BAB III

LANDASAN TEORI

3.1. Citra Digital

Citra digital merupakan sebuah fungsi $f(x, y)$ dua dimensi. Dimana x dan y merepresentasika lokasi dari sebuah *picture element* atau *pixel* dan memiliki nilai intensitas. Ketika nilai dari x, y dan intensitas diciptakan maka terbentuk apa yang dikenal sebagai citra digital. *Pixel* merupakan element terkecil dari suatu citra digital yang dihitung secara digital. *Pixel* tersusun secara berdekatan dengan teratur sehingga terbentuk suatu citra. Semakin banyak dan rapat susunan *pixel* dalam suatu citra maka semakin tajam citra yang diperoleh [1].

Setiap *pixel* memiliki nilai dalam rentang tertentu, mulai dari nilai minimal hingga nilai maksimal. Nilai yang berbeda-beda dari *pixel* ini, sangat tergantung dari warnanya. Secara umum jangkauan dari warna sebuah citra adalah 0-255. Terdapat beberapa jenis citra yang dilihat dari nilai setiap *pixel*-nya, yaitu citra biner, citra *grayscale* dan citra warna.

A. Citra Biner

Citra biner merupakan citra yang nilai intensitasnya terdiri dari 2 intensitas yaitu 0 dan 1. Nilai 0 menyatakan hitam dan nilai 1 menyatakan putih [1]. Berikut contoh citra biner :

Gambar 3. 1. Contoh Citra Biner (Sumber: [11])

B. Citra *Grayscale*

Citra *grayscale* merupakan citra digital yang nilai intensitas RED sama dengan nilai intensitas GREEN sama dengan nilai intensitas BLUE. Pada citra *grayscale* ini terdapat tiga warna yaitu hitam, putih dan keabu-abuan. Warna keabu-abuan disini mulai dari tingkat hitam hingga mendekati putih [1]. Berikut contoh citra *graysclae* :

Gambar 3.2. Contoh Citra *Grayscale* (Sumber: [11])

C. Citra Warna

Citra warna merupakan citra yang setiap *pixel*-nya, memuat informasi mengenai tiga warna sekaligus yaitu warna RED, warna GREEN dan warna BLUE. Ukuran penyimpanan warna dasar ini sebesar 8 bit, sehingga warna yang tersedia 255 untuk masing-masing warna dasar [1]. Berikut contoh citra warna:

Gambar 3.3. Contoh Citra Warna (Sumber : [10])

Dalam citra digital tentu terdapat format *file* citra yang digunakan untuk menyimpan dan menampilkan citra. Format *file* citra yang paling sering ditemukan dalam kehidupan sehari-hari adalah *JPEG*, *Bitmap*, *Tangged Image Format* dan

Portable Network Grapichs. Berikut penjelasan singkat mengenai format *file* tersebut:

1. JPEG(.jpg)

   JPEG merupakan format citra yang paling sering digunakan untuk pengkompresan citra. Format JPEG ini berbasis *bitmap*, sehingga baik untuk digunakan dalam citra panorama maupun foto. Namun kelemahan dari format ini adalah sering hilangnya informasi dari gambar ketika dilakukan perbesaran atau pengecilan pada gambar. Hal ini tentu menyebabkan gambar terlihat seperti kotak-kotak ketika dilakukan proses perbesaran maupun pengecilan [1].

2. *Bitmap*(.bmp)

   Bitmap merupakan format citra yang digunakan untuk penyimpanan standar tanpa kompresi. Dimana format citra ini sering digunakan untuk menyimpan citra biner hingga citra warna [1].

3. *Tangged Image Format(.tif)*

   Tangged Image Format merupakan format penyimpanan citra untuk menyimpan citra *bitmap* hingga citra warna palet terkompresi. Format ini juga dapat digunakan untuk menyimpan citra yang tidak terkompresi dan citra terkompresi [1].

