BAB III

LANDASAN TEORI

III.1. Citra Digital

Citra merupakan gambar yang merepresentasikan sesuatu. Citra dapat berupa gambar dari sebuah atau kumpulan obyek. Citra digital merupakan citra yang dapat diolah komputer. Citra digital memiliki informasi berupa posisi dan warna (Salomon & Breckon, 2010). Citra digital dapat diwakili oleh sebuah matriks yang terdiri dari baris dan kolom, di mana perpotongan antara baris dan kolom tersebut merupakan piksel. Piksel merupakan elemen terkecil dari sebuh citra. Setiap piksel mewakili koordinat piksel (posisi) dan intensitas piksel (warna).

Citra digital dapat dibagi ke dalam dua jenis, yaitu citra jenis *raster* dan citra jenis *vector*. Citra *raster* adalah citra yang 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 yang dibentuk oleh titik, garis dan lainnya dengan basis ekspresi matematika.

III.2. Video Digital

Video digital pada dasarnya tersusun atas serangkaian *frame* yang ditampilkan dengan kecepatan tertentu (*frame*/detik). Dengan kecepataan yang cukup tinggi, mata manusia akan melihatnya sebagai serangkaian yang kontigu sehingga tercipta ilusi gerak yang halus. Setiap *frame* merupakan gambar / citra digital sehingga segmentasi citra juga dapat dilakukan pada video mengingat bahwa video merupakan sekumpulan *frame*.

III.3. Segmentasi Citra

Segmentasi citra merupakan proses untuk membagi-bagi suatu citra menjadi daerah-daerah atau obyek-obyek yang ada pada citra tersebut.

Segmentasi citra dapat dilakukan dengan beberapa metode, yaitu: *edge-based*, *threshold, region growing* (Prosser, 2010).

1. *Edge-based*

   Metode segmentasi ini dapat digunakan untuk citra dengan wilayah yang akan disegmentasi relatif sederhana dan memiliki kontras yang kuat dengan lingkungnannya.
2. *Threshold*

   Segmentasi *threshold* mengoperasikan seluruh bagian gambar secara bersamaan, mengklasifikasikan setiap piksel apakah merupakan bagian dari yang disegmentasi atau tidak. Metode *threshold* ini mencoba berbagai cara untuk menentukan nilai piksel atau rentang piksel untuk target yang disegmentasi.
3. *Region growing*

   Metode *region growing* dimulai dari satu set point dan dilakukan rekursif untuk memeriksa piksel dengan set poin dan melihat apakah piksel tersebut memenuhi beberapa kondisi yang cocok dengan obyek yang disegmentasi. Setiap piksel yang memenuhi kriteria akan ditambahkan ke dalam set point dan untuk iterasi selanjutnya akan dimulai dengan nilai set poin yang baru.

III.4. Metode Level Set

Metode *level set* mengembangkan sebuah kontur (dua dimensi) dan permukaan (tiga dimensi) secara implisit dengan memanipulasi sebuah fungsi dimensional yang lebih tinggi yang sering disebut sebagai fungsi *level set* $\phi(x,t)$. Evolusi kontur atau permukaan dapat diekstrak dari *zero level set* $(x,t) = \{\phi(x,t) = 0\}$. Keuntungan menggunakan metode ini adalah topologi akan berubah sesuai

dengan penggabungan dan pemisahan dari kontur (*contour*) atau permukaan (*surface*). Evolusi dari kontur atau permukaan diatur oleh persamaan *level set*. Solusi mengarah pada *partial differential equation* yang dikomputasikan secara iteratif dengan mengubah nilai ϕ untuk setiap interval waktu. Berikut ini persamaan umum dari *level set* (Mostofi & College, 2009):

$$\frac{\partial \phi}{\partial t} = |\nabla \phi| . F \tag{3.1}$$

di dalam persamaan *level set* di atas *F* merupakan kecepatan yang mendeskripsikan evolusi dari *level set*. Dengan memanipulasi nilai *F*, kita dapat mengarahkan *level set* ke area atau bentuk yang berbeda dengan memberikan inisialisasi tertentu dari fungsi *level set*.

