# IMPLEMENTASI METODE ADAPTIVE MEDIAN FILTER MENGGUNAKAN PEMROGRAMAN CPU-GPU UNTUK PENGHAPUSAN NOISE PADA CITRA BERWARNA

**SKRIPSI**

**Oleh :**
**SAKINAH AMIRAH NUR ROCHMAH**
**NIM : 12650097**

**JURUSAN TEKNIK INFORMATIKA**
**FAKULTASSAINS DAN TEKNOLOGI**
**UNIVERSITAS ISLAM NEGERI MAULANA MALIK IBRAHIMMALANG**
**2016**

**HALAMAN PENGAJUAN**

**IMPLEMENTASI METODE ADAPTIVE MEDIAN FILTER MENGGUNAKAN PEMROGRAMAN CPU-GPU UNTUK PENGHAPUSAN NOISE PADA CITRA BERWARNA**

**SKRIPSI**

**Diajukan Kepada:**
**Fakultas Sains dan Teknologi**
**Universitas Islam Negeri**
**Maulana Malik Ibrahim Malang**
**Untuk Memenuhi Salah Satu Persyaratan Dalam**
**Memperoleh Gelar Sarjana Komputer (S.Kom)**

**Oleh :**
**SAKINAH AMIRAH NUR ROCHMAH**
**NIM : 12650097**

**JURUSAN TEKNIK INFORMATIKA**
**FAKULTAS SAINS DAN TEKNOLOGI**
**UNIVERSITAS ISLAM NEGERI MAULANA MALIK IBRAHIM MALANG**
**2016**

**LEMBAR PERSETUJUAN**

**IMPLEMENTASI METODE ADAPTIVE MEDIAN FILTER MENGGUNAKAN PEMROGRAMAN CPU-GPU UNTUK PENGHAPUSAN NOISE PADA CITRA BERWARNA**

**SKRIPSI**

**Oleh:**

**Sakinah Amirah Nur Rochmah**

**NIM. 12650097**

Telah Diperiksa dan Disetujui untuk Diuji :

Tanggal : 16 September 2016

| Pembimbing I | Pembimbing II |
|---|---|
| Fatchurrochman, M.Kom | A'la Syauqi, M.Kom |
| NIP. 19700731 200501 1 002 | NIP. 19771201 200801 1 007 |

Mengetahui,

Ketua Jurusan Teknik Informatika

Fakultas Sains dan Teknologi

Universitas Islam Negeri Maulana Malik Ibrahim Malang

Dr. Cahyo Crysdian

NIP. 19740424 200901 1 008

**HALAMAN PENGESAHAN**

**IMPLEMENTASI METODE ADAPTIVE MEDIAN FILTER MENGGUNAKAN PEMROGRAMAN CPU-GPU UNTUK PENGHAPUSAN NOISE PADA CITRA BERWARNA**

**SKRIPSI**

**Oleh :**

**Sakinah Amirah Nur Rochmah**

**NIM : 12650097**

Telah Dipertahankan Di Depan Dewan Penguji Skripsi

Dan Dinyatakan Diterima Sebagai Salah Satu Persyaratan

Untuk Memperoleh Gelar Sarjana Komputer (S.Kom)

Tanggal : 16 September 2016

| Susunan Dewan Penguji: | | | Tanda Tangan |
|---|---|---|---|
| 1. Penguji Utama | : | Dr. M. Amin Hariyadi<br>NIP. 19670118 200501 1 001 | ( ) |
| 2. Ketua Penguji | : | Dr. Cahyo Crysdian<br>NIP. 19740424 200901 1 008 | ( ) |
| 3. Sekretaris | : | Fatchurrochman, M.Kom<br>NIP. 19700731 200501 1 002 | ( ) |
| 4. Anggota | : | A'la Syauqi, M.Kom<br>NIP. 19771201 200801 1 007 | ( ) |

Mengetahui,

**Kepala Jurusan Teknik Informatika**

**Dr. Cahyo Crysdian**

**NIP. 19740424 200901 1 008**

## HALAMAN PERNYATAAN
## ORISINALITAS PENELITIAN

**Nama** : Sakinah Amirah Nur Rochmah

**NIM** : 12650097

**Jurusan** : Teknik Informatika

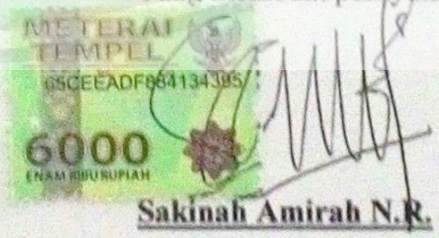
**Fakultas** : Sains dan Teknologi

**Judul Skripsi** : **IMPLEMENTASI METODE ADAPTIVE MEDIAN FILTER MENGGUNAKAN PEMROGRAMAN CPU-GPU UNTUK PENGHAPUSAN NOISE PADA CITRA BERWARNA**

Menyatakan dengan sebenarnya bahwa skripsi yang saya tulis ini benar-benar merupakan hasil karya saya sendiri, bukan merupakan pengambil alihan data, tulisan, atau pikiran orang lain yang saya akui sebagai hasil tulisan atau pikiran saya sendiri, kecuali dengan mencantumkan sumber cuplikan pada daftar pustaka. Apabila dikemudian hari terbukti atau dapat dibuktikan skripsi ini hasil jiplakan, maka saya bersedia menerima sanksi atas perbuatan tersebut.

Malang, 14 September 2016

Yang membuat pernyataan,




**Sakinah Amirah N.R.**

**NIM : 12650097**

**HALAMAN MOTTO**

"MIMPI HARI INI

~ Adalah ~

KENYATAAN HARI ESOK"

**HALAMAN PERSEMBAHAN**

**Bismillahirrohmanirrohim, kupersembahkan sebuah karya sederhanaku ini untuk orang-orang yang paling kusayangi, kubanggakan, dan selalu memberikan energi semangat untukku..**

**Seluruh keluarga besarku**

**khususnya Ayah dan Ibu yang terkasih**

**Anang Maksum dan Laili Inayah**

**yang selalu ikhlas mendoakan putra-putrinya..**

**yang selalu mengarahkan kami dalam kebaikan..**

**yang dengan sabar membimbing kami..**

**Semoga Allah SWT melindungi dan menjaga mereka dalam naungannya..**

**Aamiin**

# KATA PENGANTAR

Segala puji bagi Allah SWT yang Maha Pengasih lagi Maha Penyayang atas Rahmat dan Hidayah-Nya sehingga penulis dapat menyelesaikan penyusunan skripsi ini. Sholawat serta Salam tetap tercurahkan kepada junjungan kita, kekasih Allah, Nabi Muhammad SAW, sang pemberi syafaat kelak di hari akhir, beserta seluruh keluarga, sahabat, dan para pengikutnya.

Penelitian skripsi yang berjudul "**Implementasi Metode *Adaptive Median Filter* Menggunakan Pemrograman CPU-GPU Untuk Penghapusan Noise Pada Citra Berwarna**" ini ditulis untuk memnuhi salah satu syarat guna memperoleh gelar Sarjana Strata Satu (S1) Fakultas Sains dan Teknologi Universitas Maulana Malik Ibrahim Malang. Karya penelitian skripsi ini tidak akan pernah ada tanpa bantuan baik moral maupun spiritual dari berbagai pihak yang telah terlibat. Untuk itu dengan segala kerendahan hati, penulis mengucapkan rasa terimakasih yang sebesar-besarnya kepada:

1. Fatchurrochman, M.Kom, selaku Dosen Pembimbing I dan Bapak Zainal Abidin, S.Kom, M.Kom yang telah bersedia meluangkan waktu, tenaga dan pikiran untuk memberikan bimbingan, berbagai pengalaman, arahan, nasihat, motivasi dan pengarahan dalam pembangunan program hingga penyusunan skripsi ini.
2. Dr. Cahyo Crysdian selaku ketua jurusan Teknik Informatika, dan Bapak A'la Syauqi selaku Dosen Pembimbing II atasbimbingannya.
3. Segenap sivitas akademika Fakultas Psikologi, Universitas Islam Negeri Maulana Malik Ibrahim Malang terutama seluruh dosen, terimakasih atas segala ilmu dan bimbingannya.

4. Mama Laili Inayah, Abi Anang Maksum, dan seluruh keluarga yang selalu memberikan doa, kasih sayang, semangat, dukungan moril, serta motivasi sampai saat ini, terimakasih banyak.
5. Zuliatul Afifa, Kurnia Rizky, Ardana Reswari, sebagai teman, sahabat bahkan saudara yang saling mendukung dalam segala keadaan dan berjuang bersama dalam memperjuangkan gelar sarjana komputer.
6. Ulfatul Aini, Dewi Afifah, Lilik Mahbuba, Tayuh, Etik, Nur Mutammimah, yang membantu internal maupun eksternal dalam bentuk semangat kepada penulis untuk menyelesaikan skripsi ini.
7. Teman-teman angkatan 2012, yang berjuang bersama-sama untuk meraih mimpi, terimakasih atas kenang-kenangan indah yang dirajut bersama.
8. Semua pihak yang tidak dapat penulis sebutkan satu-persatu atas bantuan, masukan, dukungan serta motivasi kepada penulis.

Harapan penulis semoga semua amal kebaikan dan jasa-jasa dari semua pihak yang telah membantu hingga skripsi ini selesai diterima oleh Allah SWT, serta mendapatkan balasan yang lebih baik dan berlipat ganda.

Penulis juga menyadari bahwa skripsi ini masih jauh dari kesempurnaan yang disebabkan keterbatasan Harapan penulis, semoga karya ini bermanfaat dan menambah ilmu pengetahuan bagi kita semua, Aamiin.

Malang, 1 Agustus 2016

Penulis

Sakinah Amirah N.R

## DAFTAR ISI

**HALAMAN PENGAJUAN** ..... ii

**LEMBAR PERSETUJUAN** .....**Error! Bookmark not defined.**

**HALAMAN PENGESAHAN** ..... iii

**HALAMAN PERNYATAAN**.....**Error! Bookmark not defined.**

**HALAMAN MOTTO** ..... vi

**HALAMAN PERSEMBAHAN** ..... vii

**KATA PENGANTAR**..... viii

**DAFTAR ISI**..... x

**DAFTAR GAMBAR**..... xiii

**DAFTAR TABEL** ..... xv

**DAFTAR LISTING KODE**..... xvi

**ABSTRAK** ..... xvii

**ABSTRACT**..... xviii

**الملخص**..... xix

**BAB I**..... 1

**PENDAHULUAN**..... 1

**1.1. Latar Belakang**..... 1

**1.2. Identifikasi Masalah** ..... 5

**1.3. Tujuan Penelitian**..... 5

**1.4. Batasan Masalah** ..... 5

**1.5. Manfaat**..... 6

**BAB II** ..... 8

**STUDI PUSTAKA** ..... 8

**2.1. Citra**..... 8

**2.2. Citra Warna**..... 9

**2.3. Citra Skala Keabuan (Grayscale)** ..... 10
**2.4. Noise** ..... 10
**2.4.1. Gaussian*Noise*** ..... 11
**2.4.2. *Speckle Noise*** ..... 12
**2.4.3. *Salt & PepperNoise*** ..... 12
**2.5. Pengolahan Citra** ..... 12
**2.5.1. Median Filter** ..... 14
**2.5.2. *Adaptive Median Filter*** ..... 15
**2.6. Komputasi Paralel (Parallel Computing)** ..... 17
**2.7. GPGPU** ..... 18
**2.8. OpenCL** ..... 19
**2.9. JOCL** ..... 20
**BAB III** ..... 22
**PERANCANGAN DAN IMPLEMENTASI** ..... 22
**3.1. Prosedur Penelitian** ..... 22
**3.1.1. Pengumpulan data citra** ..... 22
**3.1.2. Pembuatan data citra uji** ..... 23
**3.1.3. Penerapan metode *Adaptive Median Filter*** ..... 24
**3.1.4. Pencatatan dan perhitungan waktu proses** ..... 25
**3.1.5. Penyajian hasil** ..... 25
**3.2. Analisis dan Perancangan** ..... 26
**3.2.1. Metode *Adaptive Median Filter*** ..... 26
**3.2.2. Pengelolaan thread untuk pemetaan data** ..... 33
**3.3. Implementasi** ..... 35
**3.3.1. Implementasi dengan GPU** ..... 35
**3.3.2. Implementasi dengan CPU** ..... 46

**BAB IV** ..... 49

**UJI COBA DAN PEMBAHASAN** ..... 49

**4.1. Data Uji** ..... 49

**4.2. Langkah Pengujian** ..... 49

**4.3. Hasil Uji Coba** ..... 52

**4.3.1. Hasil Implementasi Metode *Adaptive Median Filter*** ..... 53

**4.3.2. Hasil Waktu Komputasi** ..... 65

**4.4. Integrasi Penelitian dengan Al-Qur'an** ..... 77

**BAB V** ..... 80

**PENUTUP** ..... 80

**5.1. Kesimpulan** ..... 80

**5.2. Saran** ..... 80

**DAFTAR PUSTAKA** ..... 81

## DAFTAR GAMBAR


Gambar 2. 1 Contoh Median Filter (Sumber : Lei, 2008) .....15

Gambar 2. 2 Contoh perhitungan median filter (Sumber : Lei, 2008) .....15

Gambar 2. 3 Perbandingan Arsitektur CPU/GPU (Sumber : e2matrix.com/blog/?p=133) .....18

Gambar 3. 1 Diagram alir prosedur penelitian.....24

Gambar 3. 2 Gambar uji 481× 321 px dengan perpotongan sampel berukuran 20×20 px.....26

Gambar 3. 3 Hasil baca nilai pixel berukuran 20×20 px dengan salt and pepper noise dengan 3 sampel window.....27

Gambar 3. 4 Window awal berukuran 3x3 .....28

Gambar 3. 5 Array 1 dimensi dari window 3×3 .....29

Gambar 3. 6 Array 1 dimensi yang terurut .....29

Gambar 3. 7 Array 1 dimensi yang sudah diproses .....29

Gambar 3. 8 Window awal yang berukuran 3×3 .....30

Gambar 3. 9 Array 1 dimensi dari window 3×3 .....30

Gambar 3.10 Array 1 dimensi yang terurut .....30

Gambar 3. 11 Window kedua yang berukuran 5×5 .....30

Gambar 3. 12 Array 1 dimensi dari window 5×5 .....31

Gambar 3. 13 Array 1 dimensi yang terurut .....31

Gambar 3. 14 Array 1 dimensi yang sudah terfilter.....31

Gambar 3. 15 Window yang berukuran 3×3.....32

Gambar 3. 16 Array 1 dimensi dari window 3×3 .....32

Gambar 3. 17 Array 1 dimensi yang terurut .....32

Gambar 3. 18 Array 1 dimensi yang sudah diproses .....32

Gambar 3. 19 Pembagian *work group* dan *work item* dalam pengelolaan *thread*.....34

Gambar 3. 20 Activity Diagram Implementasi dengan GPU .....36

Gambar 3. 21 Activity Diagram Implementasi dengan CPU.....37

Gambar 4. 1 Langkah pengujian .....50

Gambar 4. 2  Data citra uji.....51

Gambar 4. 3  Data hasil keluaran komputasi CPU.....51

Gambar 4. 4 Data hasil keluaran komputasi GPU .....52

Gambar 4. 5. Data citra uji indoor salt and pepper noise 10%.....53

Gambar 4. 6. Data citra hasil komputasi CPU .....54

Gambar 4. 7. Data citra hasil komputasi GPU.....54

Gambar 4. 8. Data citra uji indoor salt and pepper noise 10%.....57

Gambar 4. 9. Data citra hasil komputasi CPU .....57

Gambar 4. 10. Data citra hasil komputasi GPU.....57

Gambar 4. 11. Data citra uji indoor salt and pepper noise 20%.....59

Gambar 4. 12. Data citra hasil komputasi CPU .....59

Gambar 4. 13. Data citra hasil komputasi GPU.....59

Gambar 4. 14. Data citra uji indoor salt and pepper noise 20%.....61

Gambar 4. 15. Data citra hasil komputasi CPU .....62

Gambar 4. 16. Data citra hasil komputasi GPU.....62

Gambar 4. 17 Hasil Waktu Komputasi Citra Indoor, Salt and PepperNoise 10%.69

Gambar 4. 18 Hasil Waktu Komputasi Citra Indoor, Salt and Pepper Noise 10%70

Gambar 4. 19 Hasil Waktu Komputasi Citra Outdoor, Salt and Pepper Noise 20% .....71

Gambar 4. 20 Hasil Waktu Komputasi Citra Outdoor, Salt and Pepper Noise 20% .....72

Gambar 4. 21 Rata-rata Waktu Komputasi.....74

## DAFTAR TABEL

Tabel 4. 1. Hasil Pengurangan Gambar Input Indoor dengan Gambar Hasil pada Salt & Pepper noise 10%.....54

Tabel 4. 2. Hasil Pengurangan Gambar Input Indoor dengan Gambar Hasil pada Salt & Pepper noise 20%.....55

Tabel 4. 3. Hasil Pengurangan Gambar Input Outdoor dengan Gambar Hasil pada Salt & Pepper noise 10%.....58

Tabel 4. 4. Hasil Pengurangan Gambar Input Outdoor dengan Gambar Hasil pada Salt & Pepper noise 20%.....60

Tabel 4. 5. Nilai *Peak Signal to Noise Ratio*.....63

Tabel 4. 6. Hasil waktu komputasi citra outdoor salt and pepper noise 10% .....66

Tabel 4. 7. Hasil waktu komputasi citra outdoor salt and peppernoise 20% .....66

Tabel 4. 8. Hasil waktu komputasi citra outdoor salt and pepper noise 10% .....69

Tabel 4. 9. Hasil waktu komputasi citra outdoor salt and pepper noise 20% .....75

## DAFTAR LISTING KODE

Listing 3. 1 Kode mengambil nilai gambar.....35

Listing 3. 2 Setting platform dan perangkat.....37

Listing 3. 3 Setting context .....38

Listing 3. 4 Setting *command-queue*.....38

Listing 3. 5 Membuat memory object (input & output).....39

Listing 3. 6 Kode untuk *build* .....42

Listing 3. 7 Kode membuat ojek kernel .....43

Listing 3. 8 Mengatur argumen kernel .....44

Listing 3. 9 Membaca memori dari device ke host .....45

Listing 3. 10 Membersihkan memori objek program .....45

Listing 3. 11 Membersihkan memori untuk objek kernel, command-queue, dan context.....46

Listing 4. 1 Kode mendapatkan waktu proses CPU.....50

Listing 4. 2 Kode mendapatkan waktu proses GPU .....50

# ABSTRAK

Sakinah Amirah N.R, **Implementasi Metode *Adaptive Median Filter* Menggunakan Pemrograman CPU-GPU untuk Penghapusan Noise Citra Berwarna**
Pembimbing I:Fatchurrochman, M.Kom
Pembimbing II: A'la Syauqi, M.Kom
**Kata Kunci** : Noise, CPU, GPU, Metode *Adaptive Median Filter*

Perbaikan citra banyak dibutuhkan untuk keperluan pengolahan citra yaitu berupa penyaringan (filter) gambar. Perbaikan citra itu sendiri akan memperbaiki kualitas citra yang diperlukan karena seringkali citra yang dijadikan objek pembahasan mempunyai kualitas yang buruk. Sehingga digunakan Metode *Adaptive Median Filter* untuk menghapus noise agar kualitas citra menjadi lebih baik. Metode *Adaptive Median Filter*ini digunakan dengan cara mengaktifkan fleksibilitas filter untuk mengubah ukurannya sesuai berdasarkan perkiraan kepadatan *noise* lokal. Metode *Adaptive Median Filter*didasarkan pada pembanding trans-konduktansi, di mana arus saturasi dapat dimodifikasi untuk bertindak sebagai operator bobot lokal (Ambule et al, 2013). Namun adanya beberapa kondisi dalam *Adaptive Median Filter*, diperlukan komputasi yang lebih cepat untuk performa yang lebih baik. Sehingga ditawarkan proses filtering yang dilakukan pada GPU, namun tetap menggunakan bantuan host (CPU). Pada implementasi metode dengan komputasi GPU tersebut proses yang dilakukan awalnya dieksekusi pada host (CPU), kemudian dilanjutkan pada device (GPU), dan terakhir dilakukan pada host (CPU). Adapun data yang diujikan dalam penelitian ini dibedakan menjadi data citra indoor yang diberi *salt and pepper noise*10% sejumlah 40 buah; citra indoor dengan *salt and pepper noise*20% sejumlah 40 buah; citra outdoor dengan *salt and pepper noise*10% sejumlah 40 buah; citra outdoor dengan *salt and pepper noise*20% sejumlah 40 buah.Setelah pengujian dilakukan, didapatkan paparan data dan grafik yang menunjukkan pemrosesan dengan komputasi CPU mengkonsumsi waktu hingga 3767,7 milidetik sedangkan komputasi GPU mengkonsumsi waktu hanya mencapai 1463,72 milidetik. Waktu yang dikonsumsi komputasi CPUlebih banyak hampir 2 kali daripada komputasi GPU.Efisiensi waktu untuk implementasi komputasi GPUpada metode *Adaptive Median Filter*menunjukkan nilai peningkatan konsumsi waktu 60,46% dari komputasi CPU

# ABSTRACT

Sakinah Amirah N.R, **Implementation*Adaptive Median Filter*MethodUsing CPU-GPUProgramming forColor Image Noise Removal**
Main Counselor: Fatchurrochman, M.Kom
Second Counselor: A'la Syauqi, M.Kom
**Keywords**: Noise, CPU, GPU, *Adaptive Median Filter* Method

Many image improvement required for image processing that is image filtering. Repairing the image will improve the image quality is needed because often the image that made the object of discussion has poor quality. Adaptive Median Filter method is used for removing noise so that the image quality becomes better. Adaptive Median Filter Method is used by enabling the flexibility to resize the appropriate filter based on the local noise density estimate. Adaptive Median Filter Method is based on a comparison of trans-conductance, where the current saturation can be modified to act as operator of local weight (Ambule et al, 2013). But there are some conditions in Adaptive Median Filter, speed computing required for better performance. So that offered filtering process performed on the GPU, but still need help of host (CPU). In the implementation of the method with the parallel computing process performed initially executed on the host (CPU), and then proceed to the device (GPU), and last performed on the host (CPU). The data were tested in this study can be divided into indoor image data by salt and pepper noise 10% amount of 40 pieces; indoor image with salt and pepper noise 20% amount of 40 pieces; outdoor image with salt and pepper noise 10% amount of 40 pieces; outdoor image with salt and pepper noise 20% amount of 40 pieces. After testing done, get exposure data and graphs showing the computational processing with CPU consume up to 3767,7 milliseconds while GPU computing consumes time only reached 1463,72milliseconds. CPU computation time consumed nearly 2 times more than the GPU computing. Efficiency time for implementation of GPU computing on the methods Adaptive Median Filter shows the increased value of 60.46% of the time consumption of CPUcomputing.

## الملخص

السكينة أميرةن ر، التكيف وسيطة أسلوب التنفيذ تصفية عن طريق وحدة المعالجة المركزيةك ف اوالبرمجة لإزالة صورة الضوضاء من اللون
المشرف الأول: فتح الرحمان، .الماجتير .كوم
المشرف الثاني: أعلا شوقي.الماجتير .كوم
كلمات البحث: الموازي الحاسبات، اوفن-ج ل(فتح حساب اللغة)، إزالة الضوضاء، ألفا المشذبة متوسط

صورة حدوث تحسن اللازم لمعالجة الصور هو النموذج لتصفية الصورة (فلتر). سيتم إصلاح الصورة نفسها تحسين جودة الصورة ضروري لأنه غالباً الصور ستستخدم كموضوع المناقشة نوعية رديئة. حتى الطريقة المستخدمة "تصفية مدييان التكيفية" لإزالة الضجيج من أجل جودة الصورة على نحو أفضل. استخدمت طريقة "التكيف متوسط تصفية" مع المرونة اللازمة لتمكين عامل التصفية لتغيير حجمه مناسبة على أساس الكثافة التقريبية للضوضاء المحلية. أسلوب "التكيف متوسط تصفية" استناداً إلى مقارنة الموصلية العابرة، حيث يمكن تعديل تدفق التشبع بمثابة عامل للترجيح المحلية (أمبولي ,١٩٩٥)
لكن وجود بعض الشروط في "التكيف متوسط تصفية" أسرع العملية الحسابية المطلوبة لأداء أفضل. حتى تتم تصفية العملية المعروضة على زوجته، ولكن لا تزال تستخدم مساعدة المضيف (ج ف او) تنفيذ الأسلوب مع عملية "الحوسبة الجرافيك" أصلاً أعدم على المضيف(ج ف او) ثم المضي قدما على جهاز(ك ف او) وآخر يقوم على المضيف (ج ف او) أما بالنسبة للبيانات درست في هذه الدراسة هي الموقر بيانات الصورة داخلي إلى معطى الملح والفلفل ضوضاء ٪ ١٠عدد من الفاكهة ٤٠؛ صورة داخلي مع الملح والفلفل الضوضاء ٢٠٪ على عدد من الفاكهة ٤٠؛ الصورة في الهواء الطلق مع الملح والفلفل الضوضاء ١٠٪ على عدد من الفاكهة ٤٠؛ الصورة تجمع مع الملح والفلفل الضوضاء ٢٠٪ في عدد ٤٠ قطعة. بعد يتم إجراء اختبار، وتعرض البيانات التي تم الحصول عليها وتجهيز الرسم البياني يظهر وحدة المعالجة المركزية مع تستهلك الوقت الحوسبة إلى ٢٦٦٩٧ميلي ثانية بينما زوجته الحوسبة مضيعة للوقت فقط الوصول إلى١٥٣٩٥٧ مللي ثانية. تستهلك الوقت الحوسبة أكثر من وحدة المعالجة المركزية وكثير من الأوقات تقريبا ٢ من "الحوسبة الجرافيك". لكفاءة استخدام الوقت الحوسبة في أسلوب التنفيذ "التكيف متوسط تصفية" الجرافيك وتظهر القيمة لزيادة استهلاك الوقت ٦٠٤٦٪ من الحوسبة وحدة المعالجة المركزية.

# BAB I

# PENDAHULUAN

## 1.1. Latar Belakang

Perbaikan citra banyak dibutuhkan untuk keperluan pengolahan citra yaitu berupa penyaringan (filter) gambar. Perbaikan citra itu sendiri akan memperbaiki kualitas citra yang diperlukan karena seringkali citra yang dijadikan objek pembahasan mempunyai kualitas yang buruk. Dapat disebabkan karena ketidaksempurnaan sensor citra, gambar sering rusak oleh kebisingan (*noise*). Impuls Noise adalah jenis yang paling sering muncul pada tipe *noise*. Dalam kebanyakan kasus, impuls noise disebabkan oleh piksel rusak pada sensor kamera, lokasi memori rusak di dalam hardware, atau kesalahan dalam transmisi data. Jenis umum impuls noise dibedakan menjadi dua yaitu *salt-and-pepper noise* dan *random-valued shot noise* (Vasicek dan Sekanina, 2008). Citra yang rusak karena *salt-and-pepper noise, noise* piksel bernilai maksimum atau minimum sedangkan *random-valued shot noise, noise* piksel memiliki nilai berubah-ubah.

Beberapa algoritma filtering banyak digunakan untuk meghilangkan *noise* yang tidak diinginkan pada citra. Algoritma filter yang yang ditawarkan untuk menghilangkan noise tersebut meliputi Mean filter (Jiang, Yuan, Bao, 2007), Gaussian filter (Pei, Shin, dan Feng, 2007), dan Median Filter (Ben, 2006) dimana median filter tersebut terbagi menjadi *Median Filter* (Ben, 2006), *Adaptive Median Filter* (Hwang dan Haddad, 1995), *Switching Median Filter* (Zhang dan Karim, 2002), *Weighted Median Filter* (Brownrigg, 1984), *Adaptive Center-Weighted Median Filter* (Chen dan Wu, 2001), dan *Tristate Median Filter* (Chen, Ma, dan Chen, 1999).

