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.

Σημείωση:

Λόγω επικείμενης εξεταστικής, φόρτου εργασίας μαθημάτων και προσωπικών λόγων, ο διαθέσιμος χρόνος που είχα προσωπικά εγώ για την πραγματοποίηση της εργασίας αυτής ήταν πολύ λίγος και γι'αυτό η υλοποίηση αυτή που έκανα αποτελεί μια πρώτη έκδοση, είναι απλή, λειτουργική, αλλά και συνάμα αργή και όχι τόσο αποτελεσματική σε σχέση με αυτό που θα ήθελα να κάνω και έχω σκεφτεί πώς να κάνω ώστε να είναι βελτιστοποιημένο, όμως εξ'αιτίας του περιορισμού στον χρόνο δεν κατέστη αυτό εφικτό δυστυχώς. Λόγω προθεσμίας αναγκαστικά αυτή η έκδοση είναι η μόνη που έχω προλάβει να κάνω μέχρι στιγμής.

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 ένα μόνο block με πλήθος thread έστω [16 16], και άρα 64/16=4 pixels θα επεξεργάζεται και θα είναι υπεύθυνο το κάθε thread. Η μνήμη που χρησιμοποιείται είναι global επειδή δεν προλάβαινα να το λειτουργήσω με shared, αν και θα ήταν σαφώς πιο γρήγορο.

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.

Αρχικά ορίζονται κάποιες σταθερές, καθώς και τα array access macros με define, για διευκόλυνση στον κώδικα αργότερα κατά την αναφορά σε πίνακες, διότι αυτοί έχουν οριστεί σαν μονοδιάστατοι και όχι ως δισδιάστατοι.

Η __global__ void Babis_Kernel_Jan_2017(float const* const A, float *B, float const* const Gauss, float *Z, float *Products) 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, τους πίνακες B, Z, Products που είναι οι μηδενικοί nxm και την κατανομή Gauss G που θα δράσει σαν φίλτρο για την αποθορυβοποίηση της εικόνας.

Τα i,j είναι οι συντεταγμένες των thread, δηλαδή τα threadIdx.x, threadIdx.y.

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

Οι δείκτες a,b δείχνουν στο κάθε ένα από τα πίξελ που αφορά το κάθε νήμα, άρα παίρνουν τιμή πχ για 64χ64 εικόνα και 16x16 νήματα -> 4 πίξελ το νήμα, οπότε από 0 έως 3.

Οι δείκτες c,d δείχνουν στο κάθε πίξελ της 64x64 εικόνας και συνεπώς σαρώνουν όλον τον πίνακα/εικόνα.

Η λογική είναι ότι για κάθε νήμα, για κάθε i,j πίξελ θα βρω την απόστασή του με κάθε άλλο c,d πίξελ της εικόνας και για να γίνει αυτό θα πρέπει να σαρώνω μια περιοχή e,f γύρω από κάθε ένα από αυτά τα πίξελ, τα οποία e,f παίρνουν τιμές -1,0,1 για 3x3 περιοχή, -2,-1,0,1,2 για 5x5 κλπ, ενώ τα c,d έχουν τιμές μέχρι nxm όπως είναι επόμενο.

Στη συνέχεια, για κάθε πίξελ που εξετάζουμε και για αυτό το πίξελ, σε σχέση με κάθε ένα πίξελ από τα nxm , βρίσκουμε από ποιά πίξελ ακριβώς θα βρούμε την απόσταση. Έτσι καθορίζουμε ποιά είναι επακριβώς τα στοιχεία του πίνακα A. Επίσης προνοούμε και για περιπτώσεις mirror padding και ορίζουμε αντίστοιχα τα "κέντρα"/πίξελ, σε περίπτωση δηλαδή που βγούμε εκτός ορίων πίνακα.

Έπειτα, αφού έχουμε βρει ποια δύο πίξελ θα επεξεργαζόμαστε κάθε φορά, 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 expf((-1/s)*norm) με τη νόρμα μέσα, σε μια μεταβλητή w και προσαυξάνουμε τον πίνακα Z κατά w, στη θέση i*m_pixels_thread+a,j*n_pixels_thread+b. Αυτή η θέση ορίζεται ως εξής:

Το κάθε νήμα καθορίζεται από τα i,j άρα η θέση αυτή λαμβάνει υπ'όψιν της τα νήματα οπότε αποκλείονται race conditions. Ο πολλαπλασιασμός με n_pixels_thread ή m_pixels_thread γίνεται για να προσπεράσουμε την ομάδα των πίξελ που έχει αναλάβει το κάθε νήμα και να πάμε παρακάτω κατά τόσες θέσεις, ώστε να γράψουμε σε θέσεις του Z κατάλληλες που αντιστοιχούν στο εκάστοτε νήμα. Οι δείκτες a,b δείχνουν αυτό ακριβώς το περιθώριο επιπλέον μετακίνησης αφού φτάσουμε στην κατάλληλη αυτή περιοχή και ορίζουν πχ για 4 πίξελ/νήμα, μια τετράγωνη περιοχή 16 θέσεων, τις οποίες έχει αναλάβει αποκλειστικά ένα νήμα, το αντίστοιχο. Ουσιαστικά ο Z είναι ο πίνακας με τα συνολικά αθροίσματα για τα βάρη, μέχρι το αντίστοιχο πίξελ κάθε φορά. Κατόπιν, με παρόμοιο

τρόπο προσαυξάνουμε στην ίδια θέση με πριν, τον Products κατά w*A(c,d), σύμφωνα με τον αλγόριθμο της εκφώνησης ώστε να προκύψει το τελικό αποτέλεσμα η f^.

Τέλος, μόλις υπολογιστούν όλα αυτά, το τελικό αποτέλεσμα θα είναι αν διαιρέσουμε στοιχείο προς στοιχείο τον Products με τον Z και αποθηκεύουμε το αποτέλεσμα στον πίνακα B.

Ουσιαστικά, περάσαμε σαν επιπλέον ορίσματα στην συνάρτηση αυτή, δύο μηδενικούς πίνακες τους Z και Products για να χρησιμοποιηθούν ως πίνακες που θα αποθηκεύουμε τα ενδιάμεσα αποτελέσματα των υπολογισμών μας και ώστε να είναι global και αυτό επειδή δεν χρησιμοποιούμε shared μνήμη και λόγω threads εάν δηλώνονταν μέσα στην συνάρτηση θα υπήρχε πρόβλημα. Ο B περάστηκε σαν όρισμα σαν 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 (εδώ έχουμε όπως και είπαμε μόνο ένα block).

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, τους 3 μηδενικούς, το φίλτρο Gauss και την εικόνα με θόρυβο, τους μετατρέπουμε σε single και 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.