III.5. Segmentasi Citra Menggunakan Metode Level Set

Di dalam segmentasi citra *F* tergantung dari intensitas piksel atau nilai kelengkungan dari *level set*. *F* juga dapat bergantung pada fungsi indikator tepi yang telah didefenisikan memiliki nilai *zero* pada tepi dan *non-zero* pada bagian lain. Ini menyebabkan F sangat lambat untuk evolusi *level set* pada bagian tepi.

Untuk tujuan segmentasi citra, *F* hanya tergantung pada data piksel citra dan fungsi kelengkungan (Lefohn, et al., 2004). Oleh karena itu, kita akan mengadopsi metodologi yang sama dalam pembuatan persamaan *level set* (Mostofi & College, 2009).

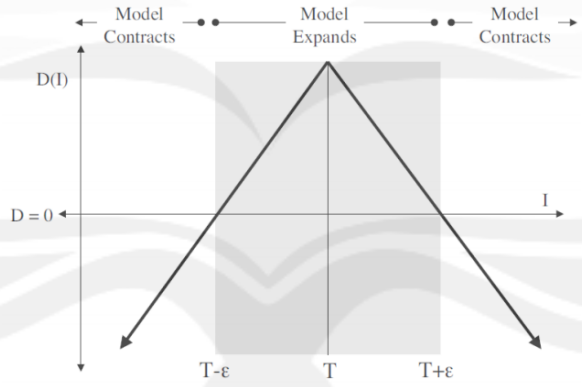
$$\frac{\partial \phi}{\partial t} = |\nabla \phi| \left[\alpha D(I) + (1 - \alpha) \nabla . \frac{\nabla \phi}{|\nabla \phi|} \right] \tag{3.2}$$

Fungsi data *D(I)* menjaga solusi terhadap target yang ditentukan dan rata-rata kelengkungan $\nabla . (\nabla \phi / |\nabla \phi|)$ untuk menjaga agar fungsi *level set* tetap halus. α adalah parameter bebas yang di-*set* sebelumnya untuk mengontrol seberapa halus kontur, dan $\alpha \in [0,1]$.

Fungsi *D(I)* akan berperan sebagai gaya dasar untuk membawa segmentasi. Dengan membuat nilai *D* positif untuk daerah yang ingin disegmentasi dan nilai negatif untuk daerah yang tidak ingin disegmentasi, segmentasi citra akan cenderung ke arah obyek yang ingin kita segmentasi. Fungsi kecepatan sederhana yang dapat memenuhi tujuan tersebut yang digunakan oleh Lefohn (Lefohn, et al., 2004) dapat ditunjukkan oleh persamaan berikut (Mostofi & College, 2009):

$$D(I) = \epsilon - |I - T| \tag{3.3}$$

Persamaan tersebut dapat digambarkan seperti pada gambar (3.1) T mendeskripsikan nilai tengah intensitas dari daerah yang akan disegmentasi, ϵ mendeskripsikan deviasi intensitas di sekitar T yang merupakan bagian yang akan disegmentasi. Oleh karena itu, jika nilai piksel memiliki nilai intensitas masih di antara $T \pm \epsilon$ maka model segmentasi akan menyebar (melebar) tetapi jika tidak maka segmentasi akan menyusut (mengecil).



Gambar 3. 1 *Speed term (Mostofi & College, 2009)*

Di dalam segmentasi citra terdapat tiga parameter yang perlu ditentukan yaitu T, ϵ dan α. Nilai inisialisasi *mask* untuk fungsi *level set* juga dibutuhkan di mana akan menggunakan sebuah gambar kotak untuk segmentasi dua dimensi.

III.6. Algoritma Level Set

III.6.1. Upwinding

Persamaan (3.1), persamaan *level set*, perlu untuk didiskritisasi untuk komputasi serial dan untuk komputasi paralel. Diskritisasi dapat dilakukan dengan menggunakan skema *up-wind differencing* (Osher & Fedkiw, 2003).

