BAB II
TINJAUAN PUSTAKA & LANDASAN TEORI

II.1. Tinjauan Pustaka

Pengolahan citra sudah ada sejak dahulu, pengolahan citra dilakukan untuk meningkatkan kualitas dari suatu citra atau bahkan memodifikasi bagian tertentu pada citra. Salah satu pengolahan citra yang bisa dibilang sudah tua adalah deteksi tepi menggunakan diffusi anisotropik (Perona & Malik, 1990). Pada tahun 1990 Pietro Perona dan Jitendra Malik mengembangkan sebuah algoritma diffusi anisotropik yang digunakan untuk melakukan proses deteksi tepi pada citra. Algoritma ini dikenal dengan nama *Perona-Malik Equation*. Selain untuk deteksi tepi, algoritma tersebut sangat berpotensi untuk *image segmentation, noise removing, image enhancement*, dan *image restoration* (Wei, 1999). Salah satu bentuk *image restoration* adalah *inpainting*.

Bertalmio mengembangkan teknik *inpainting* tersebut dengan menggunakan transformasi Laplacian(*Heat*). Setelah dipelopori oleh Bertalmio pada tahun 2000, beberapa peneliti yang tertarik pada bidang *computer vision* mengadopsi teknik *inpainting* untuk mengembangkan metode tersebut menjadi lebih baik. Pada tahun 2001 algoritma yang dikenal sederhana, cepat dan digunakan untuk menyelesaikan masalah *inpainting* dikenalkan oleh Oliveira dan kawan-kawan yang dikenal dengan *Fast*

Digital Image Inpainting (Oliveira et al., 2001), pada tahun 2003 Cant mencoba untuk memecahkan masalah *inpainting* dengan metode multi skala untuk otomatisasi *inpainting* (Cant, 2003). Pada tahun 2004 Criminisi dan Toyama memperkenalkan sebuah algoritma *inpainting* yang digunakan untuk menghilangkan obyek besar yang didasarkan pada perhitungan warna menggunakan exemplar sintesis (Criminisi et al., 2004), masih pada tahun 2004 Telea mengembangkan algoritma *inpainting* yang didasarkan pada apa yang disebut *Level Set and Fast Marching Method* (Telea, 2004). Kemudian pada tahun 2005 Grossauer dan Scherzer menggunakan teknik persamaan Ginzburg-Landau pada ruang 2 dimensi dan 3 dimensi untuk melakukan *inpainting* (Grossauer & Scherzer, 2005).

Pada tahun 2006, telah diluncurkan sebuah bahasa pemrograman baru oleh NVIDIA dengan nama CUDA(*Compute Unified Device Architecture*). CUDA adalah sebuah *API* yang digunakan untuk mengoperasikan GPU untuk melaksanakan sebuah komputasi secara paralel. Pada awalnya komputasi paralel hanya dapat dilakukan dengan menggunakan *grid computing*, di mana suatu pekerjaan didistribusikan ke banyak komputer yang saling terhubung melalui jaringan. Sekarang hal tersebut tidak menjadi masalah lagi, berkat adanya CUDA yang dapat mengoperasikan GPU dengan banyak *core* dalam sebuah komputer saja, sehingga proses komputasi dapat berjalan lebih cepat.

CUDA sangat berpotensi untuk dimanfaatkan dalam bidang pengolahan citra terutama untuk memproses citra dengan algoritma tertentu menggunakan komputasi paralelnya. Dalam penelitian ini, penulis akan mengembangkan *image inpainting* yang mengimplementasikan algoritma *Heat* dengan sedikit modifikasi pada algoritmanya, yaitu mengganti algoritma Laplacian dengan algoritma *Perona-Malik*. Keseluruhan proses tersebut akan dilakukan secara paralel dengan memanfaatkan GPU CUDA, sehingga diharapkan dapat meningkatkan kecepatan proses komputasinya.

II.2. Landasan Teori

II.2.1. Citra Digital

Citra merupakan gambar yang merepresentasikan sesuatu. Citra dapat berupa gambar dari seseorang, orang banyak atau hewan, atau suatu pemandangan luar, atau *microphotograph* dari suatu komponen elektronik atau juga hasil dari pencitraan medis (McAndrew, 2004).