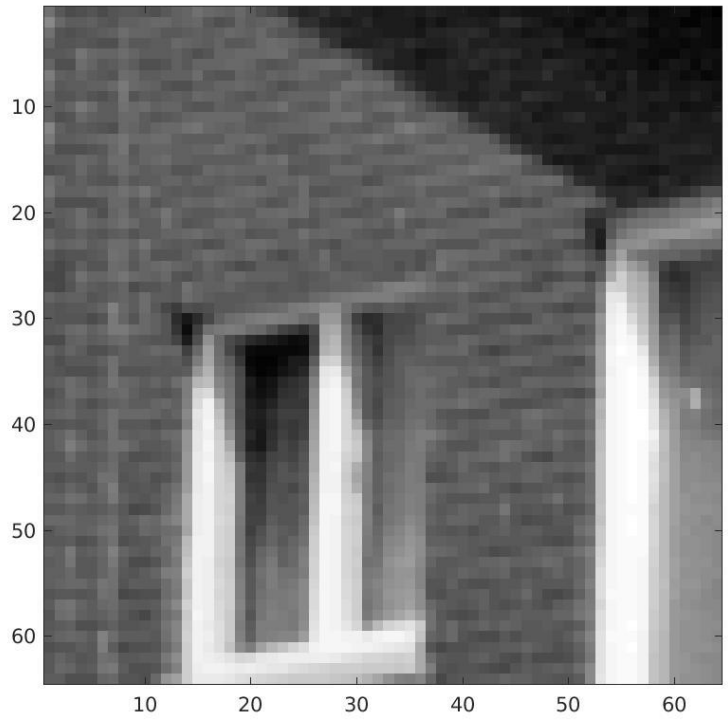
Τέλος, εμφανίζουμε τις τελικές εικόνες και το υπόλοιπο, καθώς και τους χρόνους αυτούς.

ΑΠΟΤΕΛΕΣΜΑΤΑ

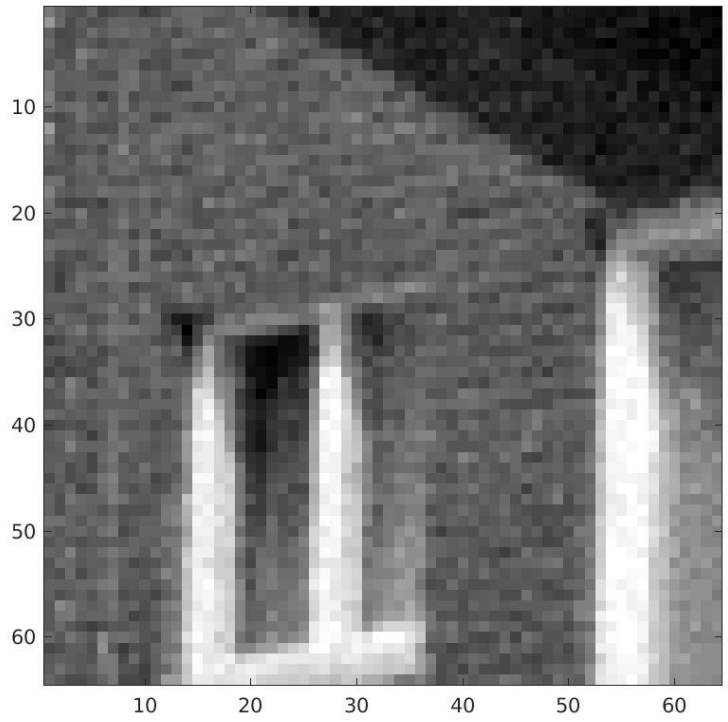
Παραθέτουμε τα αποτελέσματα από τον Diades για πλήθος threads σε όλες τις περιπτώσεις ίσο με 16.

