Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης

Πολυτεχνική Σχολή

Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Φοιτητής : **Ιγγλέζος Χαράλαμπος**

ΑΕΜ : **8145**

Εξάμηνο : 7ο

Έτος : 2016-2017 (4ο)

ΜΑΘΗΜΑ: ΠΑΡΑΛΛΗΛΑ ΚΑΙ ΔΙΑΝΕΜΗΜΕΝΑ ΣΥΣΤΗΜΑΤΑ

ΕΡΓΑΣΙΑ 3η

Παραλληλοποίηση με χρήση CUDA.

ΖΗΤΟΥΜΕΝΑ ΤΗΣ ΕΡΓΑΣΙΑΣ

Στο συνοδευτικό αρχείο παρέχεται κώδικας σε περιβάλλον `MATLAB`[1] που υλοποιεί το pipeline του αλγορίθμου *Non Local Means* [1] για την αποθορυβοποίηση εικόνας. Στόχος είναι η βελτίωση της απόδοσής του με τη χρήση `CUDA`[2].

Σε αντίθεση με φίλτρα *local mean*, που υπολογίζουν την μέση τιμή σε μία γειτονιά κάθε pixel για να εξομαλύνουν την εικόνα, ο *Non Local Means* υπολογίζει τον μέσο όρο όλων των pixels στην εικόνα, σταθμισμένο με το βαθμό ομοιότητας με το pixel αναφοράς. Το αποτέλεσμα είναι καλύτερη ευκρίνεια και διατήρηση των λεπτομερειών της αρχικής εικόνας[3].

Ο αλγόριθμος βασίζεται στην εύρεση παρόμοιων γειτονιών σε όλη την εικόνα και στον υπολογισμό της αποθορυβοποιημένης τιμής ως εξής:

$$\hat{f}(\mathbf{x}) = \sum_{\mathbf{y}\in\Omega} w(\mathbf{x},\mathbf{y}) f(\mathbf{y}), \qquad \forall \mathbf{x} \in \Omega,$$

όπου $\Omega \subset \mathbb{R}^2$ το πεδίο ορισμού της εικόνας, $f : \Omega \mapsto \mathbb{R}$ η θορυβώδης εικόνα και $\hat{f} : \Omega \mapsto \mathbb{R}$ η προσέγγιση της αποθορυβοποιημένης εικόνας.

Ο πίνακας βαρών $w(i, j)$ ορίζεται από την σχέση:

$$w(i,j) = \frac{1}{Z(i)} e^{-\frac{||f(\mathcal{N}_i) - f(\mathcal{N}_j)||^2_{G(a)}}{\sigma^2}},$$
$$Z(i) = \sum_j e^{-\frac{||f(\mathcal{N}_i) - f(\mathcal{N}_j)||^2_{G(a)}}{\sigma^2}},$$

όπου ως $\mathcal{N}_k$ ορίζεται μία τετράγωνη γειτονιά σταθερού μεγέθους με κέντρο το pixel k.

Χρησιμοποιώντας τον κώδικα που δίνεται σε `MATLAB` για επαλήθευση, το πρόγραμμά σας θα πρέπει να:

- Υλοποιεί τον υπολογισμό του $\hat{f}$ με χρήση δικού σας `CUDA kernel`, για τύπο δεδομένων `float` (η εικόνα παίρνει τιμές στο διάστημα $[0, 1]$).
- Αξιοποιεί την `shared memory` για μείωση των αναγνώσεων από την `global memory`, ώστε να επιταχυνθεί περαιτέρω η υλοποίηση.

ΕΙΣΑΓΩΓΗ

Αυτό που θέλουμε είναι ουσιαστικά να αντικαταστήσουμε τον κώδικα του non local means του Matlab με δικό μας Kernel σε γλώσσα C, στον οποίον θα υλοποιούμε παράλληλη αποθορυβοποίηση της εικόνας με επεξεργασία στην GPU με χρήση CUDA. Έπειτα, αυτόν τον Kernel θα τον καλούμε από το Matlab, από όπου αρχικά θα έχουμε εισάγει την εικόνα και θα της έχουμε προσθέσει τον θόρυβο και τελικά θα πρέπει να παράγεται η επεξεργασμένη εικόνα χωρίς τον θόρυβο ιδανικά, φυσικά

με κάποια περιθώρια σφάλματος. Αυτό επιτυγχάνεται με τη χρήση του παραπάνω αλγορίθμου της εκφώνησης, ο οποίος περιγράφει μαθηματικώς τα βήματα για την ομαλή λειτουργία της αποθορυβοποίησης και αυτόν υλοποιούμε στον Kernel όσο το δυνατόν αυτό γίνεται. Τα δεδομένα εκτελέστηκαν στο σύστημα Diades του ΑΠΘ και τα αποτελέσματα που θα παρουσιαστούν στην συνέχεια προέρχονται από την επεξεργασία στον Diades.

ΠΕΡΙΓΡΑΦΗ ΤΟΥ ΚΩΔΙΚΑ

Τα αρχεία κώδικα προς σχολιασμό είναι το Kernel και το pipeline_non_local_means.

Η λογική του κώδικα είναι η εξής:

Αν οι διαστάσεις της εικόνας είναι για παράδειγμα [n m] = [64 64], τότε δημιουργώ thread έστω [16 16] για κάθε μπλοκ , με πλήθος μπλοκ όσο είναι η διάσταση του γενικευμένου πίνακα μετά το padding διά τα νήματα ανά μπλοκ. Έτσι, ουσιαστικά κάθε μπλοκ περιέχει μια πληθώρα νημάτων που επεξεργάζονται και υπολογίζουν δεδομένα για μερικά πίξελ, παράλληλα.

Babis_Kernel_Jan_2017.cu

Αποτελεί τον Kernel σε γλώσσα C και υλοποιεί την κυρίως επεξεργασία της εικόνας με θόρυβο, ώστε να αφαιρεθεί ο θόρυβος τελικά και το τελικό αποτέλεσμα να μοιάζει σχετικά στην αρχική εικόνα μας χωρίς θόρυβο.

Η __global__ void Babis_Kernel(double const *A, double *B, double const *G, int n, int m, int patchSize_x, int patchSize_y, double filtSigma) είναι η βασική συνάρτηση που θα τρέχει στην GPU και θα εφαρμόζει CUDA.

Σαν ορίσματα έχει τον πίνακα A που είναι η εικόνα με τον θόρυβο προς επεξεργασία μετά το padding που έχει γίνει εκ των προτέρων στο matlab, τον πίνακα B που είναι ο μηδενικός nxm και την κατανομή Gauss G που θα δράσει σαν φίλτρο για την αποθορυβοποίηση της εικόνας. Επίσης τις διαστάσεις του γενικευμένου πίνακα, το σ για το φίλτρο και την περιοχή των γειτόνων, το patch area, κατά το ήμισυ, δηλαδή αν πχ είναι η 3x3. εδώ θα είναι οι δύο μεταβλητές 1 αφού παίρνουμε το floor της πράξης.

Τα i,j είναι οι συντεταγμένες των thread λαμβάνοντας υπ'όψιν τα μπλοκ φυσικά, δηλαδή είναι τα blockIdx.x*blockDim.x + threadIdx.x και blockIdx.y*blockDim.y + threadIdx.y.

Στη συνέχεια η όλη διαδικασία γίνεται μέσα σε 4 επαναλήψεις for.

Οι δείκτες x,y δείχνουν στο κάθε ένα από τα πίξελ της εικόνας ώστε να υπολογίσουν δεδομένα σε σχέση με το βασικό πίξελ i,j, σαρώνοντας ουσιαστικά όλην την εικόνα.

Οι δείκτες area_x, area_y δείχνουν σε κάθε πίξελ εντός της γειτονιάς (πχ 3x3, 5x5, 7x7) του πίξελ x,y.

Η λογική είναι ότι για κάθε νήμα, για κάθε i,j πίξελ θα βρω την απόστασή του με κάθε άλλο x,y πίξελ της εικόνας και για να γίνει αυτό θα πρέπει να σαρώνω μια περιοχή area_x,area_y γύρω από κάθε ένα από αυτά τα πίξελ, τα οποία area_x,area_y παίρνουν τιμές -1,0,1 για 3x3 περιοχή, -2,-1,0,1,2 για 5x5 κλπ, ενώ τα x,y έχουν τιμές μέχρι nxm όπως είναι επόμενο, ξεκινώντας από την περιοχή του πίνακα εντός του padding και προφανώς εντός ορίων του.

Έπειτα, κάνουμε την αφαίρεση των τιμών του A στις θέσεις αυτές, πολλαπλασιάζουμε το αποτέλεσμα με την αντίστοιχη θέση της περιοχής πχ 3x3 του φίλτρου Gauss, υψώνουμε στο τετράγωνο και αθροίζουμε το αποτέλεσμα σε μια τοπική μεταβλητή που υποδηλώνει τη νόρμα. Μόλις υπολογιστεί αυτή για όλα τα πίξελ της περιοχής αυτής, περνάμε το εκθετικό exp((-1/s)*norm) με τη νόρμα μέσα, σε μια μεταβλητή w_temp και προσαυξάνουμε την W κατά w_temp, ενώ ακόμα προσαυξάνουμε την Products κατά w_temp*A[x*m+y].