Citra digital merupakan citra yang dapat diolah oleh komputer, citra digital memiliki informasi berupa posisi dan warna (Solomon et al., 2010). Citra digital dapat diwakili oleh sebuah matriks yang terdiri dari kolom dan baris, dimana perpotongan antara kolom dan baris disebut piksel. Piksel adalah elemen terkecil dari sebuah citra. Setiap piksel pada citra menyimpan

dua buah nilai atau data, yaitu data lokasi kordinat piksel(posisi) dan data intensitas(warna).

Citra digital dapat dibagi kedalam dua jenis, yaitu citra jenis *raster* dan citra jenis *vector*. Citra *raster* adalah seperti yang telah dijelaskan sebelumnya, yaitu citra yang diwakili oleh sebuah matriks, dan memiliki informasi pada setiap perpotongan antara kolom dan barisnya yang disebut dengan piksel. Sedangkan citra *vector* adalah citra berdasarkan bentuk geometri, citra yang dibentuk oleh titik, garis, dan lainnya dengan basis ekpresi matematika.

II.2.2. Pengolahan Citra

Pengolahan Citra merupakan sebuah proses untuk meningkatkan kualitas citra dan mengekstrak informasi penting pada citra tersebut (Saxena et al., 2013). Pengolahan citra juga dapat disebut sebuah ilmu pengetahuan yang mempelajari tentang manipulasi citra digital yang meliputi beberapa teknik dalam meningkatkan atau bahkan mengubah citra tersebut (Crane, 1997). Pengolahan citra dapat diaplikasikan ke beberapa kasus dasar, seperti *image representation and modeling, image enhancement, image restoration, image analysis, image reconstruction, dan image data compression* (Jain, 1989).

Image representation and modeling terkait dengan peranan dari kuantitas setiap piksel yang menggambarkan

sebuah citra. Citra merepresentasikan cahaya dari objek yang berada di alam(seperti memotret pemandangan), penyerapan karakteristik dari organ tubuh manusia(seperti *X-ray imaging*), dan lain sebagainya. *Image enhancement* terkait dengan peningkatan kontras dan tepi pada citra, *pseudocoloring, noise filtering, sharpening, dan magnifying*.

Image restoration berkenaan dengan menghapus atau meminimalisir degradasi dari sebuah citra. Terkait juga *deblurring* dari citra yang terdegradasi oleh batas lingkungan, *noise filtering*, dan pembetulan dari penyimpangan geometris. *Image analysis* terkait dengan membuat ukuran kuantitatif dari sebuah citra untuk dideskripsikan kedalam bentuk yang lebih sederhana. *Image reconstruction* adalah sebuah kelas yang spesial dari *image restoration*, dimana objek dua dimensi direkonstruksi dari beberapa proyeksi satu dimensi, dan *image data compression* adalah banyaknya data yang berhubungan dengan informasi visual.

II.2.3. OpenCV

OpenCV adalah *open source library* dari fungsi pemrograman yang terutama fungsi ini ditujukan untuk keperluan pengolahan citra secara *real-time*, dimana OpenCV ini dikembangkan oleh pusat penelitian Intel Rusia di Nizhny Novgorod, dan sekarang didukung oleh Willow Garage dan Itseez. OpenCV memiliki lebih dari

500 algoritma yang sudah dioptimalkan untuk pengolahan citra dan video (Laganiere, 2011).

OpenCV diciptakan atau dibangun dengan sangat kuat dan cukup bagus untuk menyelesaikan kebanyakan masalah *computer vision* yang sudah ditetapkan. Hal yang dapat dilakukan adalah memotong citra(*crop*), meningkatkan kualitas citra dengan memodifikasi kecerahan, ketajaman, dan kontras, mendeteksi bentuk, segmentasi citra, mendeteksi objek bergerak, mengenali objek yang sudah diketahui, dan masih banyak lagi (Brahmbhatt, 2013). Sebuah aspek dari modul openCV membutuhkan ketegasan untuk mencapai tingkat optimasi yang tinggi. Hal tersebut dimaksudkan untuk aplikasi *real-time* dan dirancang untuk proses eksekusi yang sangat cepat. OpenCV menyediakan beberapa modul yang dapat digunakan langsung sesuai fungsionalitasnya yang dipaparkan pada tabel dibawah ini.