Mean filtering (Chandel dan Gupta 2013) merupakan filter dengan proses yang paling sederhana serta efektif untuk penipisan *noise* karena perata-rataan menghilangkan variasi kecilnya. Adapun mean filter ini diaplikasikan dengan kernel, jika tidak terpenuhi maka gambar yang difilter akan menjadi lebih terang dari gambar aslinya. Selain itu Gaussian filter (Pei, Shin, dan Feng, 2007) efektif untuk menghilangkan *noise* Gaussian, dan termasuk linear low pass filter. Komputasi Gaussian Filter juga efisien karena filter besar diimplementasikan menggunakan filter 1D kecil. Namun pengimplementasian filter yang kecil akan membutuhkan waktu yang setidaknya sama besar seperti melakukan filter tunggal yang besar. Sehingga dengan penerapan gaussian dengan data yang kecil akan memakan bayak waktu. Sedangkan algoritma yang termasuk dalam median filter yaitu Standard Median Filter. Metode ini adalah metode yang paling sederhana dari teknik filtering median dan karena kesederhanaan; telah digunakan untuk waktu yang lama (Shrestha, 2014). Weighted Median Filter (Brownrigg, 1984) ini mampu untuk tetap menjaga detil gambar (Koo dan Lee, 1991). Namun, dalam menjaga detail gambar sangat tergantung pada koefisien bobot, dan sifat dari gambar masukan itu sendiri. Sedangkan dalam situasi praktis, sulit untuk ditemukan koefisien bobot yang cocok untuk Weighted Median Filter, dan filter ini membutuhkan waktu komputasi yang tinggi ketika bobot besar (Asano, Itoh, dan Ichioka, 1991). Switching median filter dapat meminimalkan perubahan pada piksel yang tidak rusak oleh proses filter (Varade, Dhotre, Pahurkar, 2013). Namun prosedur deteksi *noise* yang digunakan oleh peneliti biasanya tergantung pada model *noise* digunakan. Adaptive Center-Weighted Median Filter (ACWMF) jika dibandingkan dengan Median Filter, Center Weighted Median

Filter, Adaptive Center Weighted, ACWMF dapat lebih menekan *noise* dan impuls *noise*. Namum waktu komputasi yang dibutuhkan meningkat setara dengan peningkatan kepadatan *noise* (Ambule et al, 2013). Tristate Median Filter menunjukkan kinerja yang stabil dengan berbagai macam gambar dan dapat menjaga detail gambar (Chen, Ma, dan Chen, 1999). Selain itu, performa bagus jika kepadatan *noise* rendah. Namun jika ukuran window ditingkatkan, gambar cenderung menunjukkan efek kabur (Shrestha, 2014).

*Adaptive Median Filter* memecahkan tujuan ganda yaitu menghilangkan impuls *noise* dari gambar dan mengurangi penyimpangan pada gambar. *Adaptive Median Filtering* dapat menangani operasi filter gambar yang rusak karena impuls *noise* dengan probabilitas lebih besar dari 0,2. *Adaptive Median Filter*ini digunakan untuk mengaktifkan fleksibilitas filter untuk mengubah ukurannya sesuai berdasarkan perkiraan kepadatan *noise* lokal. Filter median adaptif didasarkan pada pembanding trans-konduktansi, di mana arus saturasi dapat dimodifikasi untuk bertindak sebagai operator bobot lokal (Ambule et al, 2013). Namun adanya beberapa kondisi dalam *AdaptiveMedianFilter*, diperlukan komputasi yang lebih cepat untuk performa yang lebih baik.

Allah SWT memerintahkan kepada manusia untuk menggunakan waktu yang diberikan-Nya untuk menjadikannya hal-hal yang bermanfaat. Dimana Allah SWT memberikan input waktu yang sama kepada manusia, namun diantara mereka menghasilkan pencapaian yang berbeda. Hal tersebut dikarenakan kesempatan waktu yang dilakukan manusia berbeda-beda. Allah SWT pun juga telah bersumpah atas nama waktu yang seharusnya manusia tidak menyia-nyiakan

waktu yang diberikan Allah kepada seluruh hambanya. Allah SWT berfirman pada Surat Al-Ashr ayat 1-2

وَٱلۡعَصۡرِ ۝١ إِنَّ ٱلۡإِنسَٰنَ لَفِي خُسۡرٍ ۝٢ إِلَّا ٱلَّذِينَ ءَامَنُواْ وَعَمِلُواْ ٱلصَّٰلِحَٰتِ وَتَوَاصَوۡاْ بِٱلۡحَقِّ وَتَوَاصَوۡاْ بِٱلصَّبۡرِ ۝٣

*Artinya: (1) demi masa. (2) Sesungguhnya manusia itu benar-benar dalam kerugian, (3) kecuali orang-orang yang beriman dan mengerjakan amal saleh dan nasehat menasehati supaya mentaati kebenaran dan nasehat menasehati supaya menetapi kesabaran.*

Dari input waktu yang sama, seharusnya manusia dapat memanfaatkannya secara efisien dalam pencapaiannya. Begitupun juga dengan dunia pemrograman, dalam pengembangannya, adapun pemrograman yang dapat dilakukan sehingga pekerjaan menjadi lebih efisien. Pemrograman tersebut adalah pemrograman GPU. Pemrograman GPU memiliki kinerja lebih cepat dari pemrograman yang dilakukan dengan CPU seperti pemrograman yang dilakukan dari periode terdahulu. Pemrograman GPU pada penelitian ini digunakan untuk meningkatkan kinerja dari proses filter citra yang terdapat noise.

Hasil penelitian dari Sanchez et al (2014) menunjukkan bahwa penggunaan komputasi GPUmemperoleh kinerja terbaik. Kecepatan komputasi ini memungkinkan efisiensi*Adaptive Median Filter* untuk menghilangkan impuls *noise* pada citra. Sehingga mencapai performa yang maksimal dalam kinerjanya.

Pada skripsi ini komputasi GPUakan diterapkan dalam perbaikan citra menggunakan *Adaptive Median Filter* dimana metode filter yang lain kurang

efektif untuk menghilangkan *noise* dan dapat menghilangkan detail halus dari citra itu sendiri.

## 1.2. Identifikasi Masalah

Identifikasi masalah yang muncul dari latar belakang yang telah diuraikan diatas adalah sebagai berikut :

1. Bagaimana mengimplementasikan komputasi GPUpada metode *Adaptive Median Filter* untuk penghapusan*noise*?
2. Seberapa efisiensi waktu metode *Adaptive Median Filter* untuk penghapusan *noise* pada komputasi GPU?

## 1.3. Tujuan Penelitian

Tujuan dari penelitian ini adalah sebagai berikut.

1. Mengimplementasikan komputasi GPUpada metode *Adaptive Median Filter* untuk penghapusan *noise*
2. Untuk mengukur efisiensi waktu metode *Adaptive Median Filter* untuk penghapusan *noise* pada komputasi GPU

## 1.4. Batasan Masalah

Agar penelitian skripsi sesuai dengan pokok permasalahan, peneliti membatasi permasalahan sebagai berikut:

a. *Noise* yang digunakan adalah *salt-and-pepper noise*.
b. Platform yang digunakan adalah OpenCL dan digunakan pula library JOCL (Java bindings for OpenCL).

c. Spesifikasi laptop yang digunakan

   Platform : Laptop Toshiba Satellite C800D

   Memori : 2 GB

   Harddisk : 250 GB HDD

   VGA : AMD Radeon HD 7310 Graphics

d. Kamera yang digunakan adalah kamera smartphone Asus Zenfone 5.

e. Data citra uji yang digunakan berukuran 512×512 px.

f. Ukuran *work group* dan *work item* ditentukan secara statis.

g. Peningkatan ukuran *window* maksimal pada proses *Adaptive Median Filter* adalah 5×5.

### 1.5. Manfaat

Hasil penelitian skripsi ini diharapkan dapat bermanfaat bagi penelitian lain yang banyak menggunakan aplikasi citra dalam keperluan bisnisnya. Pengolahan citra tersebut mempunyai aplikasi yang sangat luas dalam berbagai bidang kehidupan antara lain Bidang Militer untuk mengenali sasaran peluru kendali melalui sensor visual, mengidentifikasi pesawat musuh melalui radar, Teropong malam hari (night vision); Bidang Medis / Kedokteran untuk mendeteksi retak/patah tulang dengan CT Scan, rekonstuksi foto janin (USG), mendeteksi kanker (kanker otak); Bidang Biologi untuk pengenalan jenis kromosom melalui gambar mikroskopis; Bidang Pendidikan untuk pengolahan pendaftaran mahasiswa menggunakan scanner; Bidang Geografi dan Geologi untuk pemetaan batas wilayah melalui foto udara / Landsat, mengenali jenis dan bentuk lapisan batuan bawah permukaan bumi melalui rekonstruksi hasil seismik; Bidang

Kepolisian / Hukum untuk Pengenalan pola sidik jari (finger print), rekonstruksi wajah pelaku kejahatan, pengenalan pola hasil uji balistik; Bidang Perdagangan untuk pembacaan barcode pada barang di swalayan, mengenali huruf / angka pada suatu formulir secara otomatis; Bidang Hiburan untuk pemampatan video (MPEG); Bidang Komunikasi data untuk pemampatan citra yang ditransmisi (Internet).

# BAB II

# STUDI PUSTAKA

Dalam sebuah penelitian diperlukan pustaka yang dapat membantu dalam penelitian tersebut. Untuk itu dijelaskan mengenai acuan-acuan untuk membangun penelitian ini meliputi citra dan noise, yang merupakan objek dari penelitian ini ; metode yang diimplementasikan untuk penghapusan *noise* yaitu metode *Adaptive Median Filter* ; komputasi yang direkomendasikan dalam penelitian ini yaitu komputasi GPU; *platform*, *framework*, dan *library* untuk membangun komputasi GPU.

## 2.1. Citra

Citra (*image*) adalah bidang dalam dwimatra (dua dimensi) (Munir, 2004). Sebagai salah satu komponen multimedia, citra memegang peranan sangat penting sebagai bentuk informasi visual. Seiring dengan perkembangan teknologi pengolahan citra (*imageprocessing*) telah banyak dipakai di berbagai bidang. Citra adalah gambar dua dimensi yang dihasilkan dari gambar analog dua dimensi yang kontinu menjadi gambar diskrit melalui proses sampling. Gambar analog dibagi menjadi N baris dan M kolom sehingga menjadi gambar diskrit. Dimana setiap pasangan indeks baris dan kolom menyatakan suatu titik pada citra. Nilai matriksnya menyatakan nilai kecerahan titik tersebut. Titik tersebut dinamakan sebagai elemen citra atau piksel. Dalam kamus komputer, gambar atau foto diistilahkan sebagai citra digital yang mempunyai representasi matematis berupa matriks m x n = $(c_{ij})$.

Citra digital merupakan representasi dari citra yang diambil oleh mesin dengan bentuk pendekatan berdasarkan sampling dan kuantisasi (Basuki dan Palandi, 2005) Setiap citra digital memiliki beberapa karakteristik, antara lain ukuran citra, resolusi dan format nilainya. Umumnya citra digital berbentuk persegi panjang yang memiliki lebar dan tinggi tertentu. Ukuran ini biasanya dinyatakan dalam banyaknya titik atau piksel, sehingga ukuran citra selalu bernilai bulat.

## 2.2. Citra Warna

RGB adalah suatu model warna yang terdiri dari merah, hijau, dan biru, digabungkan dalam membentuk suatu susunan warna yang luas. Setiap warna dasar, misalnya merah, dapat diberi rentang nilai. Untuk monitor komputer, nilai rentangnya paling kecil=0 dan paling besar=255. Pilihan skala 256 ini adalah didasarkan pada cara mengungkap 8 digit bilangan biner yang digunakan oleh mesin komputer. Dengan cara ini, akan diperoleh warna campuran sebanyak 256×256×256 = 16777216 jenis warna. Sebuah jenis warna, dapat dibayangkan sebagai suatu vektor di ruang 3 dimensi yang biasanya dipakai dalam matematika, koordinatnya dinyatakan dalam bentuk tiga bilangan, yaitu komponen-x, komponen-y, dan komponen-z. misalkan sebuah vektor dituliskan sebagai r = (x,y,z). untuk warna, komponen-komponen tersebut digantikan oleh komponen R (*Red*), G (*Green*), B (*Blue*). Jadi, sebuah jenis warna dapat dituliskan sebagai berikut: warna RGB (30, 75, 255). Putih = RGB (255, 255, 255), sedangkan untuk hitam=RGB (0, 0, 0).

## 2.3. Citra Skala Keabuan (Grayscale)

Citra skala keabuan memberi kemungkinan warna yang lebih banyak dari pada citra biner, karena terdapat kemungkinan nilai-nilai lain antara nilai minimum (0) hingga nilai maksimum. Banyaknya kemungkinan nilai tergantung dari jumlah bit yang digunakan. Contoh, jika skala keabuan yang digunakan bernilai 4 bit, maka jumlah kemungkinan nilai adalah 24 = 16, dan nilai maksimum adalah 24 -1 = 15. Sedangkan untuk skala keabuan 8 bit, maka jumlah kemungkinan nilainya adalah 28 = 256, dengan nilai maksimumnya 28 -1 = 255 Format citra ini umumnya memiliki warna antara hitam sebagai warna minimal dan warna putih sebagai warna maksimal, sedangkan warna diantaranya adalah warna kelabu. Dalam prakteknya warna yang dipakai tidak terbatas pada warna kelabu, sebagai contoh dipilih warna minimalnya adalah warna putih dan warna maksimalnya adalah warna merah, maka semakin besar nilainya maka semakin besar pula intensitas warna merahnya. Beberapa buku menyebut format citra ini sebagai citra intensitas (Achmad dan Firdausy, 2005).

## 2.4. Noise

*Noise* adalah citra atau gambar atau piksel yang mengganggu kualitas citra. *Noise* dapat disebabkan oleh gangguan fisis (optik) pada alat akuisisi maupun secara disengaja akibat proses pengolahan yang tidak sesuai, selain itu *noise* juga dapat disebabkan oleh kotoran-kotoran yang terjadi pada citra. Terdapat beberapa *noise* sesuai dengan bentuk dan karakteristik jenis, yaitu *salt&pepper, gaussian, uniform*, dan *noise speckle*. Banyak metode yang ada dalam pengolahan citra yang bertujuan untuk mengurangi atau menghilangkan *noise*. *Noise* muncul biasanya

sebagai akibat dari pembelokkan yang tidak bagus (sensor *noise, photographic gainnoise*). Gangguan tersebut umumnya berupa variasi intensitas suatu piksel dengan piksel-piksel tetangganya. Secara visual, gangguan mudah dilihat oleh mata karena tampak berbeda dengan piksel tetangganya. Piksel yang mengalami gangguan umumnya memiliki frekuensi tinggi. Komponen citra yang berfrekuensi rendah umumnya mempunyai nilai piksel konstan atau berubah sangat lambat. Operasi *denoise* dilakukan untuk menekan komponen yang berfrekuensi tinggi dan meloloskan komponen yang berfrekuensi rendah (Munir, 2004) Reduksi *noise* adalah suatu proses menghilangkan atau mengurangi *noise* dari suatu signal. Reduksi *noise* secara konsep hampir sama penerapannya pada setiap jenis signal, tetapi untuk implementasinya, reduksi *noise* tergantung dari jenis signal yang akan diproses.

Secara umum metode untuk mereduksi *noise* dapat dilakukan dengan cara melakukan operasi pada citra digital dengan menggunakan suatu jendela ketetanggan, kemudian jendela tersebut diterapkan dalam citra. Proses tersebut dapat juga disebut proses filtering.

Berdasarkan bentuk dan karakteristiknya, *noise* pada citra dibedakan menjadi beberapa macam yaitu:

### 2.4.1. Gaussian*Noise*

*GaussianNoise* merupakan model *noise* yang mengikuti distribusi normal standar dengan rata-rata nol dan standar deviasi 1. Efek dari *noise* ini adalah munculnya titik-titik berwarna yang jumahnya sama dengan persentase *noise*.

*GaussianNoise* dapat dibangkitkan dengan cara membangkitkan bilangan acak [0,1] dengan distribusi Gaussian kemudian titik-titik yang terkena *noise*, nilai fungsi citra ditambahkan dengan *noise* yang ada.

Untuk membangkitkan bilangan acak berdistribusi Gaussian, tidak dapat langsung menggunakan fungsi rnd, tetapi diperlukan suatu metode yang digunakan untuk mengubah distribusi bilangan acak ke dalam fungsi f tertentu. Dalam buku ini digunakan metode *rejection* untuk memudahkan dalam alur pembuatan programnya. Metode *rejection* dikembangkan dengan cara membangkitkan dua bilangan acak (x,y) dan ditolak bila y>f(x).

### 2.4.2. *Speckle Noise*

*Noise* ini dapat dibangkitkan dengan cara membangkitkan bilangan 0 (warna hitam) pada titik-titik yang secara probabilitas lebih kecil dari nilai probabilitas *noise*.

### 2.4.3. *Salt & PepperNoise*

*Noise* ini dapat dibangkitkan dengan cara membangkitkan bilangan 255 (warna putih) pada titik-titik yang secara probabilitas lebih kecil dari nilai probabilitas *noise*.

## 2.5. Pengolahan Citra

*Image processing* atau pengolahan citra adalah salah bidang dalam dunia komputer yang mulai berkembang sejak manusia memahami bahwa komputer tidak hanya mampu menangani data teks, tetapi juga data citra (Ahmad, 2005).

Terminologi pengolahan citra dipergunakan bila hasil pengolahan data yang berupa citra, adalah juga berbentuk citra yang lain, yang mengandung atau memperkuat informasi khusus pada citra hasil pengolahan sesuai dengan tujuan pengolahannya. Sesuai dengan perkembangannya, pengolahan citra mempunyai dua tujuan utama, yakni sebagai berikut:

a) Memperbaiki kualitas citra, dimana citra yang dihasilkan dapat menampilkan informasi secara jelas atau dengan kata lain manusia dapat melihat informasi yang diharapkan dengan menginterpretasikan citra yang ada. Dalam hal ini interpretasi terhadap informasi yang ada tetap dilakukan oleh manusia.
b) Mengekstraksi informasi ciri yang menonjol pada suatu citra, dimana hasilnya adalah informasi citra dimana manusia mendapat informasi ciri dari citra secara numerik atau dengan kata lain komputer (mesin) melakukan interpretasi terhadap informasi yang ada pada citra melalui besaran-besaran data yang dapat dibedakan secara jelas (besaran- besaran ini berupa besaran numerik).

Secara umum, operasi-operasi pada pengolahan citra diterapkan pada citra bila:

a) Perbaikan atau memodifikasi citra perlu dilakukan untuk meningkatkan kualitas penampakan atau untuk menonjolkan beberapa aspek informasi yang terkandung di dalam citra.
b) Elemen di dalam citra perlu dikelompokkan, dicocokkan, diukur
c) Sebagian citra perlu digabung dengan citra yang lain.

Operasi-operasi pengolahan citra diklasifikasikan dalam beberapa jenis sebagai berikut (Munir, 2004)

a) Perbaikan kualitas citra (*image enhancement*)

b) Pemugaran citra (*image restoration*)

c) Pemampatan citra (*image compression*)

d) Segmentasi citra (*image segmentation*)

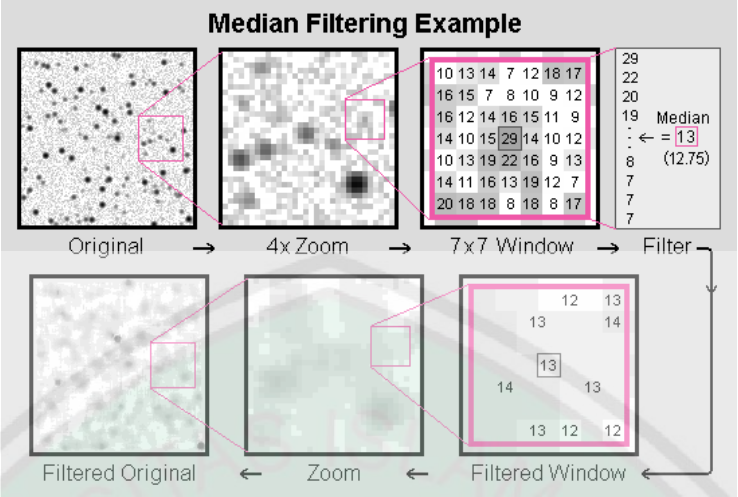
e) Pengorakan citra (*image analysis*)

f) Rekontruksi citra (*image reconstruction*)

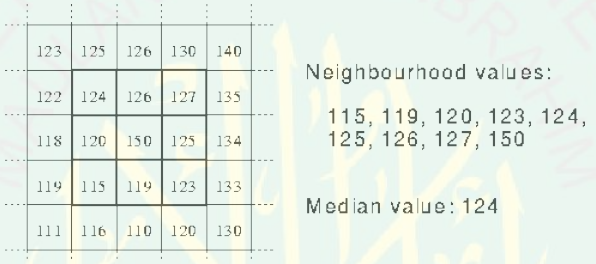
### 2.5.1. Median Filter

Filter median biasa digunakan untuk mengurangi *noise* dalam gambar, seperti halnya mean filter. Namun, efektifitas filter median lebih baik daripada mean filter dalam mempertahankan detail piksel yang berguna dalam gambar. Median filter menghaluskan data sekaligus mempertahankan detail-detail kecil dan tajam. Median adalah nilai tengah dari semua nilai-nilai piksel di lingkungan. Median tidak sama dengan rata-rata. Namun, median memiliki setengah nilai dalam lingkungan yang lebih besar dan setengah lebih kecil. Median filter hampir tidak dipengaruhi oleh perbedaan nilai sejumlah kecil antara piksel di lingkungan. Akibatnya, median filter sangat efektif menghilangkan berbagai macam *noise*. (Gambar 2.1) menggambarkan contoh median filter.

Mean filter maupun median filter menganggap setiap piksel dalam gambar pada gilirannya dan melihat tetangga terdekat untuk memutuskan apakah wakil dari sekitarnya atau tidak. Median dihitung dengan terlebih dahulu menyortir semua nilai piksel dari lingkungan sekitarnya ke dalam urutan numerik dan kemudian mengganti piksel yang dipertimbangkan dengan nilai piksel tengah. (Gambar 2.2) mengilustrasikan perhitungan contoh.




Gambar 2. 1 Contoh Median Filter (Sumber : Lei, 2008)




Gambar 2. 2 Contoh perhitungan median filter (Sumber : Lei, 2008)

### 2.5.2. *Adaptive Median Filter*

Standar Median Filter memiliki masalah yang dapat diatasi dengan *Adaptive median filter*. Perbedaan mendasar antara dua filter ini adalah bahwa pada *Adaptive Median Filter*besarnya *window* (jendela) sekitarnya setiap piksel adalah variabel. Variasi ini tergantung pada median dari piksel dalam jendela sekarang atau saat ini. Jika nilai rata-rata adalah impuls, maka ukuran jendela akan diperluas. Jika tidak, proses lebih lanjut dilakukan pada citra dalam spesifikasi jendela saat ini. Pada dasarnya pada "pengolahan" citra diperlukan : piksel pusat dari jendela (*window*) dievaluasi untuk memverifikasi apakah itu suatu impuls atau bukan. Jika itu adalah suatu impuls, maka nilai piksel baru pada gambar yang telah difilter akan menjadi nilai median dari piksel dalam jendela itu. Jika piksel

pusat bukan suatu impuls, maka nilai dari pusat piksel akan dipertahankan dalam citra yang difilter. Piksel (terkecuali) yang dipertimbangkan sebagai sebuah impuls, nilai *grayscale* dalam piksel pada gambar yang difilter adalah sama dengan citra masukan. *Adaptive Median Filter* memiliki tujuan ganda yaitu menghapus impuls *noise* pada gambar dan mengurangi distorsi pada gambar. *Adaptive Median Filter*dapat menangani operasi filter pada gambar rusak dengan impuls *noise*. Filter ini juga memperhalus *noise*. Dengan demikian, filter ini memberikan output citra jauh lebih baik dari standar median filter.

Filter ini melakukan pengolahan spasial untuk menentukan nilai mana dalam citra yang terkena *noise* dengan membandingkan setiap pikselnya terhadap tetangganya. Ukuran *window* dapat disesuaikan dengan batasan maksimum *window*. Piksel yang berbeda dengan tetangganya maka dianggap sebagai *noise* untuk kemudian digantikan dengan nilai median piksel yang ada dalam satu *window*.

Misalnya $x_{ij}$ , untuk (i,j) $\in A \equiv 1, \dots, M \times 1, \dots, N$ , adalah derajat keabuan dari citra x dengan ukuran M×N pada lokasi (i,j), dan [$S_{min}$ , $S_{max}$] adalah jangkauan dinamik dari x dengan kata lain $S_{min} \leq X_{ij} \leq S_{max}$ untuk semua $i,j \in A$. Kemudian y didefinisikan sebagai citra yang terkena *noise*. Disini akan dijelaskan tentang algoritma *Adaptive Median Filter*. Dimisalkan $S_{ij}^{w}$ adalah sebuah *window* dengan ukuran $w \times w$ dan memiliki pusat di (i,j) serta $W_{max} \times W_{max}$ adalah ukuran maksimal *window*. Tujuan dari algoritma *Adaptive Median Filter*ini adalah mengidentifikasi kandidat *noisey*$_{ij}$ kemudian mengganti setiap $yij$ dengan nilai median dari piksel yang ada pada *windows*$_{ij}^{w}$. Untuk lebih jelasnya dapat dilihat pada penjelasan di bawah ini :

Untuk setiap piksel pada lokasi (i,j) , dilakukan :

a) Inisialisasi ukuran pertama *window*, $w = w + 3$ , karakteristik matriks X.

b) Hitung nilai $S_{ij}^{min,w}$ , $S_{ij}^{med,w}$ , dan $S_{ij}^{max,w}$ yang merupakan nilai minimum, median, dan maksimum dari piksel-piksel yang ada dalam $windows_{ij}^{w}$ .

c) Jika $S_{ij}^{min,w} \leq S_{ij}^{med,w} \leq S_{ij}^{max,w}$ , maju ke langkah 5. Jika tidak, atur ukuran $w = w + 2$.

d) Jika $w \leq w_{max}$ , maka ulangi dari langkah 2. Selain itu ganti piksel $y_{ij}$ dengan $S_{ij}^{med,w}$ kemudian set $x_{ij} = 0$.

e) Jika $S_{ij}^{min}$ ,$w \leq y_{ij} \leq S_{ij}^{max}$ ,$w$ maka $y_{ij}$ bukan *noise* dan tidak perlu diganti nilainya kemudian, set $x_{ij} = 1$. Jika tidak, ganti $y_{ij}$ dengan $S_{ij}^{med,w}$ dan set $x_{ij} = 0$.

