρχικό, διδιάστατο φίλτρο.

Συνεπώς, επιτυγχάνουμε μείωση της πολυπλοκότητας κατά 75%.

Η διαχωρισιμότητα του φίλτρου υλοποιήθηκε για την μετατόπιση κατά μη ακεραίους. Όσον αφορά την περιστροφή δεν είναι κάτι εμφανές ότι το φίλτρο είναι διαχωρίσιμο. Ο λόγος που δεν επιτεύχθηκε να διαχωριστεί το φίλτρο σε 2 μικρότερα φίλτρα είναι το γεγονός ότι σε μια περιστροφή δεν έχουν όλα τα pixel τις ίδιες μετατοπίσεις (dx,dy).

6.ΥΠΕΡΘΕΣΗ (REGISTRATION)

Πολύ σημαντικό είναι το πρόβλημα της υπέρθεσης. Δηλαδή η εύρεση των παραμέτρων (θ,dx,dy), όπου θ η γωνία περιστροφής, dx η οριζόντια μετατόπιση και dy η κάθετη μετατόπιση, που αντιστοιχούν σε κάθε εικόνα σε σχέση με την εικόνα αναφοράς. Ως εικόνα αναφοράς θέτουμε την πρώτη εικόνα.

Έστω ότι έχουμε K φωτογραφίες με την ίδια σκηνή και ας θεωρήσουμε την i-στη εικόνα ως y_i , $1 \leq i \leq K$ με μέγεθος $M \times N$.
Στην υπέρθεση χρησιμοποιούμε για την επεξεργασία εικόνας μόνο την συνιστώσα φωτεινότητας, Y, η οποία γνωρίζουμε ότι έχει την περισσότερη πληροφορία.

Το πρόβλημα βελτιστοποίησης που επιλέξαμε να λύσουμε για κάθε εικόνα y_i , $2 \leq i \leq K$, εκτός της πρώτης εικόνας που την θεωρήσαμε σημείο αναφοράς, είναι το παρακάτω.

$$(\theta_i^*, dx_i^*, dy_i^*) = arg \min_{\theta_i, dx_i, dy_i} \|y_1 - W(\theta_i, dx_i, dy_i) y_i\|_1$$

Εδώ, το $W(\theta_i, dx_i, dy_i)$ είναι ο τελεστής περιστροφής και μετατόπισης. Όπως είδαμε παραπάνω στην *μετακίνηση μιας εικόνας* αυτόν τον τελεστή τον χωρίσαμε σε 3 μέρη. Στην περιστροφή, στην μετατόπιση κατά ακέραιους και στην μετατόπιση κατά δεκαδικούς. Επίσης είπαμε ότι χρησιμοποιούμε το φίλτρο Lanczos στην περιστροφή και στην μετατόπιση κατά δεκαδικούς αριθμούς.

Αντίστοιχα και εδώ διαιρέσαμε το πρόβλημα σε 3 μέρη για λόγους απόδοσης.

Έστω

- $R(\theta_i)$ ο τελεστής περιστροφής.
- $T(sx_i, sy_i)$ ο τελεστής μετατόπισης κατά ακέραιους αριθμούς (sx_i, sy_i)
- $D(px_i, py_i)$ ο τελεστής μετατόπισης κατά μη ακέραιους αριθμούς (px_i, py_i) όπου $-1 < px_i, py_i < 1$

Το πρόβλημα που δημιουργήσαμε είναι το εξής:

$$(\theta_i^*, sx_i^*, sy_i^*, px_i^*, py_i^*) = arg \min_{\theta_i, sx_i, sy_i, px_i, py_i} \|y_1 - D(px_i, py_i)T(sx_i, sy_i)R(\theta_i)\ y_i\|_1$$

Όπου, $dx_i^* = sx_i^* + px_i^*$

$$dy_i^* = sy_i^* + py_i^*$$

6.1.MULTI-STEP Αλγόριθμος

Για την γρήγορη λύση αυτού του προβλήματος χρησιμοποιήσαμε έναν αλγόριθμο αναζήτησης multi-step. Τον οποίο χρησιμοποιήσαμε τρεις φορές. Μία φορά για μία μεταβλητή για την εύρεση γωνίας περιστροφής θ_i^*, μια δεύτερη φορά για την εύρεση των δύο ακέραιων μετατοπίσεων (sx_i^*, sy_i^*), και τέλος για την εύρεση των δεκαδικών μετατοπίσεων (px_i^*, py_i^*).

Αρχικά θα περιγράψουμε τον αλγόριθμο multi-step για 2 μεταβλητές (dx,dy).

Πρώτα θέτουμε ένα αρχικό βήμα step1 και ένα τελικό βήμα stepN ($stepN = step1/2^{N-1}$) που ορίζουν το εύρος των ορίων που ψάχνουμε. Το τελικό βήμα ορίζει και την ακρίβεια που έχουμε στο αποτέλεσμα.

Ξεκινάμε παίρνωντας ως κεντρικό σημείο το (dx=0,dy=0) και εξετάζουμε όλους τους δυνατούς συνδιασμούς για dx={0-step1, 0 , 0+step1} και dy={0-step1, 0 , 0+step1} , δηλαδή 9 σημεία. Από αυτά τα 9 σημεία επιλέγουμε το σημείο που μας δίνει το καλύτερο αποτέλεσμα, δηλαδή τη μικρότερη διαφορά ανάμεσα στις δύο συγκρινόμενες εικόνες. Στην συνέχεια ορίζουμε αυτό το σημείο ως κεντρικό σημείο και αλλάζουμε το βήμα σε step2 = (step1) /2. Δηλαδή αλλάζουμε το βήμα στο μισό του προηγούμενου βήματος.

Αυτή η διαδικασία συνεχίζεται μέχρις ότου να χρησιμοποιήσουμε και το τελικό βήμα. Το σημείο που θα βρούμε στο τέλος είναι και το αποτέλεσμα του προβλήματος μας.

Παρακάτω έχουμε ένα παράδειγμα του αλγορίθμου με αρχικό βήμα step1 = 8 και τελικό (μικρότερο) βήμα step4 = 1, δηλαδή τα όρια που ψάχνουμε είναι $(8 + 4 + 2 + 1 = 15)$

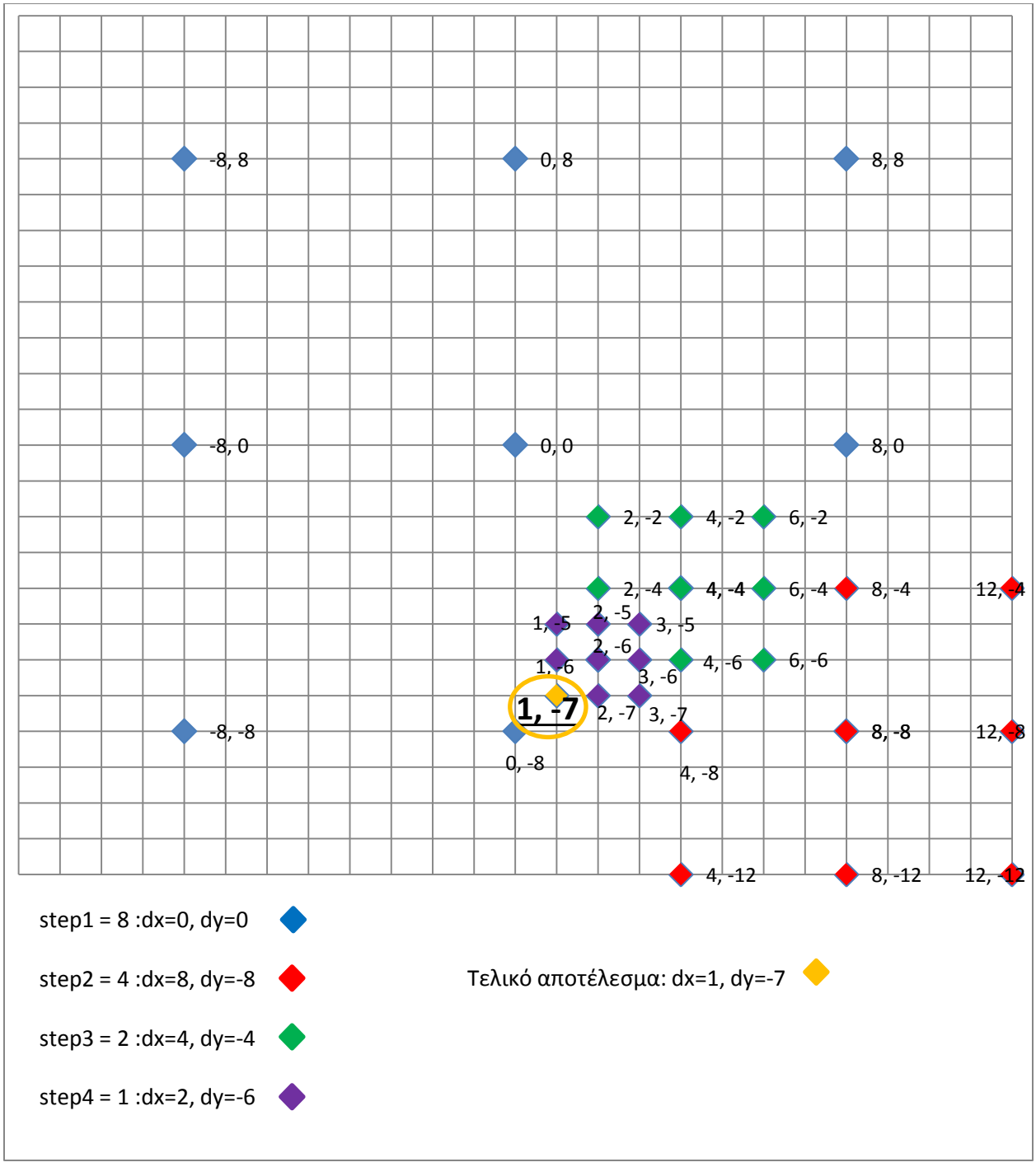
$-15 \leq dx, dy \leq 15$ και η ακρίβεια που έχουμε είναι 1.