Module	Functionality
Core	Core data structures, data types, and memory management
Imgproc	Image filtering, geometric image transformations, structure, and shape analysis
Highgui	GUI, reading and writing images and video
Video	Motion analysis and object tracking in video
Calib3d	Camera calibration and 3D reconstruction from multiple views
Features2d	Feature extraction, description, and matching
Objdetect	Object detection using cascade and histogram-of-gradient classifiers
ML	Statistical models and classification algorithms for use in computer vision applications
Flann	Fast Library for Approximate Nearest Neighbors—fast searches in high-dimensional (feature) spaces
GPU	Parallelization of selected algorithms for fast execution on GPUs
Stitching	Warping, blending, and bundle adjustment for image stitching
Nonfree	Implementations of algorithms that are patented in some countries

Tabel 2.1. Modul dan fungsi openCV

II.2.4. Image Inpainting dengan Metode Perona-Malik

Bertalmio mengatakan *Inpainting* adalah suatu teknik untuk memodifikasi citra dalam bentuk yang tidak terdeteksi, dalam hal seni murni (lukis) (Bertalmio et al., 2000). Tujuan dan aplikasi dari kebanyakan *inpainting*, mulai dari restorasi lukisan dan foto yang rusak untuk menghilangkan atau mengganti objek yang dipilih.

II.2.4.1. Image Inpainting

Seperti yang telah dijelaskan pada latar belakang, *inpainting* adalah sebuah seni dalam merestorasi lukisan, dan kini telah dikembangkan dalam dunia digital oleh Bertalmio. Dalam disertasi (Mahalingam, 2010) bagian citra yang hilang atau rusak pada *digital inpainting* sering disebut dengan *hole*(lobang), dan biasanya disediakan oleh pengguna dalam bentuk *mask* atau diperoleh dengan cara yang otomatis atau semi-otomatis. Tujuan terakhir dari *inpainting* adalah untuk menyusun kembali bagian citra yang hilang atau rusak dan membuatnya lebih mudah dibaca serta mengembalikannya menjadi satu kesatuan citra yang lebih baik.

Algoritma differensial parsial(*partial differential equation*) berdasarkan pada algoritma iteratif yang diusulkan oleh Bertalmio et al (2000)

telah membuka jalan untuk *digital image inpainting*. Proses algoritma *iterative* ini yaitu menyebar ke sekeliling area yang dioperasikan dan mengumpulkan setiap nilai yang ada disekelilingnya lalu mengisi pada bagian *hole* setelah diproses oleh algoritma (Mahalingam, 2010). Persamaan algoritma inpainting tertulis pada persamaan 2.1.

$$I^{n+1}(i,j) = I^{n}(i,j) + \Delta t . I_t^n(i,j), \forall (i,j) \in \Omega \qquad (2.1)$$

Dimana n adalah indeks iterasi, i dan j adalah kordinat piksel citra, Δt nilai perubahan *inpainting*, dan $I_t^n(i,j)$ adalah faktor penentu perubahan pada citra. Faktor penentu pada lagoritma diatas adalah sebuah citra yang sudah dihaluskan dengan menggunakan perhitungan Laplacian. Perhitungan didapatkan dari daerah *isophote* yaitu daerah disekitar piksel yang dioperasikan. Untuk itu persamaan 2.1 yang juga dapat dikatakan sebagai algoritma *Heat* dapat ditulis seperti persamaan 2.2.

$$I^{n+1}(i,j) = I^{n}(i,j) + \Delta t[\ (I^{n}(i+1,j) + I^{n}(i-1,j) + I^{n}(i,j+1) + I^{n}(i,j-1) - 4I^{n+1}(i,j)\ ) + \lambda^{n}(i,j)(u^{n}(i,j) - I^{n}(i,j))] \qquad (2.2)$$

Dimana $I(i,j)$ adalah citra hasil proses inpainting, $u(i,j)$ adalah citra yang rusak, dan $\lambda(i,j)$ adalah citra *mask*.