Diskritisasi terhadap waktu untuk orde pertama dari persamaan (3.1) dapat menggunakan metode *Euler* (Osher & Fedkiw, 2003) :

$$\frac{\phi^{t+\Delta t}-\phi^t}{\Delta t} + F^t \,.\, \nabla\phi^t = 0 \qquad (3.4)$$

ϕ^t merepresentasikan nilai dari ϕ pada waktu *t*, F^t merepresentasikan nilai kecepatan pada waktu *t* dan $\nabla\varphi^t$ merepresentasikan nilai dari gradien ϕ pada waktu *t*. Di dalam menghitung nilai gradien, kita harus menggunakan turunan spasial dari ϕ untuk mendapatkan hasil yang baik (Mostofi & College, 2009). Dengan menggunakan turunan spasial dari ϕ persamaan (3.4) menjadi

$$\frac{\phi^{t+\Delta t}-\phi^t}{\Delta t} + u^t\phi_x^t + v^t\phi_y^t + w^t\phi_z^t = 0 \qquad (3.5)$$

di mana u, v dan w adalah x, y dan z yang merupakan komponen dari F. Untuk penyederhanaan, dengan menggunakan satu dimensi pada persamaan (3.5) pada *grid* yang spesifik yaitu titik x_i (Mostofi & College, 2009).

$$\frac{\phi^{t+\Delta t}-\phi^t}{\Delta t} + u_i^t(\phi_x)_i^t = 0 \qquad (3.6)$$

di mana $(\phi_x)_i$ adalah turunan spasial dari ϕ pada x_i. Karakteristik dari metode mengindikasikan apakah akan menggunakan metode *forward difference* atau *backward difference* untuk ϕ berdasarkan tanda dari u_i pada titik x_i. Jika $u_i > 0$, nilai dari ϕ akan berpindah dari kiri ke kanan dan akan menggunakan metode *backward difference*. Sebaliknya, jika $u_i < 0$ akan menggunakan metode *forward difference*.

Proses tersebut dilakukan untuk memperkirakan nilai turunan spasial dari ϕ menggunakan tanda dari u_i yang dikenal sebagai *upwinding*.

Penjabaran untuk dua dimensi dengan asumsi menggunakan *isotropic resolution* dapat menghasilkan turunan yang dapat memenuhi perubahan pada persamaan *level set* (Lefohn, et al., 2004).

$$D_x = (\phi_{i+1,j} - \phi_{i-1,j})/2 \qquad D_x^+ = (\phi_{i+1,j} - \phi_{i,j})/2 \qquad D_x^- = (\phi_{i,j} - \phi_{i-1,j})/2$$

$$D_y = (\phi_{i,j+1} - \phi_{i,j-1})/2 \qquad D_y^+ = (\phi_{i,j+1} - \phi_{i,j})/2 \qquad D_y^- = (\phi_{i,j} - \phi_{i,j-1})/2 \tag{3.7}$$

$\nabla\phi$ adalah nilai perkiraan dengan menggunakan metode *upwinding*.

$$\nabla\phi_{max} = \begin{bmatrix} \sqrt{\max(D_x^+, 0)^2 + \max(-D_x^-, 0)^2} \\ \sqrt{\max\left(D_y^+, 0\right)^2 + \max\left(-D_y^-, 0\right)^2} \end{bmatrix} \tag{3.8}$$

$$\nabla\phi_{min} = \begin{bmatrix} \sqrt{\min(D_x^+, 0)^2 + \min(-D_x^-, 0)^2} \\ \sqrt{\min\left(D_y^+, 0\right)^2 + \min\left(-D_y^-, 0\right)^2} \end{bmatrix} \tag{3.9}$$