1. Step1=8 (dx=0,dy=0) έστω καλύτερο αποτέλεσμα (dx=8,dy=-8)
2. Step2=4 (dx=8,dy-8) έστω καλύτερο αποτέλεσμα (dx=4,dy=-4)
3. Step3=2 (dx=4,dy-4) έστω καλύτερο αποτέλεσμα (dx=2,dy=-6)
4. Step4=1 (dx=2,dy-6) έστω καλύτερο αποτέλεσμα (dx=1,dy=-7)

Αποτέλεσμα στο παράδειγμα είναι το σημείο (dx=1,dy=-7)

Στο συγκεκριμένο παράδειγμα εξετάσαμε ως πιθανά αποτελέσματα $4 \times 9 = 36$ σημεία

Την στιγμή που με ένα full-search θα εξετάζαμε $31 \times 31 = 961$ σημεία για να βρούμε το σωστό αποτέλεσμα.


-8, 8
0, 8
8, 8
-8, 0
0, 0
8, 0
2, -2
4, -2
6, -2
2, -4
4, -4
6, -4
8, -4
12, -4
1, -5
2, -5
3, -5
2, -6
1, -6
3, -6
4, -6
6, -6
1, -7
2, -7
3, -7
-8, -8
0, -8
4, -8
8, -8
12, -8
4, -12
8, -12
12, -12
step1 = 8 :dx=0, dy=0
step2 = 4 :dx=8, dy=-8
step3 = 2 :dx=4, dy=-4
step4 = 1 :dx=2, dy=-6
Τελικό αποτέλεσμα: dx=1, dy=-7

6.2.Ο Αλγόριθμος με τα 3 Multi-step

Όπως αναφέρθηκε παραπάνω, για την λύση του προβλήματος της υπέρθεσης χρησιμοποιήσαμε 3 φορές τον Multi-step αλγόριθμο που περιγράψαμε προηγουμένως.

Κάναμε multi-step για μία μεταβλητή, για την εύρεση του θ_i^* όπου για κάθε θ_i που ελέγχαμε κάναμε περιστροφή της εικόνας κατά θ_i και για αυτήν την περιστραμένη εικόνα πρώτα κάναμε multi-step κατά ακέραιους αριθμούς(δηλαδή τελικό βήμα = 1) βρίσκοντας ένα σημείο (sx_i, sy_i), μετατοπίζαμε την περιστραμένη εικόνα κατά (sx_i, sy_i) οριζοντίως και καθέτως, και μετά κάναμε multi-step για μη ακέραιους αριθμούς (αρχικό βήμα =0.5) βρίσκοντας ένα σημείο (px_i, py_i).

Όποτε για κάθε θ_i βρίσκαμε τις βέλτιστες μετατοπίσεις (sx_i, sy_i, px_i, py_i). Οπότε, συγκρίνοντας τα βέλτιστα αποτελέσματα που μας δίνει ο αλγόριθμος για κάθε γωνία, επιλέγουμε κάθε φορά το ελάχιστο σφάλμα και συνεχίζουμε υποδιπλασιάζοντας το βήμα της γωνίας. Κατ'αυτό τον τρόπο, στο τέλος του αλγορίθμου multi-step για την μεταβλητή θ_i βρίσκουμε το αποτέλεσμα της υπέρθεσης $(\theta_i^*, sx_i^*, sy_i^*, px_i^*, py_i^*)$.

Ψευδοκώδικας:

rotation multi step(image1,image2):

```
min_step = 1/8; // μικρότερο βήμα 1/8 της μοίρας
step = 8; // αρχικό βήμα 8 μοίρες
theta = 0; //αρχικό κεντρικό theta

//multi_step κατά ακέραιους αποθηκεύοντας την μετατοπισμένη εικόνα στο image_temp
(best_sx, best_sy, image_temp) = integer_multi_step(image1, image2);

// multi_step κατά μη ακέραιους αριθμούς
(best_px, best_py, min_norm) = float_milti_step(image1,image_temp);

best_theta = 0;

while(step >= min_step)
{
       for(i = theta-step; i <= theta+step; i += step)
       {
              // περιστρέφει την εικόνα image2 κατά i μοίρες και την επιστρέφει ως
              //image_temp
              image_temp = rotation(image2,i);

              (sx,sy,image_temp) = integer_multi_step(image1, image_temp);
              (px, py, norm) = float_multi_step(image1,image_temp);

              if(norm < min_norm)
              {
                     (best_theta,best_sx,best_sy,best_px,best_py,min_norm) =
                     (i,sx,sy,px,py,norm);
```

```
                }
            }

            theta = best_theta;  // αλλάζουμε κεντρικό theta
            step = step/2;  //αλλάζουμε το step στο μισό
        }

        //επιστρέφει την περιστροφή και την μετατόπιση που πρέπει να γίνει στην εικόνα
        //image2
        return (best_theta,best_sx,best_sy,best_px,best_py);
```

integer multi_step(image1, image2):

```
            // μικρότερο βήμα 1 καθώς ελέγχουμε μόνο ακέραιες μετατοπίσεις
            min_step = 1;

            step = 16; //αρχικό βήμα 16

            //αρχικο κεντρικό σημείο (0,0)
            pointX = 0;
            pointY = 0;

            bestX = pointX - step;
            bestY = pointY - step;

            // μετατόπιση εικόνας image2 κατά ακέραιους (χωρίς φίλτρο)
            image_best = translation(image2, bestX, bestY)

            min_norm = L1_norm(image1, image_best);


            while(step >= min_step)
            {

                for(x=pointX-step; x<=pointX+step; x+=step)
                for(y=pointY-step; y<=pointY+step; y+=step)
                {
                    image_temp = translation(image2, x, y)
                    norm = L1_norm(image1, image_temp);

                    if(norm < min_norm)
                    {
                        (image_best,bestX,bestY,min_norm) =
                        (image_temp,x,y,norm);
                    }


                }

                //αλλαγή του κεντρικού σημείου και του βήματος
                pointX = bestX;
                pointY = bestY;
                step = step/2;
            }

            return(bestX,bestY,image_best);
```

float multi_step(image1 , image2):

```
min_step = 1/8 // μικρότερο βήμα
step = 0.5 // αρχικό βήμα 0.5

pointX = 0;
pointY = 0;

bestX = pointX - step;
bestY = pointY - step;

// μετατόπιση εικόνας image2 κατά μη ακέραιους με το φίλτρο lanczos
image_temp = lanczos_translation(image2, bestX, bestY)

min_norm = L1_norm(image1, image_temp);

while(step >= min_step)
{

	for(x=pointX-step; x<=pointX+step; x+=step)
	for(y=pointY-step; y<=pointY+step; y+=step)
	{
		image_temp = lanczos_translation(image2, x, y)
		norm = L1_norm(image1, image_temp);

		if(norm < min_norm)
		{
			(bestX,bestY,min_norm)=(x,y,norm);
		}


	}

	//αλλαγή του κεντρικού σημείου και του βήματος
	pointX = bestX;
	pointY = bestY;
	step = step/2;
}

return (bestX,bestY,min_norm);
```