- 64x64, 3x3

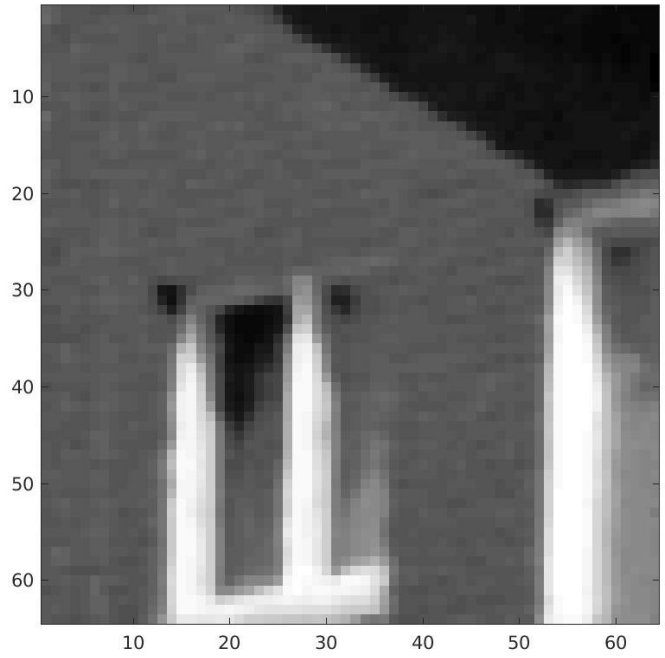
Original image



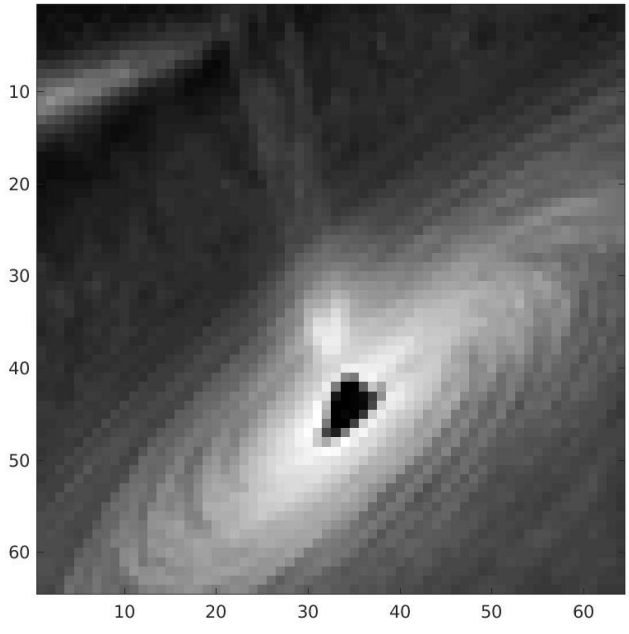
Original image with noise


10
20
30
40
50
60
10
20
30
40
50
60

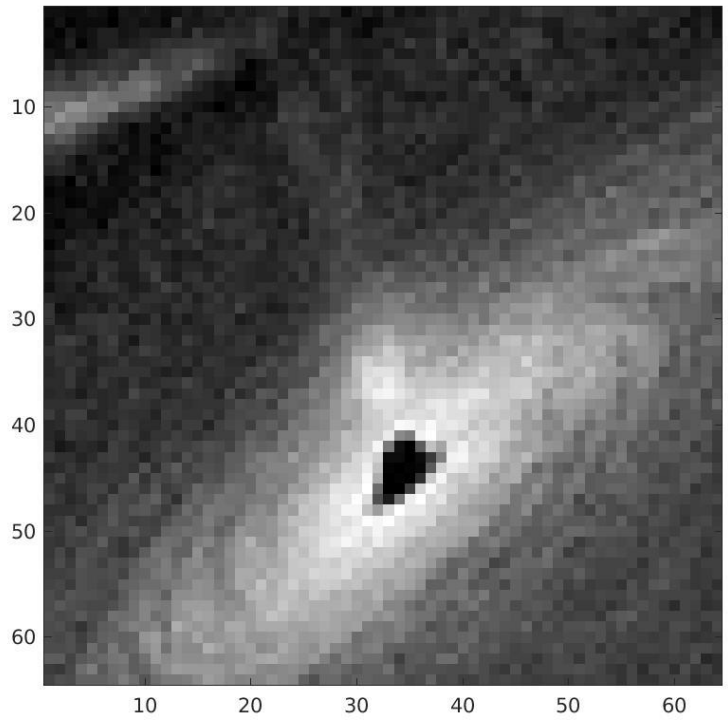
Filtered image



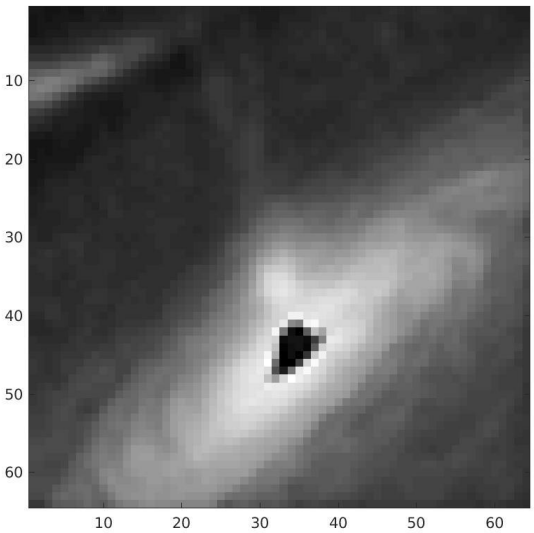
Original image



Original image with noise



Filtered image



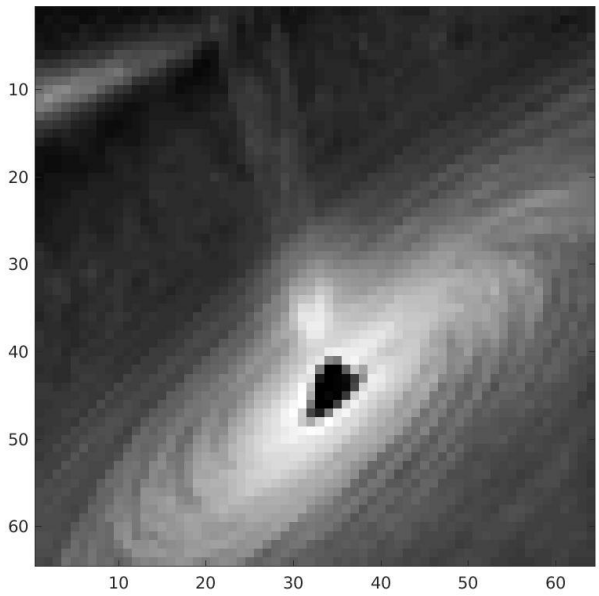
Χρόνος CPU->GPU: 0.002085 sec

Χρόνος επεξεργασίας GPU: 0.451957 sec

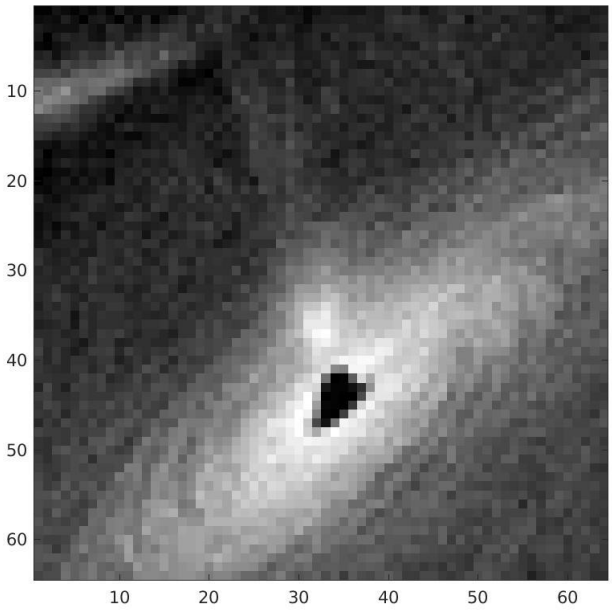
Χρόνος GPU->CPU: 0,000619

- 64x64, 5x5

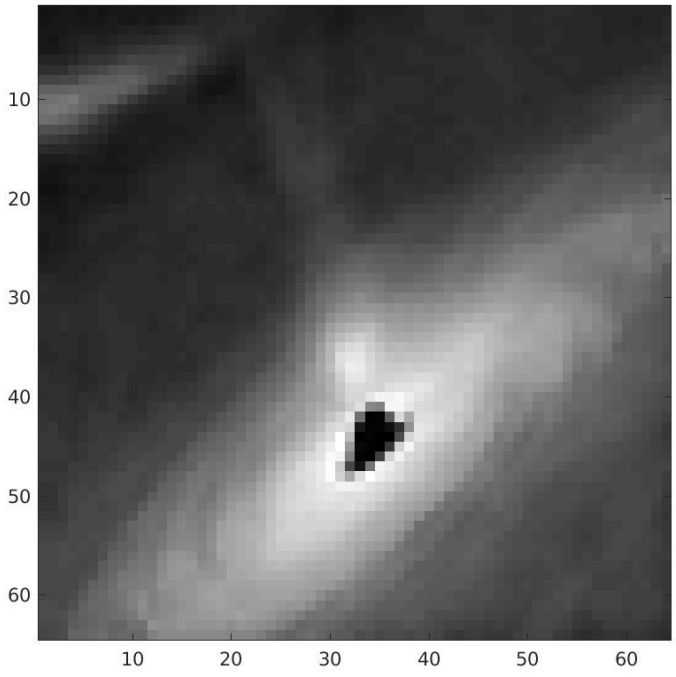
Original image



Original image with noise



Filtered image