4. *Portable Network Grapichs*(.png)

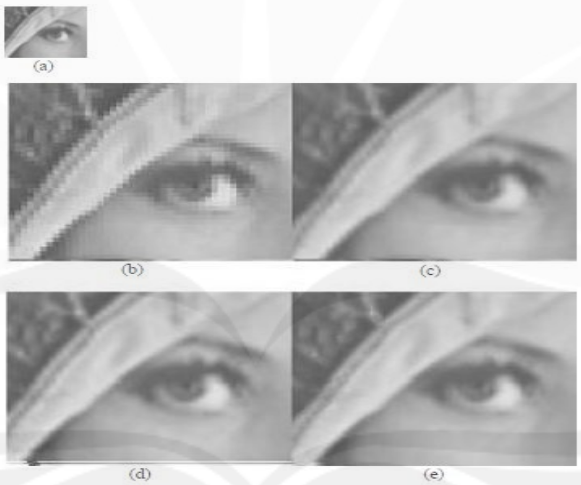
   Portable Network Grapichs merupakan format citra penyimpanan yang terkompresi. Format citra ini digunaan untuk menyimpan citra *grayscale* dan citra warna [1].

3.2. Magnifikasi

Pengolahan citra merupakan suatu proses yang dilakukan oleh komputer untuk mengolah citra tertentu. Dimana masukkannya adalah sebuah citra dan keluarannya adalah sebuah citra yang telah dimanipulasi sesuai kebutuhan. Pengolahan citra bertujuan untuk memperbaiki kualitas citra menjadi lebih baik. Pengolahan citra meliputi proses filtering, konversi citra, perbaikan citra dan magnifikasi [2].

Magnifikasi adalah sebuah proses untuk memperoleh citra dengan resolusi yang tinggi dari citra yang memiliki resolusi rendah. Proses magnifiksi pada sebuah citra bertujuan agar objek pada sebuah citra terlihat lebih jelas dibandingkan sebelum dilakukan proses magnifikasi. Proses magnifikasi pada sebuah citra pun sering terdapat kabur *(bluring)* dan juga bayangan *(shadowing)* yang menyebabkan citra terlihat seperti kotak-kotak [3].

Proses magnifikasi pada citra memiliki banyak manfaat. Selain *output* citra proses magnifikasi menjadi lebih besar dari ukuran semula, proses magnifikasi juga dapat membantu memperjelas sebuah citra. Kejelasan sebuah citra yang terlihat juga sangat bergantung pada metode yang digunakan [4]. Berikut ditampilkan contoh-contoh proses magnifikasi pada citra:



Gambar 3.4. Contoh Proses Magnifikasi pada Citra (Sumber : [3])

3.3.Interpolasi

Syarat yang penting dalam fungsi interpolasi adalah harus ada nilai data pada gambar yang sudah diketahui. Sehingga interpolasi *image* dapat dideskripsikan sebagai proses menggunakan data yang sudah diketahui untuk mengstimasi nilai pada lokasi yang belum diketahui [4]. Interpolasi *image* berhubungan dengan tugas untuk mendapatkan gambar yang beresolusi tinggi dari gambar yang beresolusi lebih rendah. Ada beberapa teknik dalam proses interpolasi tersebut yaitu *linear*, *bilinear*, *spline, bilinear* dan *nearest neighbor* [12].

3.4. Magnifikasi dengan metode interpolasi *Bicubic Basis Spline*

Interpolasi *bicubic* merupakan sebuah metode interpolasi yang menggunakan 16 *pixel* dalam *pixel* 4*x*4 tetangga terdekat pada citra aslinya. Dengan menggunakan metode interpolasi *bicubic* ini dapat membuat tepi-tepi citra hasil lebih halus. Sehingga metode interpolasi *bicubic* sering digunakan dalam pengeditan perangkat lunak dan banyak kamera digital lainnya. Seiring perkembangannya maka mulai dibentuk berbagai metode interpolasi, salah satunya adalah interpolasi *bicubic basis spline,* yang mana metode ini juga memanfaatkan 16 *pixel* tetangga terdekatnya [3].

Konsep metode interpolasi *bicubic basis spline* dan representasinya dalam matematika pertama kali dideskripsikan oleh Schoenberg pada tahun 1946. Menurut definisinya, metode interpolasi *bicubic basis spline* dimaksudkan sebagai potongan-potongan polinominal dengan potongan-potongan yang lebih halus dan digabungkan secara bersamaan. Berikut ditampilkan fungsi dari *bicubic basis spline* untuk *n* = 3 [8].

$$\beta^3((x)) = \begin{cases} (2/3) - 0.5\,|x|^2\,(2 - |x|) & 0 \le |x| < 1 \\ (1/6) * (2 - |x|)^3 & 1 \le |x| < 2 \\ 0 & 2 \le |x| \end{cases} \tag{3.1}$$