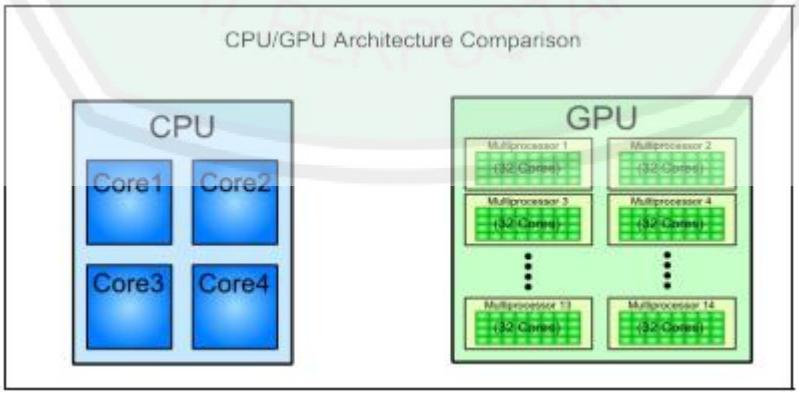
Adapun penelitian yang terkait dengan *Adaptive Median Filter* yaitu (Hwang dan Haddad, 1995) mengusulkan algoritma yang diklaim sederhana dan memberikan performa yang lebih baik dari algoritma rivalnya. Dalam penelitian tersebut diterangkan tentang algoritma yang disebut *Adaptive Median Filter* beserta hasil implementasinya. Algoritma tersebut terdiri dari dua algoritma lagi yaitu *Ranked-Order Based Adaptive Median Fllter* (RAMF) dan *Impulse Size Based Adaptive Median Fllter* (SAMF).

## 2.6. Komputasi Paralel (Parallel Computing)

Kecepatan komputer sekuensial konvensional terus meningkat agar sesuai dengan kebutuhan aplikasi, sampai mencapai batas fisik. Tapi di banyak area, masih perlu daya komputasi yang lebih tinggi, seperti modeling dan solusi numerik dari masalah dalam sains dan teknik, atau perhitungan berulang dari

sejumlah data besar dengan kendala temporal yang tinggi. Sistem tersebut menjadi lebih kompleks yang membutuhkan kapasitas komputasi yang lebih besar. Namun tidak selalu dimungkinkan karena keterbatasan fisikal yang dikenakan oleh pengembang prosesor. Untuk itu, sistem yang dipilih yaitu menggunakan beberapa prosesor yang membentuk sistem paralel. Sistem paralel menyediakan berbagai macam pilihan untuk meningkatkan kinerja seperti, penggunaan pipeline, paralel level instruksi, eksekusi yang berlebih dan spekulasi. Pemrograman paralel didasarkan pada penggunaan beberapa prosesor untuk memecahkan tugas bersama. Cara di mana setiap prosesor akan menghadapi permasalahan yang didefinisikan oleh pengembang, sehingga setiap prosesor bekerja pada bagiannya, bertukar hasil yang diperlukan melalui memori bersama (*shared memory*) atau menggunakan jaringan interkoneksi.

Hasil penelitian dari Sanchez et al (2014) menunjukkan bahwa penggunaan komputasi paralel memperoleh kinerja terbaik. Kecepatan komputasi ini memungkinkan efektifitas pemrograman pada pengolahan citra sehingga mencapai performa yang maksimal dalam kinerjanya.




Gambar 2. 3 Perbandingan Arsitektur CPU/GPU (Sumber :e2matrix.com/blog/?p=133)

**2.7. GPGPU**

GPGPU singkatan *General-Purpose computation on Graphics Processing Units*, yang berarti, tujuan umum komputasi pada *Graphics Processing Unit* (GPU). GPU adalah prosesor kinerja tinggi terdiri dari beberapa core yang mampu melakukan operasi besar pada berbagai data dengan performa yang hebat. Meskipun tahun lalu GPU yang ditujukan terutama pada grafik dan sangat sulit dalam pemrogramannya, saat ini, telah menjadi paralel prosesor yang memiliki banyak tujuan yang mendukung antarmuka tingkat tinggi yang memungkinkan bahasa pemrograman tingkat rendah seperti C/C++. GPU adalah platform yang cocok untuk pelaksanaan tugas-tugas (*tasks*) yang dapat dinyatakan sebagai data komputasi paralel, yang menjadi perangkat yang sangat efisien untuk masalah yang mungkin paralel, kehilangan efektivitas terhadap semua masalah sekuensial ini.

## 2.8. OpenCL

Ada 2 teknologi komputasi GPU yang umum yaitu OpenCL dan CUDA. OpenCL dapat berjalan pada semua GPU dari Intel, AMD, NVIDIA, termasuk yang biasa digunakan pada *mobile device*. Sedangkan CUDA hanya didukung oleh kartu grafis NVIDIA, meski kinerjanya lebih cepat dari OpenCL, karena cara kerjanya dengan mengkompilasi *source codefunction* yang diinginkan dalam bahasa C ke *object binary proprietary* CUDA yang dapat langsung dieksekusi oleh prosesor GPU NVIDIA.

OpenCL (*Open Computing Language*) adalah nama yang diberikan untuk standar arsitektur pemrograman paralel dikembangkan dan dirilis oleh Khronos. Hal ini didukung oleh perusahaan-perusahaan besar yang memproduksi perangkat

keras dan perangkat lunak yang terkait dengan komputasi paralel, seperti AMD, NVIDIA, Apple, IBM, Intel. Teknologi ini mulai mendapatkan penting dalam dunia komputasi tujuan umum di GPU.

Selain itu, menjadi standar yang diakui tidak perlu belajar bahasa pemrograman pada kartu untuk sebuah perusahaan tertentu dan sama sekali berbeda lain dari kartu untuk perusahaan yang berbeda. Hal ini hanya diperlukan untuk memiliki *driver* yang kompatibel dan perpustakaan yang memungkinkan pengembangan OpenCL. Ini telah menyebabkan peningkatan yang signifikan OpenCL digunakan sejak peluncurannya.

Titik utama OpenCL adalah portabilitas. Penting untuk dicatat bahwa ini hanya portabilitas fungsional. Hal ini karena meskipun hasil dari aplikasi yang benar pada perangkat yang berbeda, untuk kinerja terbaik itu perlu untuk mengoptimalkan kode untuk digunakan dalam perangkat tertentu. Untuk alasan ini, aplikasi yang sama, masih berjalan pada dua perangkat yang berbeda, Anda tidak akan mendapatkan kinerja yang sama di kedua.

Untuk arsitektur berbasis OpenCL, 3 *library* yang dapat dipakai adalah ViennaCL, JOCL, dan MAGMA.

ViennaCL dibangun menggunakan C++ dan tersedia sebagai *library* C++ maupun Python. ViennaCL tidak tersedia untuk Java, meski ada kemungkinan menggunakan JNA untuk mengakses native *library* ViennaCL. JOCL adalah *library* OpenCL yang dapat dipakai di Java. MAGMA dikhususkan pada komputasi aljabar linear.

## 2.9. JOCL

*Library* ini menawarkan Java-*Bindings* untuk OpenCL yang sangat mirip dengan API OpenCL yang asli. Fungsi yang disediakan bersifat statis, dan semantik dan penanda dari fungsi disini telah disimpan konsisten dengan fungsi *library* asli, kecuali untuk keterbatasan spesifik bahasa Java. Namun OpenCL API ini pada beberapa titik memiliki sifat yang membosankan. API dari JOCL ini mirip dengan API OpenCL asli, sehingga terdapat berbagai sumber online yang menyediakan OpenCL yang diaplikasikan secara langsung pada program JOCL.

# BAB III

# PERANCANGAN DAN IMPLEMENTASI

Untuk membangun implementasi Metode *Adaptive Median Filter* dengan pemrograman GPU yang akan dibandingkan dengan pemrograman CPU, berikut ini dijelaskan mengenai langkah-langkah untuk melakukan penelitian secara menyeluruh, perancangan untuk metode *AdaptiveMedianFilter*, perancangan dalam mengelola pemecahan data dengan suatu thread, dan implementasi dalam membangun metode *AdaptiveMedianFilter* dengan komputasi CPU dan GPU.

## 3.1. Prosedur Penelitian

Prosedur penelitian dideklarasikan agar tahapan pengerjaan dilakukan secara berurutan. Langkah-langkah untuk implementasi metode *Adaptive Median Filter* dengan komputasi GPUOpenCL dimulai dari pengumpulan data citra, pembuatan data citra uji dengan memberi *salt and pepper noise*, penerapan metode *Adaptive Median Filter*, pencatatan waktu proses, hingga penyajian hasil perbandingan. Prosedur penelitian dapat dilihat pada (Gambar 3.1). Berikut ini rincian langkah prosedur penelitian.

### 3.1.1. Pengumpulan data citra

Data yang digunakan dalam penelitian ini berupa citra digital. Data yang dikumpulkan meliputi data citra RGB dengan format .jpg sejumlah 80 gambar. Data citra didapatkan dari kamera *smartphone* Asus Zenfone 5. Data citra tersebut berukuran 3264×1836. Namun dalam penggunaannya, data yang dipakai sebagai data uji coba adalah citra berukuran 512×512 px. Sehingga terdapat proses untuk

pengubahan ukuran citra. Pengambilan data citra dibedakan menjadi dua lokasi yaitu dari foto luar ruangan(*outdoor*) dan foto dalam ruangan (*indoor*). Data *outdoor* yang digunakan adalah foto pemandangan pada jarak minimal 5 meter dan jarak maksimal mencapai 90 kilometer. Sedangkan yang digunakan sebagai data *indoor* adalah foto dalamruangan seperti di dalam aula, maupun ruang perkantoran. Sehingga diperkirakan jarak maksimal mencapai 50 meter.

Perubahan ukuran citra dari 3264×1836 ke ukuran 512×512 px, dilakukan proses pemotongan gambar agar memiliki perbandingan panjang dan lebar yang sama ; dan proses *resize* agar ukuran menjadi 512×512 px. Pengubahan ukuran dilakukan dengan proses manual menggunakan perangkat lunak *Microsoft Office Picture Manager*. Setelah proses pengubahan ukuran citra, hasil citra 512×512 px disimpan sebagai data citra asli.

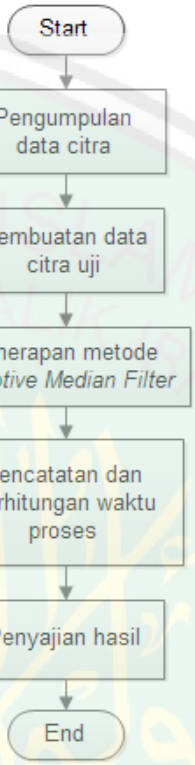
### 3.1.2. Pembuatan data citra uji

Data citra asli yang disimpan seperti pada langkah sebelumnya, belum dapat digunakan untuk proses pengujian karena citra asli yang terdapat noise jarang ditemukan pada lingkungan sekitar. Sehingga perlu dibuat data yang digunakan sebagai data citra uji yaitu denganmemberi *noise* pada data citra asli. Adapun noise yang digunakan dalam penelitian ini yaitu *salt and pepper noise*.

Pemberian *noise* dilakukan menggunakan perangkat lunak Matlab R2013a. *Noise* yang ditambahkan meliputi *salt and peppernoise* 10% dan *salt and peppernoise* 20%. Sehingga didapatkan citra untuk disimpan sebagai data uji berjumlah 160 gambar. Rincian data citra uji adalah sebagai berikut :

1. 40 data citra *indoor* dengan persentase *noise* 10%

2. 40 data citra *indoor* dengan persentase *noise* 20%
3. 40 data citra *outdoor* dengan persentase *noise* 10%
4. 40 data citra *outdoor* dengan persentase *noise* 20%




Gambar 3. 1 Diagram alir prosedur penelitian

### 3.1.3. Penerapan metode *Adaptive Median Filter*

Penerapan metode yang dilakukan adalah penulisan *source code* ke dalam komputasi CPU dan GPU dengan metode *Adaptive Median Filter*. Pada komputasi CPU, *source code* yang ditulis adalah menggunakan bahasa java.

Sedangkan komputasi GPU, menggunakan*framework* yang mengeksekusi dengan lintas *platform* (CPU dan GPU) yang dinamakan Opencl. Dalam Opencl tersebut, source code yang ditulis berekstensi .cl yang merupakan proses menggunakan Opencl. Namun tidak hanya dengan file Opencl (.cl) tersebut, namun juga ada proses penulisan *source code* pada host. Dalam penelitian ini

pemrograman pada host menggunakan bahasa java dengan library yang menawarkan *bindings* untuk OpenCL.Komputasi GPU dapat membagi piksel menjadi beberapa kelompok untuk diproses secara bersamaan.

Hasil penerapan metode *Adaptive Median Filter*pada source code yang telah dibuat, dijalankan pada data uji satu persatu dengan komputasi CPU maupun GPU.

### 3.1.4. Pencatatan dan perhitungan waktu proses

Setelah proses penerapan metode *Adaptive Median Filter*dan uji coba setiap gambar selesai, maka akan dinilai performa pemrosesannya per uji coba.Pencatatan waktu yang diambil untuk menilai komputasi CPU ataukah GPU yang paling cepat. Data waktu proses dicatat ke dalam sebuah tabel menurut macamnya. Tabel yang dibuat meliputi tabel data uji *indoor* dengan *salt and pepper noise* 10%, tabel data uji *indoor* dengan *salt and pepper noise* 20%,tabel data uji *outdoor* dengan *salt and pepper noise* 10%, tabel data uji *outdoor* dengan *salt and pepper noise*20%. Kolom tabel yang disediakan meliputi Nama Data dan Kecepatan pada CPU dan GPU.

### 3.1.5. Penyajian hasil

Untuk mempermudah dalam membaca data yang telah didapat, maka data harus dibuat diagram. Diagram yang disajikan digunakan untuk melihat perbandingan proses komputasi CPU dan GPU. Data tabel yang telah dibuat pada langkah sebelumnya digunakan sebagai acuan untuk membuat diagram.

## 3.2. Analisis dan Perancangan

Sebelum membuat implementasi metode *Adaptive Median Filter*menggunakan komputasi GPU, pembuatan rancangan diperlukan sebagai pokok-pokok perencanaan penelitian dan dasar dalam melakukan penelitian. Perancangan yang diperlukan untuk penelitian ini meliputi proses metode *Adaptive Median Filter.*

Gambar 3. 2 Gambar uji 481× 321 px dengan perpotongan sampel berukuran 20×20 px

### 3.2.1. Metode *Adaptive Median Filter*

Untuk melakukan implementasi penghapusan *noise* dengan Metode *Adaptive Median Filter,*perlu dilakukan perancangan perhitungan secara manual. Data yang digunakan dalam perancangan perhitungan secara manual sebagai data uji coba adalah gambar outdoor. Dari gambar tersebut diambil satu layer (layer *blue*), sehingga nilai piksel menunjukkan skala keabuan (*grayscale*) seperti pada (Gambar 3.3) dengan ukuran 481×321 piksel. Dalam pengujian ini, sampel yang

diujikan adalah nilai piksel dalam ukuran 20×20 piksel perpotongan dari gambar outdoor utuh pada (Gambar 3.2). Kemudian pada potongan gambar sampel ditambahkan *salt and pepper noise*.

(Gambar 3.3) menunjukkan hasil baca nilai piksel gambar uji yang berukuran 20×20 px. Berikut tahapan proses metode *Adaptive Median Filter*. Data sampeldiaplikasikan dalam tiga kondisi kasus untuk menyesuaikan algoritma metode *Adaptive Median Filter*pada (Gambar 3.2). Ketiga kondisi tersebut yaitu jika piksel pada koordinat x,y ($Z_{xy}$) adalah *noise* berukuran kecil, jika $Z_{xy}$*noise* berukuran besar, dan jika piksel pada koordinat x,y ($Z_{xy}$) bukan *noise*. Untuk menyesuaikan algoritma metode *Adaptive Median Filter*pada ketiga kasus, maka pada (Gambar 3.3) diatur untuk ditambahkan noise dengan ketiga kondisi tersebut.

| | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 191 | 188 | 185 | 182 | 182 | 183 | 186 | 190 | 191 | 198 | 204 | 206 | 205 | 207 | 214 | 217 | 215 | 210 | 207 | 205 |
| 191 | 189 | 186 | 184 | 184 | 185 | 187 | 191 | 190 | 197 | 203 | 205 | 204 | 206 | 212 | 216 | 205 | 205 | 207 | 210 |
| 192 | 190 | 188 | 186 | 185 | 186 | 188 | 191 | 190 | 195 | 199 | 202 | 203 | 204 | 210 | 214 | 211 | 213 | 216 | 218 |
| 191 | 189 | 187 | 185 | 185 | 185 | 188 | 189 | 189 | 194 | 197 | 199 | 202 | 203 | 207 | 210 | 216 | 218 | 218 | 216 |
| 190 | 187 | 186 | 185 | 185 | 185 | 186 | 187 | 190 | 192 | 195 | 196 | 199 | 201 | 206 | 208 | 203 | 206 | 209 | 208 |
| 189 | 188 | 187 | 184 | 184 | 185 | 185 | 186 | 191 | 191 | 192 | 195 | 198 | 202 | 205 | 206 | 196 | 203 | 209 | 208 |
| 191 | 189 | 189 | 188 | 186 | 0 | 0 | 0 | 193 | 192 | 193 | 196 | 200 | 204 | 205 | 205 | 206 | 206 | 202 | 187 |
| 192 | 192 | 192 | 190 | 189 | 0 | 0 | 0 | 194 | 194 | 193 | 196 | 201 | 203 | 205 | 205 | 211 | 202 | 180 | 148 |
| 195 | 193 | 191 | 186 | 183 | 182 | 188 | 194 | 192 | 185 | 196 | 207 | 201 | 199 | 208 | 212 | 190 | 160 | 62 | 41 |
| 193 | 192 | 190 | 191 | 192 | 191 | 192 | 194 | 190 | 207 | 214 | 201 | 200 | 214 | 197 | 147 | 75 | 74 | 34 | 21 |
| 193 | 192 | 191 | 192 | 196 | 198 | 196 | 193 | 202 | 199 | 200 | 200 | 198 | 177 | 110 | 28 | 15 | 38 | 62 | 45 |
| 194 | 194 | 193 | 191 | 192 | 197 | 197 | 196 | 187 | 206 | 212 | 175 | 108 | 60 | 29 | 0 | 11 | 43 | 101 | 74 |
| 194 | 197 | 195 | 190 | 190 | 198 | 203 | 204 | 202 | 202 | 156 | 68 | 12 | 31 | 55 | 40 | 11 | 39 | 112 | 99 |
| 192 | 197 | 197 | 192 | 193 | 200 | 201 | 197 | 196 | 119 | 31 | 0 | 4 | 61 | 92 | 77 | 16 | 35 | 98 | 123 |
| 193 | 196 | 196 | 194 | 199 | 199 | 183 | 162 | 106 | 43 | 5 | 8 | 17 | 35 | 75 | 103 | 43 | 33 | 53 | 102 |
| 198 | 199 | 197 | 198 | 204 | 196 | 164 | 129 | 51 | 40 | 40 | 38 | 26 | 32 | 63 | 82 | 91 | 51 | 22 | 72 |
| 205 | 206 | 203 | 201 | 200 | 198 | 183 | 168 | 112 | 46 | 48 | 76 | 74 | 57 | 47 | 54 | 58 | 77 | 53 | 49 |
| 195 | 198 | 203 | 207 | 215 | 220 | 214 | 206 | 158 | 86 | 50 | 45 | 65 | 91 | 77 | 50 | 46 | 61 | 38 | 35 |
| 200 | 205 | 209 | 214 | 220 | 224 | 223 | 220 | 186 | 124 | 80 | 51 | 52 | 79 | 84 | 78 | 58 | 65 | 49 | 44 |
| 213 | 216 | 218 | 217 | 215 | 214 | 214 | 215 | 197 | 138 | 108 | 85 | 59 | 46 | 61 | 106 | 85 | 81 | 70 | 66 |

Gambar 3. 3 Hasil baca nilai pixel berukuran 20×20 px dengan salt and pepper noise dengan 3 sampel window

Berikut ini adalah varibel yang akan digunakan dalam proses .

- $Z_{min}$ adalah nilai *grayscale* yang paling kecil setelah diurutkan pada piksel-piksel *window*
- $Z_{max}$adalah nilai *grayscale* yang paling besar setelah diurutkan pada piksel-piksel *window*
- $Z_{med}$adalah nilai tengah dari nilai *grayscale* setelah diurutkan pada piksel-piksel *window*
- $Z_{xy}$adalah nilai *grayscale* pada posisi baris x, dan kolom y (nilai ini biasanya berada pada posisi tengah array 1 dimensi)
- $A_1$ $A_2$ $B_1$ $B_2$digunakan sebagai variabel pembantu sementara
- n adalah ukuran baris atau kolom *window*
- $n_{max}$ adalah ukuran maksimal *window* (dalam hal ini ditentukan nilainya sama dengan 5×5)

3.2.1.1. *Noise* berukuran kecil

*Windowing* dimulai dengan ukuran minimal n=3 sehingga tampak nilai pixel pada (Gambar 3.4).

| | | |
|---|---|---|
| 156 | 68 | 12 |
| 31 | 0 | 4 |
| 5 | 8 | 17 |

Gambar 3. 4 Window awal berukuran 3x3

Kemudian *window* tersebut diubah menjadi array satu dimensi sehingga menjadi seperti (Gambar 3.5). Kemudian array tersebut diurutkan untuk mendapatkan nilai minimum, median, dan maksimum seperti pada (Gambar 3.6).

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 156 | 68 | 12 | 31 | 0 | 4 | 5 | 8 | 17 |

Gambar 3. 5 Array 1 dimensi dariwindow 3×3

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0 | 4 | 5 | 8 | 12 | 17 | 31 | 68 | 156 |

Gambar 3. 6 Array 1 dimensi yang terurut

a. Inisialisasikan variabel :

$Z_{min}$ =0

$Z_{med}$ =12

$Z_{max}$ =156

$Z_{xy}$ = 0

$n_{max}$=5

b. Karena memenuhi Zmin < Zmed < Zmax , maka ditanyakan apakah Zmin < Zxy < $Z_{max}$ .

c. Karena tidak memenuhi, maka Zxy dikatakan sebagai *noise* sehingga $Z_{xy}$ = 12 . Kemudian nilai array sebelumnya berubah menjadi array seperti (Gambar 3.7).

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 156 | 68 | 12 | 31 | 12 | 4 | 5 | 8 | 17 |

Gambar 3. 7 Array 1 dimensi yang sudah diproses

3.2.1.2. *Noise* berukuran besar

*Windowing* dimulai dengan ukuran minimal n=3 sehingga tampak nilai pixel pada (Gambar 3.8). Kemudian *window* tersebut diubah menjadi array satu dimensi sehingga menjadi seperti (Gambar 3.9). Kemudian array tersebut diurutkan untuk mendapatkan nilai minimum, median, dan maksimum seperti pada (Gambar 3.10).

| | | |
|---|---|---|
| 185 | 185 | 186 |
| 0 | 0 | 0 |
| 0 | 0 | 0 |

Gambar 3. 8 Window awal yang berukuran 3×3

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 185 | 185 | 186 | 0 | 0 | 0 | 0 | 0 | 0 |

Gambar 3. 9 Array 1 dimensi dari window 3×3

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 185 | 185 | 186 |

Gambar 3. 10 Array 1 dimensi yang terurut

a. Inisialisasikan variabel :

$Z_{min}$ =0

$Z_{med}$ =0

$Z_{max}$ =186

$Z_{xy}$ = 0

$n_{max}$=5

b. Karena tidak memenuhi $Z_{min} < Z_{med} < Z_{max}$ , maka ukuran *window* diperbesar

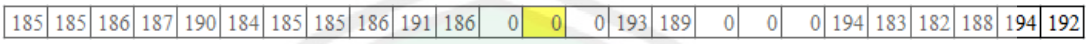
n = +2 → n = 5

maka ukuran *window* menjadi 5×5 seperti pada (Gambar 3.11)

| | | | | |
|---|---|---|---|---|
| 185 | 185 | 186 | 187 | 190 |
| 184 | 185 | 185 | 186 | 191 |
| 186 | 0 | 0 | 0 | 193 |
| 189 | 0 | 0 | 0 | 194 |
| 183 | 182 | 188 | 194 | 192 |

Gambar 3. 11 Window kedua yang berukuran 5×5

Seperti pada langkah sebelumnya, *window* tersebut diubah menjadi array satu dimensi sehingga menjadi seperti (Gambar 3.12). Kemudian array tersebut diurutkan untuk mendapatkan nilai minimum, median, dan maksimum seperti pada (Gambar 3.13).




Gambar 3. 12 Array 1 dimensi dari window 5×5


Gambar 3. 13 Array 1 dimensi yang terurut

c. Inisialisasikan variabel :

$Z_{min} = 0$

$Z_{med} = 185$

$Z_{max} = 194$

$Z_{xy} = 0$

$n_{max} = 5$

d. Karena memenuhi $Z_{min} < Z_{med} < Z_{max}$ , maka ditanyakan apakah $Z_{min} < Z_{xy} < Z_{max}$ .

e. Karena tidak memenuhi, maka Zxy dikatakan sebagai *noise* sehingga $Z_{xy} = 185$ . Kemudian nilai array sebelumnya berubah menjadi array seperti pada (Gambar 3.14).


Gambar 3. 14 Array 1 dimensi yang sudah terfilter

### 3.2.1.3. Tidak terdeteksi*noise*

*Windowing* dimulai dengan ukuran minimal n=3 sehingga tampak nilai piksel pada (Gambar 3.15).

| | | |
|---|---|---|
| 185 | 187 | 191 |
| 186 | 188 | 191 |
| 185 | 188 | 189 |

Gambar 3. 15 Window yang berukuran 3×3

Kemudian *window* tersebut diubah menjadi array satu dimensi sehingga menjadi seperti (Gambar 3.16). Kemudian array tersebut diurutkan untuk mendapatkan nilai minimum, median, dan maksimum seperti pada (Gambar 3.17).

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 185 | 187 | 191 | 186 | 188 | 191 | 185 | 188 | 189 |

Gambar 3. 16 Array 1 dimensi dari window 3×3

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 185 | 185 | 186 | 187 | 188 | 188 | 189 | 191 | 191 |

Gambar 3. 17 Array 1 dimensi yang terurut

a. Inisialisasikan variabel :

$Z_{min}$ =185

$Z_{med}$ =188

$Z_{max}$ =191

$Z_{xy}$ = 188

$n_{max}$=5

b. Karena memenuhi $Z_{min} < Z_{med} < Z_{max}$ , maka ditanyakan apakah $Z_{min} < Z_{xy} < Z_{max}$ .

c. Karena memenuhi, maka Zxy dianggap bukan sebagai *noise* sehingga $Z_{xy}$ = 188 dan nilai array sebelumnya tidak berubah seperti yang ditunjukkan pada (Gambar 3.18).

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 185 | 187 | 191 | 186 | 188 | 191 | 185 | 188 | 189 |

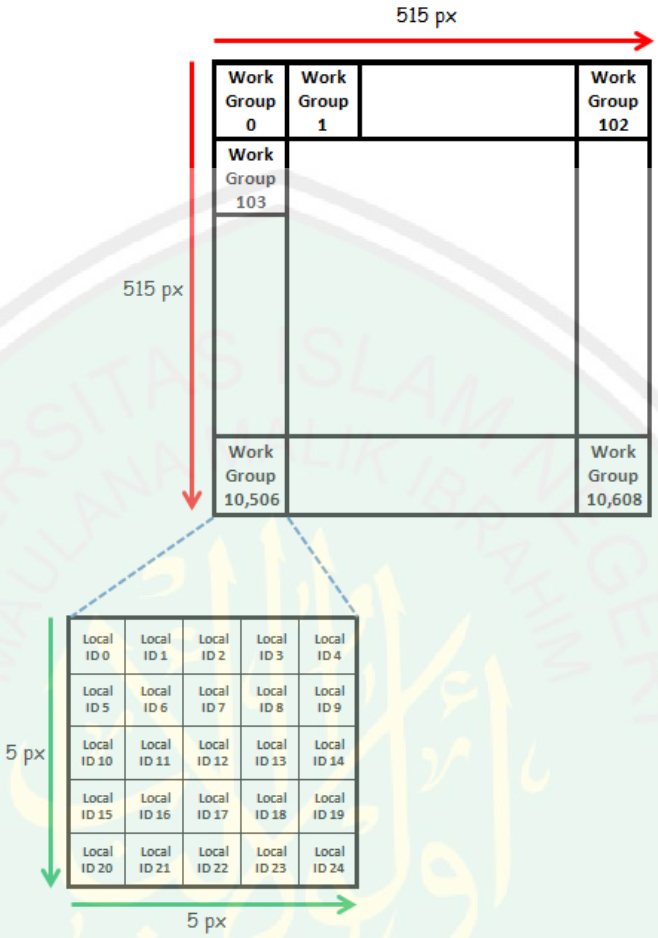
Gambar 3. 18 Array 1 dimensi yang sudah diproses

### 3.2.2. Pengelolaan thread untuk pemetaan data

Dalam penelitian ini dikembangkan metode *Adaptive Median Filter* yang diimplementasikan ke dalam pemrograman GPU. Pemrograman GPUsendiri menggunakan teknik untuk memecah struktur data vektor, matriks, dan bidang pengolahan citra. Teknik memecah data merupakan dasar dalam pemrograman GPUdimana satu tugas dengan banyak data dijalankan dalam satu kali pemrosesan.