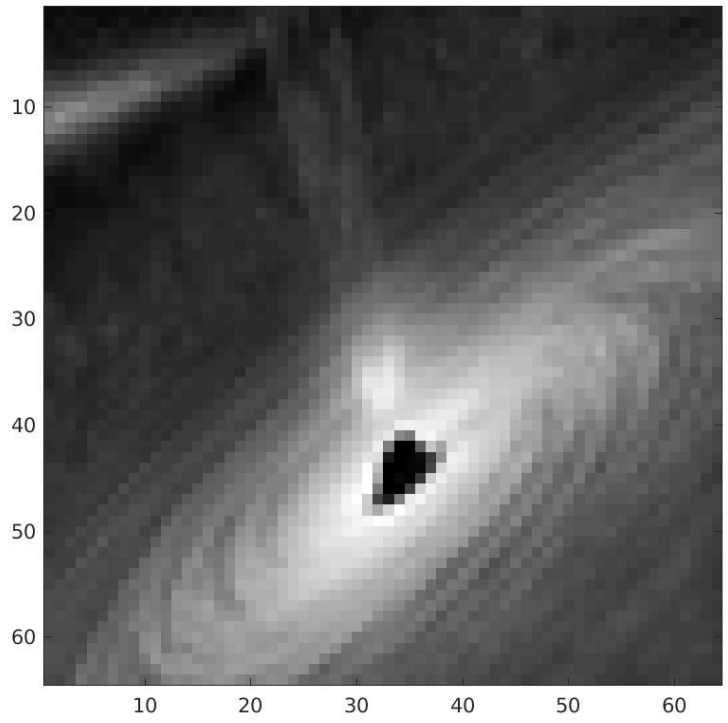
Χρόνος CPU->GPU: 0.002072 sec

Χρόνος επεξεργασίας GPU: 1.126360 sec

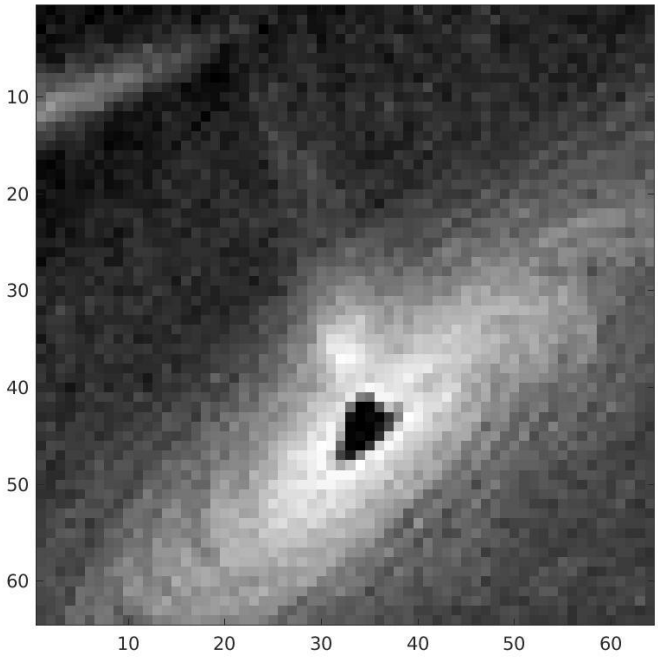
Χρόνος GPU->CPU: 0.000667

- 64x64, 7x7

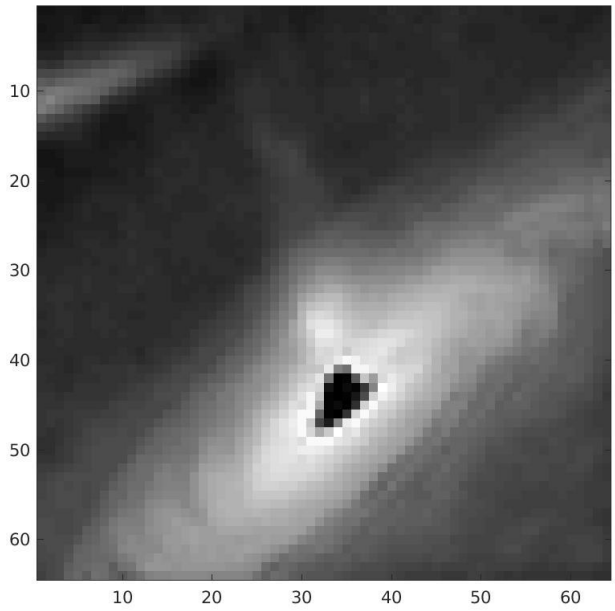
Original image



Original image with noise



Filtered image



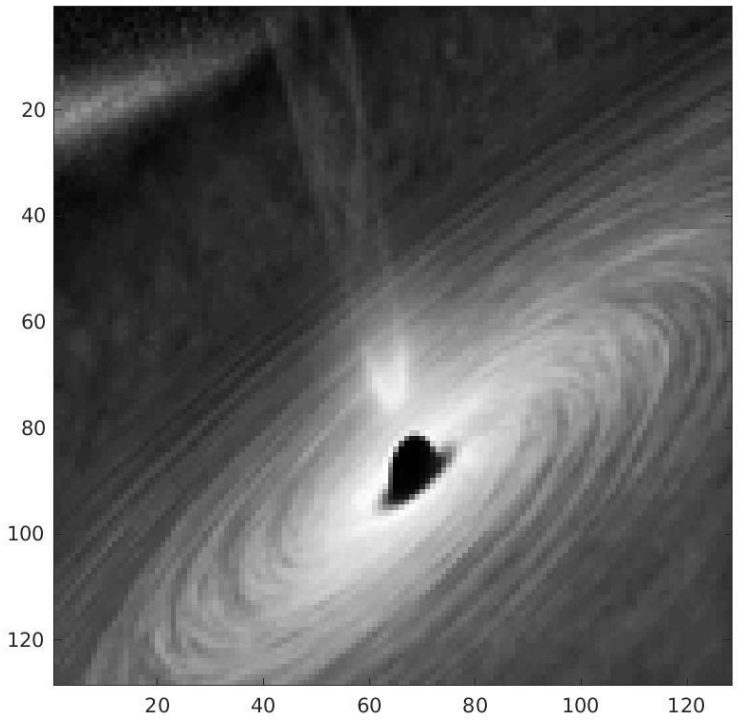
Χρόνος CPU->GPU: 0.002113 sec

Χρόνος επεξεργασίας GPU: 2.186092 sec

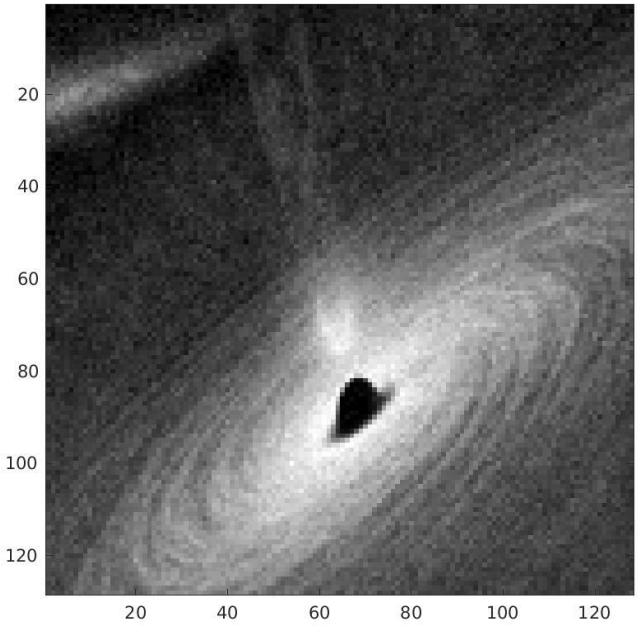
Χρόνος GPU->CPU: 0.000708

- 128x128, 3x3

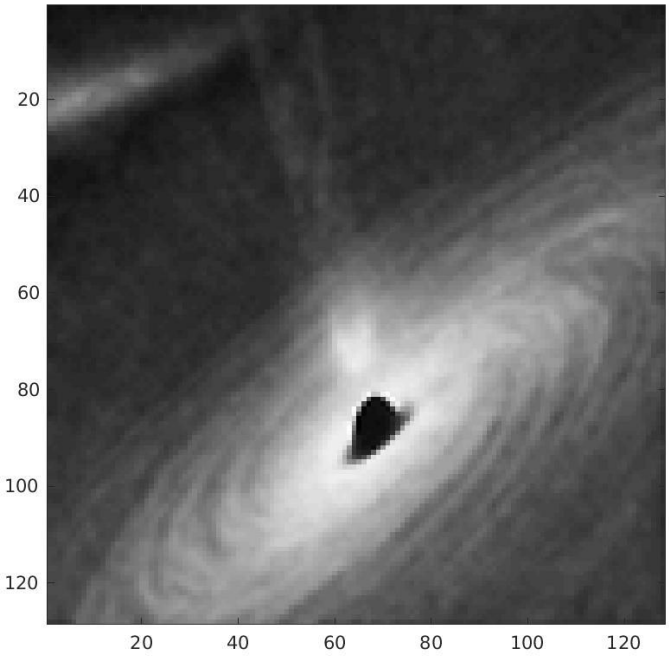
Original image



Original image with noise



Filtered image



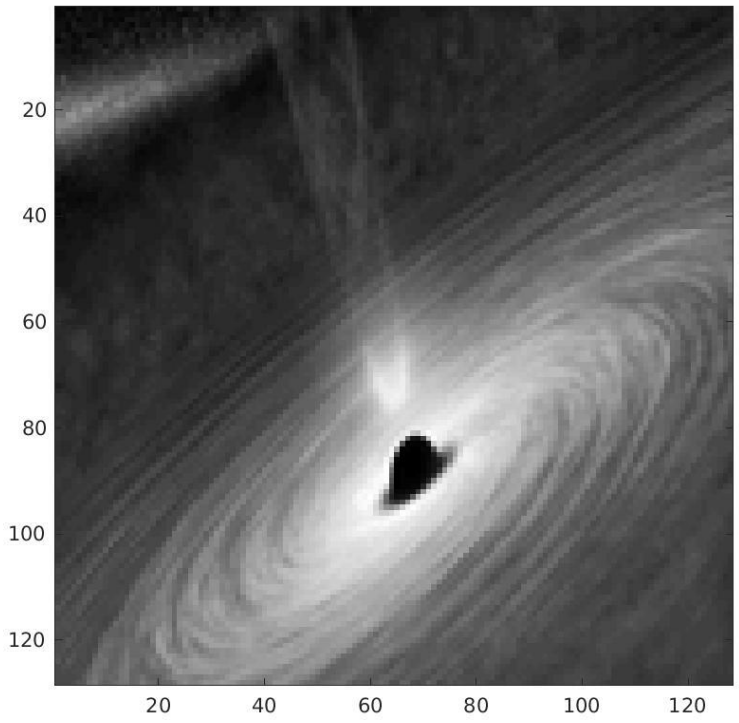