Dari fungsi *bicubic basis spline* diatas tentu akan diimplemantasikan dalam proses magnifikasi citra dengan metode interpolasi *bicubic basis spline*. Metode interpolasi *bicubic basis spline* menggunakan 16 *pixel* tetangga terdekat yang terdiri dari 4*x*4 *pixel* pada sumbu x dan sumbu y, sehingga fungsi *bicubic basis spline* akan diterapkan pada masing-masing sumbu x dan y. Dimana dengan ketentuan jika $0 \le |x| < 1$ maka menggunakan persamaan $(2/3) - 0.5\,|x|^2\,(2 - |x|)$ dan jika $1 \le |x| < 2$ maka menggunakan persamaan $(1/6) * (2 - |x|)^3$ serta jika $2 \le |x|$ maka sama dengan 0.

Metode interpolasi *bicubic* yang menggunakan 16 *pixel* tetangga terdekat tersebut tentu menghasilkan citra *output* yang memiliki kualitas yang baik. Dimana

metode interpolasi *bicubic* menghasilkan citra *output* yang lebih tajam, sedikit kabur dan dapat menghindari efek kotak-kotak. Lebih jelasnya dibawah ini ditampilkan hasil pembesaran citra menggunakan metode *Bicubic*.

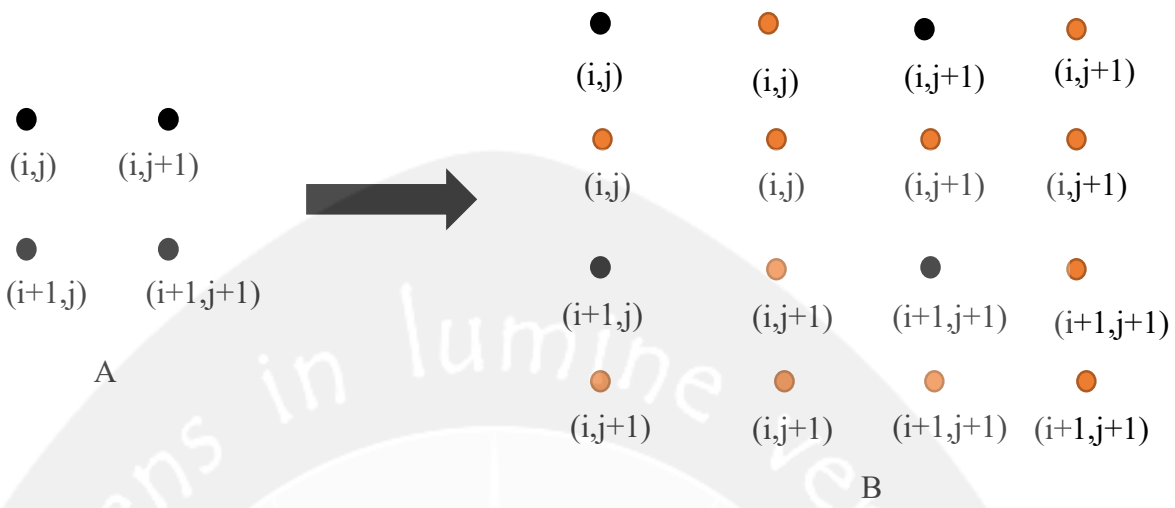
Gambar 3.5. Hasil Interpolasi Bicubic (Sumber: [3])

3.5. Magnifikasi dengan metode interpolasi pembanding

Adapun metode interpolasi lainnya yang dibahas dalam penelitian, dengan tujuan sebagai metode pembanding dengan metode interpolasi yang digunakan oleh penulis. Metode-metode interpolasi lainnya yang digunakan sebagai pembanding dalam penelitian ini adalah metode interpolasi *nearest neighbor* dan metode inteprolasi *bilinear*.