Dalam penelitian ini, pemecahan data yang dilakukan adalah sebuah citra yang merupakan matriks dalam jumlah tertentu dengan memiliki empat komponen warna (*Red, Green, Blue, Alpha*). Dalam pemecahan data citra adapun *work group* dan *work item* ditentukan seperti terlihat pada (gambar 3.19).Data citra yang digunakan berukuran 512×512 px. Untuk dapat membagi data citra menjadi blok-blok eksekusi, perlu ditentukan ukuran keseluruhan (*global work size*) dengan tepat. Data citra yang berukuran 512×512 jika per blok (*work group*) dibagi 5×5, maka *global work size* ditentukan menjadi 515×515. Dan jika per *work group* dibagi 5×5 *work item*, maka work group berjumlah 103×103 work group. Dari pembagian tersebut, maka dapat ditentukan memori local (*local work size*) adalah 5×5.

Batasan jumlah work group pada setiap work item tergantung pada ukuran lokal memori. Kode kernel OpenCL harus dipastikan bahwa penggunaan memori lokal tidak keluar dari batas. Jika melewati batas yang diizinkan maka kinerja akan menurun. Untuk hampir semua perangkat AMD, memori lokal 32 K dapat dialokasikan.




Gambar 3. 19 Pembagian *work group* dan *work item* dalam pengelolaan *thread*

Setelah perintah-perintah OpenCL kernel dieksekusi pada *device* (GPU), memori yang ada pada *device* (GPU) dibaca kembali oleh host (CPU). Kemudian dilakukan ekseskusi program secara keseluruhan, dan objek-objek seperti memori, perintah, dan context pada OpenCL dan host buffer yang sudah dialokasikan di awal dibersihkan.

## 3.3. Implementasi

Implementasi metode *Adaptive Median Filter*dilakukan dengan komputasi CPU dan komputasi GPU. Sehingga dalam skripsi ini dibangun dua implementasi dalam pemrosesannya, yaitu alur yang menerangkan tentang implementasi pemrosesan pada GPU dan alur yang menerangkan tentang implementasi pemrosesan pada CPU. Pemrosesan kedua implementasi pada data citra uji dengan *salt and pepper noise* tersebut dibangun agar diketahui perbedaan waktu proses (*consuming time*) pada masing-masing implementasi. Berikut ini rincian proses kedua implementasi.

### 3.3.1. Implementasi dengan GPU

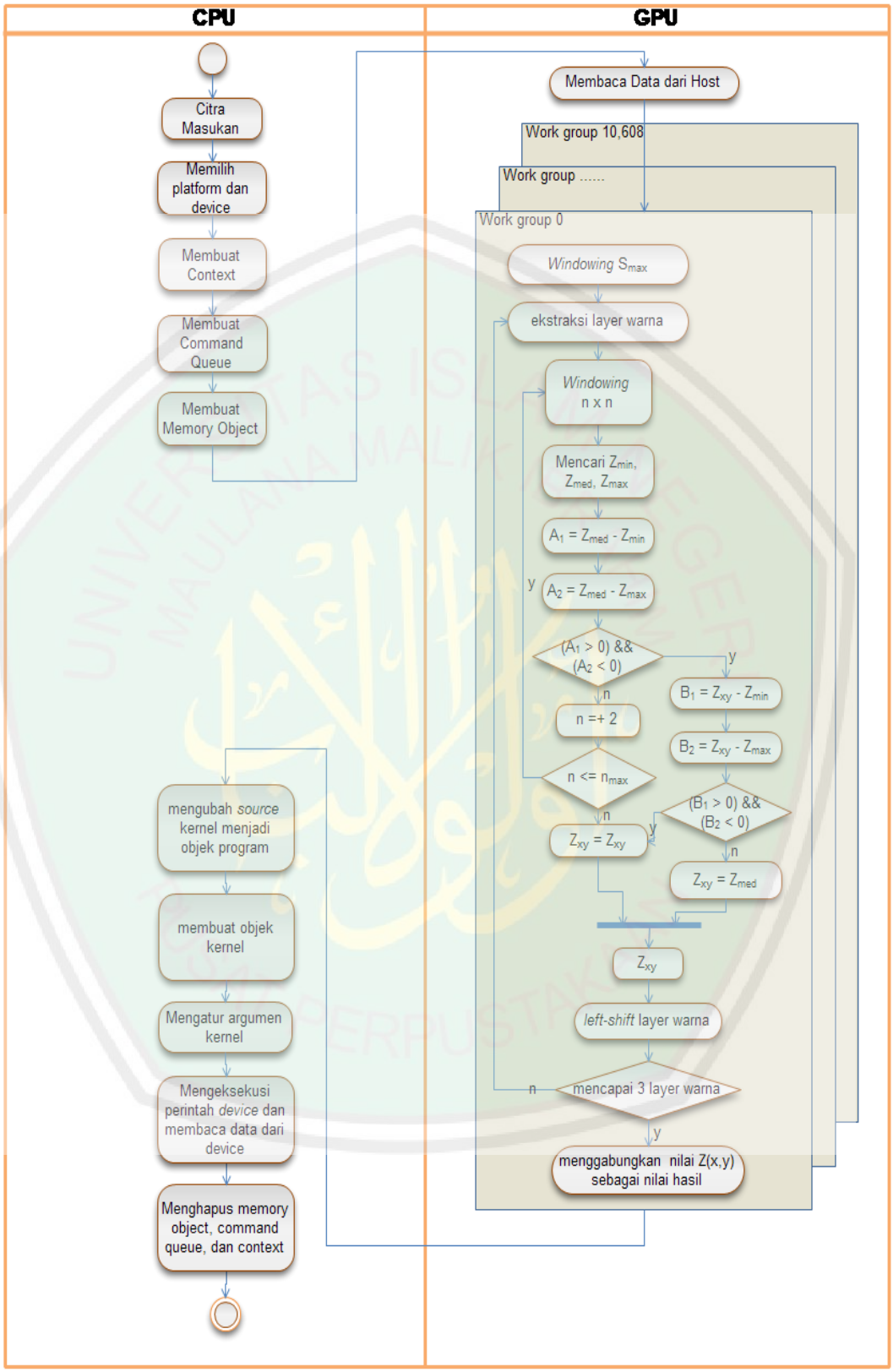
Implementasi dengan GPUterlihat pada *activitydiagram* (Gambar 3.20), proses filtering dilakukan pada GPU, namun tetap menggunakan bantuan host (CPU). Pada implementasi ini proses dilakukan pertama pada host (CPU), kemudian dilanjutkan pada device (GPU), dan terakhir dilakukan pada host (CPU). Berikut ini adalah tahapan proses Implementasi dengan GPU:

3.3.1.1. Ambil data gambar yang akan diolah.

Dari gambar tersebut, diambil nilai pixelnya untuk dialokasikan ke dalam memori host buffer (CPU) dan diinisialisasi.Listing program untuk implementasinya dapat dilihat pada (listing 3.1).

```
BufferedImage input = new BufferedImage(sizeX, sizeY,
        BufferedImage.TYPE_INT_RGB);
```

Listing 3. 1 kode mengambil nilai gambar




Gambar 3. 20 Activity Diagram Implementasi dengan GPU

3.3.1.2. Ambil informasi platform dan perangkat (device) yang tersedia.

Perintah untuk mendapatkan informasi tersebut adalah dengan `clGetPlatformIDs`dan `clGetDeviceIDs`. Dari daftar perangkat tersebut diatur platform (OpenCL) kemudian dipilih tipe perangkat yang akan dijalankan.Listing program untuk implementasinya dapat dilihat pada (listing 3.2).

```
//memperoleh jumlah device pada platform
int jumlahDevicesArray[] = new int[1];
clGetDeviceIDs(platform, tipeDevice, 0, null, jumlahDevicesArray);
int jumlahDevices = jumlahDevicesArray[0];

//memperoleh ID device
cl_device_id devices[] = new cl_device_id[jumlahDevices];
clGetDeviceIDs(platform, tipeDevice, jumlahDevices, devices, null);
cl_device_id device = devices[indexDevice];
```

Listing 3. 2 Setting platform dan perangkat

3.3.1.3. Membuat OpenCL context untuk perangkatyang telah didefinisikan.

Sebuah*context* mendefinisikan seluruh lingkungan OpenCL, termasuk perangkat, obyek program, kernel OpenCL, hingga *memory object*. Sebuah konteks dapat dikaitkan dengan beberapa perangkat atau dengan hanya satu perangkat. Antrian perintah dan kernel harus dari konteks OpenCL yang sama, tidak bisa dari konteks yang berbeda. Sebelum dapat membuat suatu konteks, dilakukan*queryruntime* OpenCL untuk menentukan vendor platform yang tersedia dalam sistem. Setelah vendor platform dipilih, inisialisasi pelaksanaan OpenCL untuk membuat konteks. Sebuah konteks dapat memiliki sejumlah perangkat terkait, yang dapat berupa CPU atau GPU atau keduanya. Konteks di OpenCL

diinisialisasi menggunakan `clCreateContext`.Listing program untuk implementasinya dapat dilihat pada (listing 3.3).

```
//membuat context untuk device yang dipilih
cl_context context = clCreateContext(contextProperties, 1, new
            cl_device_id[]{device}, null, null, null);
```

Listing 3. 3 Setting context

3.3.1.4. Membuat antrian perintah (*command queue*).

Antrian perintah dibuat untuk setiap penggunaan perangkat OpenCL untuk eksekusi kernel. Sebuah Opencl kernel dapat memiliki beberapa antrian perintah untuk tugas yang berbeda dalam aplikasi. Sehingga dapat menjalankan tugas-tugas secara independen pada antrian perintah yang berbeda.Antrian perintah dapat dibuat dengan inisialisasi `cl_command_queue`.Untuk implementasinya dapat dilihat pada (listing3.4).

```
//membuat command-queue untuk device yang dipilih
cl_command_queue cQueue = clCreateCommandQueue(context, device, 0,
null);
```

Listing 3. 4 Setting *command-queue*

Sebuah perintah merupakan sebuah transfer data, atau perintah eksekusi kernel atau hambatan (barrier) dalam antrian perintah. Host akan mengantrikan perintah tersebut pada antrian perintah. Setiap perintah atau tugas dihubungkan dengan sebuah OpenCL event. Event ini dapat digunakan sebagai mekanisme sinkronisasi untuk mengkoordinasikan eksekusi antara host dan perangkat.

3.3.1.5. Membuat *memoryobject* pada perangkat.

Adapunobjek memori yang akan dibuat adalah objek *buffer (buffer object)*. Data uji citra sebagai *memoryobject* berukuran 512×512 px. Memory object yang

dibuat meliputi input dan output. Objek input berisi objek gambar yang akan diproses dan objek output disediakan untuk menampung output yang sudah terproses. Untuk membuat memori objek digunakan fungsi `clCreateBuffer` dengan argumen meliputi *context* yang disediakan ; `cl_mem_flags` sebagai variabel untuk spesifikasi tipe dari memori ; ukuran buffer yang dialokasikan dalam bytes ; pointer untuk menempatkan data dari/ke host (pointer ini ukurannya harus sama atau lebih besar dari ukuran yang dialokasikan) ; kode error jika ada. Listing program untuk implementasinya dapat dilihat pada (listing 3.5).

```
//membuat memory object untuk gambar input dan output
DataBufferInt dataBufferSrc = (DataBufferInt)
        src.getRaster().getDataBuffer();
int dataSrc[] = dataBufferSrc.getData();
inputImageMem = clCreateBuffer(context, CL_MEM_READ_ONLY |
        CL_MEM_USE_HOST_PTR, dataSrc.length * Sizeof.cl_uint,
        Pointer.to(dataSrc), null);
outputImageMem = clCreateBuffer(context, CL_MEM_WRITE_ONLY,
        imageSizeX * imageSizeY * Sizeof.cl_uint, null, null);
```

Listing 3. 5 membuat memory object (input & output)

3.3.1.6. Membuat perintah-perintah OpenCL (Opencl kernel).

Kernel ini berisi perintah pemrosesan filter dengan metode Adaptive Median. Berikut ini adalah langkah-langkah untuk menghapus noise dengan metode *Adaptive Median Filter* pada pemrosesan GPU:

3.3.1.6.1. Proses *Adaptive Median Filter* diawali dengan menentukan berapa nilai perbesaran window maksimal untuk proses filter. Dalam skripsi ini didefinisikan nilai window maksimal adalah 5×5. Kemudian, dari nilai

tersebut dapat didefinisikan juga besarnya nilai *work group* yaitu 5×5 untuk dapat diproses dengan GPU.

3.3.1.6.2. Buat perulangan untuk memproses nilai tiap layer warna yaitu Merah, Hijau, Biru.

3.3.1.6.3. Buat windowing untuk diproses. Inisialisasi window awal yaitu 3×3.

3.3.1.6.4. Kemudian dari nilai windowing tersebut, dilakukan pengurutan nilai, dan diambil nilai piksel minimal, nilai piksel tengah (median), nilai piksel maksimal, dan nilai pada posisi (x,y) sebelum diurutkan. Untuk mengatasi subproses ini, dibuat *method* untuk mengambil nilainya yaitu *method*`NilaiMMM`.

Adapun argumen yang dibutuhkan pada proses ini yaitu pointer piksel, dan ukuran window. Pointer piksel berisi nilai-nilai piksel yang akan diambil empat nilainya dengan tipe data integer. Nilai-nilai piksel didapatkan dari alamat yang mengarah pada pointer yang disediakan. Sedangkan ukuran window menerangkan window yang akan diproses. *Method* ini merupakan method dengan tipe data vektor sehingga didefinisikan dengan `int4`. Tipe data vektor int4 menampung empat nilai integer. Sehingga nilai pengembalian memiliki empat nilai integer. Nilai yang pertama diisi dengan nilai pengembalian nilai minimal; nilai tengah; nilai maksimal; serta nilai xy.

3.3.1.6.5. Selanjutnya dilakukan pengondisian untuk metode *Adaptive Median Filter*. Adapun notasi yang dipakai meliputi:

a. $Z_{min}$ adalah nilai *grayscale* yang paling kecil setelah diurutkan pada piksel-piksel *window*

b. $Z_{max}$adalah nilai *grayscale* yang paling besar setelah diurutkan pada piksel-piksel *window*

c. $Z_{med}$adalah nilai tengah dari nilai *grayscale* setelah diurutkan pada piksel-piksel *window*

d. $Z_{xy}$adalah nilai *grayscale* pada posisi baris x, dan kolom y (nilai ini biasanya berada pada posisi tengah array 1 dimensi)

e. $A_1$ $A_2$ $B_1$ $B_2$digunakan sebagai variabel pembantu sementara

f. n adalah ukuran baris atau kolom *window*

g. $n_{max}$adalah ukuran maksimal *window*

Kondisi yang pertama adalah apakah memenuhi nilai dari $Z_{min} < Z_{med} < Z_{max}$ , jika memenuhi maka dilakukan kondisi kedua yaitu apakah memenuhi nilai dari $Z_{min} < Z_{xy} < Z_{max}$, jika memenuhi pula, maka dianggap bukan noise sehingga nilai tidak diubah. Jika tidak memenuhi maka dianggap sebagai noise dan dilakukan perubahan nilai di koordinat (x,y) dengan nilai mediannya.

Namun jika kondisi awal ($Z_{min} < Z_{med} < Z_{max}$) tidak terpenuhi, maka akan dilakukan pembesaran window (jika window kurang dari window maksimal). Setelah dilakukan pembesaran window, dilakukan proses pengondisian mulai dari awal lagi dengan ukuran selanjutnya (window+2).

3.3.1.6.6. Setelah didapatkan nilai untuk $Z_{xy}$ , layer warna berganti untuk layer selanjutnya hingga mencapai tiga layer warna.

3.3.1.7. Source kernel yang sudah dibuat dijadikan objek program dengan menggunakan fungsi `clCreateProgramWithSource`.

Objek program dibuat untuk device yang berkaitan dengan *context* Opencl. Setelah objek program baru diciptakan untuk sebuah *context*,baik menggunakan biner atau *source* OpenCL, langkah berikutnya adalah membangun program. Program kernel perlu untuk dibangun dan dihubungkan pada saat runtime. Untuk membangun program, fungsi yang digunakan yaitu `clBuildProgram`. Tahap pembangunan (*build*) ini melibatkan kompilasi source code karena program dibuat menggunakan fungsi `clCreateProgramWithSource`. Jika program dibuat menggunakan fungsi `clCreateProgramWithBinary` maka hanya langkah menghubungkan runtime yang dijalankan. listing program untuk implementasinya dapat dilihat pada (listing 3.6).

```
cl_program program = clCreateProgramWithSource(context, 1, new
        String[]{sumber}, null, null);
String compileOption = "-cl-mad-enable";
clBuildProgram(program, 0, null, compileOption, null, null);
```

Listing 3. 6kode untuk build

3.3.1.8. Setelah proses *building* program, tahap selanjutnya adalah membuat objek kernel dengan fungsi `clCreateKernel`.

Setiap program adalah kumpulan kernel. Objek program dapat dikatakan sebagai *library* kernel-kernel. Sebuah kernel ketika diantrekan pada antrian perintah, *runtime* OpenCL menghasilkan biner untuk eksekusi pada perangkat. Jika kernel lebih dari satu, maka setiap kernel dijalankan pada perangkat yang berbeda. Dalam hal ini kernel yang dibuat hanya satu kernel. Sebuah objek kernel adalah enkapsulasi untuk satu kesatuan yang dieksekusi secara paralel. Objek kernel digunakan sebagai jalan untuk melewatkan argumen menggunakan API

clSetKernelArg, sebelum *running* kernel menggunakan API clEnqueueNDRangeKernel.

```
clKernel = clCreateKernel(program, "MedianFilter", null);
```

Listing 3. 7 kode membuat ojek kernel

3.3.1.9. Mengatur argumen kernel sebelum pengeksekusian kernel.

Argumen kernel yang diatur meliputi **objek gambar input** dan **output** serta **ukuran gambar** diberi pointer ke *device* dari host dengan menggunakan fungsi clSetKernelArg. Adapun argumen yang ada pada fungsi clSetKernelArg meliputi objek kernel yang argumennya akan diatur ; index dari argumen dimulai dari indeks ke 0 hingga akhir argumen ; spesifikasi ukuran dari arg_value ; arg_valuesebagai pointer untuk data. Listing program untuk implementasinya dapat dilihat pada (listing 3.8).Objek cl_kernel yang telah dibuat dan telah diatur argumen, kemudian dilakukaneksekusi pada perangkat.

Untuk eksekusi objek kernel, digunakan fungsi clEnqueueNDRangeKerneldimana fungsi tersebut untuk menyebarkan opencl kernel ke perangkat dan mendefinisikan berapa banyak work item yang dibuat untuk mengeksekusi kernel (global_work_size) dan jumlah work item dalam setiap *work group*(local_work_size). Adapun fungsi clEnqueueNDRangeKernelmeliputi command_queue; objek Opencl Kernel; dimensi dari NDRange; global_work_offset yang juga digunakan untuk menghitung global_id dari work item, jika argumen ini bernilai null, maka nilai default adalah 0; global_work_size yang mendefinisikan global work item dalam setiap dimensi; local_work_size untuk mndefinisikan local work item dalam setiap dimensi; event_wait_list; event.

```
//menentukan work group dan argumen pada opencl kernel code
long localWorkSize[] = new long[2];
localWorkSize[0] = 5;
localWorkSize[1] = 5;

long globalWorkSize[] = new long[2];
globalWorkSize[0] = round(localWorkSize[0], imageSizeX);
globalWorkSize[1] = round(localWorkSize[1], imageSizeY);

int imageSize[] = new int[]{imageSizeX, imageSizeY};

clSetKernelArg(clKernel, 0, Sizeof.cl_mem,
        Pointer.to(inputImageMem));
clSetKernelArg(clKernel, 1, Sizeof.cl_mem,
        Pointer.to(outputImageMem));
clSetKernelArg(clKernel, 2, Sizeof.cl_int2,
        Pointer.to(imageSize));
```

Listing 3. 8 mengatur argumen kernel

Selanjutnya proses yang dilakukan adalah membaca kembali memori dari device (GPU) ke host buffer (CPU) menggunakan fungsi `clEnqueueReadBuffer` dengan argumen yang berisi `command_queue` ; objek buffer `cl_mem` yang akan dibaca dan akan ditulis pada pointer ; `blocking_read; offset` ; total bytes yang dibaca dari device yang dipointerkan oleh buffer ; host memory pointer dari dimana data dibaca ; `event_wait_list` ; `event`. Buffer yang dibaca dari perangkat ke host adalah memori buffer untuk output dari memori input yang sudah diproses pada kernel code.

```
clEnqueueNDRangeKernel(queue, clKernel, 2, null, globalWorkSize,
        localWorkSize, 0, null, null);
```

```
//membaca pixel data ke bufferedimage
DataBufferInt dataBufferDest = (DataBufferInt)
        dest.getRaster().getDataBuffer();
int dataDest[] = dataBufferDest.getData();
clEnqueueReadBuffer(queue, outputImageMem, CL_TRUE, 0,
        dataDest.length * Sizeof.cl_uint, Pointer.to(dataDest), 0,
        null, null);
```

Listing 3. 9 membaca memori dari device ke host

3.3.1.10. Kemudian bersihkan objek-objek seperti memori, perintah, dan context pada OpenCL dan host buffer yang sudah dialokasikan di awal.

Setiapobjek program seharusnya dilepas (*released*) dari penyimpanan Opencl setelah digunakan. Untuk melepas objek program, fungsi yang digunakan adalah `clRetainProgram`.Listing program untuk implementasinya dapat dilihat pada (listing 3.10) dan (listing 3.11).

```
clRetainProgram(program);
```

Listing 3. 10 membersihkan memori objek program

Sama halnya dengan objek program, objek kernel juga seharusnya dilepas (*released*) setelah proses selesai. Berbeda dengan objek program, objek kernel menggunakan fungsi `clReleaseKernel`. Adapun untuk *release command queue* menggunakan fungsi `clReleaseCommandQueue`dan untuk *release context* menggunakan fungsi `clReleaseContext`.

```
void shutdown() {
    clReleaseKernel(clKernel);
    clReleaseCommandQueue(queue);
    clReleaseContext(context);
```

```
}
```

Listing 3. 11 membersihkan memori untuk objek kernel, command-queue, dan context

### 3.3.2. Implementasi dengan CPU

Implementasi dengan CPU terlihat pada Gambar 3.21, seluruh proses filtering dilakukan pada CPU. Implementasi listing kode dengan CPU dapat dilihat pada lampiran. Proses ini diawali dengan mendapatkan nilai RGB pada citra masukan. Selanjutnya dilakukan proses windowing dan proses *Adaptive Median Filter*. Dari hasil filtering, didapatkan citra keluaran dari proses pada CPU.Berikut ini adalah tahap-tahap implementasi metode *Adaptive Median Filter*dengan menggunakan pemrograman CPU:

3.3.2.1. Buat perulangan untuk pemrosesan setiap piksel sebanyak panjang×lebar gambar input (512×512).

Sehingga di dalam perulangan tersebut dilakukan proses *Adaptive Median Filter*.

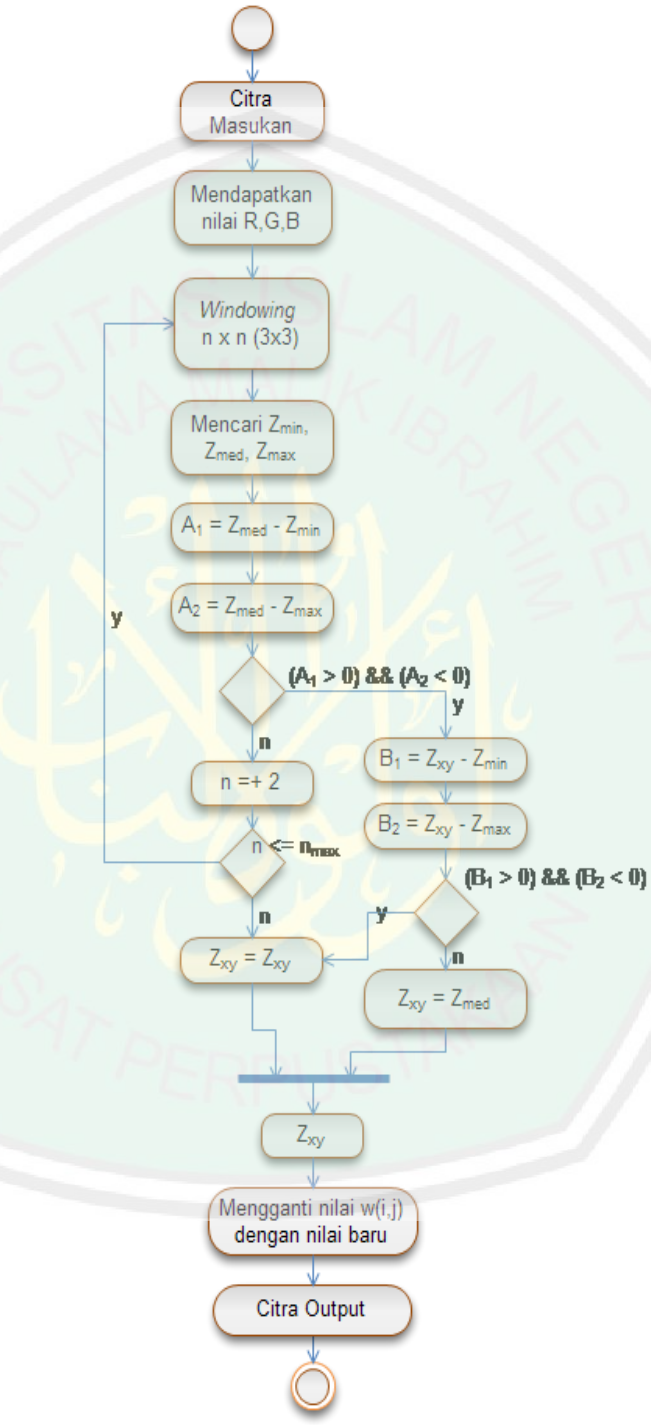
3.3.2.2. Proses*Adaptive Median Filter* Diawali dengan windowing matriks dengan ukuran awal 3×3.

Dari matriks tersebut, didapatkan nilai piksel minimal, median, dan maksimal.