II.2.4.2. Metode Perona-Malik

Perona-Malik adalah salah satu persamaan yang cocok digunaan untuk proses *filtering*. Setiap piksel tetangga dalam persamaan diffusi anisotropik adalah beberapa set piksel yang sudah didefinisikan oleh lokasi secara *relative* sesuai dengan piksel yang sedang dioperasikan. Pietro Perona dan Jitendra Malik adalah pencetus pertama dari persamaan diffusi anisotropik atau yang sering disebut dengan *Perona-Malik Equation* pada tahun 1990 (Perona & Malik, 1990). Ide dasar dari diffusi anisotropik adalah meningkatkan kehalusan pada citra digital (Chen et al., 2010). Persamaan diffusi anisotropik tertulis pada persamaan 2.3.

$$I_t = div\,(c(x,y,t)\nabla I) = (c(x,y,t)\Delta I + \nabla c\,.\,\nabla I) \qquad (2.3)$$

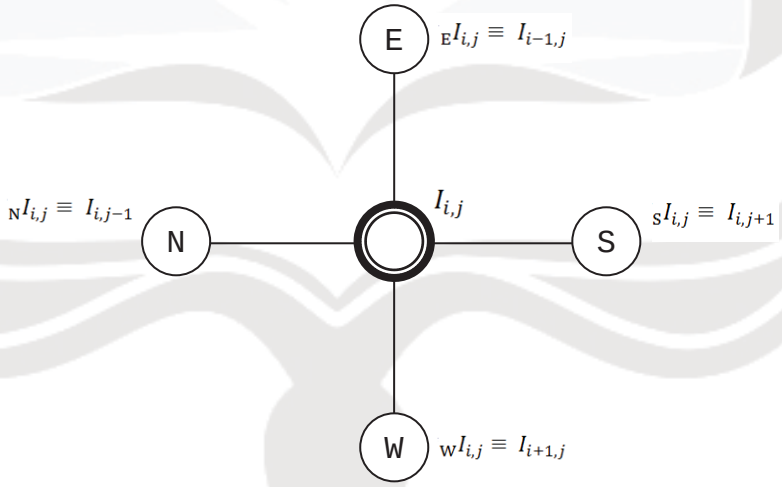
Awal dari persamaan *Perona-Malik* adalah dari diskretisasi operator laplacian. Persamaan tersebut tertulis pada persamaan 2.4.

$$I_{i,j}^{n+1} = I_{i,j}^{n} + \Delta t\,[c_N.\nabla_{\mathrm{N}} I + c_S.\nabla_{\mathrm{S}} I + c_E.\nabla_{\mathrm{E}} I + c_W.\nabla_{\mathrm{W}} I]_{i,j}^{n} \qquad (2.4)$$

Dimana n adalah indeks iterasi, lalu Δt adalah nilai konstanta permanen, dan rumus didalam tanda “[ ]”

adalah 4 tetangga terdekat, dimana ∇ adalah selisih nilai dari tetangga terdekat(N,S,E,W). Selisih dari persamaan tersebut adalah tertulis pada persamaan 2.5 dan dapat digambarkan strukturnya pada gambar 2.1.

$$\begin{aligned}
\nabla_{\mathrm{N}} I_{i,j} &\equiv I_{i,j-1} - I_{i,j} \\
\nabla_{\mathrm{S}} I_{i,j} &\equiv I_{i,j+1} - I_{i,j} \\
\nabla_{\mathrm{E}} I_{i,j} &\equiv I_{i-1,j} - I_{i,j} \\
\nabla_{\mathrm{W}} I_{i,j} &\equiv I_{i+1,j} - I_{i,j}
\end{aligned} \tag{2.5}$$