3.5.1 Metode Interpolasi *Nearest Neighbor*

Interpolasi *nearest neighbor* merupakan metode yang sederhana karena mengganti nilai piksel yang baru dengan tetangga terdekat. Selain itu juga metode *nearest neighbor* tidak memiliki perhitungan yang berat. Pada proses pembesaran ukuran citra metode *nearest neighbor* dilakukan dengan menggunakan teknik replikasi piksel. Replikasi piksel ini akan mengganti nilai piksel yang ada dengan mengulang-ulang nilai piksel yang telah ditentukan sesuai faktor pembesaran yang telah dimasukkan [13]. Ilustrasi metode interpolasi *nearest neighbor* dapat dilihat dibawah



Gambar 3.6. Ilustrasi Interpolasi Nearest Neighbor (Sumber: [11])

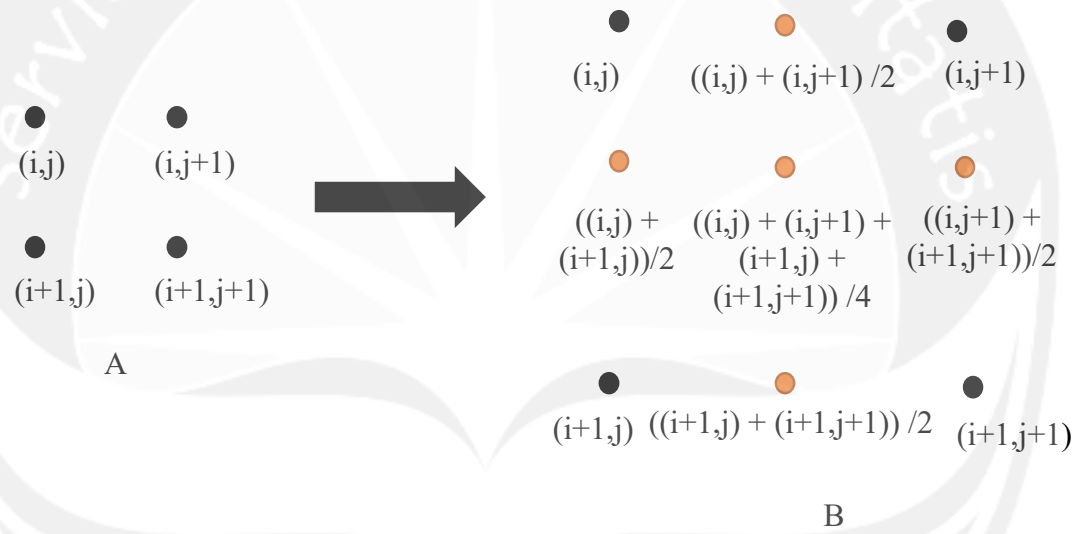
Dari ilustrasi diatas bagian A merupakan piksel-piksel pada citra asli, sementara bagian B merupakan piksel-piksel dari citra hasil. Piksel yang berwarna *orange* pada bagian B merupakan piksel-piksel hasil magnifikasi yang akan dicari nilai pikselnya. Berdasarkan ilustrasi diatas, untuk mendapatkan nilai piksel yang berada diantara (i,j) dan (i,j+1) adalah mengambil nilai piksel tetangga terdekatnya dan mereplikasinya serta mengganti nilai piksel tersebut dengan nilai piksel tetangga terdekatnya yaitu piksel (i,j).

Interpolasi *nearest neighbor* menghasilkan citra hasil yang kurang baik karena pada citra hasilnya akan terdapat efek kabur dan bayangan. Interpolasi *nearest neighbor* merupakan metode yang cepat dalam pembesaran citra, namun menghasilkan efek kotak-kotak yang tinggi pada citra asli pada saat dimagnifikasi[8]. Dibawah ini ditampilkan hasil pembesaran citra menggunakan metode *nearest neighbor*.

Gambar 3.7. Hasil Interpolasi Nearest Neighbor (Sumber : [3])

3.5.2 Metode Interpolasi *Bilinear*

Pada Interpolasi *bilinear* membatasi nilai dari piksel yang baru berdasarkan pada bobot rata-rata dari 4 piksel tetangga terdekat 2*x*2 dari piksel pada gambar aslinya[3]. Pada intepolasi *bilinear* fungsi dasar adalah sepotong garis linear merupakan setiap keluaran piksel dihitung dari kombinasi linear dari 4 piksel input. Interpolasi bilinear menentukan nilai sebuah piksel yang baru berdasarkan pada rata-rata bobot dari 4 piksel pada 2*x*2 tingkat ketetanggaan pada citra asli. Proses ini mirip dengan perataan yang memiliki karakteristik efek *anti-alias* [4]. Ilustrasi metode interpolasi *bilinear* dapat dilihat dibawah.