Χρόνος CPU->GPU: 0.002165 sec

Χρόνος επεξεργασίας GPU: 9.639287 sec

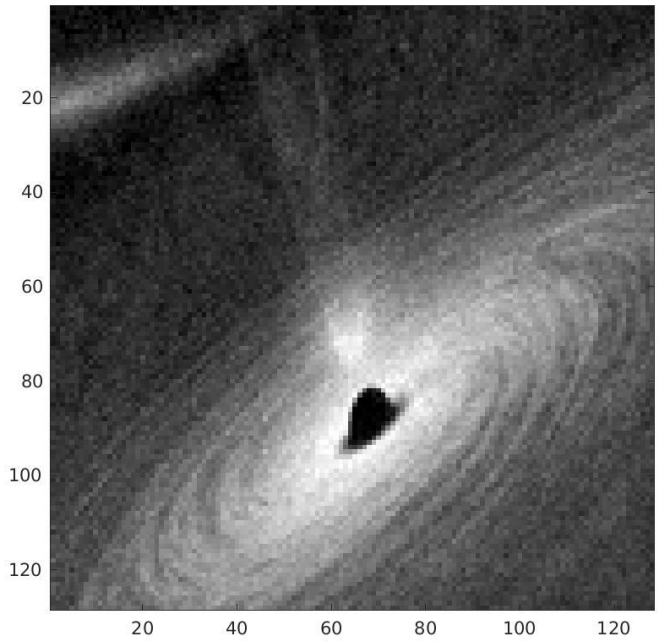
Χρόνος GPU->CPU: 0.000887

- 128x128, 5x5

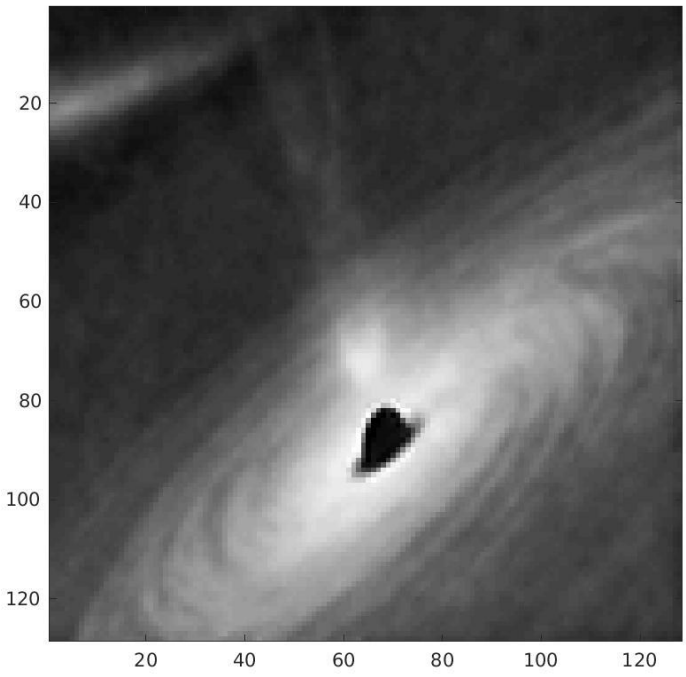
Original image



Original image with noise



Filtered image



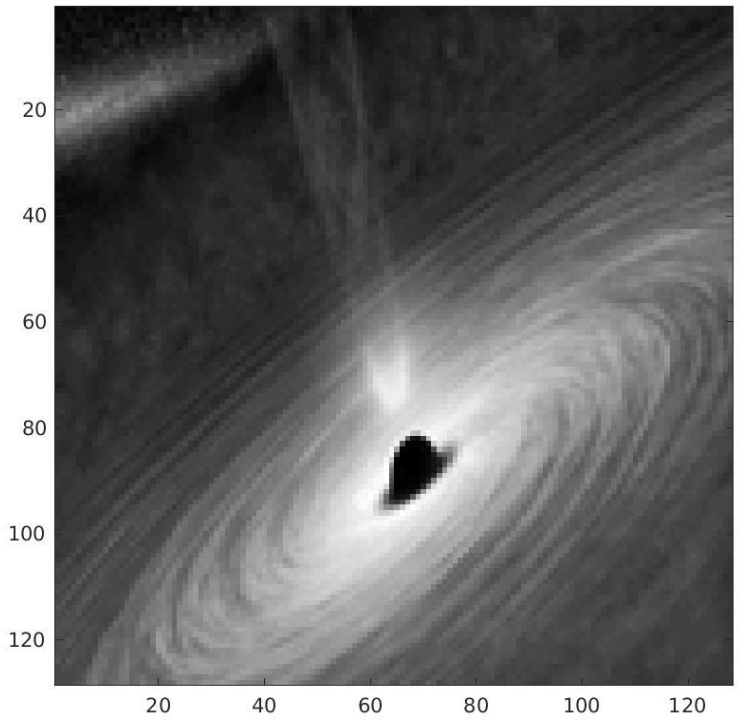
Χρόνος CPU->GPU: 0.002183 sec

Χρόνος επεξεργασίας GPU: 24.4152 sec

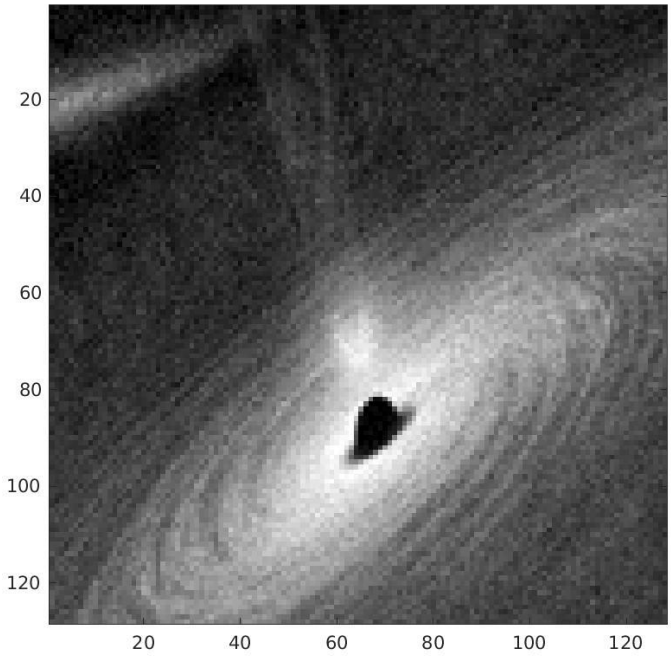
Χρόνος GPU->CPU: 0.000923

- 128x128, 7x7

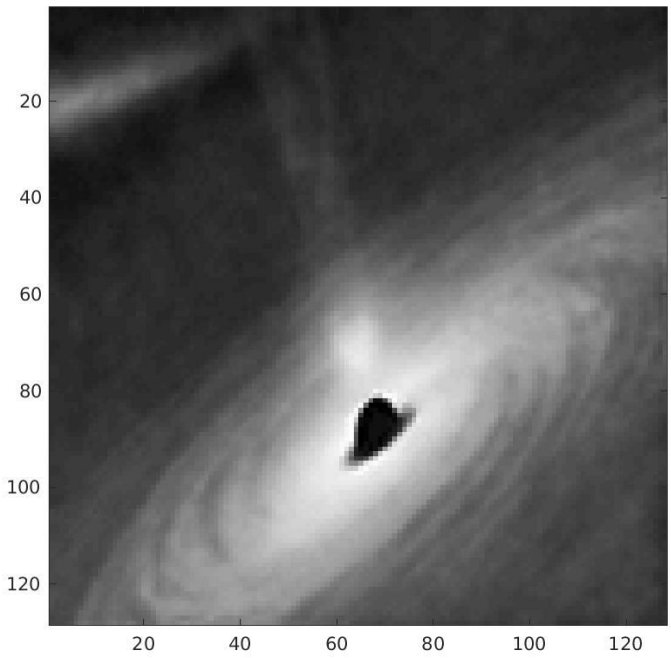
Original image



Original image with noise



Filtered image



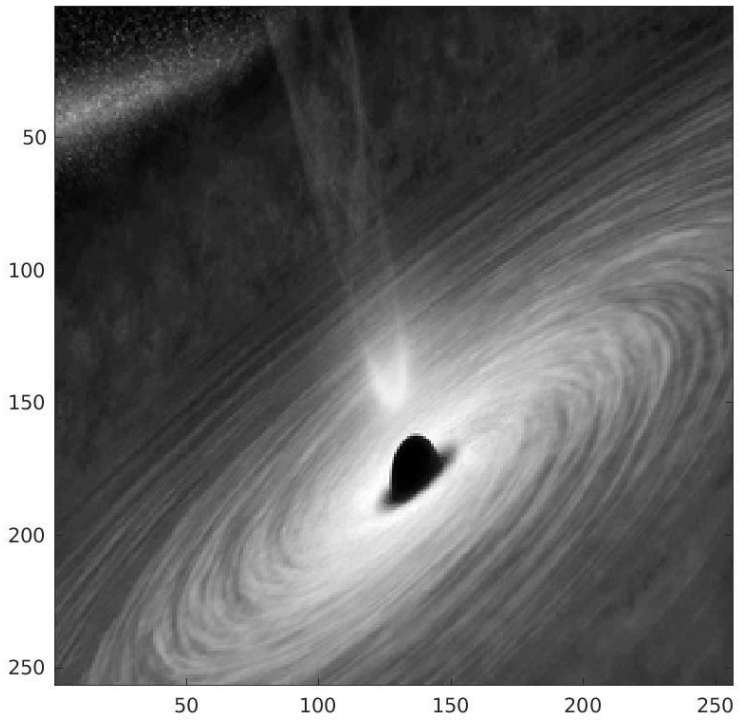
Χρόνος CPU->GPU: 0.002167 sec