Gambar 2.1. struktur daerah *isophote*

$\nabla_{\mathrm{N}} I_{i,j}$ adalah selisih nilai dengan tetangga disebelah kiri(utara), $\nabla_{S} I_{i,j}$ adalah selisih nilai dengan tetangga

disebelah kanan(selatan), $\nabla_{\mathrm{E}} I_{i,j}$ adalah selisih nilai dengan tetangga disebelah atas(timur), dan $\nabla_{\mathrm{W}} I_{i,j}$ adalah selisih nilai dengan tetangga disebelah bawah(barat). Kemudian c pada persamaan 2.4 adalah yang nanti akan menentukan apakah persamaan akan dibuat secara linier atau nonlinier tergantung dari nilai yang dimasukan pada variable tersebut.

Jika nilai yang dimasukkan pada c adalah konstanta maka diperoleh persamaan diffusi linier, tetapi jika nilai c diisi dengan sebuah fungsi *I(x,y)* maka diperoleh persamaan diffusi nonlinier yaitu ada dua persamaan diffusi *Perona-Malik* seperti tertulis pada persamaan 2.6.

$$c_{PM1} = \exp\left(-(|\nabla I| k_{PM})^2\right)$$

$$c_{PM2} = \frac{1}{(1+(|\nabla I| k_{PM})^2)} \qquad (2.6)$$

Koefisien-koefisien dari kedua persamaan diatas adalah sebagai berikut yang tertulis pada persamaan 2.7 dan 2.8.

$$c_N = \exp\left(-(|I_{i,j-1} - I_{i,j}| k_{PM})^2\right)$$

$$c_S = \exp\left(-(|I_{i,j+1} - I_{i,j}| k_{PM})^2\right)$$

$$c_E = \exp\left(-(|I_{i-1,j} - I_{i,j}|k_{PM})^2\right)$$

$$c_W = \exp\left(-(|I_{i+1,j} - I_{i,j}|k_{PM})^2\right) \quad (2.7)$$

$$c_N = \frac{1}{(1+(|I_{i,j-1} - I_{i,j}|k_{PM})^2)}$$

$$c_S = \frac{1}{(1+(|I_{i,j+1} - I_{i,j}|k_{PM})^2)}$$

$$c_E = \frac{1}{(1+(|I_{i-1,j} - I_{i,j}|k_{PM})^2)}$$

$$c_W = \frac{1}{(1+(|I_{i+1,j} - I_{i,j}|k_{PM})^2)} \quad (2.8)$$

Seperti yang telah dijelaskan pada latar belakang, bahwa dalam penelitian ini akan diteliti penerapan metode *Perona-Malik* pada algoritma *inpainting* dengan mengganti operator laplacian. Kedua algoritma tersebut telah dikombinasikan dan menjadi seperti persamaan 2.9, dimana $c_{N,S,E,W}$ adalah nilai dari metode Perona-Malik 1 atau Perona Malik 2 yang dapat dilihat pada persamaan 2.7 dan 2.8, serta $\nabla_{\mathrm{N,S,E,W}} I_{i,j}$ adalah nilai selisih daerah *isophote* yang dapat dilihat pada persamaan 2.5.