6.3. L-1 Νόρμα

Από την παραπάνω ανάλυση, είναι φανερό ότι για τη δοκιμή των διαφόρων παραμέτρων του αλγόριθμου και επίλυση του προβλήματος της υπέρθεσης, χρησιμοποιείται το άθροισμα των απολύτων τιμών των διαφορών, το οποίο είναι γνωστό και ως L1 νόρμα, ο υπολογισμός της οποίας είναι σημαντικός.

Στον υπολογισμό της νόρμας δεν πρέπει και δεν είναι απαραίτητο να συμπεριλάβουμε όλα τα pixel των εικόνων. Ο λόγος είναι ότι πολλά από αυτά που βρίσκονται στα όρια μιας εικόνας δεν εμφανίζονται σε όλες τις εικόνες.

Έτσι επιλέγουμε η νόρμα ουσιαστικά να περιλαμβάνει έναν μικρότερο αριθμό από pixels τα οποία βρίσκονται στο κέντρο των εικόνων.

Στην υλοποίηση μας επιλέξαμε για τον υπολογισμό της νόρμας να χρησιμοποιούμε τα ¾ της κάθε διάστασης, δηλαδή το κεντρικό κομμάτι μιας εικόνας.

Ο υπολογισμός της νόρμας δίνεται από την σχέση:

$$\|y_1 - D(px_i, py_i)T(sx_i, sy_i)R(\theta_i)\ y_i\|_1 = \sum_{q=\lfloor M/8 \rfloor}^{\lfloor (7M)/8 \rfloor} \sum_{j=\lfloor N/8 \rfloor}^{\lfloor (7N)/8 \rfloor} |y_1[q,j] - \ y_i'[q,j]|$$

7.Μετακίνηση των εικόνων

Αφού έχει λυθεί το πρόβλημα της υπέρθεσης και έχουμε βρει όλες τις παραμέτρους, μπορούμε να κάνουμε την μετακίνηση των εικόνων, όχι μόνο για την συνιστώσα Y, αλλά και για τις U,V.

Οπότε για $2 \leq i \leq K$:

$$y_i = D(px_i^*,\ py_i^*)T(sx_i^*, sy_i^*)R(\theta_i^*)\ y_i$$

$$u_i = D(px_i^*,\ py_i^*)T(sx_i^*, sy_i^*)R(\theta_i^*)\ u_i$$

$$v_i = D(px_i^*,\ py_i^*)T(sx_i^*, sy_i^*)R(\theta_i^*)\ v_i$$

8.Μεγέθυνση (Upsampling)

Μετά την υπέρθεση, το ερώτημα που δημιουργείται είναι πως θα «ενώσουμε» τις K εικόνες $M \times N$ που έχουμε για να δημιουργήσουμε μια καλύτερη εικόνα με $2M \times 2N$ pixels.

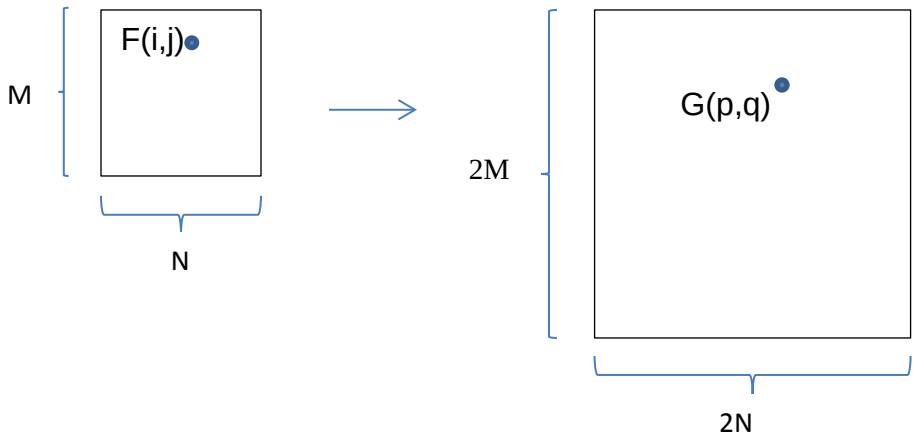
Για να γίνει αυτό, πρέπει σε πρώτο στάδιο να κάνουμε μεγέθυνση στις εικόνες που έχουμε και να αλλάξουμε τις διαστάσεις τους από $M \times N$ σε $2M \times 2N$

Για την μεγέθυνση χρησιμοποιήσαμε εκ νέου το φίλτρο Lanczos ακολουθώντας την ίδια λογική όπως με την μετατόπιση κατά μη ακέραιους αριθμούς, κάνοντας οριζόντιο φιλτράρισμα πρώτα και μετά κάθετο.

Έστω F(i,j) εικόνα εισόδου , G(p,q) εικόνα εξόδου

$$p = 2 * i\,, \qquad 0 \le p \le 2M - 1, \qquad 0 \le i \le M - 1$$

$$q = 2 * j\,, \qquad 0 \le q \le 2N - 1, \qquad 0 \le i \le N - 1$$



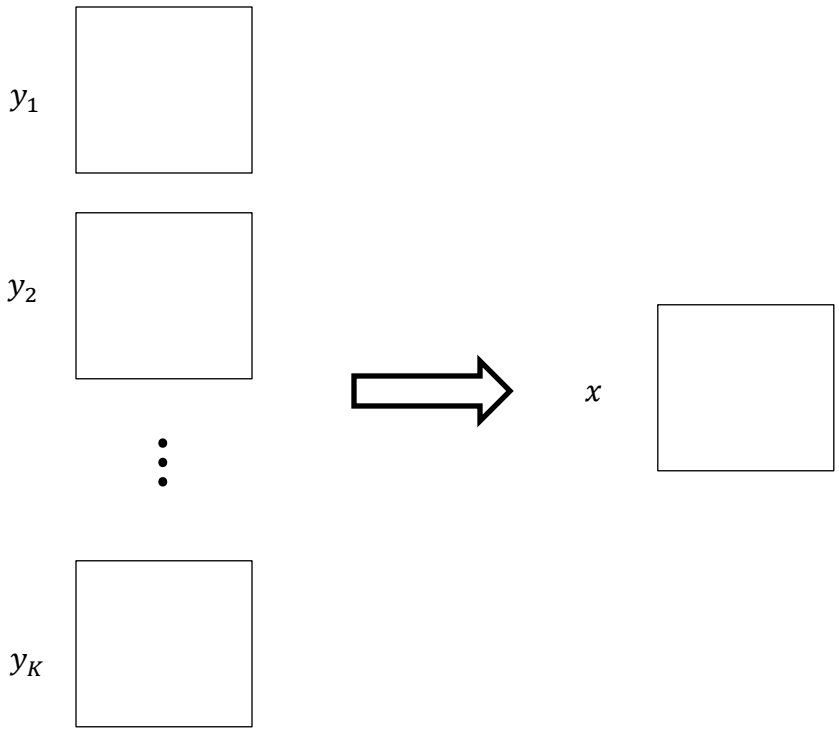
9.ΔΗΜΙΟΥΡΓΙΑ ΤΕΛΙΚΗΣ ΕΙΚΟΝΑΣ

Τέλος αυτό που μένει είναι να «ενώσουμε» τις K εικόνες $2M \times 2N$ και να δημιουργήσουμε την τελική εικόνα υπερανάλυσης(Super Resolution) , αθροίζοντας τις συνεισφορές των επιμέρους εικονών.