Gambar 3.8. Ilustrasi Interpolasi Bilinear (Sumber: [5])

Dari ilustrasi diatas bagian A merupakan piksel-piksel pada citra asli, sementara bagian B merupakan piksel-piksel dari citra hasil. Piksel yang berwarna *orange* pada bagian B merupakan piksel kosong yang dicari dengan menghitung bobot dari keempat piksel yang diketahui bagian A. berdasarkan ilustrasi diatas untuk mencari piksel yang belum diketahui dilakukan dengan cara mencari rata-rata dari dua buah titik yang mengapitinya. Contohnya untuk mencari nilai piksel diantara piksel (i,j) dan piksel (i,j+1) maka dilakukan dengan cara ((i,j) + (i,j+1))/2. Sementara untuk piksel yang berada ditengah-tengah maka dilakukan dengan cara ((i,j) + (i,j+1) + (i+1,j) + (i+1,j+1)) /4.

Interpolasi *bilinear* memberikan keluaran citra yang sedikit *jaggies* dan relatif lebih halus dibandingkan dengan interpolasi *nearest neighbor* [3]. Dibawah ini ditampilkan hasil pembesaran citra menggunakan metode *bilinear*.

Gambar 3.9. Hasil Interpolasi Bilinear (Sumber: [3])

3.6. Mengukur kualitas citra

Ada 2 cara pengukuran kualitas citra yaitu secara subjektif dan secara objektif. Pengukuran kualitas citra secara subjektif adalah pengukuran kualitas citra sesuai persepsi manusia. Pengukuran secara subjektif dinilai kurang efektif dan efisien serta memerlukan cukup banyak waktu karena dibutuhkan waktu untuk mencari para penilai dan waktu untuk menunggu nilai dari para penilai berdasarkan pendapat mereka masing-masing [14]. Sementara pengukuran kualitas citra secara objektif adalah pengukuran kualitas citra dengan memanfaatkan algoritma matematika untuk memprediksi kualitas citra secara tepat dan otomatis. Pengukuran kualitas citra secara objektif yang paling sering digunakan adalah *Mean Squared Error* (MSE) dan *Peak Signal to Noise Ratio* (PSNR). Namun kelemahan dari pengukuran kualitas citra secara objektif adalah nilai yang ditampilkan tidak selalu mencerminkan apa yang dilihat oleh mata manusia [14].

3.6.1 Mean Squared Error

Mean Squared Error (MSE) adalah salah satu algoritma matematika secara objektif yang paling sering digunakan untuk mengukur kualitas citra. Persamaan MSE dapat dilihat dibawah ini [13]:

$$\text{MSE} = \sqrt{\frac{1}{mn}\sum_{i=1}^{m}\sum_{j=1}^{n}[I(i,j) - I'(i,j)]^2} \qquad (3.2)$$

Dimana $I(i,j)$ merepresentasikan citra *input* atau citra aslinya, dan $I'(i,j)$ merepresentasikan citra hasil yang telah dilakukan modifikasi. Dimana (i,j) merupakan posisi *pixel* dari sebuah citra berukuran mxn. Sebuah citra memiliki kualitas yang makin bagus jika memiliki nilai MSE semakin kecil. Jadi nilai MSE semakin kecil maka semakin baik kualits citra tersebut [13].

3.6.2 Peak Signal to Noise Ratio

Peak Signal to Noise Ratio (PSNR) dan MSE memiliki hubungan saling ketergantungan, dimana perhitungan nilai *Peak Signal to Noise Ratio* (PSNR) dapat dilakukan setelah diketahui nilai MSE. Persamaan PSNR dapat dilihat dibawah ini[13]:

$$\text{PSNR} = 10 \log_{10} \frac{s^2}{MSE^2} \tag{3.3}$$

Dimana s = 255, untuk citra 8 bit. Sedangkan MSE adalah seperti yang telah dijelaskan pada persamaan MSE diatas. Sebuah citra memiliki kualitas yang makin bagus jika memiliki nilai PSNR semakin besar. Jadi nilai PSNR semakin besar maka semakin baik kualits citra tersebut [13].

3.7. Komputasi paralel

Komputasi paralel merupakan bentuk perhitungan yang menggunakan banyak operasi yang dikerjakan secara serentak atau secara bersama-sama. Lamanya proses dalam suatu komputasi menjadi pertimbangan untuk segara menerapkan konsep paralel yang dapat menghemat waktu [15]. Dalam perkembangannya konsep paralel mulai diterapkan secara luas, hal ini dikarenakan dengan menggunakan konsep ini banyak riset industri dan akademik yang dapat diselesaikan. Selain itu dengan menerapkan konsep paralel yang bekerja secara bersama-sama dapat meningkatkan kecepatan waktu dalam melakukan suatu proses komputasi [16].