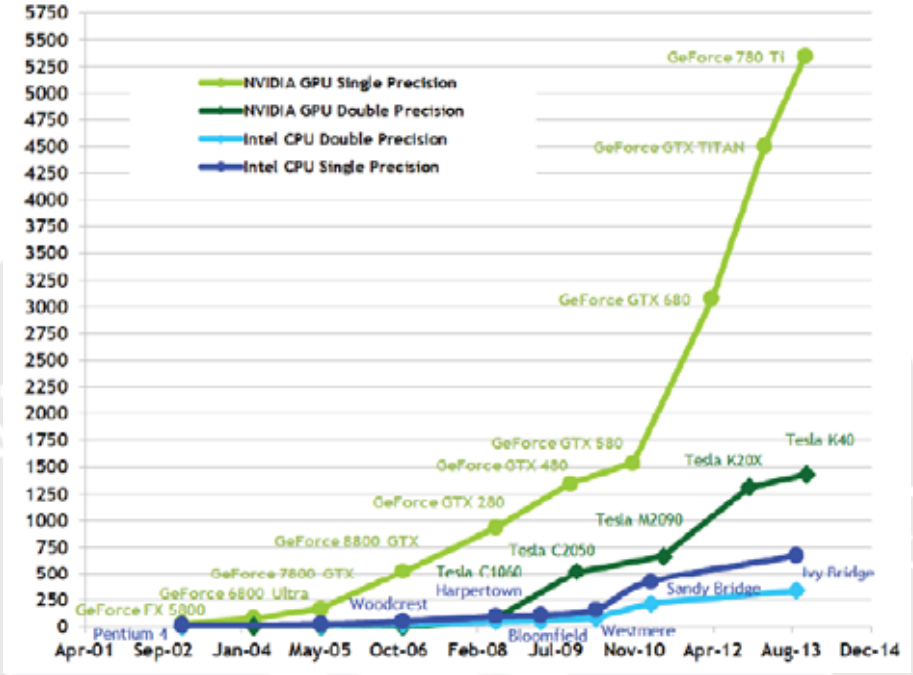
$$I^{n+1}(i,j) = I^n(i,j) + \Delta t\Big[\left((c_N * \nabla_{\mathrm{N}} I_{i,j}) + (c_S * \nabla_{\mathrm{S}} I_{i,j}) + (c_E * \nabla_{\mathrm{E}} I_{i,j}) + (c_W * \nabla_{\mathrm{W}} I_{i,j})\right) + \lambda^n(i,j)(u^n(i,j) - I^n(i,j))\Big] \quad (2.9)$$

II.2.5. Komputasi Paralel

Komputasi paralel adalah ide yang sederhana. Otak manusia benar-benar bekerja secara paralel. Konsep dasar dari komputasi paralel adalah mendistribusikan beban kerja diantara individual prosesor yang bekerja bersama-sama dalam melakukan komputasi. Bagaimanapun, sebuah *single computer* hanya dapat melakukan satu hal pada satu waktu secara sekuensial. Sebuah *parallel computer*, dimana komputer tersebut dapat melakukan banyak komputasi pada waktu yang bersamaan, dan dapat menyelesaikan masalah sederhana dalam beberapa menit saja yang seharusnya diselesaikan dengan membutuhkan waktu jam atau hari pada *single computer* (Jackson, 2009).

Sejak tahun 2003 Terdapat 2 arsitektur dalam mengembangan mikroprosesor, yaitu *multicore* dan *many-core* (Kirk & Hwu, 2010). Arsitektur *multicore* bermula dari prosesor berinti dua dan terus bertambah hingga saat ini sudah mencapai 8 inti pada satu mikroprosesor. Arsitektur *multicore* banyak digunakan pada *central processing unit* (CPU). Arsitektur ini berfokus untuk memberikan performa yang baik pada program yang didesain untuk berjalan pada banyak inti, dan juga tetap menjaga performa kecepatan eksekusi program yang didesain secara sekuensial. Arsitektur *many-core* berfokus untuk memaksimalkan kinerja program yang benar-benar didesain untuk bekerja secara paralel. Arsitektur *many-core* memiliki lebih banyak unit pemroses daripada arsitektur *multicore* dan arsitektur

ini biasanya digunakan pada *graphic processing unit* (GPU).



Gambar 2.2. Perbandingan Performa CPU dan GPU

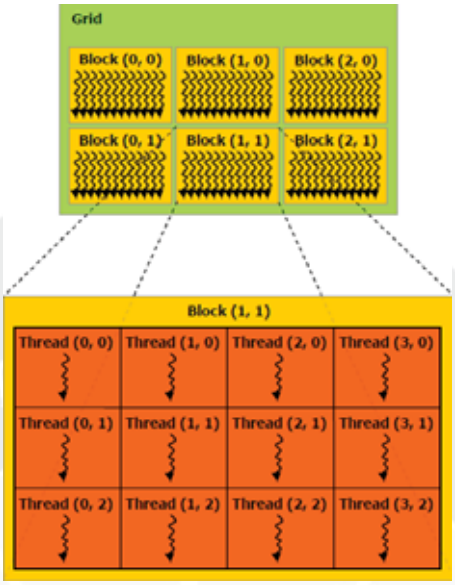
Perbedaan kecepatan antara CPU dan GPU disebabkan karena GPU dikhususkan untuk perhitungan yang intensif dan paralel (NVIDIA, 2014), tetapi GPU tidak dapat menggantikan CPU sepenuhnya karena GPU hanya didesain untuk melakukan kalkulasi numerik dan untuk memproses secara paralel saja, bukan sekuensial.