Έστω x η τελική εικόνα και $(y_1, y_2, \cdots, y_K)$ οι εικόνες που έχουμε κάνει μεγένθυση.

Ένα pixel της εικόνας x υπολογίζεται από τον μέσο όρο του άντιστοιχου pixel των εικόνων $(y_1, y_2, \cdots, y_K)$.

$$x[q,j] = \left\lfloor \frac{\sum_{i=1}^{K} y_i\,[q,j]}{K} + 0.5 \right\rfloor, \quad \forall\, 0 \le q \le 2M, 0 \le j \le 2N$$



10.SIMD (Single instruction – multiple data)

Είναι η ιδέα της παράλληλης εκτέλεσης μιας εντολής σε πολλαπλά δεδομένα. Αυτό εφαρμόζεται στους σημερινούς επεξεργαστές μέσω προσθηκών, όπως για παράδειγμα SSE, 3DNow, Altivec, Neon.

Οι SIMD εντολές αποτελούν ένα πολύ σημαντικό εργαλείο βελτιστοποίησης αλγορίθμων και μπορούν να επιφέρουν επιτάχυνση 3x-6x ανάλογα με την εφαρμογή και την ταχύτητα μνήμης.

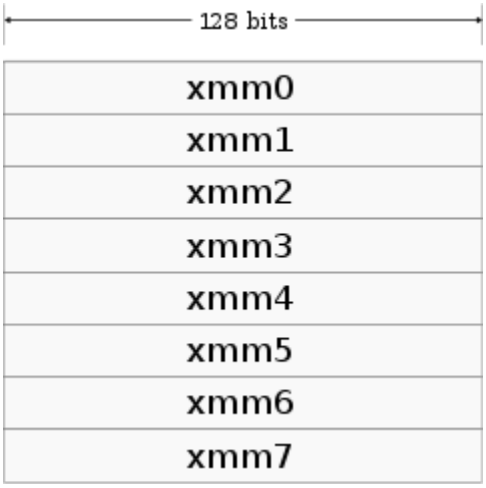
10.1.SSE Εντολές (Streaming SIMD Extensions)

Η Intel εισήγαγε πρώτα τις εντολές SSE μαζί με τους επεξεργαστές Pentium III το 1999. Η προηγούμενη προσπάθειά της ήταν οι εντολές MMX(Multi-Media-eXtension) οι οποίες επιτρέπουν πράξεις μεταξύ μόνο ακέραιων αριθμών 64-bits. Οι SSE επιτρέπουν πράξεις μεταξύ αριθμών 128-bits και αρχικά δημιουργήθηκαν ως ένα συμπληρωματικό στοιχείο των εντολών MMX ώστε να υποστηρίζονται πράξεις αριθμών κινητής υποδιαστολής.

Το SSE επεκτάθηκε από την Intel με τα SSE2,SSE3,SSSE3,SSE4. Λόγω του γεγονότος ότι υποστήριζε τόσο ακέραιους με συνολικό εύρος 128-bits όσο και αριθμούς κινητής υποδιαστολής, έγινε πιο δημοφιλές από το MMX.

Στην συνέχεια οι εντολές SSE έγιναν ακόμα πιο δημοφιλείς για τους προγραμματιστές καθώς η AMD υιοθέτησε τα SSE/SSE2 και στους δικούς της επεξεργαστές.

Για την εκτέλεση των SSE σχεδιάστηκαν 8 νέοι καταχωρητές των 128-bits οι οποίοι είναι γνωστοί ως XMM0,...,XMM7.



Στα SSE/SSE2 αυτοί οι καταχωρητές μπορούν να χρησιμοποιηθούν ως εξής:

- 4 32-bit αριθμούς float.
- 2 64-bit αριθμούς double
- 2 64-bit integers.
- 4 32-bit integers
- 8 16-bit short integers
- 16 8-bit characters

Στην υλοποίησή μας χρησιμοποιήσαμε SIMD εντολές στον υπολογισμό της L1 νόρμας, στο οριζόντιο και στο κάθετο φιλτράρισμα.

10.2.Υπολογισμός L1 νόρμας με SSE

$\|y_1 - y_i'\|_1 = \sum_{q=\lfloor M/8 \rfloor}^{\lfloor (7M)/8 \rfloor} \sum_{j=\lfloor N/8 \rfloor}^{\lfloor (7N)/8 \rfloor} |y_1[q,j] - y_i'[q,j]|$

Έχουμε 2 καταχωρητές a,b στους οποίους κάθε φορά φορτώνουμε τους επόμενους 16 8-bit αριθμούς των 2 πινάκων y_1, y_i'.

Η βασική εντολή που χρησιμοποιούμε για τον υπολογισμό της νόρμας είναι η **PSADBW** η οποία υπολογίζει την απόλυτη τιμή 16 unsigned 8-bit ακεραίων από τον καταχωρητή a και 16 unsigned 8-bit ακεραίων από τον καταχωρητή b.

Χρησιμοποιούμε τον καταχωρητή c για τους αριθμούς που επιστρέφονται από την εντολή.
Επιστρέφει το άθροισμα των 8 πρώτων απόλυτων τιμών και το άθροισμα των 8 επόμενων απόλυτων τιμών σε 2 unsigned 16-bit ακεραίους, ως εξής:

```
c0 := abs(a0 - b0) + abs(a1 - b1) +...+ abs(a7 - b7)
c1 := 0x0 ; c2 := 0x0 ; c3 := 0x0
c4 := abs(a8 - b8) + abs(a9 - b9) +...+ abs(a15 - b15)
c5 := 0x0 ; c6 := 0x0 ; c7 := 0x0
```

C7	C6	C5	C4	C3	C2	C1	C0

Ακόμα έχουμε έναν καταχωρητή e τον οποίο χρησιμοποιούμε ως αθροιστή.

Χρησιμοποιώντας την εντολή **PADDD** έχουμε:

e3	e2	e1	e0
0x0	C4	0x0	C0
e3=0x0	e2=e2+c4	e1=0x0	e0 = e0+c0

Τα αποτελέσματα αποθηκεύονται σε 2 32-bit integers.

Αφού «σκανάρουμε» τους πίνακες παίρνοντας όλες τις τιμές που χρειαζόμαστε, μας έχει μείνει ο καταχωρητής e, ο οποίος περιέχει 2 αριθμούς. Για να πάρουμε το τελικό αποτέλεσμα πρέπει να προσθέσουμε τους 2 αριθμούς. Οπότε κάνουμε πάλι PADDD στο e και στο e που το έχουμε κάνει shift δεξιά 8 bytes.

0x0	e2	0x0	e0
0x0	0x0	0x0	e2
0x0	e2	0x0	**e0+e2**

Τέλος παίρνουμε τα 32 λιγότερα σημαντικά bits που είναι το αποτέλεσμα και τα αποθηκεύουμε σε εναν integer.

10.3.Οριζόντιο φιλτράρισμα με SSE

Πρώτα πρέπει να τονίσουμε ότι στο φιλτράρισμα, οι τιμές του φίλτρου αλλά και οι τιμές της εικόνας που γίνεται το φιλτράρισμα είναι αριθμοί τύπου floating point. Για λόγους βελτιστοποίησης αυτούς τους αριθμούς τους κάναμε σταθερής υποδιαστολής (fixed-point), δηλαδή τους μετατρέψαμε σε integers 16-bit, εκ των οποίων ένας αριθμός από τα σημαντικότερα bits αποτελούν το ακέραιο μέρος, και τα υπόλοιπα bits αποτελούν το δεκαδικό μέρος.

$$S(x,y) = \sum_{i=\lfloor x \rfloor - 3}^{\lfloor x \rfloor + 4} s_{ij} L(x-i)$$

Για κάθε pixel της νέας εικόνας φορτώνουμε σε 2 καταχωρητές, a,b, τις 8 16-bit τιμές του φίλτρου και τις 8 16-bit τιμές της αρχικής εικόνας που χρειάζονται.