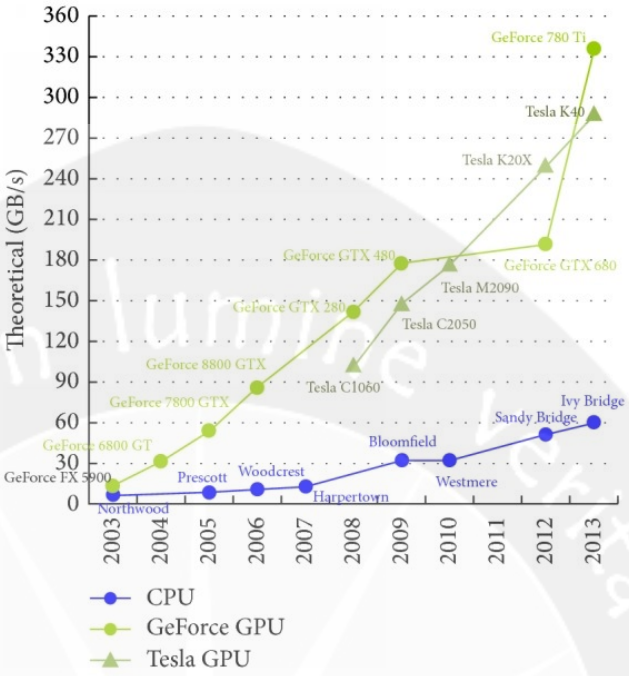
Sebelum tahun 1960-an komputer dengan banyak proseror mulai bermunculan dan komputasi paralel mulai terbentuk. Untuk terus meningkatkan

kecepatan komputasi banyak pendekatan yang digunakan antara lain dengan mereplikasi beberapa unit pemprosesan dalam satu unit. Unit-unit ini bekerja secara paralel. Dengan demikian komputer modern menjadi lebih cepat. Salah satu contoh dari arsitektur paralel adalah *Graphics Processing Unit* (GPU) [17].

3.8. GPU CUDA

GPU (*Graphics Processing Unit*) merupakan teknologi yang dimanfaatkan CUDA (*Compute Unified Device Architecture*) sebagai komputasi paralel yang bisa mempercepat kinerja komputer menjadi lebih cepat dibandingkan dengan komputasi yang dilakukan oleh CPU (*Central Processing Unit*). CUDA merupakan model pemprograman paralel yang dikembangakan oleh NVDIA pada tahun 2006. SDK CUDA pertama kali dirilis pada awal tahun 2007. Dengan adanya CUDA kartu grafis yang semula hanya dimanfaatkan sebagai pemrosesan grafis sekarang dapat dimanfaatkan sebagai komputasi [6].

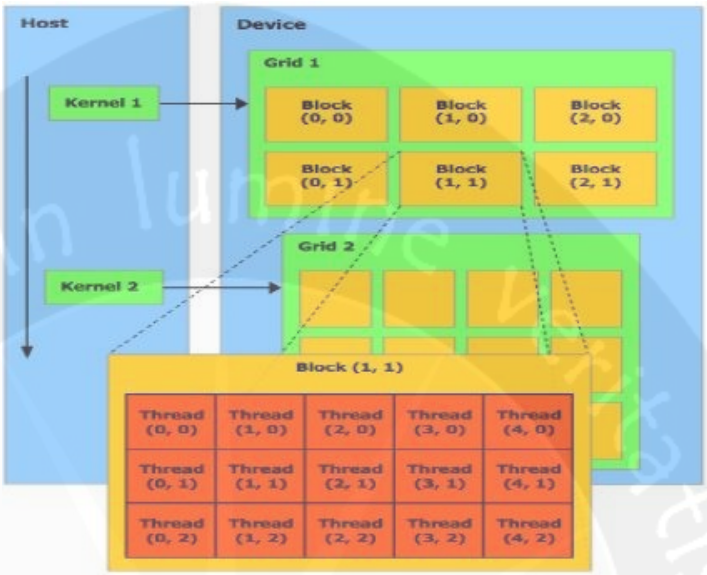
Keuntungan dengan memanfaatkan GPU CUDA selain memberikan waktu yang lebih efisien juga dapat memberikan harga yang lebih rendah dalam mengelolah komputasi yang kompleks. Banyak permasalahan yang dapat diselesaikan oleh GPU CUDA dengan lebih efisien dibandingkan dengan CPU. Hal ini dikerenakan GPU CUDA memiliki banyak *core* dengan tenaga *super* dan memiliki memori *bandwidth* yang sangat tinggi [7]. Berikut perbandingan memori *bandwidth* pada CPU dan GPU :