Χρόνος επεξεργασίας GPU: 47.501288 sec

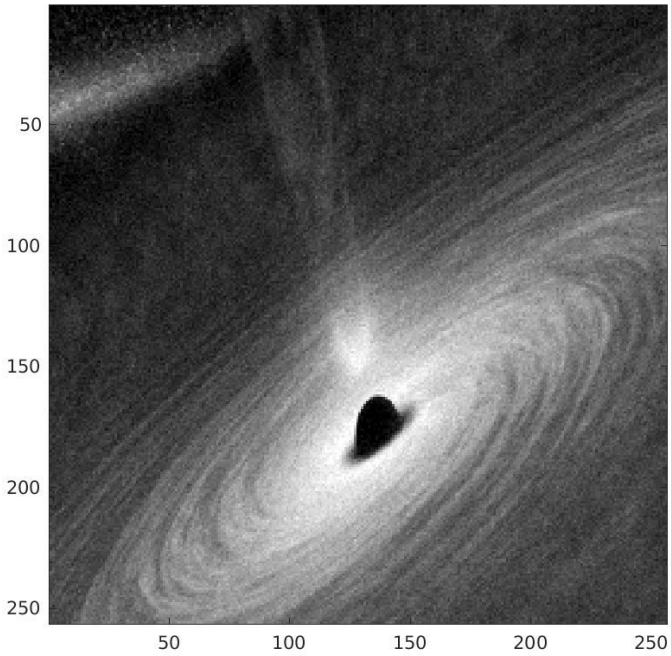
Χρόνος GPU->CPU: 0.000960

- 256x256, 3x3

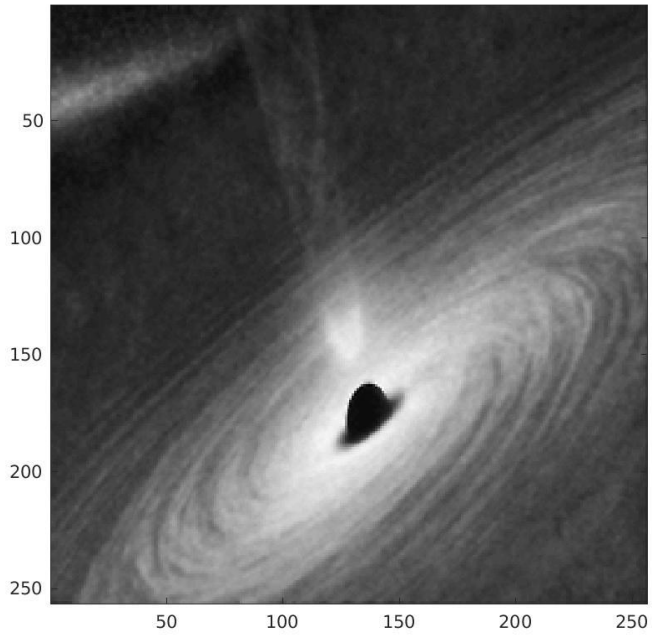
Original image



Original image with noise



Filtered image



`

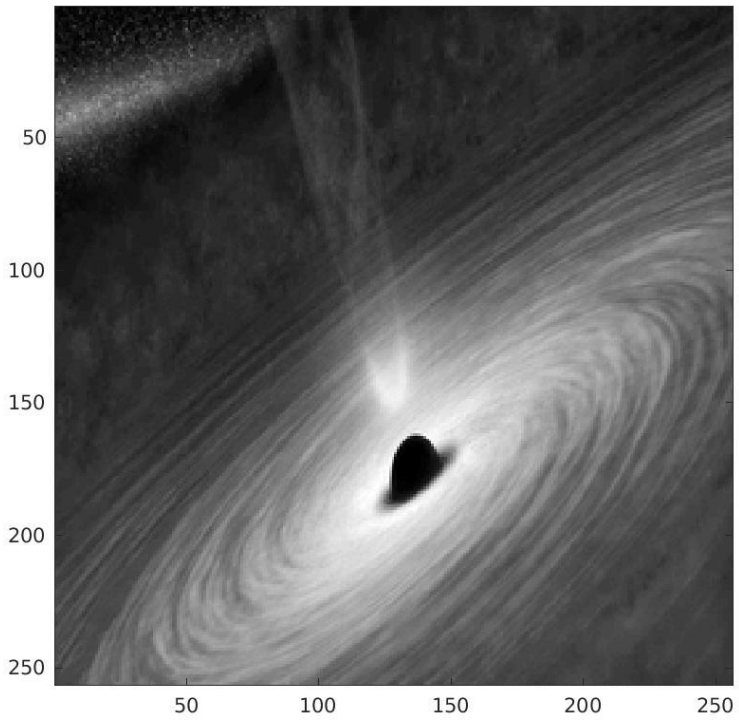
Χρόνος CPU->GPU: 0.081969 sec

Χρόνος επεξεργασίας GPU: 3.385 min

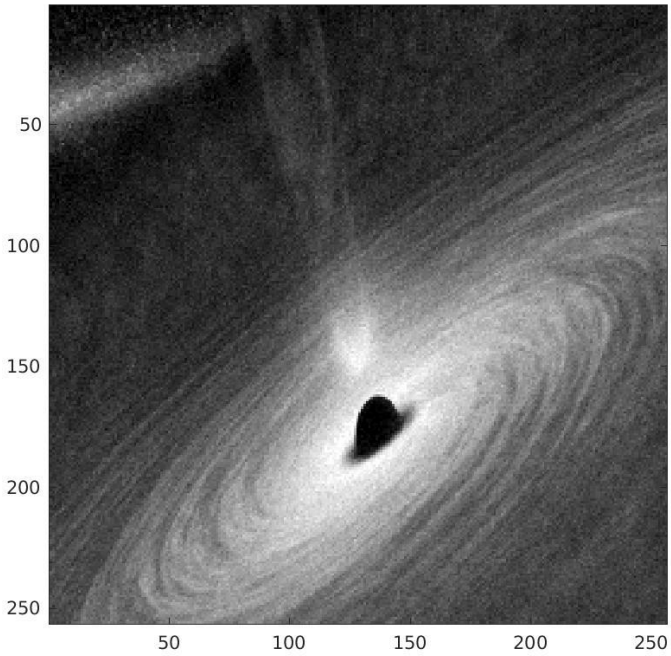
Χρόνος GPU->CPU: 0.001471 sec

- 256x256, 5x5

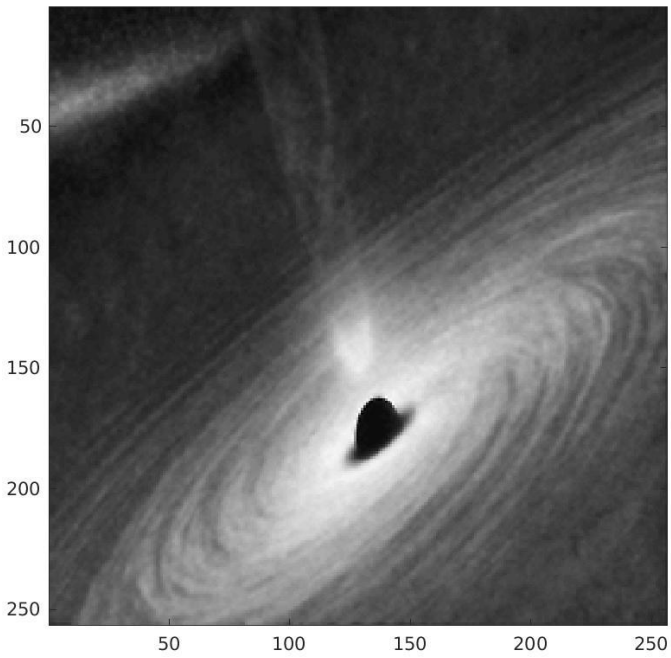
Original image



Original image with noise



Filtered image



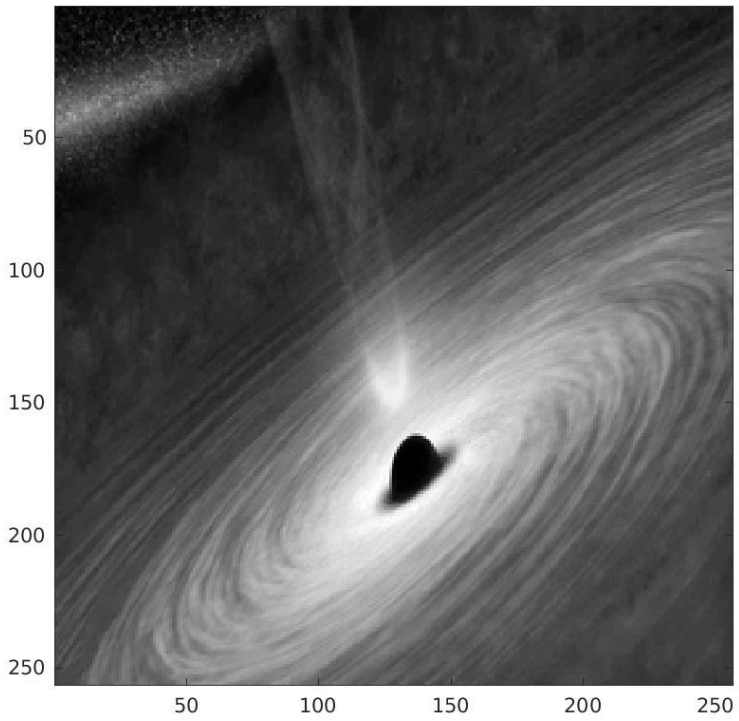
Χρόνος CPU->GPU: 0.079561 sec

Χρόνος επεξεργασίας GPU: 7.829 min