Εδώ η βασική εντολή που χρησιμοποιήσαμε είναι η PMADDWD η οποία πολλαπλασιάζει 8 signed 16-bit integers του καταχωρητή a με 8 signed 16-bit integers του καταχωρητή b. Το αποτέλεσμα που επιστρέφει το αποθηκεύουμε στον καταχωρητή c και είναι το παρακάτω.

a7	a6	a5	a4	a3	a2	a1	a0
b7	b6	b5	b4	b3	b2	b1	b0

c3=a6*b6+a7*b7	c2=a4*b4+a5*b5	c1=a2*b2+a3*b3	c0=a0*b0+a1*b1

4 32-bit integers

Στην συνέχεια κάνουμε 2 προσθέσεις με την εντολή PADDD και 2 φορές shift δεξιά, μία κατά 8 bytes και μία κατά 4 bytes.

c3	c2	c1	c0
0x0	0x0	c3	c2
c3	c2	c1+c3	c0+c2

c3	c2	c1+c3	c0+c2
0x0	c3	c2	c1+c3
c3	c2+c3	c1+c2+c3	**c0+c1+c2+c3**

Τέλος λαμβάνουμε το αποτέλεσμα για ένα pixel, από τα 32 λιγότερα σημαντικά bits.

10.4.Κάθετο φιλτράρισμα με SSE

$$S(x,y) = \sum_{j=\lfloor y \rfloor - 3}^{\lfloor y \rfloor + 4} s_{ij} L(y-j)$$

Στο κάθετο φιλτράρισμα αρχικά δημιουργούμε έναν δισδιάστατο πίνακα στην θέση του πίνακα του φίλτρου με τις 8 τιμές ως εξής:

Αν το φίλτρο είναι ο πίνακας:

λ1	λ2	λ3	λ4	λ5	λ6	λ7	λ8

Δημιουργούμε τον πίνακα 8×8 Ly

λ1	λ1	λ1	λ1	λ1	λ1	λ1	λ1
λ2	λ2	λ2	λ2	λ2	λ2	λ2	λ2
λ3	λ3	λ3	λ3	λ3	λ3	λ3	λ3
λ4	λ4	λ4	λ4	λ4	λ4	λ4	λ4
λ5	λ5	λ5	λ5	λ5	λ5	λ5	λ5
λ6	λ6	λ6	λ6	λ6	λ6	λ6	λ6
λ7	λ7	λ7	λ7	λ7	λ7	λ7	λ7
λ8	λ8	λ8	λ8	λ8	λ8	λ8	λ8

Στην ουσία πρόκειται για το ίδιο φίλτρο 8 φορές.

Για τον υπολογισμό των τιμών 8 pixel κάθε φορά, χρησιμοποιούμε 2 καταχωρητές a,b για να φορτώνουμε πρώτα την πρώτη γραμμή του Ly και την πρώτη γραμμή ενός υποπίνακα 8×8 της εικόνας, μετά την δεύτερη, την τρίτη κ.ο.κ.

Χρησιμοποιούμε τις εντολές PMULLW, PMULHW για να πολλαπλασιάσουμε 8 signed 16-bit integers του καταχωρητή a, με 8 signed 16-bit integers του καταχωρητή b.

Με την εντολή PMULLW πακετάρονται τα 16 χαμηλότερα bits του πολλαπλασιασμού, με την εντολή PMULHW τα 16 σημαντικότερα bits.

a7	a6	a5	a4	a3	a2	a1	a0
b7	b6	b5	b4	b3	b2	b1	b0
c7=a7*b7 [15:0]	c6=a6*b6 [15:0]	c5=a5*b5 [15:0]	c4=a4*b4 [15:0]	c3=a3*b3 [15:0]	c2=a2*b2 [15:0]	c1=a1*b1 [15:0]	c0=a0*b0 [15:0]

a7	a6	a5	a4	a3	a2	a1	a0
b7	b6	b5	b4	b3	b2	b1	b0
f7=a7*b7 [31:16]	f6=a6*b6 [31:16]	f5=a5*b5 [31:16]	f4=a4*b4 [31:16]	f3=a3*b3 [31:16]	f2=a2*b2 [31:16]	f1=a1*b1 [31:16]	f0=a0*b0 [31:16]

Στην συνέχεια για να ενώσουμε αυτούς τους αριθμούς χρησιμοποιούμε τις εντολές PUNPCKLWD , PUNPCKHWD

c7	c6	c5	c4	c3	c2	c1	c0
f7	f6	f5	f4	f3	f2	f1	f0
a7=f3	a6=c3	a5=f2	a4=c2	a3=f1	a2=c1	a1=f0	a0=c0

c7	c6	c5	c4	c3	c2	c1	c0
f7	f6	f5	f4	f3	f2	f1	f0
b7=f7	b6=c7	b5=f6	b4=c6	b3=f5	b2=c5	b1=f4	b0=c4

Έτσι έχουμε 2 καταχωρητές με 4 32-bit αριθμούς ο καθένας οι οποίοι είναι το αποτέλεσμα του παραπάνω πολλαπλασιασμού.

Έχουμε 2 καταχωρητές οι οποίοι είναι αθροιστές και κάθε φορά προσθέτουν το αποτέλεσμα του πολλαπλασιασμού.

d3	d2	d1	d0
a3	a2	a1	a0
d3=d3+a3	d2=d2+a2	d1=d1+a1	d0=d0+a0

g3	g2	g1	g0
b3	b2	b1	b0
g3=g3+b3	g2=g2+b2	g1=g1+b1	g0=g0+b0

Μετά την επανάληψη των παραπάνω 8 φορές, όσες είναι και οι γραμμές των πινάκων, έχουμε το αποτέλεσμα για 8 pixel του νέου πίνακα στους 2 καταχωρητές d,g, το οποίο και αποθηκεύουμε στην κατάλληλη θέση του πίνακα εξόδου.

d0	**d1**	**d2**	**d3**	**g0**	**g1**	**g2**	**g3**

Αυτή η διαδικασία γίνεται μέχρι να βρεθούν οι τιμές για όλα τα pixels της νέας εικόνας.

11.ANDROID

Άφου έγινε η υλοποίηση για PC, τα παραπάνω υλοποιήθηκαν και για το λειτουργικό Android. Οι πιο δημοφιλείς συσκευές τύπου Smartphone χρησιμοποιούν επεξεργαστές που υλοποιούν την αρχιτεκτονική της εταιρίας ARM, συγκεκριμένα την έκδοση ARMv7, η οποία συνήθως ενσωματώνει την τεχνολογία NEON, η οποία – όπως αναφέρθηκε – ανήκει στην οικογένεια των εντολών SIMD.

Το Android είναι λειτουργικό σύστημα για συσκευές κινητής τηλεφωνίας και tablets, το οποίο τρέχει τον πυρήνα του λειτουργικού Linux.

Είναι ανοικτού κώδικα που σημαίνει ότι ο πηγαίος κώδικας του είναι ελεύθερα προσβάσιμος για τροποποίηση και αναδιανομή από οποιονδήποτε (κατασκευαστές κινητών συσκευών, προγραμματιστές κτλ.). Επίσης υπάρχει μια μεγάλη κοινότητα από προγραμματιστές οι οποίοι δημιουργούν εφαρμογές («apps») με τις οποίες επεκτείνεται η λειτουργικότητα των συσκευών. Αυτοί οι παράγοντες επέτρεψαν στο Android να γίνει μια από τις πιο δημοφιλείς πλατφόρμες για έξυπνα-κινητά.

Η ανάπτυξη των εφαρμογών γίνεται κυρίως σε γλώσσα JAVA χρησιμοποιώντας το Android SDK (Software Development Kit). Το SDK περιλαμβάνει μια σειρά από εργαλεία, μεταξύ των οποίων debugger, βιβλιοθήκες, emulator. Το επίσημο περιβάλλον που υποστηρίζεται είναι το Eclipse IDE (Integrated Development Environment) με το πρόσθετο εργαλείο ADT (Android Development Tools).

Κάποιος που θέλει να ξεκινήσει την ανάπτυξη εφαρμογών για Android αρχικά χρειάζεται τα εργαλεία:

- Eclipse + ADT plugin
- Android SDK Tools

11.1.ANDROID-NDK

Ένα πρόσθετο εργαλείο είναι το Android-NDK (Native Development Kit) το οποίο δίνει την δυνατότητα στους προγραμματιστές να συμπεριλαμβάνουν κώδικα γραμμένο σε γλώσσες όπως C και C++.