Τέλος, περνάμε στον πίνακα εξόδου μας, τον B, το τελικό αποτέλεσμα για κάθε πίξελ, που είναι Products/W, στη θέση του πίνακα [(i-patchSize_x)*(m-(2*patchSize_y)) + (j-patchSize_y)].

Επειδή το κάθε νήμα καθορίζεται από τα i,j, η θέση αυτή λαμβάνει υπ'όψιν της τα νήματα οπότε αποκλείονται race conditions.

Ο B περάστηκε σαν όρισμα ως ο μηδενικός με τη λογική να κρατήσει το τελικό αποτέλεσμα.

pipeline_non_local_means.m

Αποτελεί το βασικό αρχείο/σκριπτ του matlab, στο οποίο γίνεται η εισαγωγή της εικόνας, υφίσταται την αρχική προεπεξεργασία και εισάγεται ο θόρυβος και καλείται ο Kernel για την αποθορυβοποίηση και τέλος εμφανίζει τα αποτελέσματα σε εικόνες.

Ορίζουμε τις βασικές μεταβλητές του προβλήματος και εισάγουμε κατάλληλα οποιαδήποτε εικόνα (πχ jpg) επιθυμούμε. Αυτή μετατρέπεται σε MAT αρχείο και αποθηκεύεται σε πίνακα ώστε να υποστεί την επεξεργασία στη συνέχεια.

Ορίζουμε τα νήματα ανά block και πόσα blocks θέλουμε να έχει το grid μας.

Η ουσιαστική διαφοροποίηση από τον αρχικό κώδικα είναι από το σημείο που δημιουργούμε το Kernel αντικείμενο. Σε εκείνο το σημείο ορίζουμε τα blocks και τα

threads του αντικειμένου που θα τρέξει παράλληλα CUDA στην GPU και αφού δημιουργήσουμε τους απαραίτητους πίνακες, τον μηδενικό, το φίλτρο Gauss και την εικόνα με θόρυβο, τους περνάμε στην GPU.

Μετά εκτελούμε τον Kernel και χρονομετράμε τον κώδικα για τη μεταφορά δεδομένων στην GPU, για την εκτέλεσή τους και για την επιστροφή τους στην CPU.

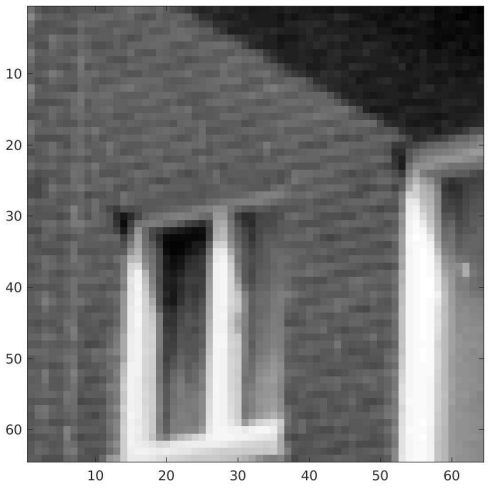
Τέλος, εμφανίζουμε τις τελικές εικόνες και το υπόλοιπο, καθώς και τους χρόνους αυτούς.

ΑΠΟΤΕΛΕΣΜΑΤΑ

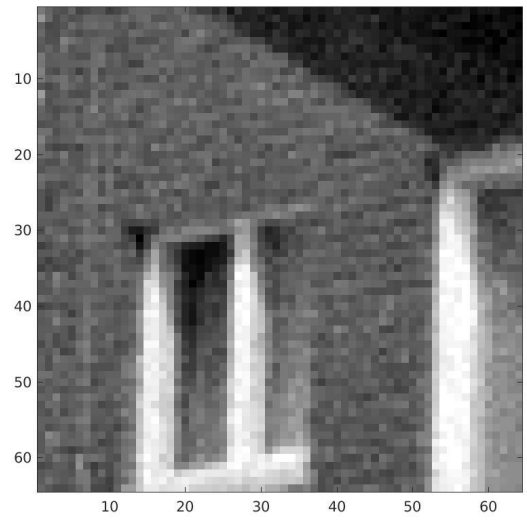
Παραθέτουμε τα αποτελέσματα από τον Diades για πλήθος threads σε όλες τις περιπτώσεις ίσο με 16.