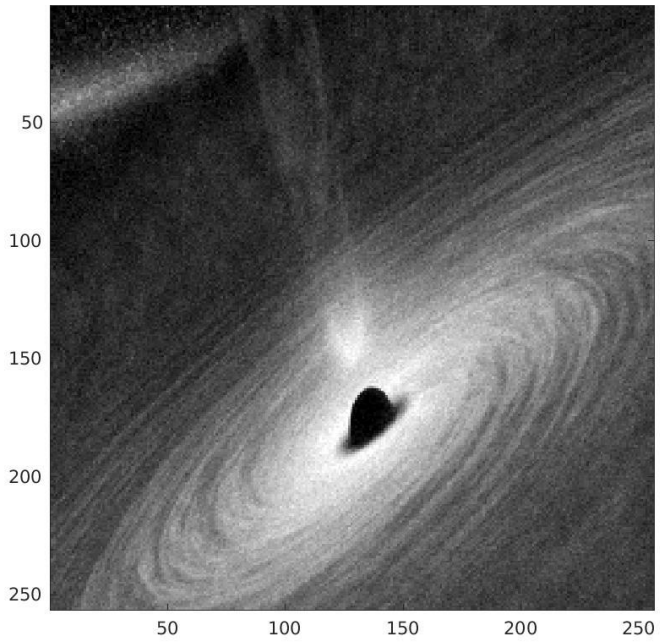
Χρόνος GPU->CPU: 0.001788 sec

- 256x256, 7x7

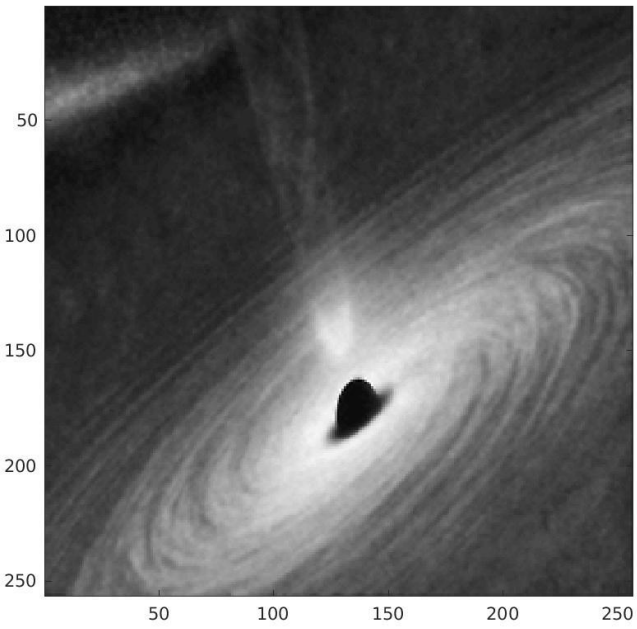
Original image



Original image with noise



Filtered image



Χρόνος CPU->GPU: 0.079535 sec

Χρόνος επεξεργασίας GPU: 15.7156 min

Χρόνος GPU->CPU: 0.001894 sec

Σημείωση:

Οι χρόνοι για 256x256 για 5x5, 7x7 δεν είναι από τον Diades αλλά εκτιμήσεις για το ποιοί ενδεχομένως να ήταν. Αυτό διότι ο server έκανε timeout την σύνδεση επειδή έπαιρνε πολλή ώρα.

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 τη λογική.

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.

Οι χρόνοι για 128x128 και μετά ξεφεύγουν αρχικά, πράγμα αρκετά αναμενόμενο, από τη στιγμή που όπως αναφέραμε η τρέχουσα υλοποίηση είναι μια πρώτη προσέγγιση λύσης της του προβλήματος και της εργασίας, αφού χρησιμοποιεί μονάχα ένα block και global memory, που δίνει σαφώς αργούς χρόνους.

Συνεπώς, το επόμενο στάδιο θα είναι να έχουμε πολλαπλά block, ενδεχομένως όσα και τα πίξελ της εικόνας, μέσα στα οποία τα thread φροντίζουν για τις επεξεργασίες και τους υπολογισμούς και στην ουσία ένα thread θα αντιστοιχεί σε ένα πίξελ. Αυτό σκέφτηκα να υλοποιήσω όμως για τους λόγους που ανέφερα στην αρχή δεν προλάβαινα και γι'αυτό έκανα αυτήν την σχετικά αποτελεσματική μεν αλλά αργή δε, λειτουργική έκδοση.