3.3.2.3. Proses*Adaptive Median Filter* selanjutnya terdapat dua level untuk membangunnya, yaitu level A dan level B.

Level A digunakan untuk mengetahui seberapa besar wilayah ketimpangan (yang diasumsikan sebagai *noise*) yang terdapat pada *window*. Level B digunakan

untuk memastikan apakah Zxy terdapat ketimpangan nilai (*noise*). Berikut ini adalah langkah level A dan level B.




Gambar 3. 21 Activity Diagram Implementasi dengan CPU

3.3.2.3.1. Level A :

a. Jika $Z_{min} < Z_{med} < Z_{max}$, maka $Z_{med}$ diidentifikasi sebagai piksel yang bukan *noise* sehingga dilanjutkan ke level B, jika tidak maka ukuran *window* diperbesar dan proses selanjutnya dijalankan.

b. Jika ukuran *window* kurang dari $n_{max}$, maka level A diulangi dengan *window* yang sudah diperbesar, Jika ukuran sudah melebihi $n_{max}$ maka nilai Zxy dihasilkan sebagai output (tidak terfilter).

3.3.2.3.2. Level B :

a. Jika $Z_{min} < Z_{xy} < Z_{max}$, maka $Z_{xy}$ bukan *noise* sehingga nilai $Z_{xy}$ dihasilkan sebagai output (tidak terfilter). Jika tidak maka nilai $Z_{med}$ dihasilkan sebagai output (terfilter).

# BAB IV

# UJI COBA DAN PEMBAHASAN

Untuk mengetahui hasil implementasi metode *Adaptive Median Filter* pada komputasi GPUmenggunakan OpenCL dalam penghapusan noise dilakukan pengujian. Dari uji coba implementasi, maka didapatkan hasil dari implementasi tersebut, serta didapatkan hasil lain berupa waktu komputasi pemrosesan CPU dan GPU. Implementasi komputasi GPUjuga diintegrasikan dengan Al-Quran.

## 4.1. Data Uji

Data uji pada penelitian ini yaitu data citra digital yang berukuran 512×512. Macam data yang diujikan dibedakan menjadi data citra indoor yang diberi *salt and pepper noise* 10% sejumlah 40 buah; citra indoor dengan *salt and pepper noise* 20% sejumlah 40 buah; citra outdoor dengan *salt and pepper noise* 10% sejumlah 40 buah; citra outdoor dengan *salt and pepper noise* 20% sejumlah 40 buah. Data citra uji dapat dilihat pada lampiran.

## 4.2. Langkah Pengujian

Implementasi metode *Adaptive Median Filter* yang sudah dilakukan, diuji coba untuk didapatkan data efisiensinya. Data efisiensi didapatkan karena adanya perbandingan antara pemrosesan CPU dan GPU. Sehingga pengujian dilakukan pada CPU dan GPU. Pengujian dilakukan pada setiap data uji seperti yang dijelaskan pada (subbab 4.1). Adapun langkah pengujian implementasi metode Adaptive Median Filter menggunakan komputasi CPU dan GPU untuk

penghapusan *noise*. Langkah pengujian secara ringkas dapat dilihat pada (gambar 4.1).

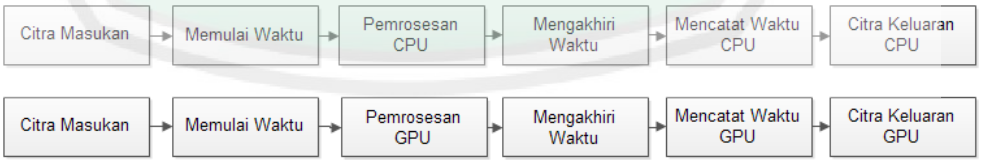
Alur pengujian untuk pemrosesan CPU dimulai dari mengambil citra masukan berupa data uji sesuai pada (subbab 4.1). Salah satu sumber data citra yang akan diproses dapat dilihat pada (gambar 4.2). Kemudian dilakukan proses penghapusan *noise* pada citra masukan menggunakan pemrosesan CPU. Waktu dijalankansebelum pemrosesandan diakhiri setelah pemrosesan berakhir seperti yang terlihat dalam (listing 4.1). Kemudian dihasilkan data waktu untuk direkapitulasi. Setelah penghapusan *noise*, data citra yang sudah terproses didapatkan sebagai data keluaran seperti pada (gambar 4.3).

```
before = System.nanoTime();
    bufferGbrOutputJV = amf.filter(bufferGbrInput, bufferGbrOutputJV);
    after = System.nanoTime();
```

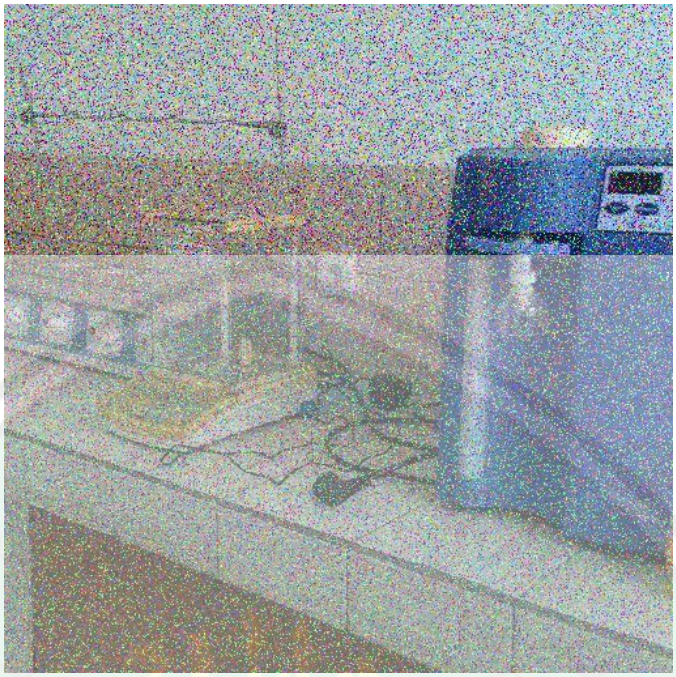
Listing 4. 1 Kode mendapatkan waktu proses CPU

```
before = System.nanoTime();
    bufferGbrOutputCL = jop.filter(bufferGbrInput, bufferGbrOutputCL);
    after = System.nanoTime();
```

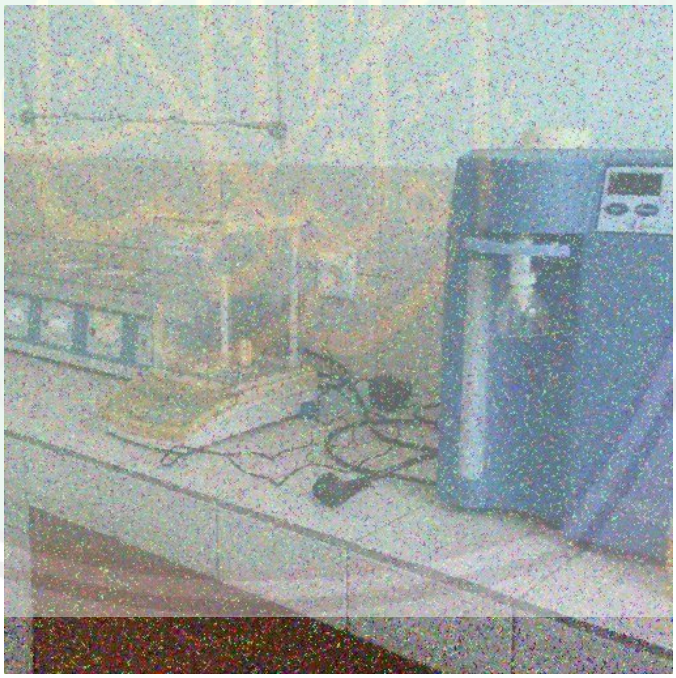
Listing 4. 2 Kode mendapatkan waktu proses GPU




Gambar 4. 1 Langkah pengujian



Gambar 4. 2 Data citra uji



Gambar 4. 3 Data hasil keluaran komputasi CPU

Data citra uji juga digunakan sebagai data masukan untuk pemrosesan GPU. Data tersebut dilakukan proses penghapusan noise dengan pemrograman lintas platform CPU dan GPU. Waktu dijalankansebelum pemrosesan dan diakhiri setelah pemrosesan berakhir seperti yang terlihat dalam (listing 4.2). Kemudian

dihasilkan data waktu untuk direkapitulasi. Setelah penghapusan *noise*, data citra yang sudah terproses didapatkan sebagai data keluaran seperti pada (gambar 4.4).

Gambar 4. 4 Data hasil keluaran komputasi GPU

### 4.3. Hasil Uji Coba

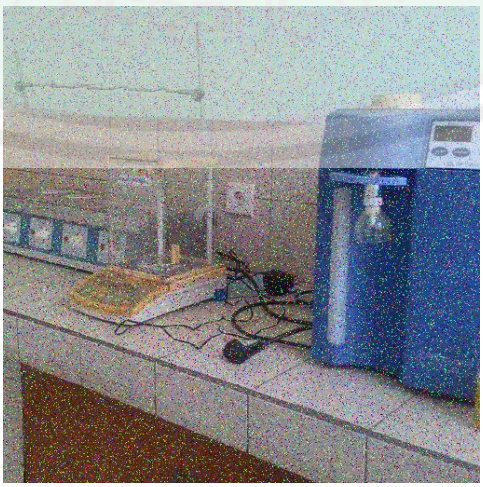
Dari pengujian yang telah dilakukan, didapatkan hasil pemrosesan komputasi CPU dan GPU berupa citra dan waktu pemrosesan. Adapun citra output selanjutnya akan dibandingkan dengan citra asli untuk mengetahui tingkat keberhasilan dari metode *Adaptive Median Filter* pada pemrograman GPU. Pembandingan tersebut dilakukan dengan cara mengurangkan gambar asli sebagai gambar referensi dengan gambar output hasil implementasi metode *Adaptive Median Filter* pada GPU. Untuk mendapatkan hasil pembandingan tersebut, yang perlu dilakukan yaitu mengurangkan rata-rata hasil dari penjumlahan tiap layer warna yang dibagi dengan jumlah pixel tiap layer warna pada gambar asli dengan gambar output implementasi GPU. Sehingga dapat dituliskan :

$$Layer\ (i) = \frac{\sum nilai\ pixel\ asli}{\sum pixel\ asli} - \frac{\sum nilai\ pixel\ output}{\sum pixel\ output}$$

Hasil terbaik setelah dilakukan pembandingan yaitu bernilai 0 pada setiap layer. Hal itu menggambarkan efektifitas metode *Adaptive Median Filter* dalam penghapusan noise 100%. Hasil pengurangan noise yang telah dilakukan denganmetode *Adaptive Median Filter*dibahas berikutnya, begitupun juga dengan penjelasan mengenai hasil waktu komputasi.

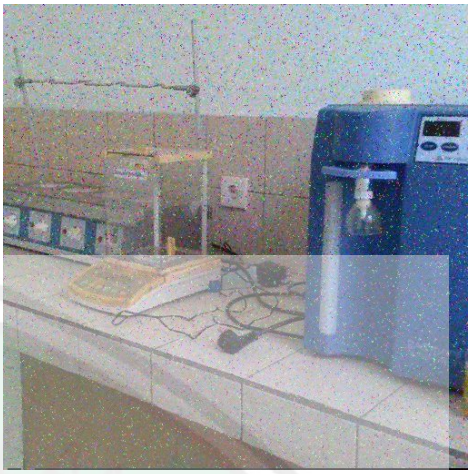
### 4.3.1. Hasil Implementasi Metode *Adaptive Median Filter*

Berdasarkan golongan data citra uji, berikut hasil implementasi dari metode *Adaptive Median Filter*. Citra pada (gambar 4.6) menunjukkan citra hasil untuk data uji pada data citra indoor dengan *salt and pepper noise*10%. Citra tersebut dihasilkan dari komputasi CPU, sedangkan (gambar 4.7) menunjukkan citra hasil dari komputasi GPU. Hasil kedua citra tersebut dihasilkan dari data citra uji pada (gambar 4.5). Waktu yang diperlukan untuk komputasi CPU adalah 1462,61 ms sedangkan untuk komputasi GPUdiperlukan waktu 864,35 ms.



Gambar 4. 5. Data citra uji *indoor salt and pepper noise*10%

Gambar 4. 6. Data citra hasil komputasi CPU



Gambar 4. 7. Data citra hasil komputasi GPU

Pengurangan noise yang dapat dilakukan dengan metode *Adaptive Median Filter*didapatkan kisaran nilainya antara 5 hingga 10 pada setiap layer data citra indoor dengan *salt and pepper noise* 10% seperti yang terlihat pada (tabel 4.1).

Tabel 4. 1 Hasil Pengurangan Gambar Input Indoor dengan Gambar Hasil pada Salt & Pepper noise 10%

| No. | Nama Data | Red | Green | Blue |
|---|---|---|---|---|
| 1 | sp A1 | 6,7386 | 5,7429 | 6,7163 |
| 2 | sp A2 | 6,6965 | 5,7591 | 6,8666 |
| 3 | sp A3 | 7,0094 | 6,2064 | 7,8973 |
| 4 | sp A4 | 6,7983 | 5,8813 | 6,7559 |
| 5 | sp A5 | 6,5920 | 5,9451 | 7,5208 |
| 6 | sp A6 | 6,2714 | 5,7524 | 7,2102 |
| 7 | sp A7 | 7,3803 | 6,8573 | 9,3651 |
| 8 | sp A8 | 5,7279 | 4,8008 | 5,7536 |
| 9 | sp A9 | 6,7359 | 6,1071 | 7,6674 |
| 10 | sp A10 | 7,1752 | 6,5081 | 8,3702 |
| 11 | sp A11 | 6,5521 | 5,9465 | 7,9476 |
| 12 | sp A12 | 6,3593 | 5,7664 | 7,0812 |
| 13 | sp A13 | 7,9843 | 6,9317 | 8,6618 |
| 14 | sp A14 | 7,9076 | 7,7635 | 8,6734 |
| 15 | sp A15 | 7,9481 | 6,7478 | 8,6756 |
| 16 | sp A16 | 6,3709 | 5,7223 | 7,5939 |
| 17 | sp A17 | 7,1126 | 6,0473 | 7,5266 |
| 18 | sp A18 | 6,7965 | 5,9214 | 7,5846 |

| No. | Nama Data | Red | Green | Blue |
|---|---|---|---|---|
| 19 | sp A19 | 6,5168 | 5,8981 | 7,3891 |
| 20 | sp A20 | 7,4415 | 6,6239 | 8,5115 |
| 21 | sp A21 | 6,9159 | 6,0616 | 7,6836 |
| 22 | sp A22 | 6,0449 | 5,9513 | 8,2094 |
| 23 | sp A23 | 6,6850 | 6,6234 | 10,0503 |
| 24 | sp A24 | 7,4405 | 6,6887 | 9,0950 |
| 25 | sp A25 | 6,7952 | 5,7563 | 7,6499 |
| 26 | sp A26 | 7,2534 | 6,6354 | 8,5098 |
| 27 | sp A27 | 6,7208 | 6,9938 | 8,4245 |
| 28 | sp A28 | 8,5486 | 7,2877 | 9,7645 |
| 29 | sp A29 | 7,2840 | 6,5533 | 7,8836 |
| 30 | sp A30 | 7,0025 | 5,9950 | 7,1267 |
| 31 | sp A31 | 7,7237 | 6,3631 | 7,3567 |
| 32 | sp A32 | 8,3196 | 5,9544 | 6,7273 |
| 33 | sp A33 | 6,2774 | 5,7193 | 7,2134 |
| 34 | sp A34 | 7,4498 | 6,1636 | 8,1143 |
| 35 | sp A35 | 7,3293 | 6,0710 | 8,1906 |
| 36 | sp A36 | 6,1545 | 5,4709 | 6,6809 |
| 37 | sp A37 | 6,4738 | 5,7987 | 7,4592 |
| 38 | sp A38 | 7,4546 | 6,0484 | 8,0921 |
| 39 | sp A39 | 7,3051 | 6,6250 | 8,9023 |
| 40 | sp A40 | 6,2416 | 5,5486 | 6,8932 |

Pengurangan noise yang dapat dilakukan dengan metode *Adaptive Median Filter* didapatkan kisaran nilai antara 8 hingga 18 pada setiap layer data citra indoor dengan *salt and pepper noise* 20% seperti yang terlihat pada (tabel 4.2).

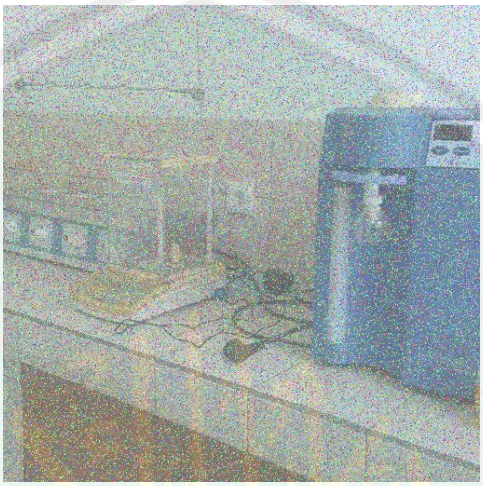
Tabel 4. 2 Hasil Pengurangan Gambar Input Indoor dengan Gambar Hasil pada Salt & Pepper noise 20%

| No. | Nama Data | Red | Green | Blue |
|---|---|---|---|---|
| 1 | sp A1 | 10,7757 | 9,4519 | 10,1881 |
| 2 | sp A2 | 10,5594 | 9,0901 | 10,4311 |
| 3 | sp A3 | 11,3166 | 10,2545 | 12,7167 |
| 4 | sp A4 | 11,1709 | 9,8547 | 10,9131 |
| 5 | sp A5 | 10,5009 | 9,9137 | 12,2123 |
| 6 | sp A6 | 10,2752 | 9,7897 | 11,8530 |

| No. | Nama Data | Red | Green | Blue |
|---|---|---|---|---|
| 7 | sp A7 | 12,7563 | 12,8968 | 16,8739 |
| 8 | sp A8 | 10,8401 | 11,2341 | 10,1699 |
| 9 | sp A9 | 10,7991 | 10,1813 | 12,3373 |
| 10 | sp A10 | 12,9385 | 11,8759 | 14,6016 |
| 11 | sp A11 | 10,8501 | 9,9511 | 13,1930 |
| 12 | sp A12 | 10,3473 | 9,7779 | 11,4557 |
| 13 | sp A13 | 14,0431 | 12,5196 | 15,3966 |
| 14 | sp A14 | 13,5965 | 14,2292 | 15,1795 |
| 15 | sp A15 | 13,6112 | 11,9841 | 15,0954 |
| 16 | sp A16 | 10,8543 | 12,7398 | 9,8316 |
| 17 | sp A17 | 10,8341 | 9,8169 | 11,8168 |
| 18 | sp A18 | 10,4550 | 9,5853 | 11,8829 |
| 19 | sp A19 | 11,7543 | 9,9423 | 12,8168 |
| 20 | sp A20 | 10,8465 | 9,8641 | 10,8158 |
| 21 | sp A21 | 11,3025 | 10,3961 | 12,5993 |
| 22 | sp A22 | 9,4739 | 10,1485 | 13,4892 |
| 23 | sp A23 | 10,7422 | 10,9140 | 16,9535 |
| 24 | sp A24 | 12,8487 | 11,9376 | 15,5535 |
| 25 | sp A25 | 11,0361 | 9,5230 | 12,0591 |
| 26 | sp A26 | 12,2338 | 11,7674 | 14,2947 |
| 27 | sp A27 | 11,3276 | 12,7780 | 14,6863 |
| 28 | sp A28 | 15,8123 | 13,6672 | 17,8967 |
| 29 | sp A29 | 12,9301 | 11,9502 | 14,0080 |
| 30 | sp A30 | 11,2282 | 9,9315 | 11,5571 |
| 31 | sp A31 | 12,8386 | 10,8263 | 9,8365 |
| 32 | sp A32 | 14,1079 | 9,9495 | 10,8387 |
| 33 | sp A33 | 10,3886 | 9,8361 | 11,5522 |
| 34 | sp A34 | 12,9165 | 11,1518 | 13,9467 |
| 35 | sp A35 | 12,8214 | 10,7340 | 14,0283 |
| 36 | sp A36 | 9,6321 | 8,7591 | 10,2841 |
| 37 | sp A37 | 11,2682 | 10,7648 | 9,2682 |
| 38 | sp A38 | 12,9706 | 10,9506 | 13,9785 |
| 39 | sp A39 | 12,8212 | 12,1184 | 15,1935 |
| 40 | sp A40 | 10,8972 | 9,8429 | 12,8624 |

Citra pada (gambar 4.9) menunjukkan citra hasil untuk data uji pada data citra indoor dengan *salt and pepper noise* 20%. Citra tersebut dihasilkan dari

komputasi CPU, sedangkan (gambar 4.10) menunjukkan citra hasil dari komputasi GPU. Hasil kedua citra tersebut dihasilkan dari data citra uji pada (gambar 4.8). Waktu yang diperlukan untuk komputasi CPU adalah 1396,56ms sedangkan untuk komputasi GPU diperlukan waktu 906,68 ms.



Gambar 4. 8. Data citra uji *indoor salt and pepper noise*20%

Gambar 4. 9. Data citra hasil komputasi CPU

Gambar 4. 10. Data citra hasil komputasi GPU

Citra pada (gambar 4.12) menunjukkan citra hasil untuk data uji pada data citra outdoor dengan *salt and pepper noise* 10%. Citra tersebut dihasilkan dari komputasi CPU, sedangkan (gambar 4.13) menunjukkan citra hasil dari komputasi GPU. Hasil kedua citra tersebut dihasilkan dari data citra uji pada

(gambar 4.11). Waktu yang diperlukan untuk komputasi CPU adalah 2235,08 ms sedangkan untuk komputasi GPU diperlukan waktu 798,74 ms.

Tabel 4. 3 Hasil Pengurangan Gambar Input Outdoor dengan Gambar Hasil pada Salt & Pepper noise 10%

| No. | Nama Data | Red | Green | Blue |
|---|---|---|---|---|
| 1 | sp A1 | 8,1511 | 7,1855 | 9,3640 |
| 2 | sp A2 | 8,0316 | 6,2344 | 7,5417 |
| 3 | sp A3 | 7,0001 | 5,6593 | 5,5832 |
| 4 | sp A4 | 6,6366 | 5,7396 | 6,3251 |
| 5 | sp A5 | 7,1622 | 5,8368 | 6,0300 |
| 6 | sp A6 | 8,4415 | 7,1222 | 8,6354 |
| 7 | sp A7 | 6,4944 | 6,0617 | 7,3592 |
| 8 | sp A8 | 7,8259 | 6,3412 | 7,4367 |
| 9 | sp A9 | 7,6080 | 6,1571 | 6,6988 |
| 10 | sp A10 | 7,6718 | 6,8353 | 7,6978 |
| 11 | sp A11 | 8,5625 | 6,6582 | 8,0458 |
| 12 | sp A12 | 8,0873 | 6,3453 | 7,1143 |
| 13 | sp A13 | 7,8254 | 6,4151 | 7,0448 |
| 14 | sp A14 | 8,1403 | 6,1929 | 5,8060 |
| 15 | sp A15 | 7,9686 | 6,8403 | 8,2436 |
| 16 | sp A16 | 8,4900 | 6,4644 | 6,4419 |
| 17 | sp A17 | 7,7300 | 6,0839 | 6,0353 |
| 18 | sp A18 | 6,5337 | 5,7601 | 9,8739 |
| 19 | sp A19 | 7,5135 | 5,4877 | 8,5214 |
| 20 | sp A20 | 8,1092 | 6,9094 | 8,7314 |
| 21 | sp A21 | 7,5342 | 6,4895 | 8,5609 |
| 22 | sp A22 | 6,6620 | 5,7352 | 7,3098 |
| 23 | sp A23 | 8,1276 | 6,7253 | 9,3415 |
| 24 | sp A24 | 7,0086 | 5,9100 | 8,6692 |
| 25 | sp A25 | 8,8915 | 7,5964 | 9,7248 |
| 26 | sp A26 | 7,0666 | 5,8092 | 7,9486 |
| 27 | sp A27 | 6,3139 | 5,6095 | 6,8730 |
| 28 | sp A28 | 7,3842 | 6,2805 | 7,7768 |
| 29 | sp A29 | 7,1488 | 6,0925 | 7,9017 |
| 30 | sp A30 | 8,6122 | 6,6780 | 7,5781 |
| 31 | sp A31 | 8,5879 | 7,0597 | 9,4764 |
| 32 | sp A32 | 9,2209 | 7,7361 | 10,5065 |
| 33 | sp A33 | 8,1634 | 6,6109 | 9,4917 |

| No. | Nama Data | Red | Green | Blue |
|---|---|---|---|---|
| 34 | sp A34 | 6,2355 | 5,3160 | 7,9137 |
| 35 | sp A35 | 8,4730 | 6,6452 | 9,9077 |
| 36 | sp A36 | 7,5722 | 6,9341 | 9,3505 |
| 37 | sp A37 | 6,3539 | 5,9131 | 10,6356 |
| 38 | sp A38 | 7,3895 | 6,5217 | 8,9139 |
| 39 | sp A39 | 8,2234 | 6,9777 | 9,5998 |
| 40 | sp A40 | 8,5759 | 7,7438 | 9,3766 |

Pengurangan noise yang dapat dilakukan dengan metode *Adaptive Median Filter* didapatkan kisaran nilai antara 5 hingga 11 pada setiap layer data citra outdoor dengan *salt and pepper noise* 10% seperti yang terlihat pada (tabel 4.3).

Gambar 4. 11. Data citra uji *indoor salt and pepper noise*10%

Gambar 4. 12. Data citra hasil komputasi CPU

Gambar 4. 13. Data citra hasil komputasi GPU

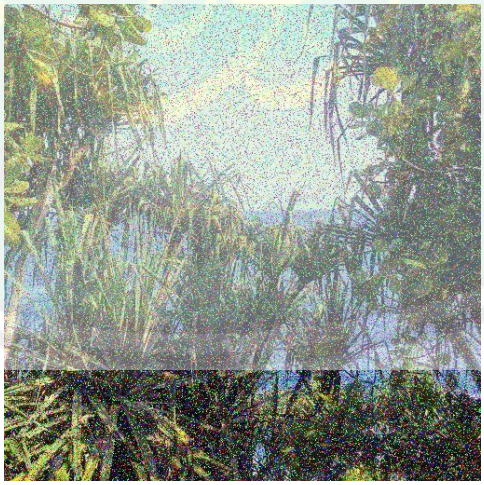
Citra pada (gambar 4.15) menunjukkan citra hasil untuk data uji pada data citra outdoor dengan *salt and pepper noise* 20%. Citra tersebut dihasilkan dari komputasi CPU, sedangkan (gambar 4.16) menunjukkan citra hasil dari komputasi GPU. Hasil kedua citra tersebut dihasilkan dari data citra uji pada (gambar 4.14). Waktu yang diperlukan untuk komputasi CPU adalah 2451,11 ms sedangkan untuk komputasi GPUdiperlukan waktu 794,43 ms.