- 64x64, 3x3

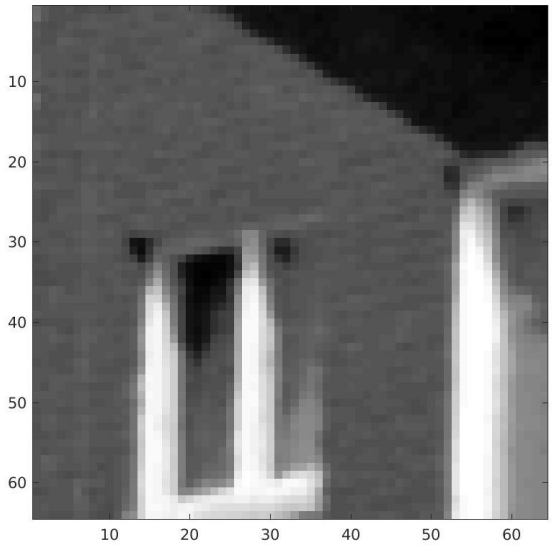
Original image



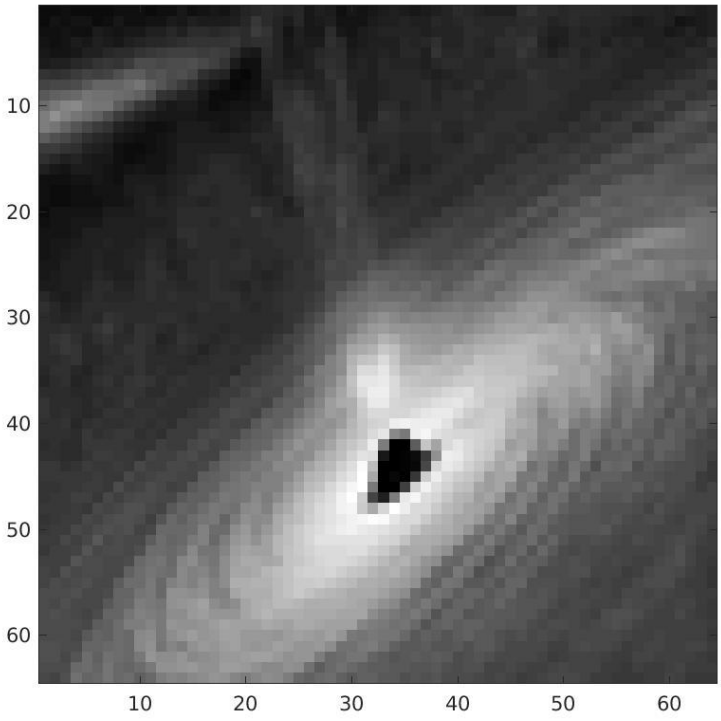
Original image with noise



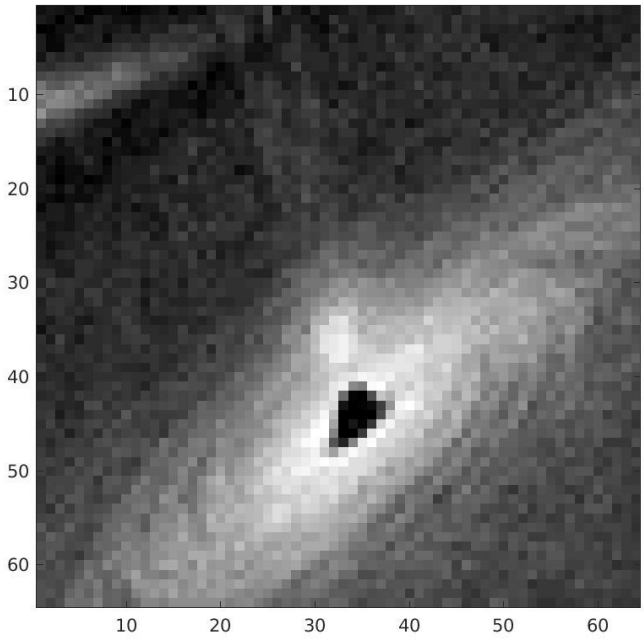
Filtered image



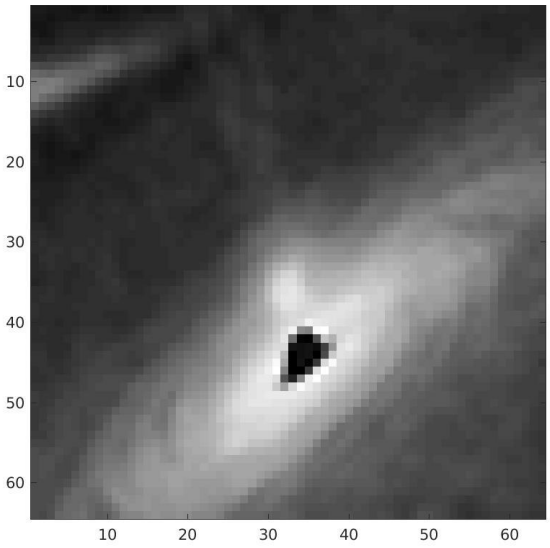
Original image



Original image with noise



Filtered image



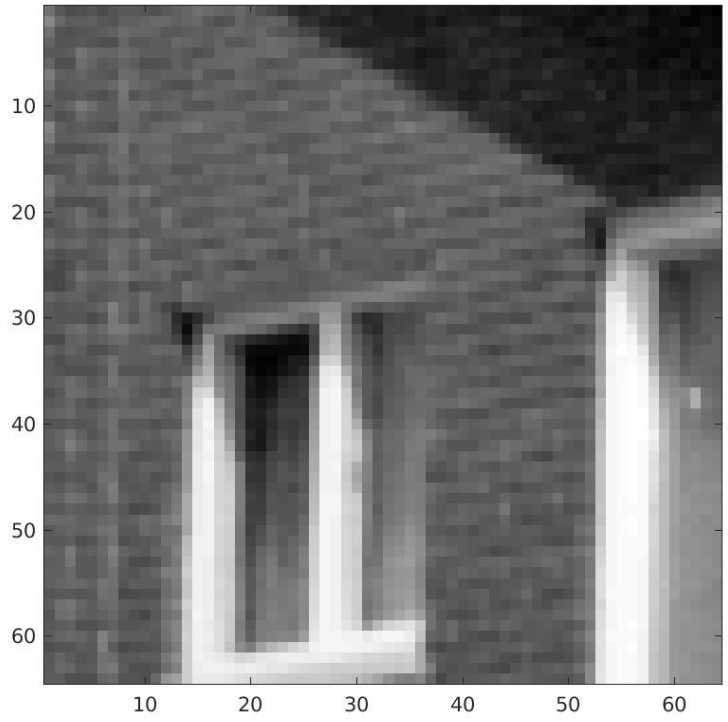
Χρόνος CPU->GPU: 0.0011095 sec

Χρόνος επεξεργασίας GPU: 0.035954 sec

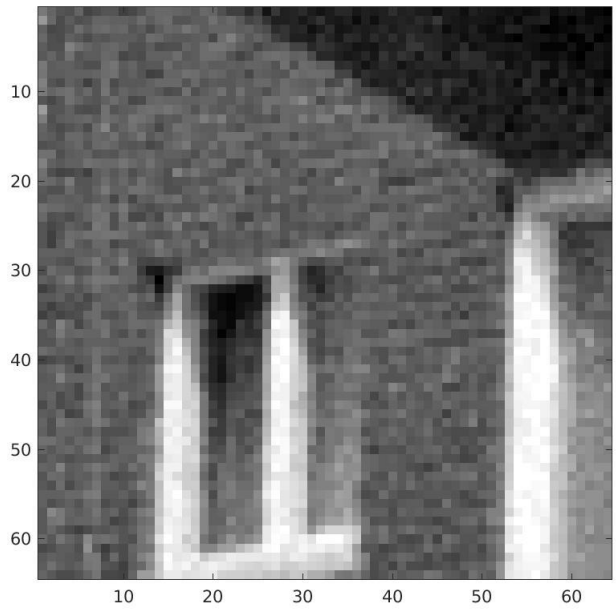
Χρόνος GPU->CPU: 0.000454 sec

- 64x64, 5x5

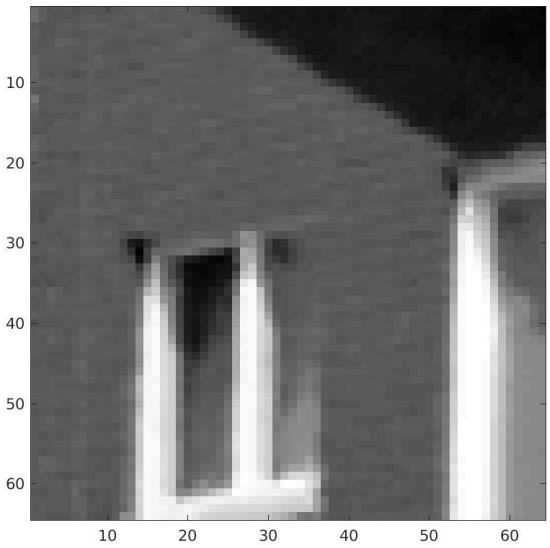
Original image



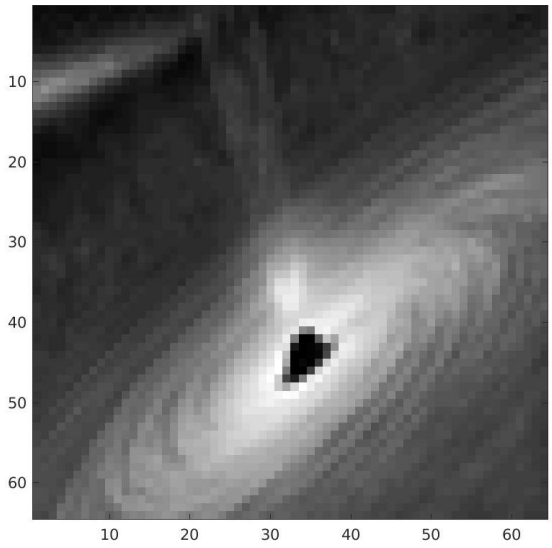
Original image with noise



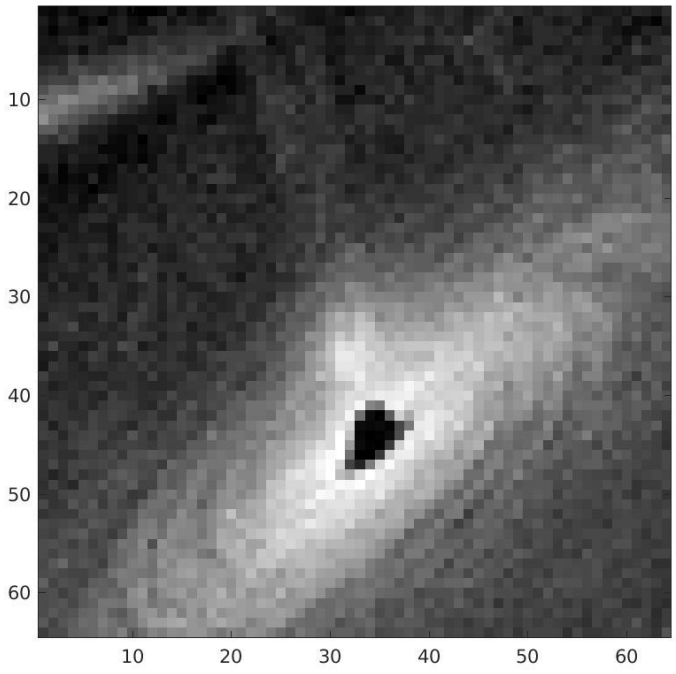
Filtered image



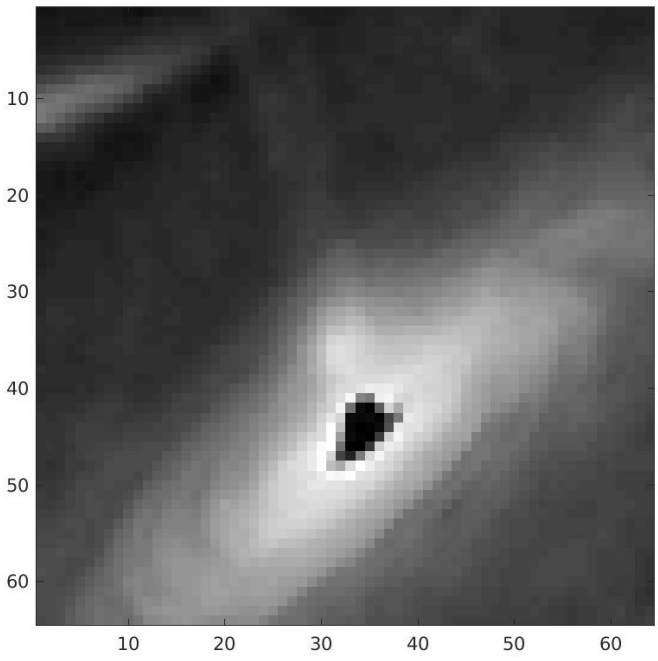
Original image:



Original image with noise:



Filtered image:



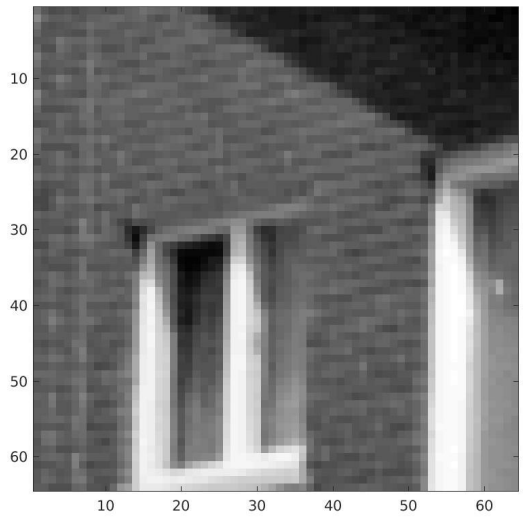
Χρόνος CPU->GPU: 0.0010105 sec

Χρόνος επεξεργασίας GPU: 0.087853 sec

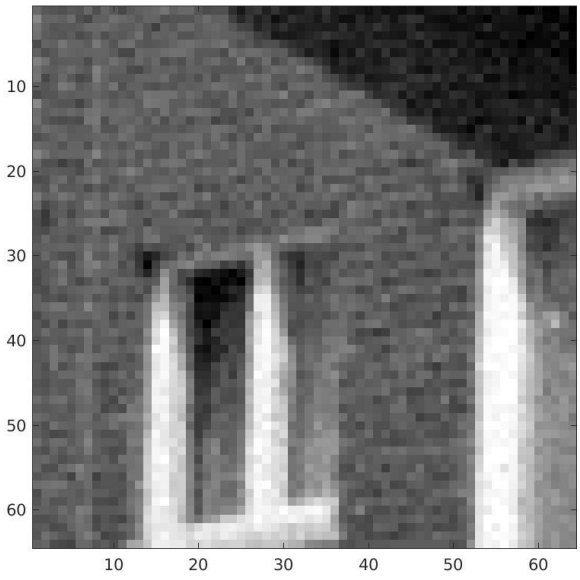
Χρόνος GPU->CPU: 0.000396 sec

- 64x64, 7x7

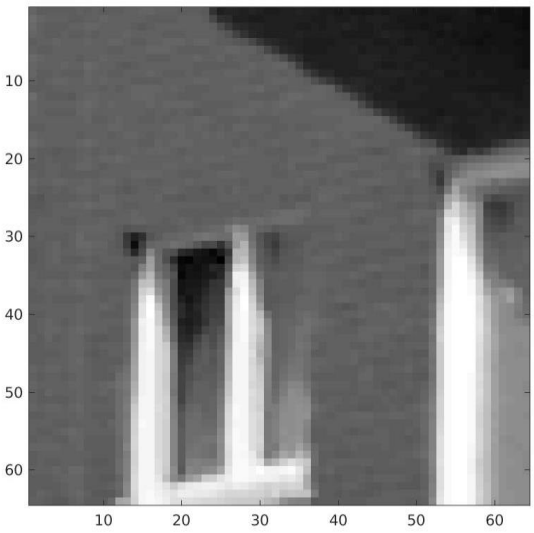
Original image



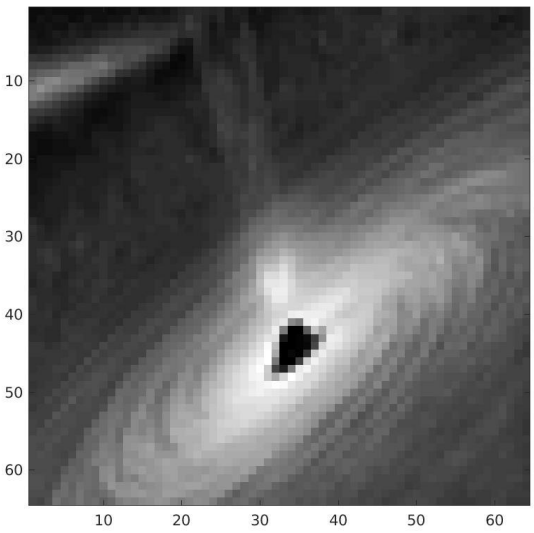
Original image with noise



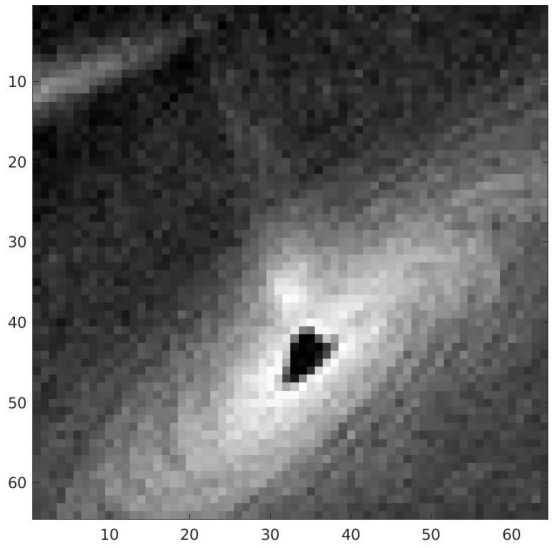
Filtered image



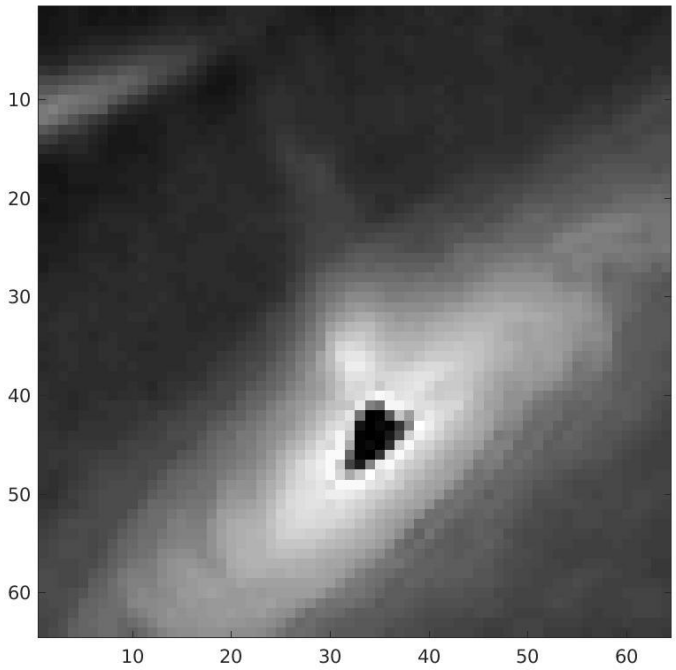
Original image:



Original image with noise:



Filtered image:



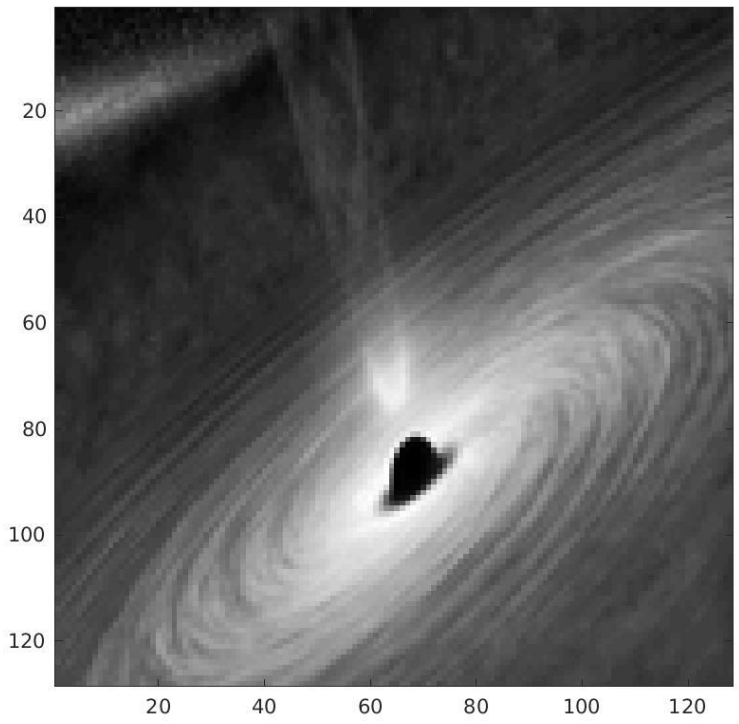
Χρόνος CPU->GPU: 0.001014 sec

Χρόνος επεξεργασίας GPU: 0.179596 sec

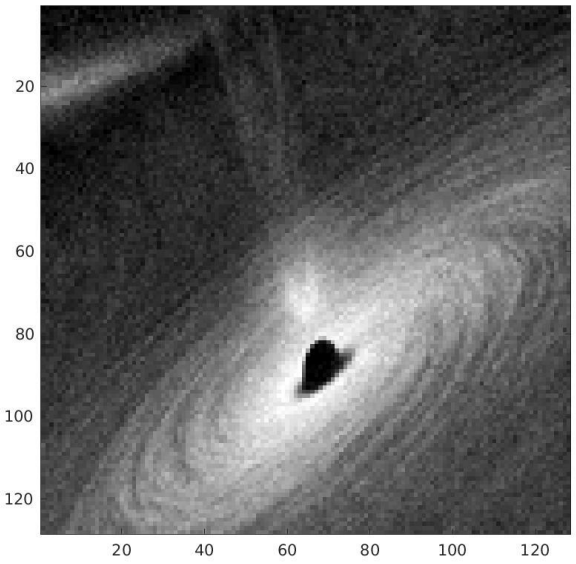
Χρόνος GPU->CPU: 0.000422 sec

- 128x128, 3x3

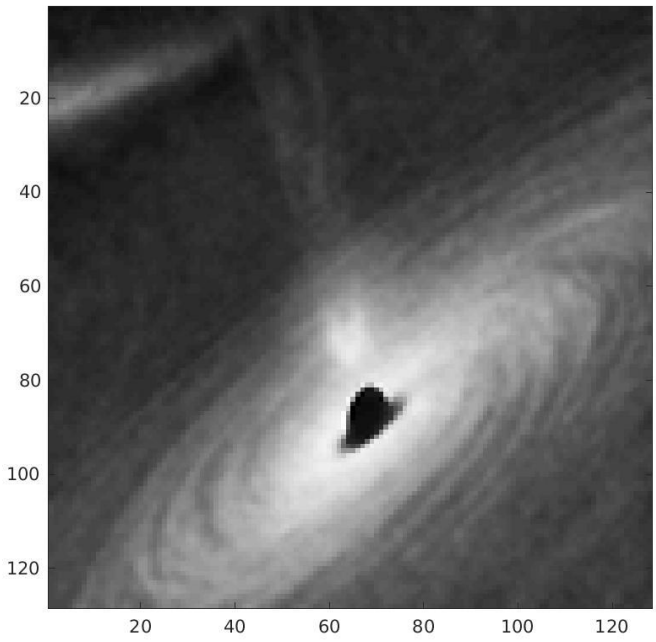
Original image



Original image with noise



Filtered image



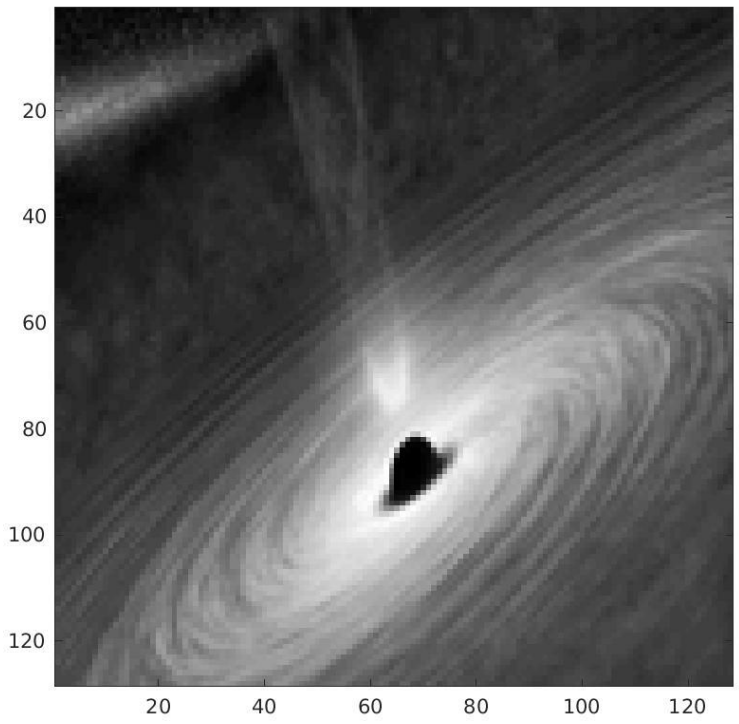
Χρόνος CPU->GPU: 0.002678 sec

Χρόνος επεξεργασίας GPU: 0.482074 sec

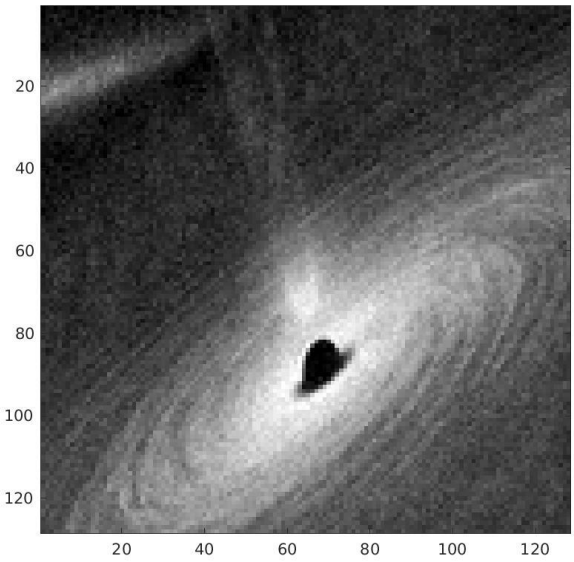
Χρόνος GPU->CPU: 0.000719

- 128x128, 5x5

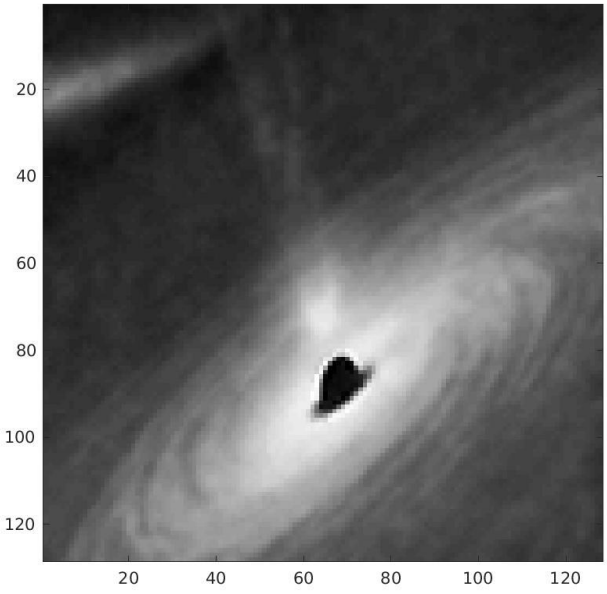
Original image