II.2.6. Nvidia CUDA

CUDA (*Compute Unified Device Architecture*) merupakan API yang dikembangkan oleh NVIDIA. CUDA adalah model pemrograman untuk komputasi paralel yang memungkinkan peningkatan dalam kinerja komputasi dengan

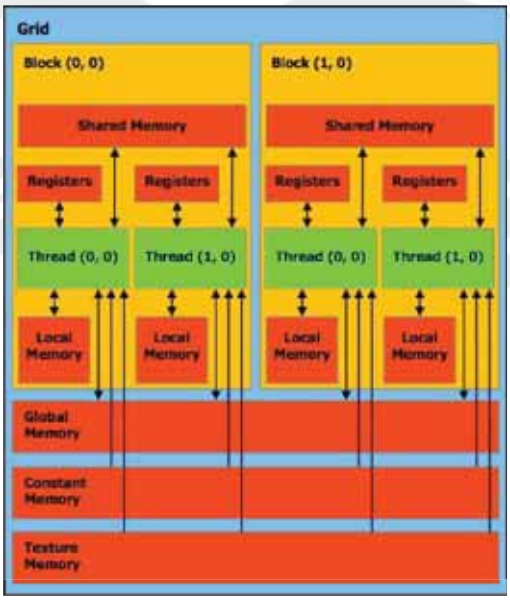
memanfaatkan kelebihan dari GPU (NVIDIA, 2014). Pada tahun 2006 , NVIDIA meluncurkan bahasa pemrograman yang bernama CUDA, dan bahasa pemrograman tersebut secara khusus untuk melakukan komputasi yang umum atau sering disebut dengan GPGPU(*General Purpose Graphic Processing Unit*). CUDA adalah ekstensi dari bahasa pemrograman C yang ditambahkan dengan beberapa sintax untuk bekerja dengan GPU (Jackson, 2009).

CUDA pada dasarnya terbagi menjadi dua bagian utama , yaitu *Thread* dan *Block(Grid)*. *Thread* memiliki *memory* sendiri dan akan mengerjakan unit pekerjaan yang kecil dan akan berjalan secara bersamaan dengan *thread* lain, sehingga waktu yang dibutuhkan untuk mengerjakan pekerjaan seluruhnya akan menjadi lebih singkat. *Thread-thread* tersebut dikelompokkan menjadi *block*, yaitu kumpulan *thread* yang memiliki satu *memory* yang dapat digunakan oleh setiap *thread* dalam *block* tersebut secara bersama-sama untuk media komunikasi antar thread tersebut yang dinamakan dengan *shared memory*. Setiap *block* tersebut akan dikelompokkan lagi menjadi sebuah *grid* yang merupakan kumpulan dari semua *block* yang digunakan dalam suatu komputasi. Dalam proses komputasi terdapat proses antrian data yang disebut sebagai *warp*. *Warp* memiliki efek pada kinerja GPU, sehingga jumlah *block* dan *thread* yang digunakan memiliki efek pada kecepatan proses komputasi.



Gambar 2.3. Struktur Unit Pemroses pada CUDA

CUDA memiliki beberapa macam *memory* yang dapat digunakan untuk proses komputasi yaitu; *Global Memory, Shared Memory, Texture Memory, Register, Local Memory,* dan *Constant Memory*.