Tabel 4. 4Hasil Pengurangan Gambar Input Outdoor dengan Gambar Hasil pada Salt & Pepper noise 20%

| No. | Nama Data | Red | Green | Blue |
|---|---|---|---|---|
| 1 | sp A1 | 12,4109 | 11,0420 | 14,7398 |
| 2 | sp A2 | 12,3852 | 9,1391 | 11,5119 |
| 3 | sp A3 | 10,8510 | 8,6533 | 8,5602 |
| 4 | sp A4 | 10,8364 | 8,8322 | 9,1962 |
| 5 | sp A5 | 11,7514 | 9,8973 | 9,4494 |
| 6 | sp A6 | 14,3949 | 12,2362 | 14,4043 |
| 7 | sp A7 | 9,9057 | 9,7112 | 11,6483 |
| 8 | sp A8 | 12,4076 | 9,9548 | 11,5753 |
| 9 | sp A9 | 12,6873 | 10,1653 | 10,6918 |
| 10 | sp A10 | 12,9724 | 11,9622 | 10,8261 |
| 11 | sp A11 | 11,8863 | 11,9279 | 9,8263 |
| 12 | sp A12 | 13,6789 | 10,5765 | 11,3771 |
| 13 | sp A13 | 12,2963 | 10,0737 | 11,3348 |
| 14 | sp A14 | 13,3854 | 10,1857 | 9,3823 |
| 15 | sp A15 | 11,8778 | 10,1568 | 12,7058 |
| 16 | sp A16 | 13,3027 | 9,5376 | 9,6309 |
| 17 | sp A17 | 12,1911 | 9,4911 | 8,9821 |
| 18 | sp A18 | 9,7185 | 8,9529 | 16,6191 |
| 19 | sp A19 | 11,8194 | 8,2636 | 13,7021 |
| 20 | sp A20 | 12,8834 | 10,7495 | 14,1327 |
| 21 | sp A21 | 11,5676 | 10,2042 | 13,4887 |
| 22 | sp A22 | 10,2778 | 8,8705 | 11,3628 |
| 23 | sp A23 | 11,8743 | 9,4742 | 14,1512 |
| 24 | sp A24 | 10,9926 | 12,9618 | 10,1689 |
| 25 | sp A25 | 14,7043 | 12,2851 | 16,0818 |

| No. | Nama Data | Red | Green | Blue |
|---|---|---|---|---|
| 26 | sp A26 | 10,9781 | 8,8612 | 12,2784 |
| 27 | sp A27 | 9,9882 | 8,7560 | 10,6732 |
| 28 | sp A28 | 12,2994 | 10,5061 | 12,7235 |
| 29 | sp A29 | 10,7768 | 8,9347 | 12,4237 |
| 30 | sp A30 | 12,9279 | 9,1826 | 11,6469 |
| 31 | sp A31 | 12,8400 | 10,0645 | 14,2791 |
| 32 | sp A32 | 14,1605 | 11,2874 | 16,5810 |
| 33 | sp A33 | 11,9860 | 9,2335 | 14,6520 |
| 34 | sp A34 | 9,6460 | 8,1783 | 12,9327 |
| 35 | sp A35 | 12,9360 | 9,5068 | 15,3194 |
| 36 | sp A36 | 11,0308 | 10,0760 | 14,5007 |
| 37 | sp A37 | 9,9416 | 9,4901 | 17,6973 |
| 38 | sp A38 | 11,9540 | 10,5111 | 14,8790 |
| 39 | sp A39 | 12,2353 | 10,1710 | 15,0555 |
| 40 | sp A40 | 12,4043 | 11,3200 | 13,7667 |

Pengurangan noise yang dapat dilakukan dengan metode *Adaptive Median Filter* didapatkan kisaran nilai antara 8 hingga 18 pada setiap layer data citra outdoor dengan *salt and pepper noise* 20% seperti yang terlihat pada (tabel 4.4).



Gambar 4. 14. Data citra uji *indoor salt and pepper noise*20%

Gambar 4. 15. Data citra hasil komputasi CPU

Gambar 4. 16. Data citra hasil komputasi GPU

Hasil implementasi Metode *Adaptive Median Filter* yang telah dilakukan,diambil nilai kualitas visual dari proses filter tersebut. Kinerja perbaikan citra dengan metode tersebut dapat dinilai kualitas visualnya ditunjukkan dengan menghitung*Peak Signal to Noise Ratio*(PSNR). PSNR didapatkan dengan membandingkan citra keluaran hasil filter dengan citra asli yang belum diberi noise. PSNR digunakan sebagai pembanding pada golongan data yang tersedia dalam menentukan kinerja metode *Adaptive Median Filter*pada GPU. Adapun data yang dibandingkan dibagi menjadi data dengan citra ber-noise Salt & Pepper 10% dan noise Salt & Pepper 20%. Masing-masing citra dengan intensitas noise-nya, dibagi lagi menjadi citra indoor dengan citra outdoor. Hasil perhitungan PSNR dapat dilihat pada (Tabel 4.5). Berikut ini adalah fungsi PSNR

$$PSNR = 10\ \log_{10} \frac{255^2}{\frac{1}{MN}\sum_{i,j}(r_{i,j} - x_{i,j})^2}$$

Dimana N×M adalah ukuran citra, r mendefinisikan citra yang sudah difilter, dan x mendefinisikan citra asli.

Tabel 4. 5 Nilai *Peak Signal to Noise Ratio*

| No. | Nama | Noise Salt & Pepper 10% | Noise Salt & Pepper 20% |
|---|---|---|---|

| | Data | Indoor | Outdoor | Indoor | Outdoor |
|---|---|---|---|---|---|
| 1 | sp A1 | 22.5071 | 20.0976 | 18.7921 | 17.0261 |
| 2 | sp A2 | 22.5409 | 20.5871 | 18.8770 | 17.5770 |
| 3 | sp A3 | 21.6521 | 20.7166 | 18.3498 | 17.6785 |
| 4 | sp A4 | 22.1387 | 20.5374 | 18.6463 | 18.6359 |
| 5 | sp A5 | 22.3549 | 22.0082 | 18.7235 | 17.8065 |
| 6 | sp A6 | 22.8496 | 21.1944 | 19.0711 | 18.2552 |
| 7 | sp A7 | 22.0900 | 21.3418 | 18.1702 | 17.4084 |
| 8 | sp A8 | 22.2607 | 20.3423 | 18.2122 | 18.3621 |
| 9 | sp A9 | 21.7980 | 21.9230 | 18.6699 | 18.2438 |
| 10 | sp A10 | 21.5023 | 19.7229 | 17.5733 | 18.3572 |
| 11 | sp A11 | 22.1369 | 21.6853 | 18.7277 | 17.9247 |
| 12 | sp A12 | 22.7875 | 22.2997 | 19.0196 | 18.4382 |
| 13 | sp A13 | 22.2489 | 21.0829 | 18.2040 | 17.9457 |
| 14 | sp A14 | 22.0601 | 21.8412 | 18.0892 | 18.3082 |
| 15 | sp A15 | 21.0620 | 21.2450 | 17.5925 | 18.2738 |
| 16 | sp A16 | 23.1146 | 21.1629 | 18.9273 | 18.0017 |
| 17 | sp A17 | 22.5363 | 21.6338 | 19.0013 | 18.3771 |
| 18 | sp A18 | 21.5551 | 21.3207 | 18.5246 | 17.8906 |
| 19 | sp A19 | 21.8383 | 21.7491 | 18.8722 | 18.3793 |
| 20 | sp A20 | 22.1044 | 20.6217 | 18.0213 | 17.5312 |
| 21 | sp A21 | 22.6795 | 21.5600 | 18.9731 | 18.3025 |
| 22 | sp A22 | 22.1184 | 21.4044 | 18.7186 | 18.0936 |
| 23 | sp A23 | 21.8171 | 20.5401 | 18.1868 | 17.9412 |
| 24 | sp A24 | 22.4583 | 20.4074 | 18.3248 | 18.2977 |
| 25 | sp A25 | 22.0105 | 20.6564 | 18.3177 | 17.6537 |
| 26 | sp A26 | 22.0949 | 21.3519 | 18.2367 | 18.1067 |
| 27 | sp A27 | 22.3877 | 21.4831 | 18.3875 | 18.1725 |
| 28 | sp A28 | 22.1419 | 22.3794 | 17.9581 | 18.5558 |
| 29 | sp A29 | 22.5658 | 21.3175 | 18.5779 | 18.1668 |
| 30 | sp A30 | 21.9195 | 22.2567 | 18.4048 | 18.1293 |
| 31 | sp A31 | 22.2319 | 20.2069 | 18.1083 | 17.6644 |
| 32 | sp A32 | 21.7602 | 20.3735 | 18.3275 | 17.4780 |
| 33 | sp A33 | 22.1145 | 20.5802 | 18.5847 | 17.6454 |
| 34 | sp A34 | 21.7447 | 20.8223 | 18.0976 | 17.4635 |
| 35 | sp A35 | 22.3792 | 21.0192 | 18.4694 | 18.0936 |
| 36 | sp A36 | 22.4286 | 21.0114 | 19.0489 | 18.2797 |
| No. | Nama Data | Noise Salt & Pepper 10% | | Noise Salt & Pepper 20% | |
| | | Indoor | Outdoor | Indoor | Outdoor |

| | | | | | |
|---|---|---|---|---|---|
| 37 | sp A37 | 22.8730 | 21.9864 | 18.1235 | 18.0121 |
| 38 | sp A38 | 22.3911 | 20.9099 | 18.2991 | 17.6031 |
| 39 | sp A39 | 22.5411 | 20.9494 | 18.4571 | 17.9634 |
| 40 | sp A40 | 20.9646 | 20.0948 | 18.4936 | 17.4902 |

Dari (tabel 4.5) didapatkan nilai rata-rata PSNR pada data citra indoor hasil filter yang berintensitas noise 10% yaitu 22.19085 ; data citra outdoor hasil filter yang berintensitas noise 10% yaitu 21.11196 ; data citra indoor hasil filter yang berintensitas noise 20% yaitu 18.4577 ; data citra outdoor hasil filter yang berintensitas noise 20% yaitu 17.97172.Di antara data tersebut yang memiliki nilai PSNR tertinggi adalah data yang diberi noise 10% dimana nilai PSNR pada interval 20 hingga 40 dapat ditentukan sebagai hasil yang baik (Rabbani,M dan Jones, P.W., 1991). Sehingga implementasi metode *Adaptive Median Filter* pada citra dengan noise 10% memiliki hasil yang baik dibandingkan dengan implementasi pada citra dengan noise 20%. *Adaptive Median Filter* bekerja dengan baik untuk intensitas noise yang rendah karena metode tersebut mencoba untuk menggunakan filter dengan window paling minimal dan hanya memodifikasi pixel yang dianggap sebagai *corrupted pixel* atau noise. Ukuran window mempengaruhi kualitas hasil filter jika intensitas noise lebih dari 40% yang juga dijelaskan pada penelitian sebelumnya oleh Vasicek dan Sekanina(2008).

Pada penelitian ini perbesaran window pixel maksimal adalah 5x5. Hal tersebut dikarenakan sumberdaya yang digunakan memiliki keterbatasan pada GPU. Sehingga perbesaran window pixel maksimal pada penelitian ini tidak dapat diperbesar lagi.Ketika nilai perbesaran window diperbesar dalam pemrograman GPU, proses tidak dapat dijalankan dengan memunculkan pesan "*Display driver*

*stopped respondingand has recovered*". Sehingga penelitian ini dilakukan dengan data citra intensitas noise 10% dan 20%.

### 4.3.2. Hasil Waktu Komputasi

Dari hasil implementasi metode *Adaptive Median Filter* pada komputasi CPU dan GPU sebelumnya, waktu komputasi juga didapatkan dari pemrosesan tersebut. Data hasil pengukuran waktu komputasi disajikan dalam tabel yang dibedakan menjadi 4 tabel yaitu tabel hasil waktu komputasi citra *indoor salt and pepper noise*10%, citra *indoor salt and pepper noise*20%, citra *outdoor salt and pepper noise*10%, dan tabel hasil waktu komputasi citra *outdoor salt and pepper noise*20%.

Dalam tabel tersebut dipaparkanwaktu komputasi secara CPU dan GPU. Selain dalam bentuk tabel, hasil pengukuran waktu komputasi disajikan pula dalam bentuk grafik. Dari rekapitulasi hasil waktu komputasi tersebut, didapatkan rata-rata waktu komputasi. Dari rata-rata tersebut, didapatkan prosentase peningkatan kecepatan komputasi metode *Adaptive Median Filter*dari eksekusi CPUdan eksekusi GPU yang kemudian disajikan dalam bentuk grafik. Satuan yang dipakai untuk memaparkan nilai konsumsi waktu komputasi yaitu milidetik.

Data citra uji yang diambil di dalam ruangan (*indoor*) yang telah diberi *salt and pepper noise*10%berjumlah 40 citra. Masing-masing citra tersebut didapatkan waktu hasil komputasi CPU dan GPU. Dalam paparan data pada (tabel 4.1) dan grafik pada (Gambar 4.17), dapat dilihat pemrosesan dengan komputasi CPU mengkonsumsi waktu hingga 3767,7 milidetik sedangkan komputasi GPU

mengkonsumsi waktu hanya mencapai 14633,7 milidetik. Waktu yang dikonsumsi komputasi CPU lebih banyak hampir 2 kali daripada komputasi GPU.

Tabel 4. 6. Hasil waktu komputasi citra *indoor salt and pepper noise*10%

| No. | Nama Data | Kecepatan | |
|---|---|---|---|
| | | CPU | GPU |
| 1 | sp A1 | 3462,12 | 568,76 |
| 2 | sp A2 | 2418,12 | 538,17 |
| 3 | sp A3 | 2472,81 | 607,67 |
| 4 | sp A4 | 2411,81 | 621,94 |
| 5 | sp A5 | 2423,15 | 565 |
| 6 | sp A6 | 2757,6 | 515,42 |
| 7 | sp A7 | 2554,82 | 850,38 |
| 8 | sp A8 | 2591,04 | 1044,94 |
| 9 | sp A9 | 2649,97 | 546,88 |
| 10 | sp A10 | 2975,78 | 1463,72 |
| 11 | sp A11 | 2456,37 | 670,1 |
| 12 | sp A12 | 2982,54 | 612,46 |
| 13 | sp A13 | 2344,79 | 968,51 |
| 14 | sp A14 | 3133,46 | 971,26 |
| 15 | sp A15 | 2856,46 | 1259,93 |
| 16 | sp A16 | 3048,72 | 605,3 |
| 17 | sp A17 | 2847,55 | 621,18 |
| 18 | sp A18 | 2916,19 | 518,34 |
| 19 | sp A19 | 2422,98 | 930,41 |
| 20 | sp A20 | 2391,78 | 959,4 |
| 21 | sp A21 | 2484,59 | 719,06 |
| 22 | sp A22 | 2840,08 | 565,56 |
| 23 | sp A23 | 2827,36 | 751,92 |
| 24 | sp A24 | 3155,34 | 1044,54 |
| 25 | sp A25 | 2444,29 | 704,01 |
| 26 | sp A26 | 2276,12 | 884,88 |
| 27 | sp A27 | 2345,65 | 732,89 |
| 28 | sp A28 | 2567,41 | 1318,75 |
| 29 | sp A29 | 2427,38 | 611,22 |
| 30 | sp A30 | 2429,29 | 830,53 |
| No. | Nama Data | Kecepatan | |
| | | CPU | GPU |

| | | | |
|---|---|---|---|
| 31 | sp A31 | 2348,11 | 944,68 |
| 32 | sp A32 | 2395,86 | 823,43 |
| 33 | sp A33 | 3767,72 | 850,65 |
| 34 | sp A34 | 2760,77 | 794,4 |
| 35 | sp A35 | 2799,21 | 767,47 |
| 36 | sp A36 | 2759,92 | 521,33 |
| 37 | sp A37 | 2438,78 | 591,4 |
| 38 | sp A38 | 3027,74 | 1089,04 |
| 39 | sp A39 | 2906,19 | 937,91 |
| 40 | sp A40 | 2505,08 | 1079,05 |

Data citra uji yang diambil di dalam ruangan (*indoor*) yang telah diberi ***salt and pepper noise***20%berjumlah 40 citra. Masing-masing citra tersebut didapatkan waktu hasil komputasi CPU dan GPU. Dalam paparan data pada (tabel 4.2) dan grafik pada (Gambar 4.18), dapat dilihat pemrosesan dengan komputasi CPU mengkonsumsi waktu hingga 2669,7 milidetik sedangkan komputasi GPU mengkonsumsi waktu hanya mencapai 1539,57 milidetik. Komputasi CPU memerlukan waktu 2 kali lebih lama dari komputasi GPU.

Tabel 4. 7. Hasil waktu komputasi citra indoor salt and pepper noise 20%

| No. | Nama Data | Kecepatan (ms) | |
|---|---|---|---|
| | | CPU | GPU |
| 1 | sp A1 | 2374,79 | 678,59 |
| 2 | sp A2 | 2551,21 | 590,41 |
| 3 | sp A3 | 2250,86 | 686,13 |
| 4 | sp A4 | 2636,53 | 690,65 |
| 5 | sp A5 | 2339,97 | 625,41 |
| 6 | sp A6 | 2242,28 | 530,5 |
| 7 | sp A7 | 2360,86 | 966,34 |
| 8 | sp A8 | 2353,27 | 1077,71 |
| 9 | sp A9 | 2228,85 | 562,8 |
| 10 | sp A10 | 2485,42 | 1539,57 |
| No. | Nama Data | Kecepatan (ms) | |
| | | CPU | GPU |

| | | | |
|---|---|---|---|
| 11 | sp A11 | 2361,23 | 706,01 |
| 12 | sp A12 | 2235,78 | 617,56 |
| 13 | sp A13 | 2298,54 | 1067,85 |
| 14 | sp A14 | 2424,16 | 1047,54 |
| 15 | sp A15 | 2389,52 | 1324,94 |
| 16 | sp A16 | 1462,61 | 864,35 |
| 17 | sp A17 | 1254,22 | 605,55 |
| 18 | sp A18 | 2255,33 | 548,16 |
| 19 | sp A19 | 2669,67 | 1015,91 |
| 20 | sp A20 | 2493,99 | 1048,12 |
| 21 | sp A21 | 2304,44 | 757,75 |
| 22 | sp A22 | 2531,62 | 596,7 |
| 23 | sp A23 | 2245,38 | 802,58 |
| 24 | sp A24 | 2379,46 | 1119,88 |
| 25 | sp A25 | 2607,46 | 827,28 |
| 26 | sp A26 | 2287,15 | 976,97 |
| 27 | sp A27 | 1301,73 | 1012,8 |
| 28 | sp A28 | 1515,96 | 1508,12 |
| 29 | sp A29 | 1398,31 | 717,96 |
| 30 | sp A30 | 1494,56 | 889,74 |
| 31 | sp A31 | 1809,61 | 1019,15 |
| 32 | sp A32 | 1459,12 | 893,08 |
| 33 | sp A33 | 1400,56 | 703,49 |
| 34 | sp A34 | 1457 | 898,37 |
| 35 | sp A35 | 1310,33 | 868,61 |
| 36 | sp A36 | 1325,09 | 509,52 |
| 37 | sp A37 | 1439,41 | 671,19 |
| 38 | sp A38 | 1715,47 | 1119,17 |
| 39 | sp A39 | 1464,43 | 976,02 |
| 40 | sp A40 | 1528,21 | 1211,31 |

Data citra uji yang diambil di luar ruangan (outdoor) yang telah diberi salt and pepper noise 10% berjumlah 40 citra. Masing-masing citra tersebut didapatkan waktu hasil komputasi CPU dan GPU. Dalam paparan data pada (tabel 4.3) dan pada grafik (Gambar 4.19), dapat dilihat pemrosesan dengan komputasi

CPU mengkonsumsi waktu hingga 3608,1milidetik sedangkan komputasi GPU mengkonsumsi waktu hanya mencapai 2566,7milidetik. Komputasi GPU memerlukan waktu sekitar hampir 2 kali lebih cepat daripada komputasi CPU.

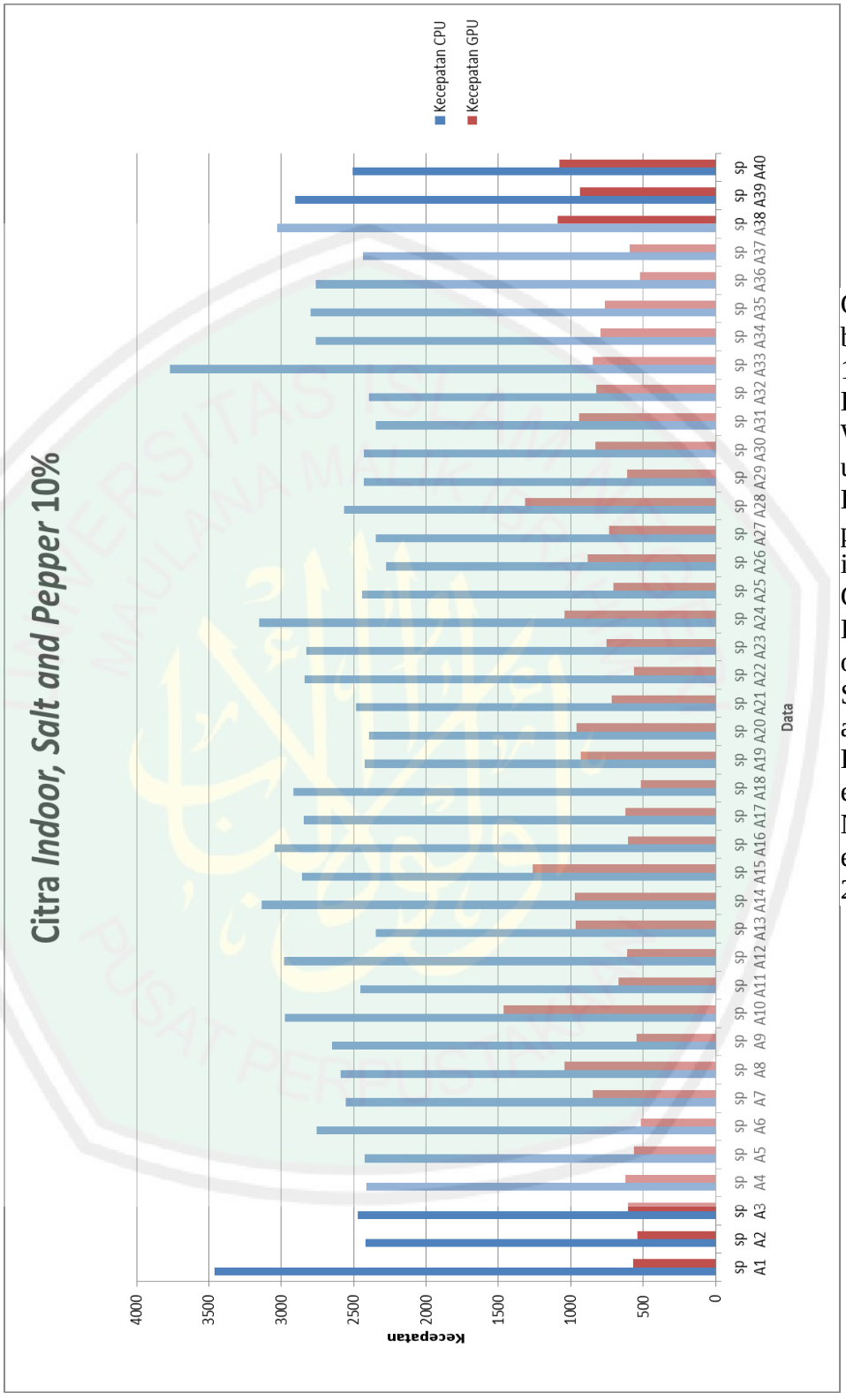
Tabel 4. 8. Hasil waktu komputasi citra outdoor salt and pepper noise 10%

| No. | Nama Data | Kecepatan | |
|---|---|---|---|
| | | CPU | GPU |
| 1 | sp A1 | 2611,57 | 1305,11 |
| 2 | sp A2 | 2690,62 | 860,91 |
| 3 | sp A3 | 3346,19 | 908,07 |
| 4 | sp A4 | 3191,09 | 867,86 |
| 5 | sp A5 | 2987,31 | 587,44 |
| 6 | sp A6 | 2854,46 | 1017,79 |
| 7 | sp A7 | 2824,05 | 577,28 |
| 8 | sp A8 | 3517,1 | 924,19 |
| 9 | sp A9 | 2634,47 | 730,97 |
| 10 | sp A10 | 2824,46 | 904,13 |
| 11 | sp A11 | 2898,73 | 1206,92 |
| 12 | sp A12 | 2566,72 | 2566,72 |
| 13 | sp A13 | 2822,95 | 909,95 |
| 14 | sp A14 | 2724,45 | 848,29 |
| 15 | sp A15 | 3608,05 | 743,26 |
| 16 | sp A16 | 3067,99 | 792,94 |
| 17 | sp A17 | 2958,75 | 743,01 |
| 18 | sp A18 | 2880,01 | 989,96 |
| 19 | sp A19 | 3019,84 | 807,57 |
| 20 | sp A20 | 2870,91 | 1021,39 |
| 21 | sp A21 | 3138,28 | 738,64 |
| 22 | sp A22 | 2880,89 | 1042,95 |
| 23 | sp A23 | 3108,61 | 629,55 |
| 24 | sp A24 | 3136,23 | 804,85 |
| 25 | sp A25 | 2999,42 | 931,58 |
| 26 | sp A26 | 2640 | 909,39 |
| 27 | sp A27 | 2711,26 | 1003,6 |
| 28 | sp A28 | 2420,56 | 779,26 |
| 29 | sp A29 | 2695,71 | 880,94 |
| No. | Nama | Kecepatan | |

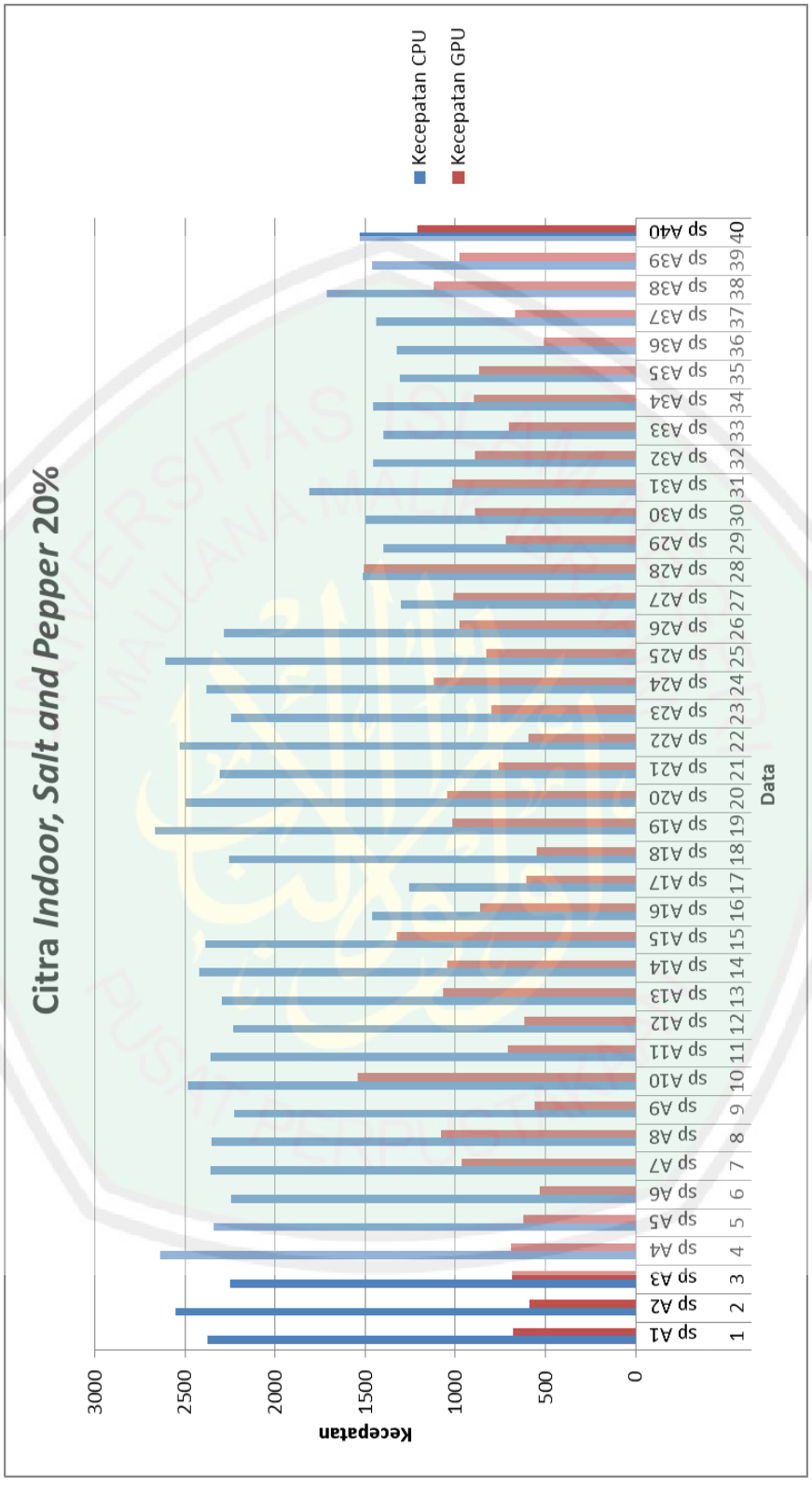
| | Data | CPU | GPU |
|---|---|---|---|
| 30 | sp A30 | 2386,86 | 906,19 |
| 31 | sp A31 | 3235,1 | 712,74 |
| 32 | sp A32 | 2618,01 | 878,64 |
| 33 | sp A33 | 2957,24 | 868,59 |
| 34 | sp A34 | 2644,76 | 1261,81 |
| 35 | sp A35 | 3043,41 | 657,98 |
| 36 | sp A36 | 3103,73 | 745,48 |
| 37 | sp A37 | 2476,13 | 1005,69 |
| 38 | sp A38 | 2484,55 | 1210,72 |
| 39 | sp A39 | 3177,1 | 742 |
| 40 | sp A40 | 2119,46 | 670,56 |

Data citra uji yang diambil di luar ruangan (outdoor) yang telah diberi salt and pepper noise 20% berjumlah 40 citra. Masing-masing citra tersebut didapatkan waktu hasil komputasi CPU dan GPU. Dalam paparan data pada (tabel 4.4) dan pada grafik (Gambar 4.20), dapat dilihat pemrosesan dengan komputasi CPU mengkonsumsi waktu hingga 2934,77milidetik sedangkan komputasi GPU mengkonsumsi waktu hanya mencapai 1400,38milidetik. Komputasi CPU memerlukan waktu lebih dari 2 kali lebih lama daripada komputasi GPU.