Original image with noise



Filtered image



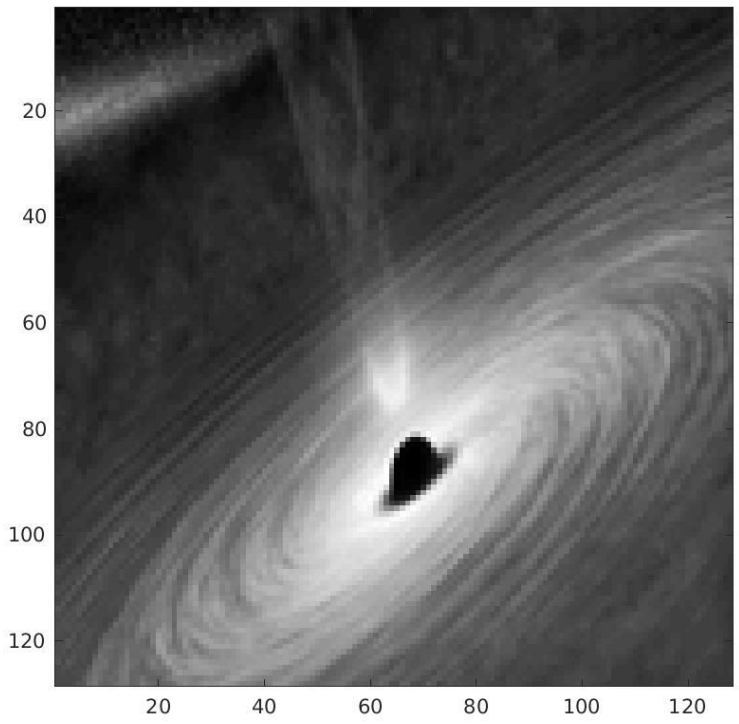
Χρόνος CPU->GPU: 0.002793 sec

Χρόνος επεξεργασίας GPU: 1.353607 sec

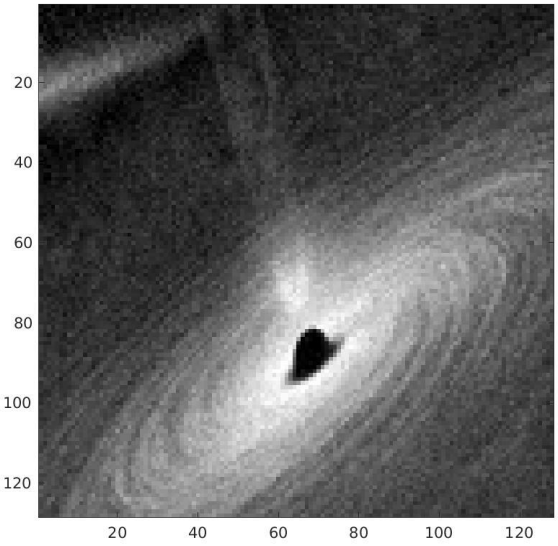
Χρόνος GPU->CPU: 0.000720 sec

- 128x128, 7x7

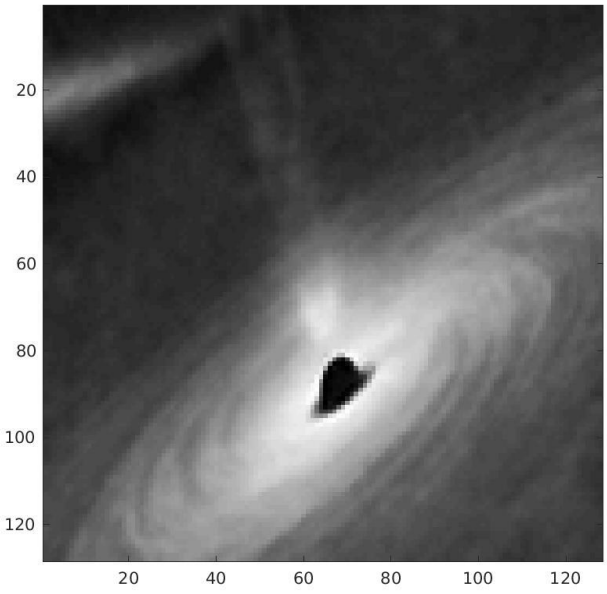
Original image



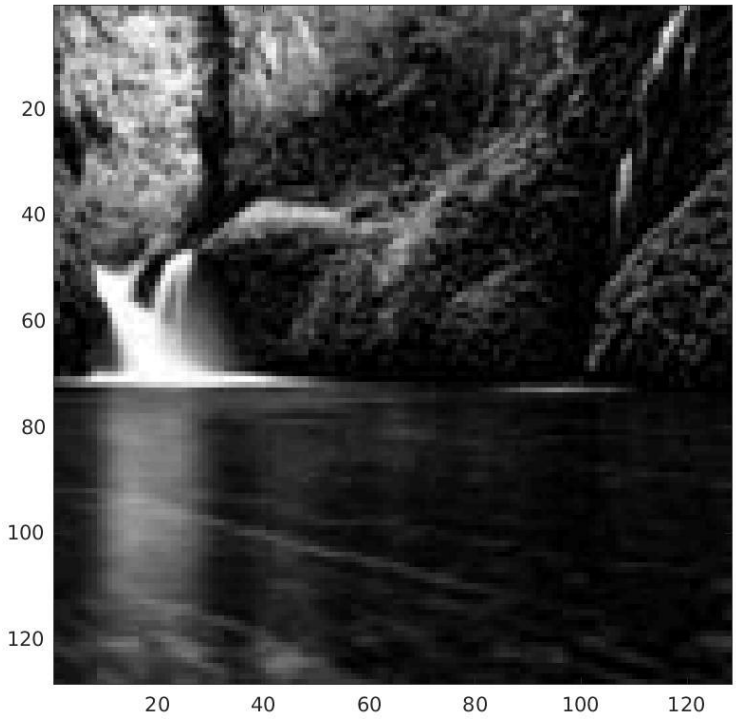
Original image with noise



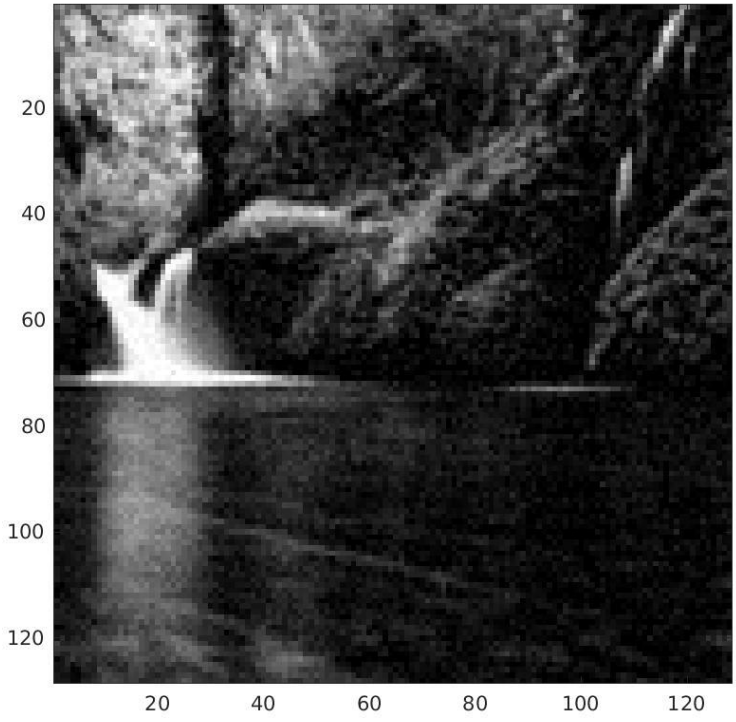
Filtered image



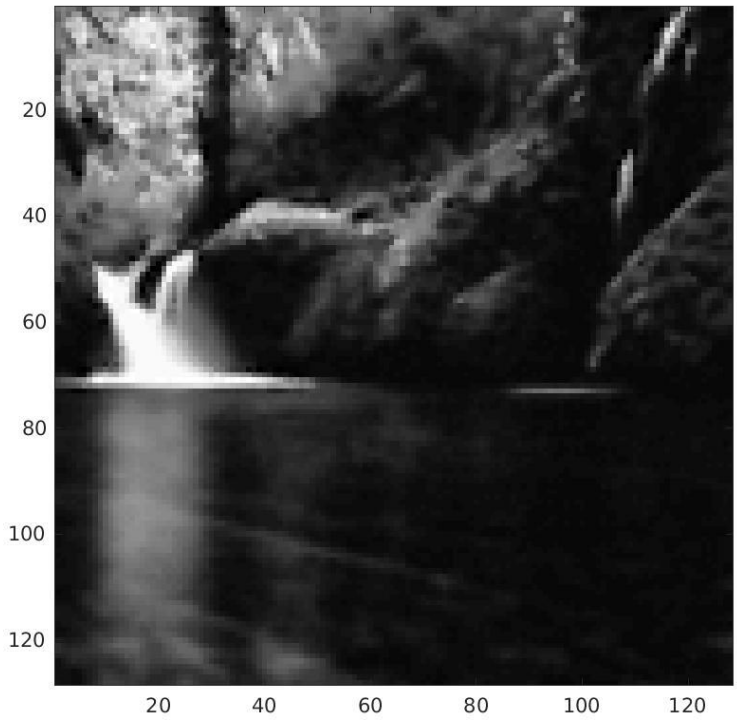
Original image:



Original image with noise:



Filtered image:



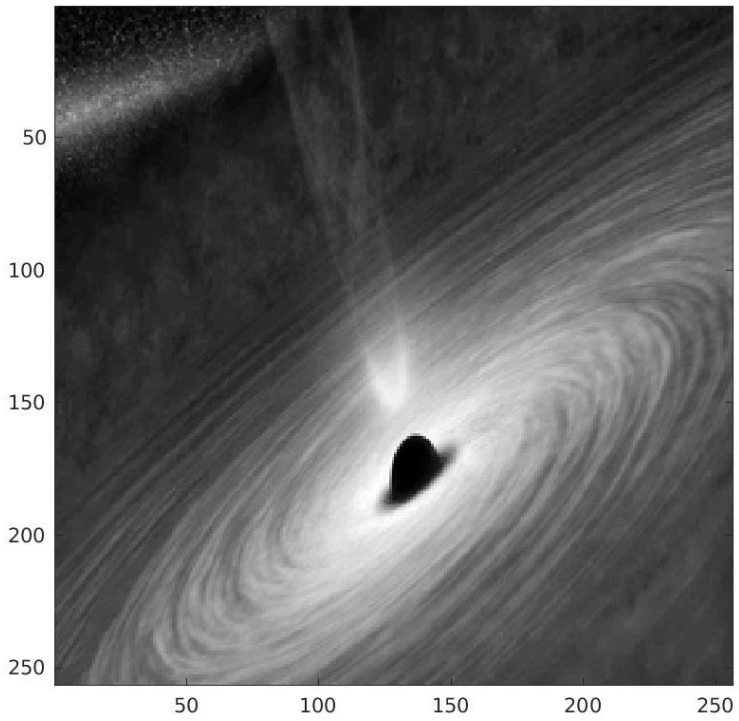
Χρόνος CPU->GPU: 0.002711 sec

Χρόνος επεξεργασίας GPU: 2.281070 sec

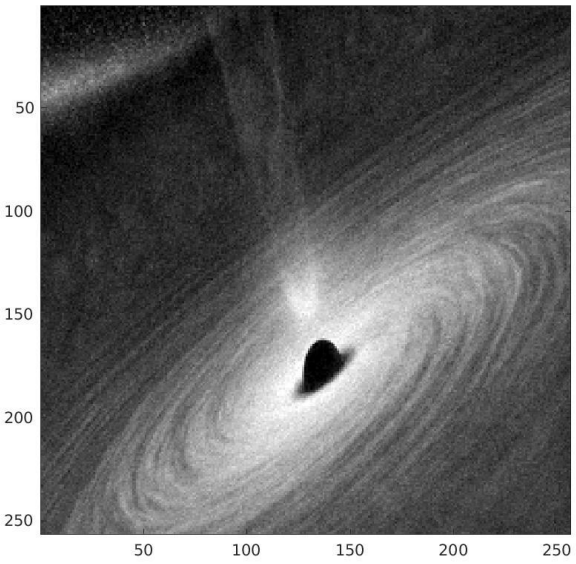
Χρόνος GPU->CPU: 0.000782 sec

- 256x256, 3x3

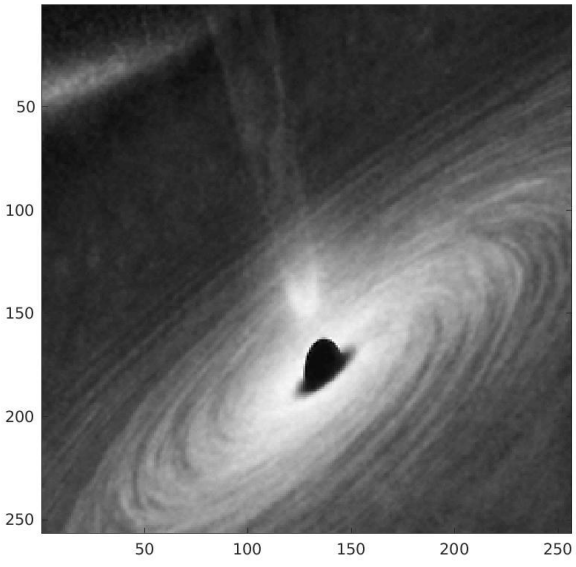
Original image



Original image with noise



Filtered image



`

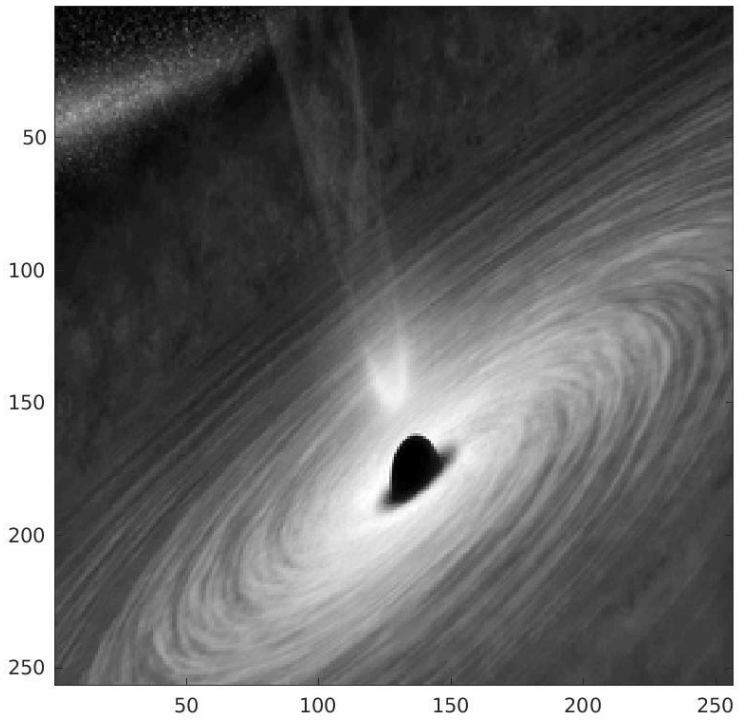
Χρόνος CPU->GPU: 0.003058 sec

Χρόνος επεξεργασίας GPU: 8.730917 sec

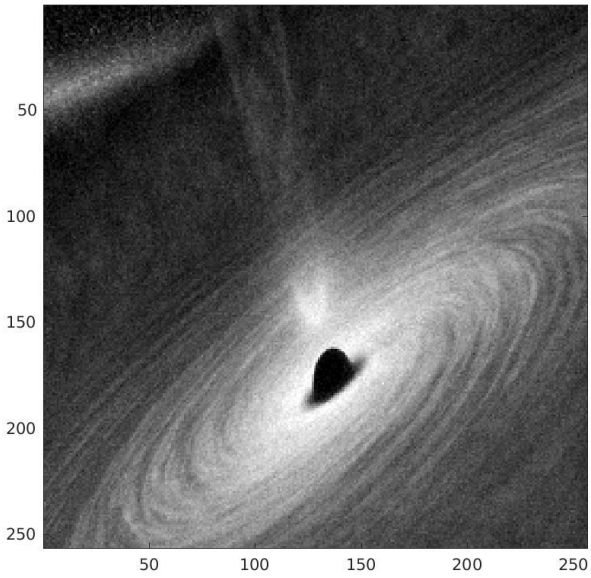
Χρόνος GPU->CPU: 0.001252 sec

- 256x256, 5x5

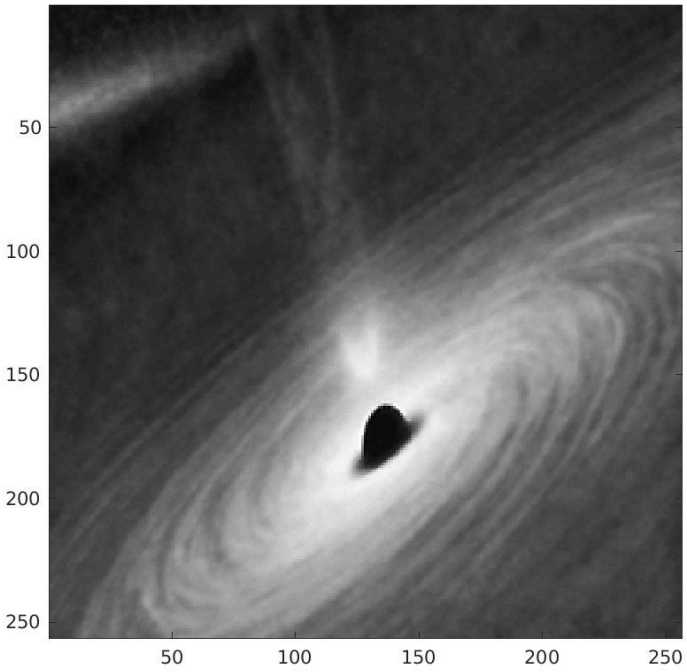
Original image



Original image with noise



Filtered image



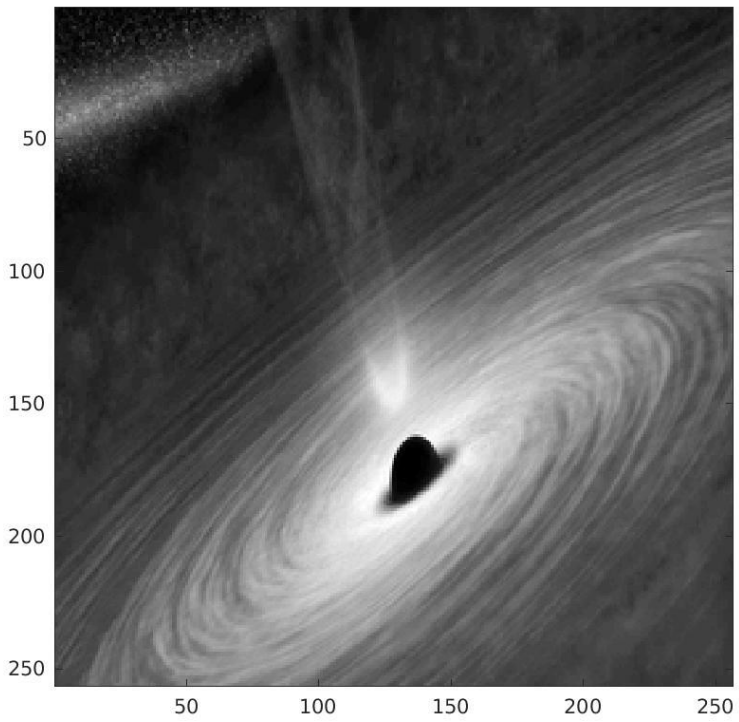
Χρόνος CPU->GPU: 0.002865 sec

Χρόνος επεξεργασίας GPU: 24.712677 sec

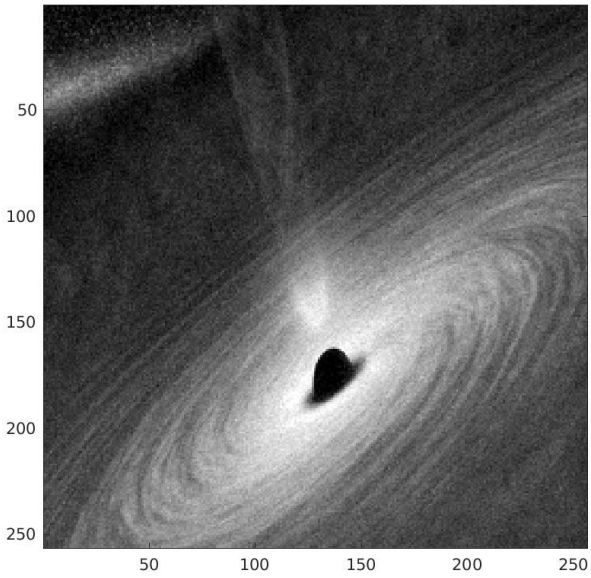
Χρόνος GPU->CPU: 0.001262 sec

- 256x256, 7x7

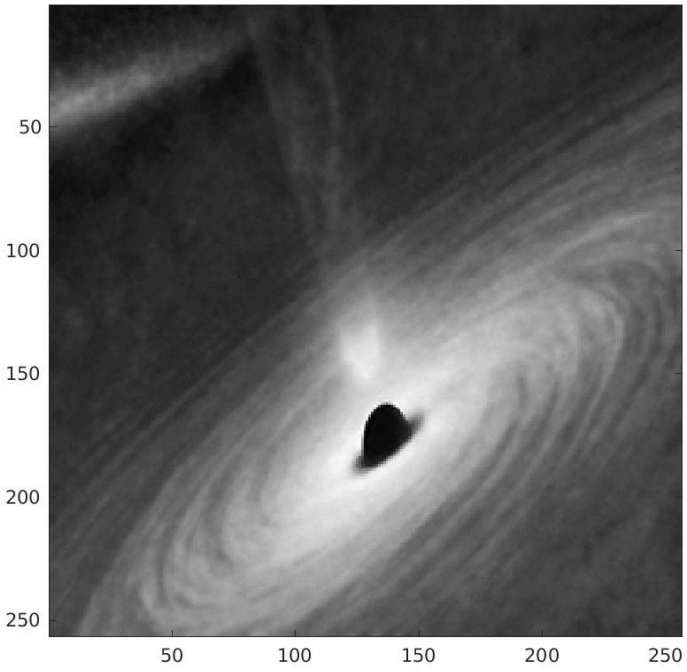
Original image



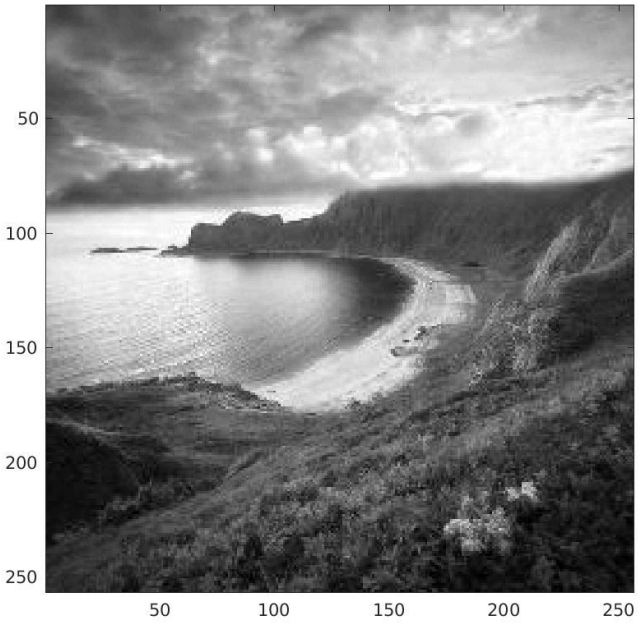
Original image with noise



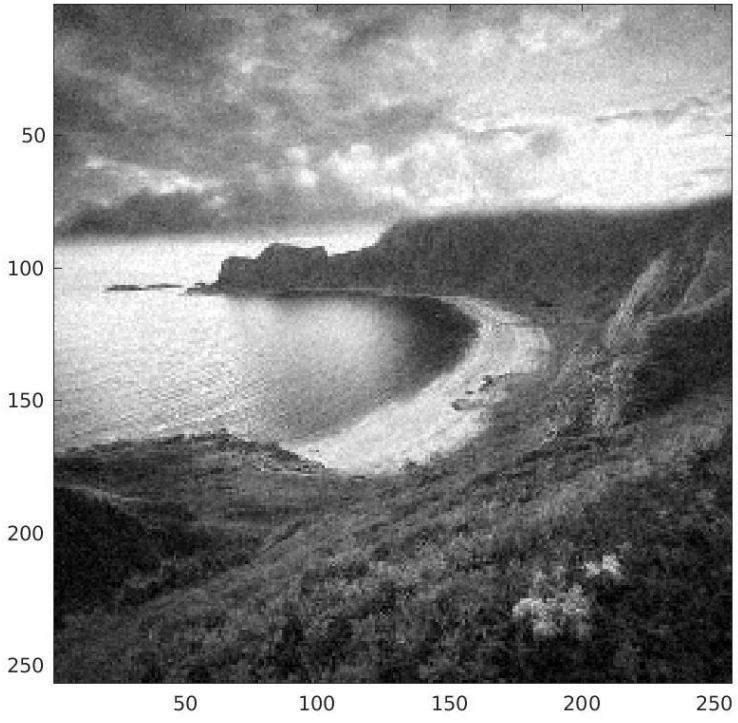
Filtered image



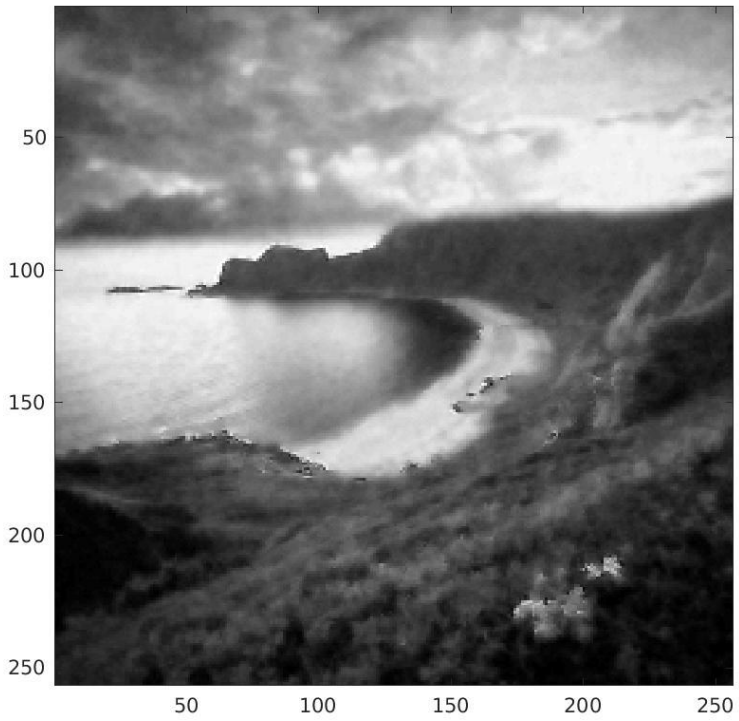
Original image:



Original image with noise:



Filtered image:



Χρόνος CPU->GPU: 0.002852 sec

Χρόνος επεξεργασίας GPU: 44.219775 sec

Χρόνος GPU->CPU: 0.001284 sec

Συγκριτικά με την υλοποίηση σε Matlab όπως είχε δοθεί αρχικά **χωρίς CUDA**, έχουμε (όλα σε sec):

	3x3	**5x5**	**7x7**
64x64	0.872735	0.896971	0.960944
128x128	12.697508	13.015113	14.091280
256x256	- (δεν γινόταν στον Diades)	-- (δεν γινόταν στον Diades	- (δεν γινόταν στον Diades

ενώ συνοπτικά **με CUDA,** *ο συνολικός χρόνος* (άθροιση και των 3 επιμέρους, οι ελάχιστοι χρόνοι από τις διάφορες εικόνες κάθε φορά, που επιτεύχθηκαν) (όλα σε sec):

	3x3	**5x5**	**7x7**
64x64	0.036347	0.089033	0.180531
128x128	0.482793	1.35712	2.284533