Gambar 2.4. Struktur Memori pada CUDA

II.2.7. Penilaian Kualitas Citra

II.2.7.1. Mean Squared Error

Mean Squared Error (MSE) adalah algoritma validasi citra yang paling sederhana, dan paling banyak digunakan, MSE adalah rekomendasi pengukuran kualitas citra yang cukup baik (Varnan et al., 2011). Algoritma ini paling sering digunakan untuk *signal processing*. Persamaan MSE dapat dilihat dibawah ini :

$$MSE = \frac{1}{MN}\sum_{i=1}^{M}\sum_{j=1}^{N}(x(i,j) - y(i,j))^2 \quad (2.10)$$

Dimana x(i, j) merepresentasikan citra yang asli dan y(i, j) merepresentasikan citra yang sudah dimodifikasi atau yang sudah dikenai proses pengolahan citra, serta i dan j adalah posisi piksel dari citra M×N. MSE bernilai 0 ketika x(i, j) = y(i, j).

CONTOH PENGUJIAN :

Original Image | Compressed Image | Contrast Image | Blurred Image | Gaussian Noise Image

IMAGE	MSE
Original Image	0
Compressed Image	35.4032
Contrast Image	3.0664
Blurred Image	30.0145
Gaussian Noise Image	49.2451

Tabel 2.2. contoh perbandingan uji MSE

II.2.7.2. Peak Signal to Noise Ratio

PSNR dievaluasi dalam nilai desibel dan berbanding terbalik dengan algoritma MSE (Varnan et al., 2011). Jika nilai MSE semakin kecil semakin bagus kualitas citra tersebut, maka PSNR semakin besar nilainya maka semakin bagus kualitas citra tersebut. persamaannya PSNR dapat dilihat dibawah ini :

$$PSNR = 10\log_{10}\frac{(2^n-1)^2}{\sqrt{MSE}} \qquad (2.11)$$

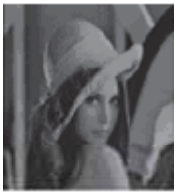
Dimana MSE adalah seperti yang telah dijelaskan sebelumnya dan dapat dilihat persamaannya pada persamaan 2.10.

CONTOH PENGUJIAN :


Original Image


Compressed Image


Contrast Image


Blurred Image


Gaussian Noise Image

IMAGE	PSNR
Original Image	∞
Compressed Image	40.3856
Contrast Image	45.6976
Blurred Image	40.7441
Gaussian Noise Image	39.6690

Tabel 2.3. contoh perbandingan uji PSNR

II.2.7.3. Structural Similarity Index Metric

SSIM adalah algoritma validasi citra yang paling bagus daripada MSE dan PSNR (Varnan et al., 2011). SSIM dikhususkan untuk membandingkan citra dengan ukuran yang sama. Agar dapat membandingkan ukuran citra yang berbeda dikembangkan menjadi MS-SSIM(Multi Scale - SSIM). Persamaan SSIM dapat dilihat dibawah ini :

$$SSIM = \frac{(2\times\bar{x}\times\bar{y}+C1)(2\times\sigma_{xy}+C2)}{(\sigma_x^2+\sigma_y^2+C2)\times((\bar{x})^2+(\bar{y})^2+C1)} \qquad (2.12)$$

Dimana C1 dan C2 adalah nilai konstan permanen, dan $\bar{x}, \bar{y}, \sigma_x^2, \sigma_y^2, \sigma_{xy}$ adalah :

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$$

$$\bar{y} = \frac{1}{N}\sum_{i=1}^{N} y_i$$

$$\sigma_x^2 = \frac{1}{N-1}\sum_{i=1}^{N} (x_i - \bar{x})^2$$

$$\sigma_y^2 = \frac{1}{N-1}\sum_{i=1}^{N} (y_i - \bar{y})^2$$

$$\sigma_{xy} = \frac{1}{N-1}\sum_{i=1}^{N} (x_i - \bar{x})(y_i - \bar{y}) \qquad (2.13)$$

CONTOH PENGUJIAN :

Original Image | Compressed Image | Contrast Image | Blurred Image | Gaussian Noise Image

IMAGE	SSIM
Original Image	1
Compressed Image	0.8719
Contrast Image	0.8323
Blurred Image	0.7606
Gaussian Noise Image	0.1359

Tabel 2.4. contoh perbandingan uji SSIM