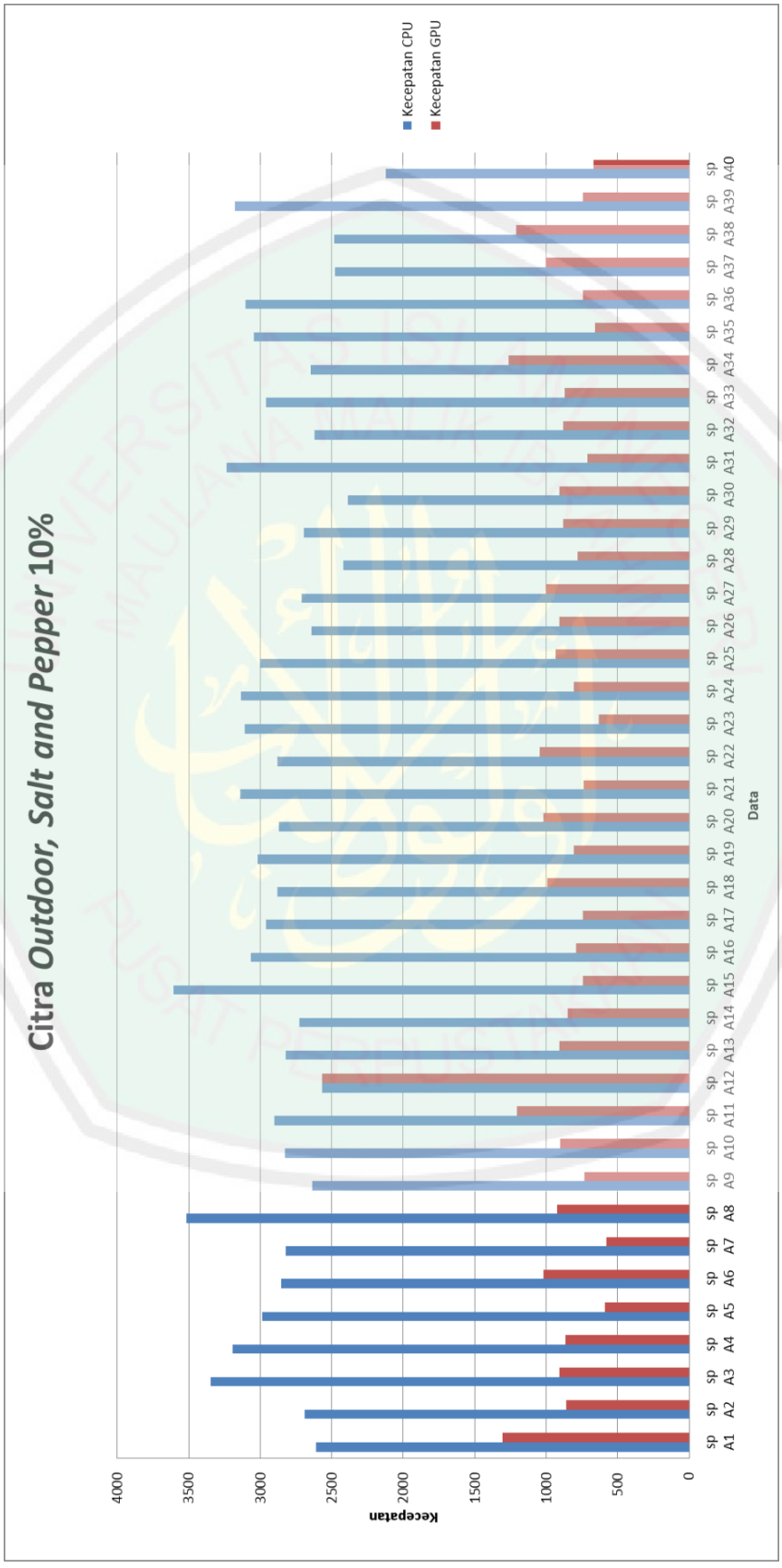
Pengukuran waktu didapatkan dari 160 data dengan tiap datanya dieksekusi secara CPU dan GPU. Pada umumnya pemrosesan yang dilakukan dengan CPU mempunyai nilai waktu komputasi yang hampir sama yaitu sekitar 2000 milidetik. Sedangkan pemrosesan yang dilakukan dengan GPU didapatkan nilai waktu komputasi yang bervariasi yaitu dalam rentang sekitar 500 hingga 1500 milidetik.



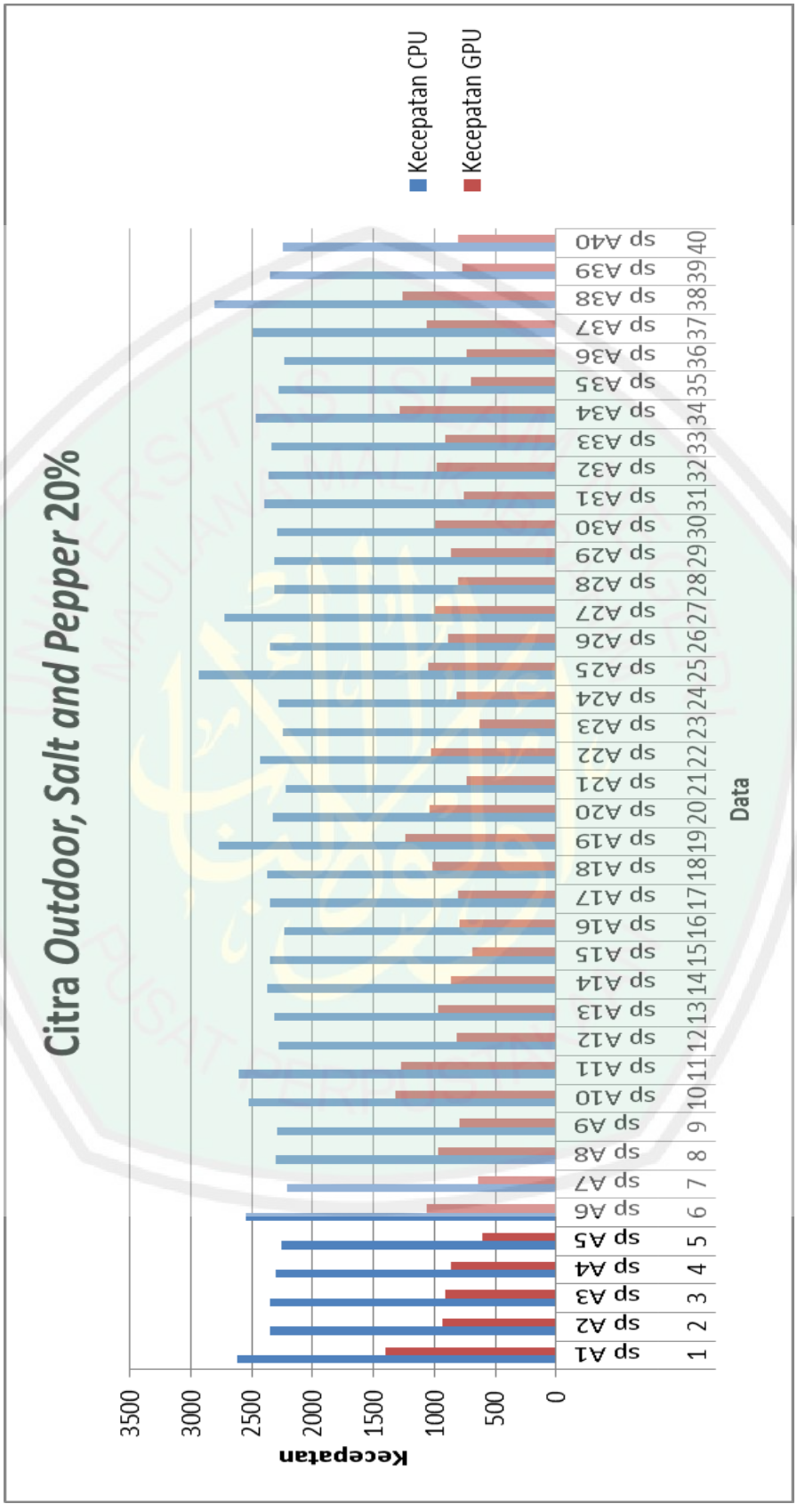

Gambar 4.17 Hasil Waktu Komputasi Citra Indoor, Salt and Pepper Noise 20%




Gambar 4.18 Hasil Waktu Komputasi Citra Indoor, Salt and Pepper Noise 20%




Gambar 4.19 Hasil Waktu Komputasi Citra Outdoor, Salt and Pepper Noise 20%

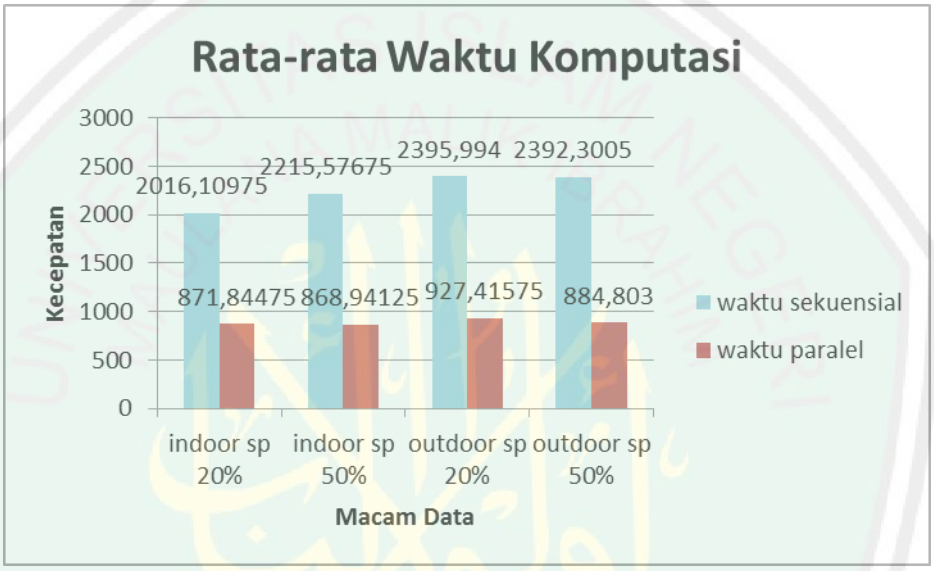



Gambar 4.20 Hasil Waktu Komputasi Citra Outdoor, Salt and Pepper Noise 20%

Tabel 4. 9. Hasil waktu komputasi citra outdoor salt and pepper noise 20%

| No. | Nama Data | Kecepatan(ms) | |
|---|---|---|---|
| | | CPU | GPU |
| 1 | sp A1 | 2622,35 | 1400,38 |
| 2 | sp A2 | 2349,83 | 932,74 |
| 3 | sp A3 | 2352,26 | 908,19 |
| 4 | sp A4 | 2308,19 | 864,18 |
| 5 | sp A5 | 2259,68 | 608 |
| 6 | sp A6 | 2549,05 | 1062,47 |
| 7 | sp A7 | 2205,8 | 645,64 |
| 8 | sp A8 | 2306,5 | 974,54 |
| 9 | sp A9 | 2287,74 | 790,49 |
| 10 | sp A10 | 2522,96 | 1323,72 |
| 11 | sp A11 | 2611,41 | 1272,7 |
| 12 | sp A12 | 2279,62 | 819,31 |
| 13 | sp A13 | 2317,11 | 970,73 |
| 14 | sp A14 | 2375,75 | 860,1 |
| 15 | sp A15 | 2354,72 | 690,03 |
| 16 | sp A16 | 2235,08 | 798,74 |
| 17 | sp A17 | 2344,08 | 806,97 |
| 18 | sp A18 | 2371,68 | 1020,4 |
| 19 | sp A19 | 2772,6 | 1233,43 |
| 20 | sp A20 | 2324,21 | 1043,44 |
| 21 | sp A21 | 2226,66 | 737,65 |
| 22 | sp A22 | 2432,95 | 1029,31 |
| 23 | sp A23 | 2246,39 | 635,4 |
| 24 | sp A24 | 2284,82 | 812,5 |
| 25 | sp A25 | 2934,77 | 1046,93 |
| 26 | sp A26 | 2347,06 | 885,98 |
| 27 | sp A27 | 2725,76 | 992,16 |
| 28 | sp A28 | 2312,77 | 800,4 |
| 29 | sp A29 | 2316,07 | 866,01 |
| 30 | sp A30 | 2287,08 | 997,96 |
| 31 | sp A31 | 2395,05 | 762,92 |
| 32 | sp A32 | 2362,03 | 979,47 |
| 33 | sp A33 | 2338,98 | 907,15 |
| 34 | sp A34 | 2465,24 | 1289,53 |
| 35 | sp A35 | 2283,84 | 699,02 |

| No. | Nama Data | Kecepatan (ms) | |
|---|---|---|---|
| | | CPU | GPU |
| 36 | sp A36 | 2227,87 | 730,61 |
| 37 | sp A37 | 2493,89 | 1063,73 |
| 38 | sp A38 | 2807,14 | 1260,41 |
| 39 | sp A39 | 2351,94 | 768,22 |
| 40 | sp A40 | 2248,83 | 805,07 |




Gambar 4. 21 Rata-rata Waktu Komputasi

Dari hasil uji coba dengan keempat macam data di atas, diambil nilai rata-ratanya tiap macam data uji seperti terlihat pada grafik (Gambar 4.20). Macam data uji yang pertama memiliki selisih 1144,265; macam data yang kedua memiliki selisih 1346,6355; macam data yang ketiga memiliki selisih 1468,57825, macam data yang keempat memiliki selisih 1507,4975. Data tersebut menunjukkan bahwa perbandingan waktu antara CPU dengan GPU dari data pertama hingga keempat mengalami peningkatan kesenjangan waktu. Sedangkan macam data menunjukkan kerumitan studi kasus. Semakin tinggi prosentase noise, selisih komputasi CPU dengan komputasi GPU semakin banyak. Dan

semakin tinggi intensitas citra (citra *indoor* dan citra *outdoor*), maka selisih komputasi CPU dengan komputasi GPU juga semakin banyak.

Untuk mengetahui peningkatan kecepatan komputasi GPU pada metode *Adaptive Median Filter* digunakan (rumus 4.1) (Pierce, 2016).

$$\frac{X'-X}{|X|} * 100\% \quad (4.1)$$

Dimana :
X = Nilai Lama
X' = Nilai Baru

Berdasarkan perhitungan, didapatkan nilai peningkatan kecepatan 56,76% untuk rata-rata data hasil pertama, 60,78% untuk rata-rata data hasil kedua, 61,29% untuk rata-rata data hasil ketiga, 63,01% untuk rata-rata data hasil keempat. Dari keempat sampel data tersebut dapat dilihat rata-rata prosentase peningkatan kecepatan komputasi GPU yaitu kurang lebih 60,46% dari komputasi CPU.

### 4.4. Integrasi Penelitian dengan Al-Qur'an

Kemajuan teknologi seiring dengan perkembangan zaman, dunia Teknologi Informasi terus melakukan perubahan-perubahan demi meningkatkan efektifitas dan efisiensi dalam kinerjanya. Di sisi lain, peningkatan kinerja seharusnya juga diikuti dengan penurunan energi yang dibutuhkan agar tercipta harmonisasi manusia dengan alam. Dimana harmonisasi dapat tercipta jika tidak ada lingkungan terjaga dari kerusakan oleh ulah manusia itu sendiri. Namun demikian, tidak semua produk teknologi yang ramah terhadap kelestariannya lingkungan. Hal ini dikarenakan adanya pemborosan energi yang diperlukan karena penggunaan teknologi banyak membutuhkan energi listrik ternyata menjadi faktor

utama penyumbang emisi gas rumah kaca (Stern, 2007). Allah SWT berfirman dalam Al-Quran surat Al-Baqarah ayat 30

ِيهَايُفۡسِدُ مَن فِيهَآ أَتَجۡعَلُ قَالُوٓاْ خَلِيفَةً ٱلۡأَرۡضِ فِي جَاعِلٌ إِنِّي لِلۡمَلَٰٓئِكَةِ رَبُّكَ قَالَ وَإِذۡ

﴿٣٠﴾ تَعۡلَمُونَ لَا مَآ أَعۡلَمُ إِنِّيٓ قَالَ لَكَ وَنُقَدِّسُ بِحَمۡدِكَ نُسَبِّحُ وَنَحۡنُ ٱلدِّمَآءَ وَيَسۡفِكُ

*Artinya : (30). ingatlah ketika Tuhanmu berfirman kepada Para Malaikat: "Sesungguhnya aku hendak menjadikan seorang khalifah di muka bumi." mereka berkata: "Mengapa Engkau hendak menjadikan (khalifah) di bumi itu orang yang akan membuat kerusakan padanya dan menumpahkan darah, Padahal Kami Senantiasa bertasbih dengan memuji Engkau dan mensucikan Engkau?" Tuhan berfirman: "Sesungguhnya aku mengetahui apa yang tidak kamu ketahui."*

Dari ayat tersebut dapat diambil intisari bahwa manusia diciptakan untuk melindungi alam yang telah diberikan oleh Yang Maha Kuasa.Oleh karenanya perlu diambil langkah untuk mengurangi kerusakan lingkungan seperti yang telah Allah perintahkan kepada seluruh umat manusia. Dalam perkembangan teknologi informasi, telah dirilis pemrograman yang mengizinkan pemrosesan komputasi secara paralel.Pemrograman paralel tersebut merupakan teknik untuk melakukan proses eksekusi perintah secara bersamaan.

Pemrograman paralel ditujukan untuk meningkatkan performa komputasi. Performa dalam pemrograman paralel diukur dari berapa banyak peningkatan kecepatan (speed up) yang diperoleh dalam menggunakan teknik GPU. Sehingga teknologi ini diyakini dapat mengurangi kerusakan lingkungan akibat penggunaan energy listrik yang berlebihan. Dengan teknologi paralel pula permasalahan yang komplek dapat diselesaikan dengan waktu yang minimum. Sehingga dalam industri dan komersial yang biasanya menggunakan sumber daya secara berlebih, kini dapat dikurangi secara maksimal demi menjaga kelestarian lingkungan.

# BAB V

# PENUTUP

## 5.1. Kesimpulan

Berdasarkan penelitian yang telah dilakukan, adapun kesimpulan yang dapat diambil:

1. Mengimplementasikan komputasi GPUpada metode *Adaptive Median Filter* untuk penghapusan*noise*diawali dengan perancangan metode dan thread untuk pemetaan data; kemudian pemrosesan pada host, dilanjutkan proses pada device (GPU) untuk pemrograman secara paralel, dan penyelesaian pemrograman dilakukan pada host.
2. Efisiensi waktu untuk implementasi komputasi GPU pada metode *Adaptive Median Filter*menunjukkan nilai peningkatan konsumsi waktu 60,46% dari komputasi CPU

## 5.2. Saran

Untuk pengembangan implementasi komputasi GPU, diperlukan perbaikan untuk mencapai hasil yang lebih maksimal, meliputi:

1. Perangkat yang digunakan seharusnya lebih dari spesifikasi dari perangkat pada penelitian ini karena spesifikasi perangkat yang terbatas mempengaruhi efisiensi waktu dalam komputasi GPU.
2. Ukuran citra yang digunakan uji coba diperbesar
3. Ukuran *window* maksimal juga diperbesar agar penghapusan *noise* dapat dilakukan secara efektif untuk *noise* dengan prosentase tinggi

# DAFTAR PUSTAKA


Achmad, B., Firdausy, K. 2005. *Teknik Pengolahan Citra Digital Menggunakan Delphi*. Jogjakarta: Ardi Publishing.

Ahmad, Usman. 2005. *Pengolahan Citra Digital dan Teknik Pemrogramannya*. Edisi Pertama. Yogyakarta : Graha Ilmu

Ambule Vicky, et al. 2013. *Adaptive Median Filter for Image Enhancement*. International Journal of Engineering Science and Innovative Technology (IJESIT) Volume 2, Issue 1

Asano A., Itoh K., Ichioka Y. 1991. *Optimization of the weighted median filter by learning*. Optics Express, vol. 16, no. 3, pp. 168–170

Basuki A., Palandi, J.F.F. 2005. *Pengolahan Citra Digital Menggunakan Visual Basic*. Jogjakarta: Graha Ilmu

Ben W., 2006. *Fast median and bilateral filtering*. SIGGRAPH 2006 Papers, ACM

Brownrigg D.R.K. 1984. *The Weighted Median Filter*. Communications of the ACM, Vol. 27, No. 8

Chandel Ruchika, Gupta Gaurav. 2013. *Image Filtering Algorithms and Techniques: A Review*. International Journal of Advanced Research in Computer Science and Software Engineering Vol. 3, Issue 10

Chen T., Ma K.-K. dan Chen L.-H. 1999. *Tri-State Median Filter for Image Denoising*. IEEE Transactions on Image Processing, vol. 8, no. 12.

Hwang H., Haddad R. A. 1995. *Adaptive median filters : new algorithms and results*. IEEE Transactions on Image Processing, vol. 4, no. 4.

Jiang JP, Yuan YT, Bao CP. 2007. *The Algorithm of Fast Filtering*. International Conference on Wavelet Analysis and Pattern Recognition. ICWAPR

Ko S.-J., Lee Y. H. 1991. *Center weighted median filters and their applications to image enhancement.* IEEE Transactions on Circuitsand Systems, vol. 38, no. 9, pp. 984-993.

Lei Peng. 2008. *Adaptive Median Filtering*. Machine Vision on Massey University

Munir Rinaldi. 2004. *Pengolahan Citra Digital dengan Pendekatan Algoritmik*. Bandung : Informatika

Pei YH, Shin SC, Feng CH. 2007. *Generic 2D Gaussian smoothing filter for noisy image processing*. IEEE Region 10 Conference (TENCON)

Pierce Rod. *Percentage Change.* www.mathisfun.com/numbers/percentage-change.html Diakses tanggal 30 Agustus 2016

Rabbani,M dan Jones P.W. 1991. Digital Image Compression Techniques, Vol TT7. Washington: SPIEOptical Engineering Press.

Sanchez Maria G., Vidal Vicente, et al. 2014. *Image Noise Removal on Heterogenous CPU-GPU Configuration*. 14th International Conference on Computational Science Vol. 29

Shrestha Suman. 2014.*Image Denoising Using New Adaptive Based Median Filter.* Signal & Image Processing : An International Journal (SIPIJ) Vol.5

Stern, N. H., & Great Britain. 2007. *The economics of climate change: The Stern review*. Cambridge, UK: Cambridge University Press.