256x256	8.735227	24.716804	44.223911

από τα οποία κια συμπεραίνουμε τελικά τις εξής επιταχύνσεις:

	3x3	**5x5**	**7x7**
64x64	24.011	10.075	5.323
128x128	26.3	9.6	6.17
256x256	-	-	-

ΣΥΜΠΕΡΑΣΜΑΤΑ

Παρατηρούμε από τις εικόνες των αποτελεσμάτων πως όντως η τελική εικόνα είναι αποθορυβοποιημένη και μοιάζει αρκετά στην αρχική, οπότε η υλοποίηση είναι αρκετά σωστή ως προς την υλοποίηση.

Βέβαια, υπάρχει ένα μικρό φαινόμενο θολώματος, το οποίο πιθανόν να οφείλεται στην συγκεκριμένη εικόνα που χρησιμοποιήθηκε, καθώς προέκυψε από διαδικασία resize και επεξεργασία/παραμόρφωση μιας διαφορετικών διαστάσεων.

Το θόλωμα αυτό μάλιστα δεν εμφανίζεται όπως βλέπουμε στην εικόνα με το house την πρώτη, και μάλιστα η filtered εκεί είναι καλύτερη και ευκρινέστερη της αρχικής, όπως το ίδιο συμβαίνει και με την διαφορετική φωτογραφία στο τέλος με την φύση.

Δηλαδή, συνολικά το αποτέλεσμα είναι αρκετά ικανοποιητικό και οπτικά, σαν εικόνα μετά την αποθορυβοποίηση με τπ φίλτρο, αλλά και σαν ταχύτητα υλοποίησης, αφού οι επιταχύνσεις είναι καλές και κυρίως οι χρόνοι είναι αρκετά μικροί. Αρκεί να αναφέρω για παράδειγμα ότι χωρίς CUDA χρειαζόταν ακόμα και λεπτό για να ολοκληρωθεί, ενώ με CUDA το κάνει πολύ πιο γρήγορα. Δυστυχώς, λόγω περιορισμών στο Matlab στον Diades δεν είναι εφικτό να δούμε στην περίπτωση 256x256 και ειδικά για 7x7 τι χρόνο θα έκανε χωρίς CUDA. Εικάζω ότι η επιτάχυνση εκεί θα ήταν σημαντικότατη και ο χρόνος με CUDA πολύ μικρός σε σχέση με αυτόν χωρίς CUDA.

Τέλος, παρατηρούμε ότι σε μεγαλύτερης ανάλυσης εικόνες έχουμε αρκετά καλύτερα και οπτικά αποτελέσματα, αισθητικά, στην ευκρίνεια, πράγμα αναμενόμενο αφού όσο μικρότερη η ανάλυση τόσες λιγότερες λεπτομέρειες αποτυπώνονται εύκολα, ενώ ο θόρυβος είναι περισσότερος.