Για κάποιους συγκεκριμμένους τύπους εφαρμογών το NDK μπορεί να φανεί χρήσιμο έτσι ώστε τυχόν υπάρχον κώδικας να μπορεί να επαναχρησιμοποιηθεί. Στις περισσότερες εφαρμογές όμως δεν συνιστάται η χρήση του NDK.

Το NDK δεν θα ωφελήσει, γενικά, στις περισσότερες εφαρμογές. Ένας προγραμματιστής πρέπει να ζυγίσει τα υπέρ και τα κατά πριν το χρησιμοποιήσει. Πολλές φορές η βελτίωση της απόδοσης δεν θα είναι αισθητή, αλλά η πολυπλοκότητα της εφαρμογής θα είναι σχετικά αυξημένη. Γενικά το NDK πρέπει να χρησιμοποιείται μόνο όταν είναι απαραίτητο για την εφαρμογή και ποτέ επειδή απλά ο προγραμματιστής προτιμάει να γράφει σε C/C++.

Το NDK είναι κατάλληλο για εφαρμογές που είναι ιδιαίτερα απαιτητικές υπολογιστικά, όπως είναι η επεξεργασία σημάτων. Έπισης, υποστηρίζει βιβλιοθήκες γραφικών, γεγονός το οποίο το καθιστά πολύ χρήσιμο εργαλείο για την ανάπτυξη παιχνιδιών για κινητές συσκευές, τομέας ο οποίος γνωρίζει μεγάλη άνθιση τα τελευταία χρόνια.

Το Super Resolution, όπως είδαμε, είναι ένα πολύπλοκο πρόβλημα, πολύ απαιτητικό υπολογιστικά. Με το NDK χρησιμοποιήσαμε τον υπάρχοντα κώδικα που είχε ήδη γραφτεί σε γλώσσα C, με κάποιες μικρές διαφοροποιήσεις.

- Στην θέση των SSE εντολών χρησιμοποιήσαμε τις εντολές NEON
- Χρησιμοποιήσαμε 2 νήματα (pthreads) σε κάποια σημεία του κώδικα για διπύρηνους επεξεργαστές.
- Χρησιμοποιήσαμε ένα επιπλέον διαθέσιμο εργαλείο, την βιβλιοθήκη OpenCV.

Εδώ, αξίζει να αναφέρουμε ότι η πρόσβαση στον Native κώδικα γίνεται μέσω του JNI (Java Native Interface).

Το JNI επιτρέπει στον κώδικα C/C++ να επικοινωνεί με το Android VM(Virtual Machine).

Για ανάπτυξη εφαρμογών σε Android δείτε τα links που παραθέτουμε στις αναφορές.

12. Εντολές NEON

Οι Neon εντολές είναι SIMD για επεξεργαστές ARM. Η τεχνολογία Neon έχει 32 καταχωρητές των 64-bits ή αλλιώς 16 καταχωρητές των 128-bits (μπορούν να χρησιμοποιηθούν και με τους 2 τρόπους). Υποστηρίζει δεδομένα signed/unsigned

8-bit,16-bit,32-bit,64-bit και αριθμούς κινητής υποδιαστολής.

Όπως γράψαμε τα κομμάτια κώδικα με SSE έτσι τα γράψαμε και με Neon. Η λογική που ακολουθήσαμε ήταν η ίδια, παρά το γεγονός ότι σε πολλές περιπτώσεις δεν υπήρχαν ακριβώς ίδιες εντολές Neon με αυτές της οικογένειας SSE. Επειδή όμως η λογική προγραμματισμού SIMD είναι κοινή, αυτό μας βοήθησε σε μεγάλο βαθμό.

12.1.Υπολογισμός L1 νόρμας με NEON

Δεν υπάρχει εντολή ίδια με την PSADBW SSE που χρησιμοποιήσαμε. Η υλοποίησή μας είναι η παρακάτω:

1. Χρησιμοποιούμε την εντολή VABD.U8 για να παίρνουμε τις απόλυτες τιμές μεταξύ των 16 unsigned 8-bit ακεραίων του καταχωρητή a και των 16 unsigned 8-bit ακεραίων του καταχωρητή b

$$c[i] = |a[i] - b[i]|$$

2. Με την εντολή VADDL.U8 προσθέτουμε τις 8 πρώτες τιμές του c με τις 8 επόμενες

c7	c6	c5	c4	c3	c2	c1	c0
c15	c14	c13	c12	c11	c10	c9	c8
d7=c7+c15	d6=c6+c14	d5=c5+c13	d4=c4+c12	d3=c3+c11	d2=c2+c10	d1=c1+c9	d0=c0+c8

3. Με την εντολή VADDL.U16 προσθέτουμε τις 4 πρώτες τιμές του d με τις επόμενες 4.

d3	d2	d1	d0
d7	d6	d5	d4
f3=d3+d7	f2=d2+d6	f1=d1+d5	f0=d0+d4

4. Χρησιμοποιούμε έναν αθροιστή e στον οποίο κάθε φορά προσθέτουμε τον καταχωρητή f με την εντολή VADD.I32

e3	e2	e1	e0
f3	f2	f1	f0
e3=e3+f3	e2=e2+f2	e1=e1+f1	e0=e0+f0

5. Στο τέλος για να πάρουμε το τελικό αποτέλεσμα προσθέτουμε τα στοιχεία του καταχωρητή e με την εντολή VADD.I32

e3	e2	e1	e0
0x0	0x0	e3	e2
e3	e2	e1=e1+e3	e0=e0+e2

e3	e2	e1+e3	e0+e2
0x0	e3	e2	e1+e3
e3	e2+e3	e1+e2+e3	**e0+e1+e2+e3**

Μπορούμε να δούμε ότι ακολουθήσαμε την ίδια τακτική όπως με SSE για να βρούμε το αποτέλεσμα.

12.2.Οριζόντιο φιλτράρισμα με NEON

Και σε αυτή την περίπτωη, δεν υπάρχει ακριβώς ίδια εντολή με την PMADDWD.

Χρησιμοποιήσαμε την εντολή VMLAL.S16 η οποία κάνει την εξής πράξη:

Vr[i] := Va[i] + Vb[i] * Vc[i]

Πολλαπλασιάζει 4 16-bit signed integers ενός καταχωρητή Vb με 4 16-bit signed integers ενός καταχωρητή Vc και τα προσθέτει με έναν καταχωρητή με 4 32-bit signed integers.

Οπότε χρησιμοποιήσαμε την συγκεκριμένη εντολή 2 φορές ως εξής:

a3	a2	a1	a0
b3	b2	b1	b0
0	0	0	0
d3=a3*b3	d2=a2*b2	d1=a1*b1	d0=a0*b0

a7	a6	a5	a4
b7	b6	b5	b4
d3	d2	d1	d0
d3=a3*b3+a7*b7	d2=a2*b2+a6*b6	d1=a1*b1+a5*b5	d0=a0*b0+a4*b4

Το μόνο που μένει είναι να προσθέσουμε τα d0,d1,d2,d3 και έχουμε το ζητούμενο αποτέλεσμα.

Με την εντολή `VADD.I32` και με τις κατάλληλες ολισθήσεις:

d3	d2	d1	d0
0x0	0x0	d3	d2
d3	d2	d1+d3	d0+d2

d3	d2	d1+d3	d0+d2
0x0	d3	d2	d1+d3
d3	d2+d3	d1+d2+d3	**d0+d1+d2+d3**

Λαμβάνουμε το επιθυμητό αποτέλεσμα από τα 32 λιγότερα σημαντικά bits.