Gambar 3.10. Memori Bandwidth dari CPU dan GPU (Sumber:[7])

Sistem paralel menggunakan CUDA terdiri dari *host* (CPU) dan *device* (GPU). Arsitektur GPU untuk *thread* terdiri dari dua level hirarki yaitu *block* dan *grid*. *Block* merupakan kumpulan *thread-thread* yang mana tiap *thread*-nya diidentifikasikan dengan *ID thread*. Sementara *grid* adalah kumpulan dari beberapa *block* yang memiliki ukuran dan dimensi yang sama [6]. *Thread*, *block* dan *grid* dapat dimanfaatkan secara bersama-sama untuk mempercepat proses komputasi. Contohnya ketika hendak menjumlahkan 10 buah bilangan, jika menggunakan CPU untuk menjumlahkan 10 bilangan tersebut digunakan konsep perulangan dimana eksekusi penjumlahan ini dilakukan secara bergantian, namun dengan menggunakan GPU CUDA proses penjumlahan ini dapat dilakukan secara bersamaan dengan menggunakan 10 *block* dalam 1 *grid* untuk menjumlahkan 10 buah bilangan tersebut sekaligus. Atau solusi lainnya adalah dengan menggunakan 10 *thread* dalam sebuah *block* untuk melakukan proses penjumlahan ini secara bersama-sama. Hal ini dapat dilakukan karena arsitektur CUDA terdiri dari *thread*,

block dan *grid* yang dapat dimanfaatkan untuk melakukan proses komputasi menjadi lebih cepat [18]. Dibawah ini ditampilkan arsitektur dari CUDA.



Gambar 3.11. Arsitektur CUDA (Sumber: [6])

Dalam mengimplementasikan kode-kode CUDA ada beberapa hal-hal dasar yang perlu diperhatikan. Berikut adalah dasar-dasar implementasi kode CUDA menurut [6] :

1. Mengalokasikan memori CPU
2. Mengalokasikan jumlah memori yang sama untuk GPU dengan menggunakan fungsi *CudaMalloc*
3. Menginput data ke memori CPU
4. Meng-*copy* data ke GPU dengan menggunakan fungsi *CudaMemCpy* dengan parameter *CudaMemcpyHostToDevice*
5. Melakukan pemprosesan di memori GPU dengan menggunakan kernel
6. Meng-*copy* data hasil ke CPU dengan menggunakan fungsi *CudaMemCpy* dengan parameter *CudaMemcpyDeviceToHost*
7. Menghapus memori GPU dengan menggunakan fungsi *CudaFree*

3.9. OpenCV

Open Source Computer Vision (*OpenCV*) merupakan *open source library* yang mengandung lebih dari 500 algoritma yang dioptimalkan untuk menganalisis gambar dan video. *OpenCV* diperkenalkan pada tahun 1999 dan awalnya dikembangkan oleh Intel dengan sebuah team yang dikepalai oleh Gray Bradski. Team ini telah melakukan penelitian dibidang *Computer Vision*. Seri *beta*, versi 1.0 dirilis pertama kali tahun 2006 dan diikuti pada tahun 2009 dirilisnya yang kedua[19]. *OpenCV* dapat ditulis dengan menggunakan bahasa pemprograman C++, Python dan Java dan dapat dijalankan pada sistem operasi windows, Linux, Mac OS, iOS and Android [20].

Fitur-fitur utama yang dimiliki oleh *OpenCV* adalah kelas *input output* dan kelas *mat*. Kelas *input output* terdiri dari kelas *imread*, yang digunakan untuk membaca citra dan kelas *imwrite*, yang digunakan untuk menulis citra. Serta *imshow*, yang digunakan untuk menampilkan gambar. Sementara kelas *mat*, adalah kelas yang merupakan tipe data yang dapat menyimpan *n* dimensi data. Kelas *mat* digunakan untuk menampung nilai vektor atau matriks suatu gambar [21].