T. Chen, H. R. Wu. 2001. *Adaptive Impulse Detection Using Center-Weighted Median Filters.* IEEE Signal Process. Lett., vol. 8, no. 1

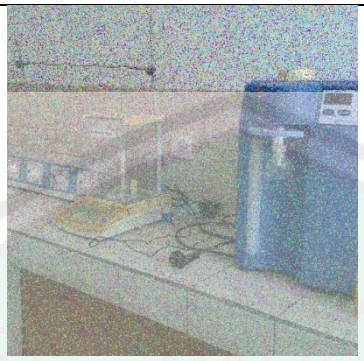
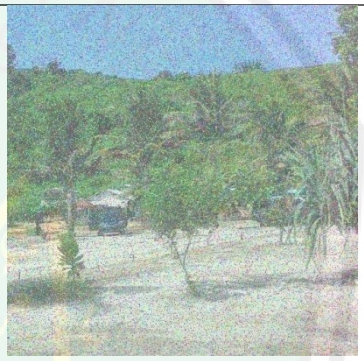
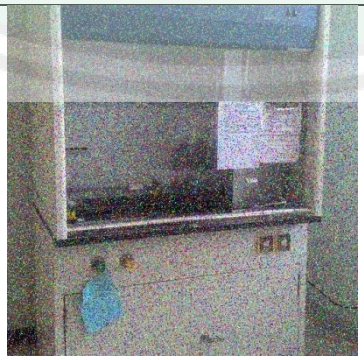
Varade Rohini R., Dhotre M. R., Pahurkar Archana B. 2013. *A Survey on Various Median Filtering Techniques for Removal of Impulse Noise from Digital Images.*International Journal of Advanced Research in Computer Engineering & Technology (IJARCET) Volume 2, Issue 2

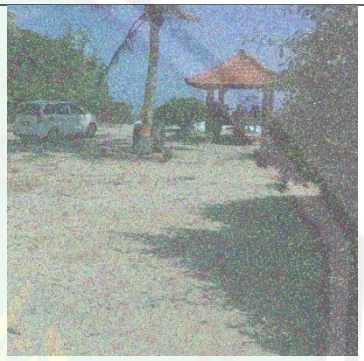
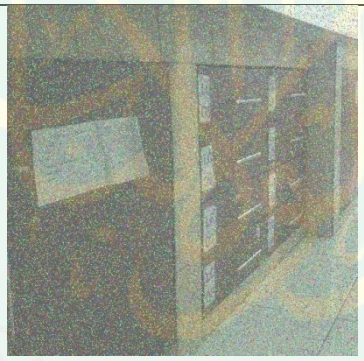
Vasicek Zdenek, Sekanina Lukas. 2008. *Novel Hardware Implementation of Adaptive Median Filters*. Design and Diagnostics of Electronic Circuits and Systems DDECS 11th IEEE

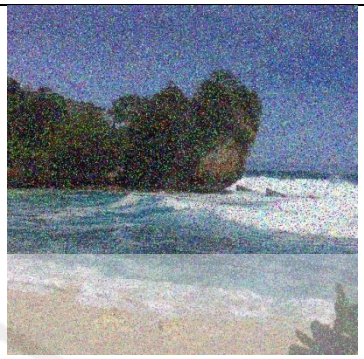
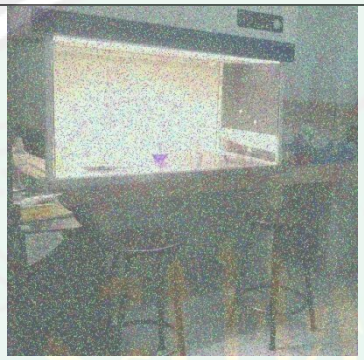
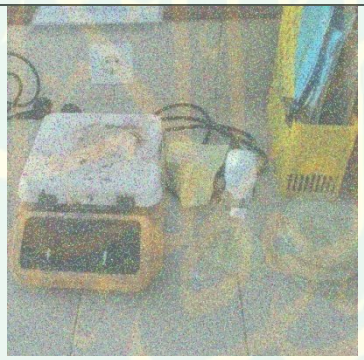
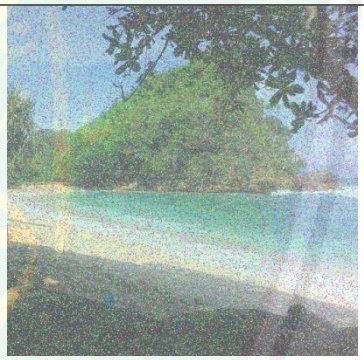
Zhang S., Karim M.A. 2002. *A New Impulse Detector for Switching Median Filter*. IEEE Signal Processing Letters, vol. 9, no. 11, pp. 360-363

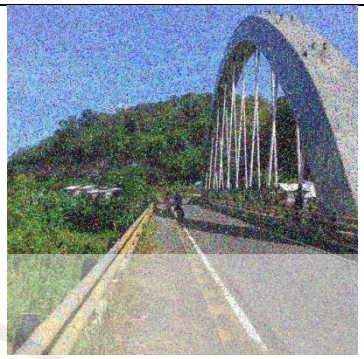
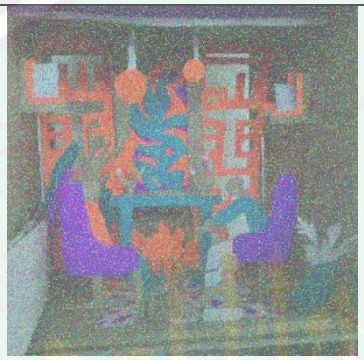
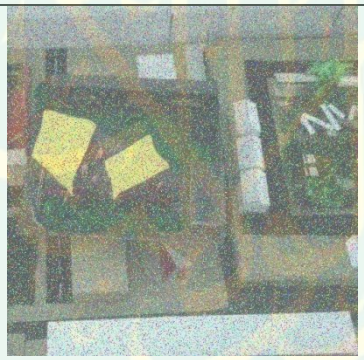
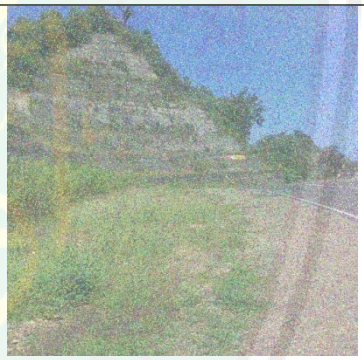
**Lampiran Data Uji Citra**

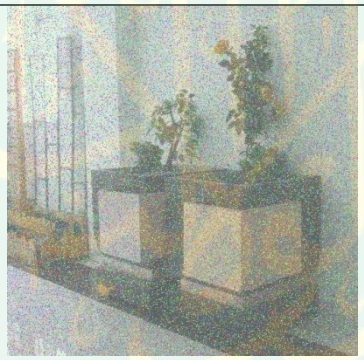
**1. Data Citra dengan Salt and Pepper Noise 10%**

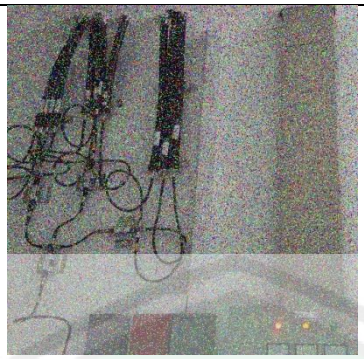
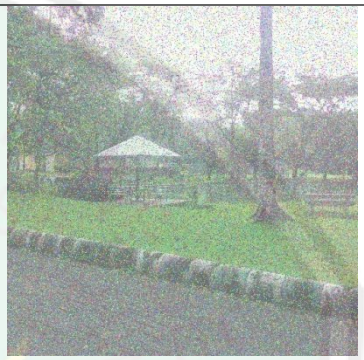
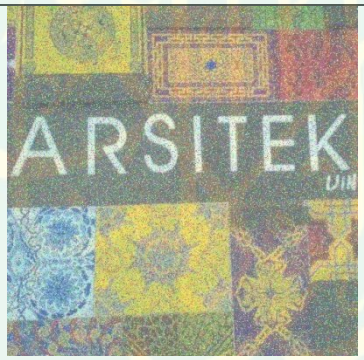
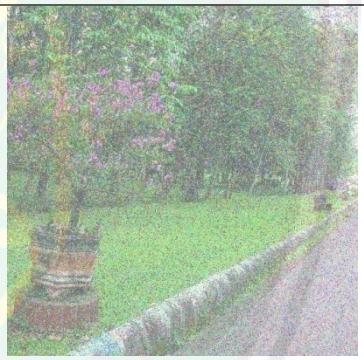
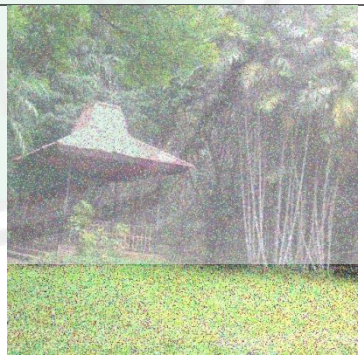
| | Nama Data | Citra *Indoor* | Citra*Outdoor* |
|---|---|---|---|
| 1 | **sp10 A1** |  |  |
| 2 | **sp10 A2** |  |  |
| 3 | **sp10 A3** |  |  |
| 4 | **sp10 A4** |  |  |

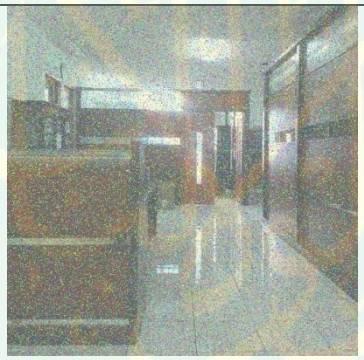
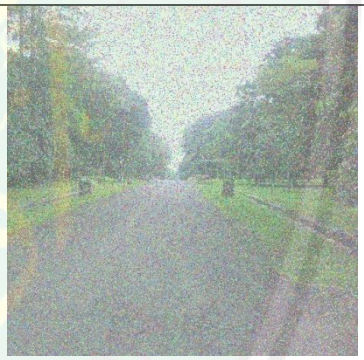
| | | | |
|---|---|---|---|
| 5 | **sp10 A5** |  |  |
| 6 | **sp10 A6** |  |  |
| 7 | **sp10 A7** |  |  |
| 8 | **sp10 A8** |  |  |

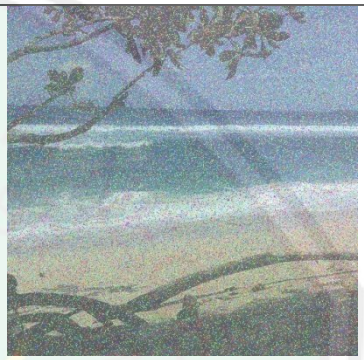
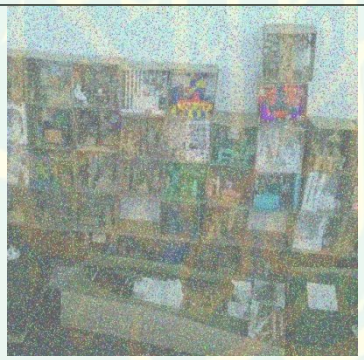
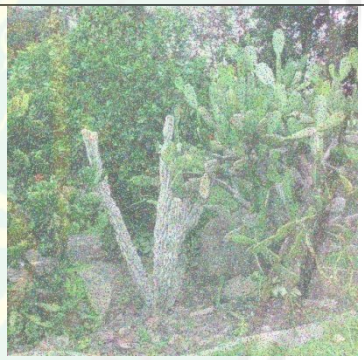
| | | | |
|---|---|---|---|
| 9 | **sp10 A9** |  |  |
| 10 | **sp10 A10** |  |  |
| 11 | **sp10 A11** |  |  |
| 12 | **sp10 A12** |  |  |

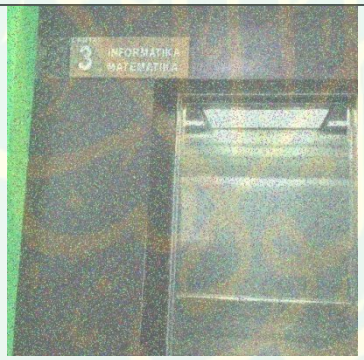
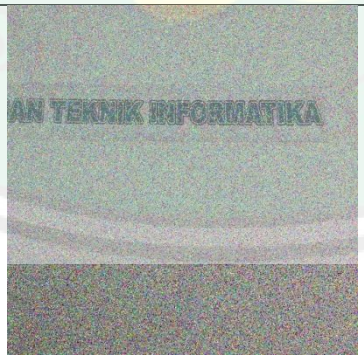
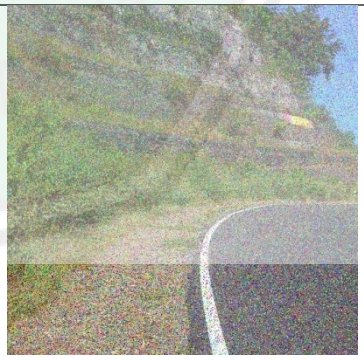
| | | | |
|---|---|---|---|
| 13 | **sp10 A13** |  |  |
| 14 | **sp10 A14** |  |  |
| 15 | **sp10 A15** |  |  |
| 16 | **sp10 A16** |  |  |

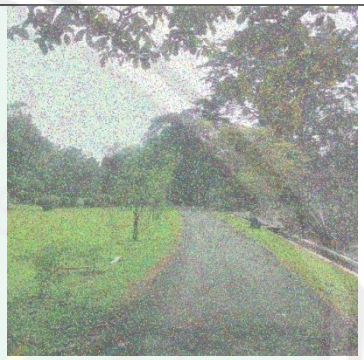
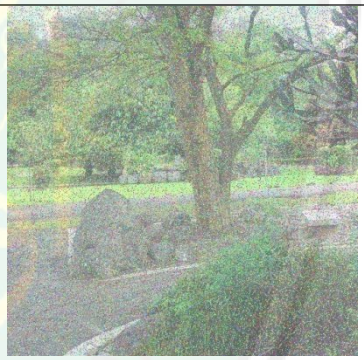
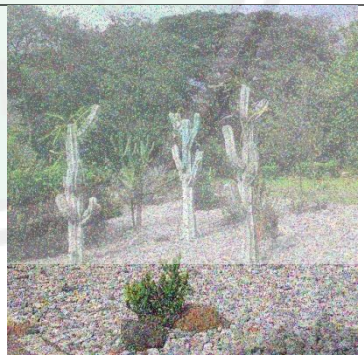
| | | | |
|---|---|---|---|
| 17 | **sp10 A17** |  |  |
| 18 | **sp10 A18** |  |  |
| 19 | **sp10 A19** |  |  |
| 20 | **sp10 A20** |  |  |

| | | | |
|---|---|---|---|
| 21 | **sp10 A21** |  |  |
| 22 | **sp10 A22** |  |  |
| 23 | **sp10 A23** |  |  |
| 24 | **sp10 A24** |  |  |

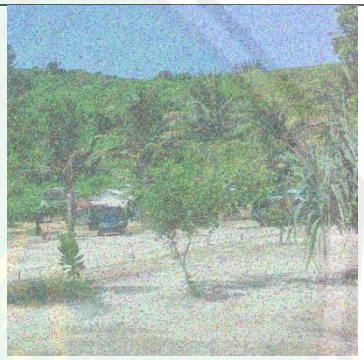
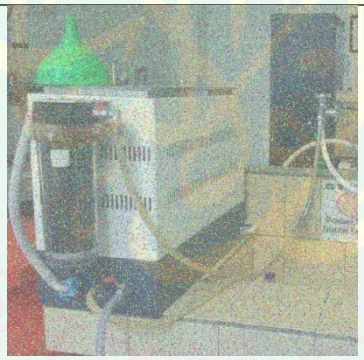
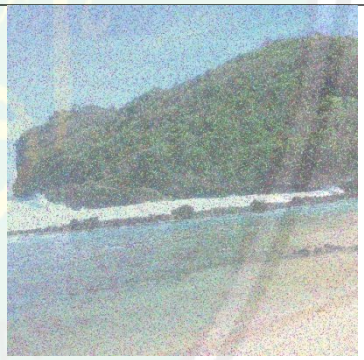
| | | | |
|---|---|---|---|
| 25 | **sp10 A25** |  |   |
| 26 | **sp10 A26** |  |  |
| 27 | **sp10 A27** |  |  |
| 28 | **sp10 A28** |  |  |

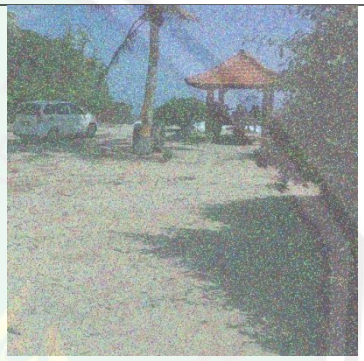
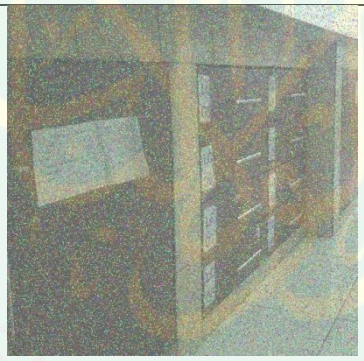
| | | | |
|---|---|---|---|
| 29 | **sp10 A29** |  |  |
| 30 | **sp10 A30** |  |  |
| 31 | **sp10 A31** |  |  |
| 32 | **sp10 A32** |  |  |

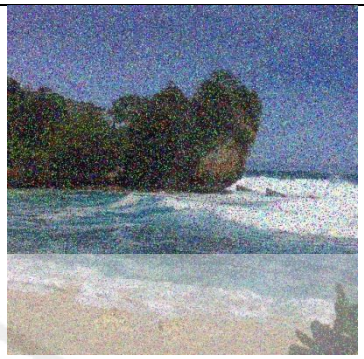
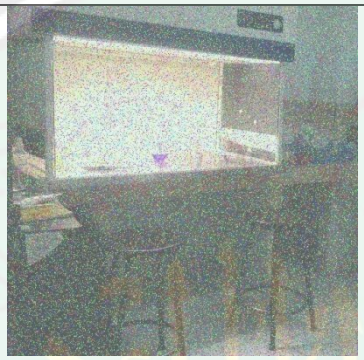
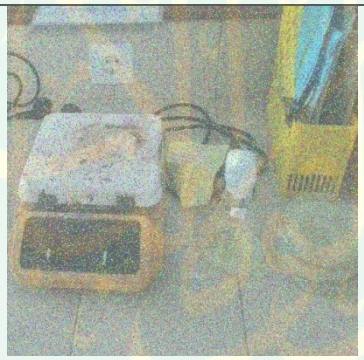
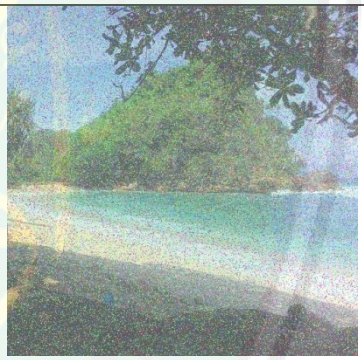
| | | | |
|---|---|---|---|
| 33 | **sp10 A33** |  |  |
| 34 | **sp10 A34** |  |  |
| 35 | **sp10 A35** |  |  |
| 36 | **sp10 A36** |  |  |

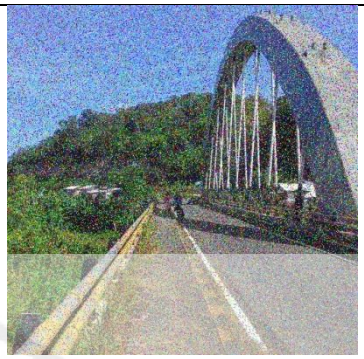
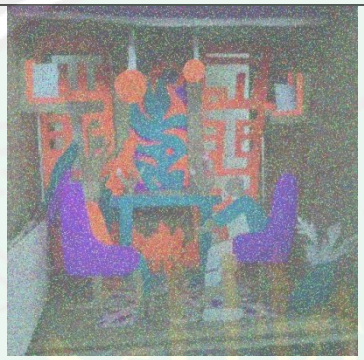
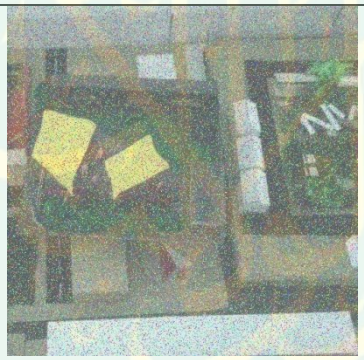
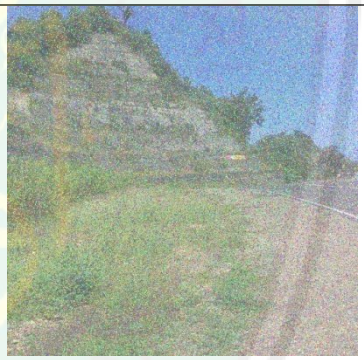
| | | | |
|---|---|---|---|
| 37 | **sp10 A37** |  |  |
| 38 | **sp10 A38** |  |  |
| 39 | **sp10 A39** |  |  |
| 40 | **sp10 A40** |  |  |

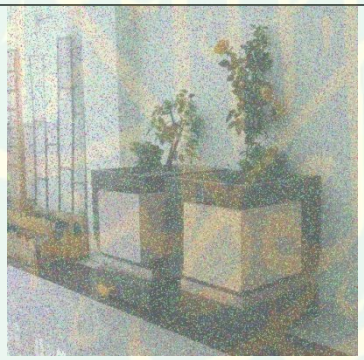
**2. Data Citra dengan Salt and Pepper Noise 20%**

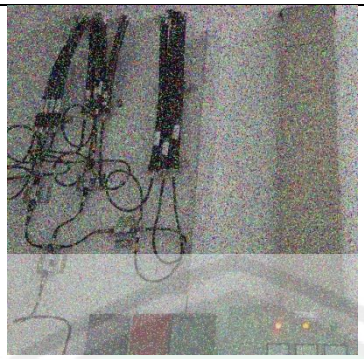
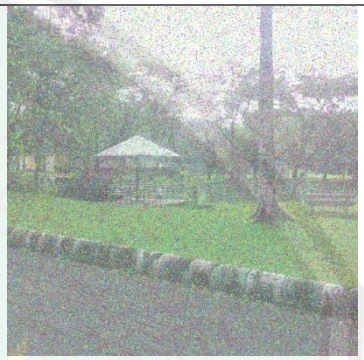
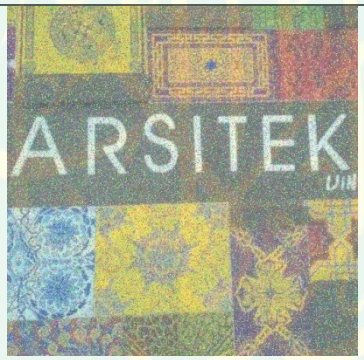
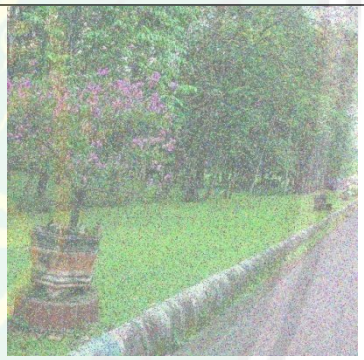
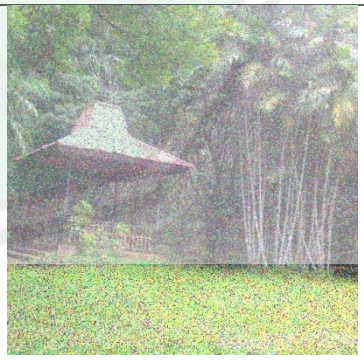
| | Nama Data | Citra *Indoor* | Citra*Outdoor* |
|---|---|---|---|
| 1 | **sp20 A1** |  |  |
| 2 | **sp20 A2** |  |  |
| 3 | **sp20 A3** |  |  |
| 4 | **sp20 A4** |  |  |

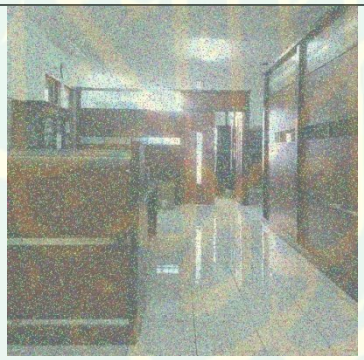
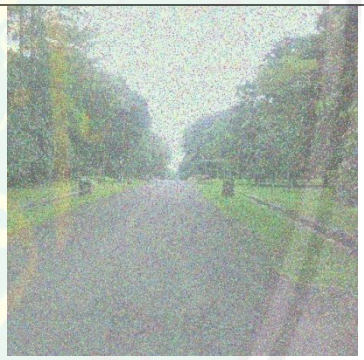
| | | | |
|---|---|---|---|
| 5 | **sp20 A5** |  |  |
| 6 | **sp20 A6** |  |  |
| 7 | **sp20 A7** |  |  |
| 8 | **sp20 A8** |  |  |

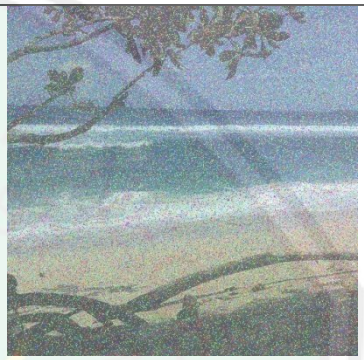
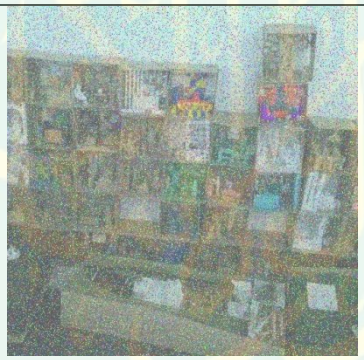
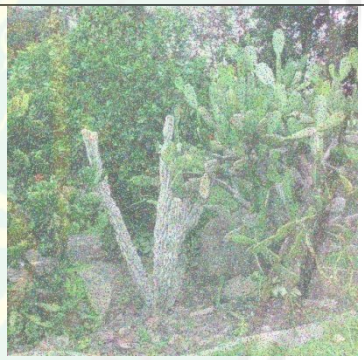
| | | | |
|---|---|---|---|
| 9 | **sp20 A9** |  |  |
| 10 | **sp20 A10** |  |  |
| 11 | **sp20 A11** |  |  |
| 12 | **sp20 A12** |  |  |

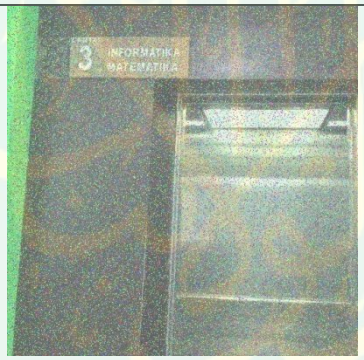
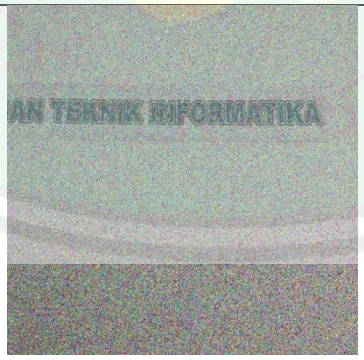
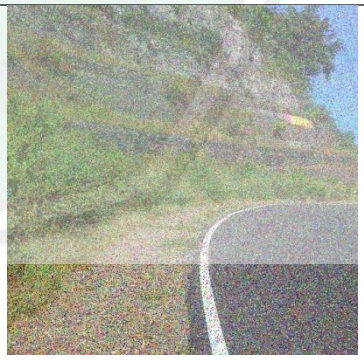
| | | | |
|---|---|---|---|
| 13 | **sp20 A13** |  |  |
| 14 | **sp20 A14** |  |  |
| 15 | **sp20 A15** |  |  |
| 16 | **sp20 A16** |  |  |

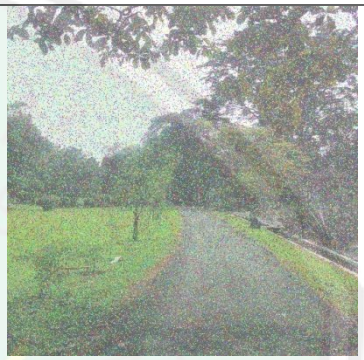
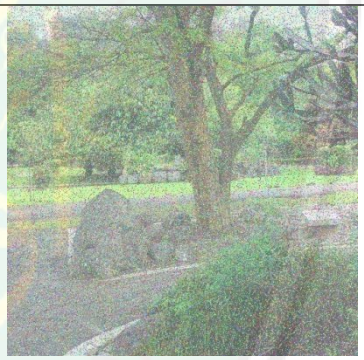
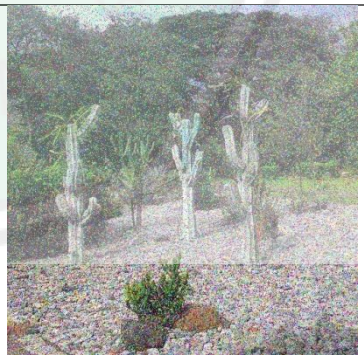
| | | | |
|---|---|---|---|
| 17 | **sp20 A17** |  |  |
| 18 | **sp20 A18** |  |  |
| 19 | **sp20 A19** |  |  |
| 20 | **sp20 A20** |  |  |

| | | | |
|---|---|---|---|
| 21 | **sp20 A21** |  |  |
| 22 | **sp20 A22** |  |  |
| 23 | **sp20 A23** |  |  |
| 24 | **sp20 A24** |  |  |

| | | | |
|---|---|---|---|
| 25 | **sp20 A25** |  |  |
| 26 | **sp20 A26** |  |  |
| 27 | **sp20 A27** |  |  |
| 28 | **sp20 A28** |  |  |

| | | | |
|---|---|---|---|
| 29 | **sp20 A29** |  |  |
| 30 | **sp20 A30** |  |  |
| 31 | **sp20 A31** |  |  |
| 32 | **sp20 A32** |  |  |

| | | | |
|---|---|---|---|
| 33 | **sp20 A33** |  |  |
| 34 | **sp20 A34** |  |  |
| 35 | **sp20 A35** |  |  |
| 36 | **sp20 A36** |  |  |

| | | | |
|---|---|---|---|
| 37 | **sp20 A37** |  |  |
| 38 | **sp20 A38** |  |  |
| 39 | **sp20 A39** |  |  |
| 40 | **sp20 A40** |  |  |