12.3.Κάθετο φιλτράρισμα με NEON

Αφού δημιουργήσαμε τον 8x8 πίνακα με το φίλτρο, όπως κάναμε στο κάθετο φιλτράρισμα με SSE, ακολουθούμε την ίδια διαδικασία.
Χρησιμοποιούμε πάλι την εντολή `VMLAL.S16` ως εξής:

a3	a2	a1	a0
b3	b2	b1	b0
c3	c2	c1	c0
c3 = c3+a3*b3	c2 = c2+a2*b2	c1 = c1+a1*b1	c0 = c0+a0*b0

a7	a6	a5	a4
b7	b6	b5	b4
d3	d2	d1	d0
d3 = d3+ a7*b7	d2 = d2 + a6*b6	d1= d1+a5*b5	d0=d0+a4*b4

Οι καταχωρητές c,d είναι αθροιστές οι οποίοι προσθέτουν τα γινόμενα των στοιχείων των δυο πινάκων για κάθε γραμμή.

Στο τέλος των επαναλήψεων, οι δυο αυτοί καταχωρητές περιέχουν το αποτέλεσμα για 8 pixels.

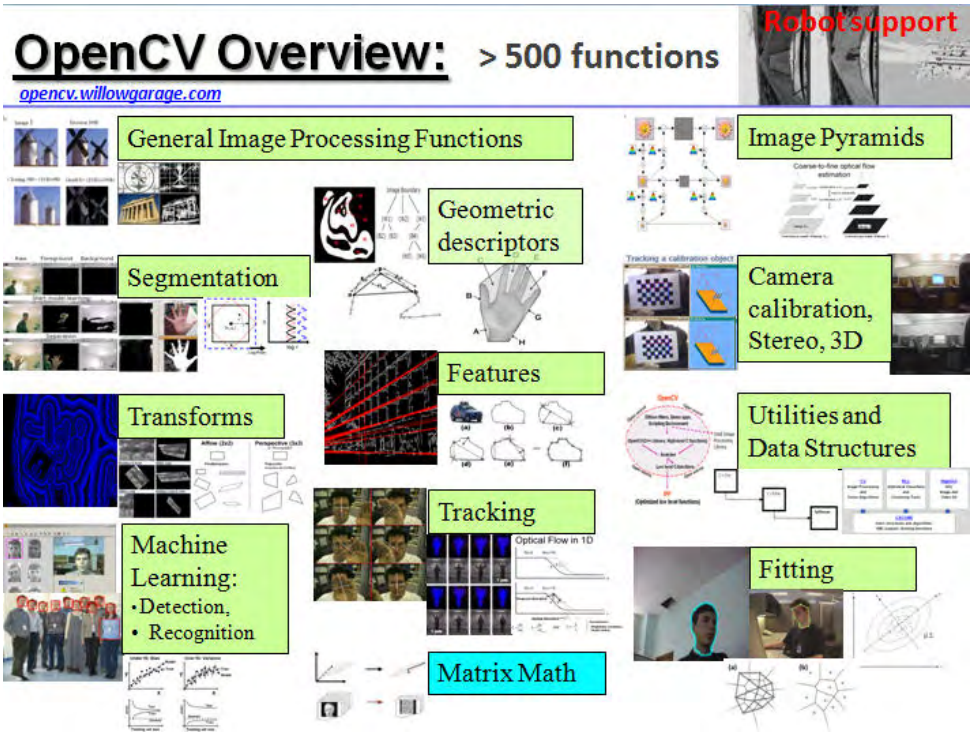
c0	c1	c2	c3	d0	d1	d2	d3

13.OpenCV (Open Source Computer Vision)

Το συγκεκριμένο εργαλείο χρησιμοποιήθηκε απλά για το «διάβασμα» και το «γράψιμο» των εικόνων. Παρ' όλα αυτά, θα θέλαμε να κάνουμε μια σύντομη αναφορά σε αυτό γιατί πρόκειται για ένα πολύ χρήσιμο εργαλείο το οποίο είναι διαθέσιμο για προγραμματισμό σε Android (για NDK και SDK) καθώς επίσης και για τις γλώσσες C++, C, Python, Java ενώ υποστηρίζει Windows, Linux, Mac OS, iOS.

Είναι μια βιβλιοθήκη από συναρτήσεις, η οποία σχεδιάστηκε για υπολογιστική αποδοτικότητα και εστιάζει κυρίως σε εφαρμογές «πραγματικού χρόνου».

Χρησιμοποιείται σε πολλούς τομείς μερικοί από τους οποίους φαίνονται στην παρακάτω εικόνα.



14.Pthreads(POSIX Threads)

Με την βιβλιοθήκη “pthread.h”, χρησιμοποιήσαμε δυο νήματα που τρέχουν παράλληλα, για διπύρηνους επεξεργαστές, στα πιο χρονοβόρα τμήματα του κώδικα μας. Στην περιστροφή, στο οριζόντιο και στο κάθετο φιλτράρισμα των εικόνων, χωρίζοντας την εικόνα σε δύο μισά, που αντιστοιχούν στο πάνω και κάτω μέρος της. Το αποτέλεσμα ήταν να μειωθεί ό χρόνος «τρεξίματος» του προγράμματος κατά τουλάχιστον 30%.

Τα Pthreads είναι ένα POSIX πρότυπο για νήματα. Το πρότυπο αυτό ορίζει ένα API για την δημιουργία και τον χειρισμό νημάτων.

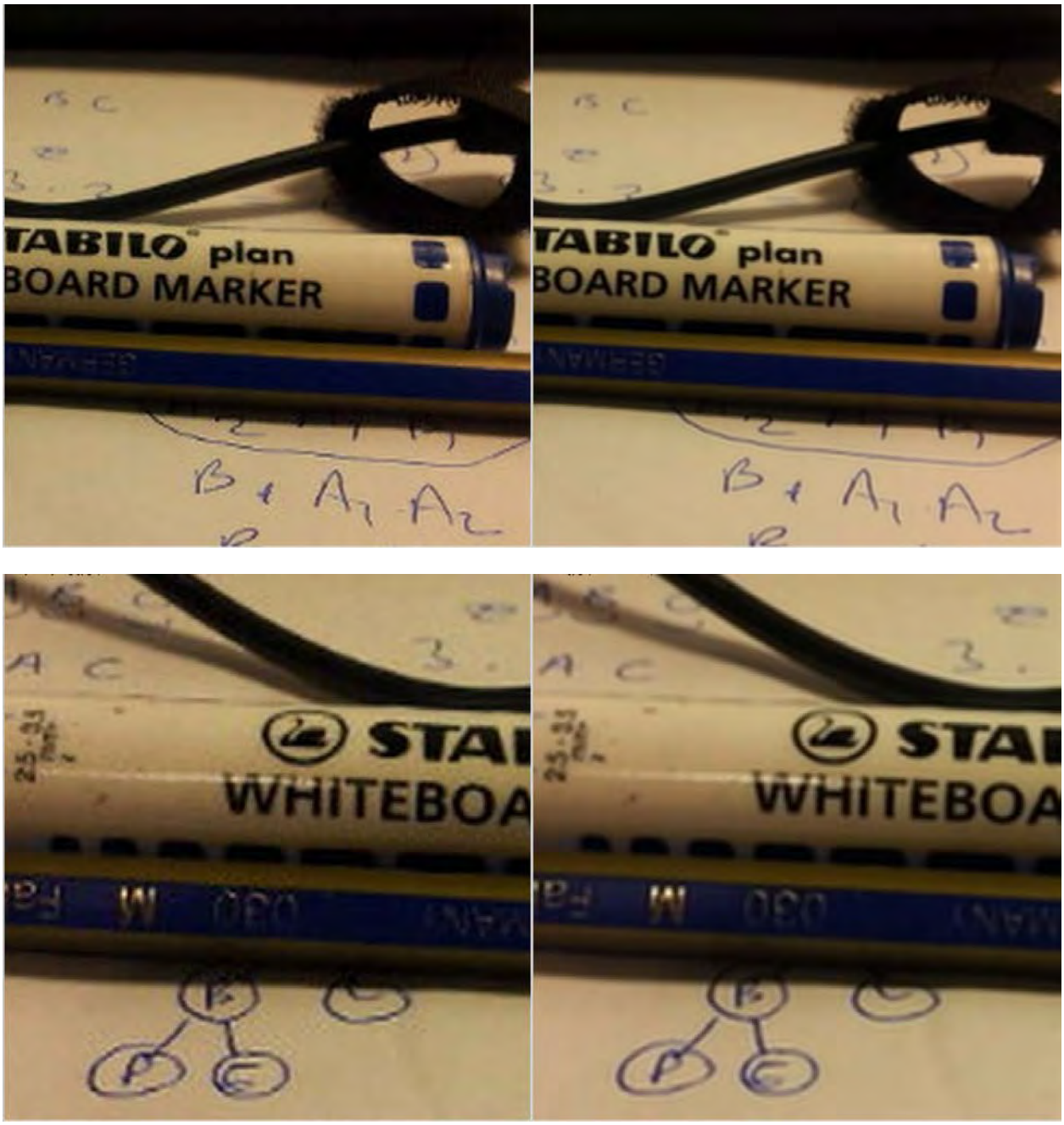
POSIX (Portable Operating System Interface) είναι μια οικογένεια από πρότυπα τα οποία καθορίζουν τις βασικές υπηρεσίες που παρέχει ένα λειτουργικό σύστημα, και δημιουργήθηκαν με στόχο την εξασφάλιση της συμβατότητας μεταξύ των διαφόρων λειτουργικών συστημάτων Unix καθώς και άλλων λειτουργικών.

15.ΑΠΟΤΕΛΕΣΜΑΤΑ

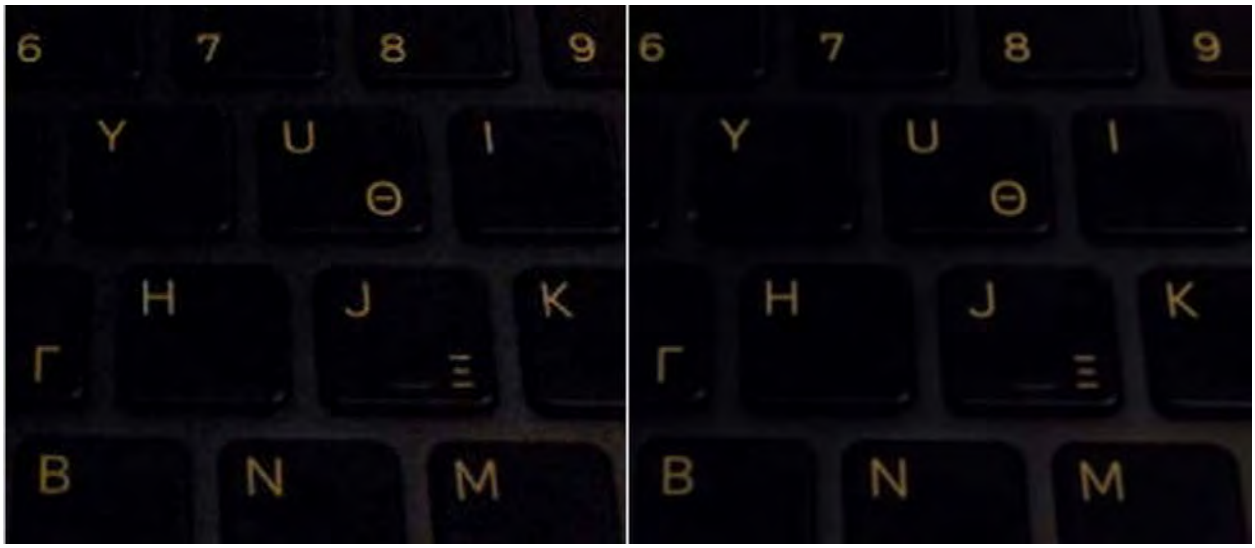
- 10 φωτογραφίες 640×480

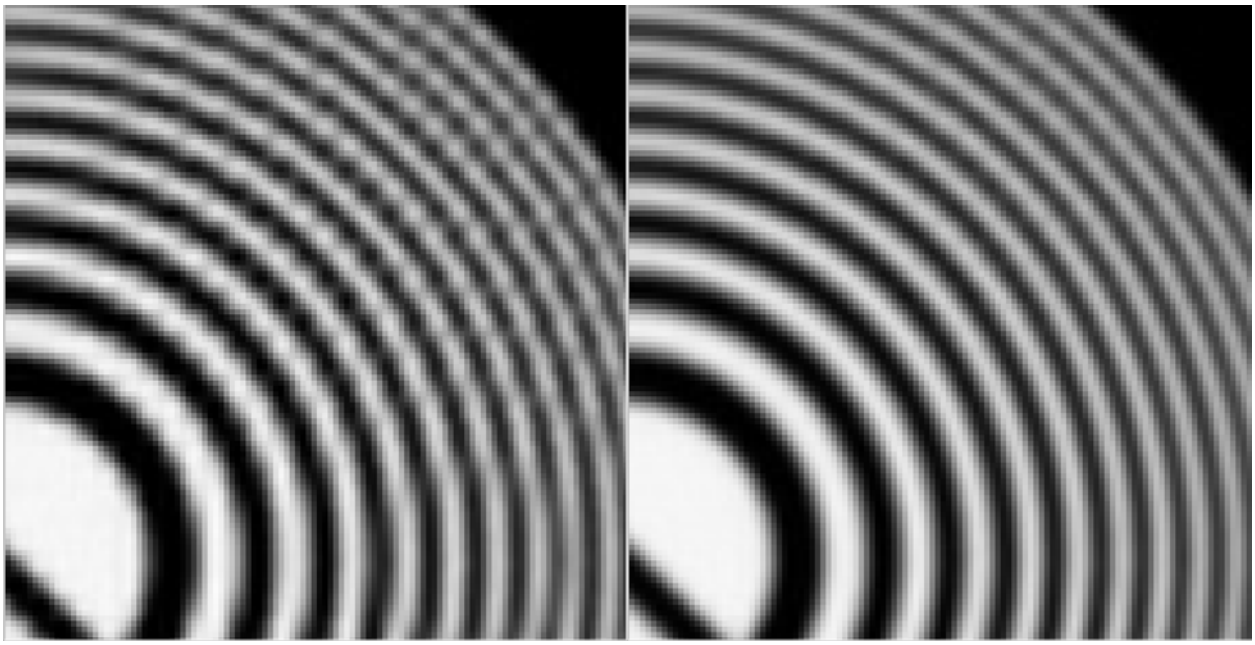
	CPU	Χρόνος
PC	Intel Core i7-3517U @1.90GHz 2,40GHz	~28 sec
Android	Dual-core 1.4 GHz ARM Cortex-A9	~196 sec

Παρακάτω μπορείτε να δείτε κομμάτια από την αρχική εικόνα και την τελική εικόνα που παράγεται.


TABILO plan
BOARD MARKER
TABILO plan
BOARD MARKER
WHITEBOA
WHITEBOA


SILO plan
RD MARKE
SILO plan
RD MARKE


6 7 8 9
Y U I
Θ
H J K
Γ Ξ
B N M
6 7 8 9
Y U I
Θ
H J K
Γ Ξ
B N M



Αναφορές

[1] Park, S., Park, M., Kang, M.: Super-resolution Image Reconstruction: a Technical Overview.

[2]Giannis Chantas, Nikolaos Galatsanos, Nathan Woods: Super-Resolution Based on Fast Registration and Maximum a Posteriori Reconstruction

[3] Giannis Chantas: Variational Bayesian Image Super-Resolution with GPU Acceleration

[4] WILLIAM K. PRATT: DIGITAL IMAGE PROCESSING

[5] Rafael C. Gonzalez: Digital Image Processing

[6] Converting Between YUV and RGB: http://msdn.microsoft.com/en-us/library/ms893078.aspx

[7] Lanczos resampling: https://en.wikipedia.org/wiki/Lanczos_resampling

[8] Get the Android SDK: https://developer.android.com/sdk/index.html

[9] Android NDK: https://developer.android.com/tools/sdk/ndk/index.html

[10]OpenCV for Android: http://opencv.org/platforms/android.html

[11]ARM NEON Intrinsics: http://gcc.gnu.org/onlinedocs/gcc/ARM-NEON-Intrinsics.html

[12] MMX, SSE, and SSE2 Intrinsics:
http://msdn.microsoft.com/en-us/library/y0dh78ez%28v=vs.80%29.aspx

[13] POSIX thread (pthread) libraries:
http://www.yolinux.com/TUTORIALS/LinuxTutorialPosixThreads.html