Setelah itu bergantung pada apakah Fi,j,k > 0 atau Fi,j,k < 0 (Mostofi & College, 2009), sehingga $\nabla\phi$ adalah:

$$\nabla\phi = \begin{cases} \|\nabla\phi_{max}\|_2 & if\ F_{i,j,k} > 0 \\ \|\nabla\phi_{min}\|_2 & if\ F_{i,j,k} > 0 \end{cases} \tag{3.10}$$

$$\phi(t + \Delta t) = \phi(t) + \Delta t F|\nabla\phi| \tag{3.11}$$

III.6.2. Curvature

Curvature dikomputasikan berdasarkan nilai *level set* yang menggunakan turunan. Di dalam dua dimensi yang dibutuhkan ialah nilai dari D_x^{+y}, D_x^{-y}, D_y^{+x}, D_y^{-x} dan juga nilai turunan yang ditentukan sebelumnya (Mostofi & College, 2009).

$$D_x^{+y} = (\phi_{i+1,j+1} - \phi_{i-1,j+1})/2 \qquad D_x^{-y} = (\phi_{i+1,j-1} - \phi_{i-1,j-1})/2$$

$$D_y^{+x} = (\phi_{i+1,j+1} - \phi_{i+1,j-1})/2 \qquad D_y^{-x} = (\phi_{i-1,j+1} - \phi_{i-1,j-1})/2 \tag{3.12}$$

Dengan menggunakan metode *difference of normal* (Lefohn, et al., 2004), curvature dihitung menggunakan turunan dengan 2 normal yaitu n^+ dan n^-.

$$n^+ = \begin{bmatrix} \frac{D_x^+}{\sqrt{(D_x^+)^2 + \left(\frac{D_y^{+x} + D_y}{2}\right)^2}} \\ \frac{D_y^+}{\sqrt{(D_y^+)^2 + \left(\frac{D_x^{+y} + D_x}{2}\right)^2}} \end{bmatrix} \tag{3.13}$$

$$n^- = \begin{bmatrix} \frac{D_x^-}{\sqrt{(D_x^-)^2 + \left(\frac{D_y^{-x} + D_y}{2}\right)^2}} \\ \frac{D_y^-}{\sqrt{(D_y^-)^2 + \left(\frac{D_x^{-y} + D_x}{2}\right)^2}} \end{bmatrix} \tag{3.14}$$

Kedua normal tersebut digunakan untuk menghitung perbedaan, sehingga *mean curvature* dapat dihitung menggunakan persamaan (3.6.12) (Mostofi & College, 2009).

$$H = \frac{1}{2}\nabla . \frac{\nabla\phi}{|\nabla\phi|} = \frac{1}{2}((n_x^+ - n_x^-) + (n_y^+ - n_y^-)) \tag{3.15}$$

III.6.3. Stabilitas

Pendekatan dengan menggunakan *finite difference* terhadap sebuah persamaan *linear diferensial parsial* adalah konvergen jika dan hanya jika keduanya konsisten dan stabil (Osher & Fedkiw, 2003). Stabilitas mengartikan bahwa eror kecil tidak diperbesar (semakin besar) selama iterasi. Stabilitas dihitung menggunakan *Courant-Friedrichs-Lewy* (CFL) yaitu kondisi di mana kecepatan gelombang numerik harus lebih besar dari kecepatan gelombang fisik, $\Delta x/\Delta t > |u|$. Jika kita menyusun ulang persamaan tersebut menjadi (Mostofi & College, 2009):

$$\Delta t < \frac{\Delta x}{\max\{|u|\}} \tag{3.16}$$

di mana biasanya nilai CFL berada diantara 0 dan 1 untuk menjamin stabilitas. Cara lain, selain menggunakan CFL untuk memastikan stabilitas, adalah dengan memastikan penggunaan dari nilai ambang relatif di bagian penyebut pada setiap pecahan untuk menghindari error yang terjadi karena penyebut menjadi 0.Cara ini sudah ada pada persamaan (3.13) dan (3.14) untuk memastikan bahwa nilai *n* tidak akan menjadi tidak berhingga jika nilai di dalam akar ialah 0.

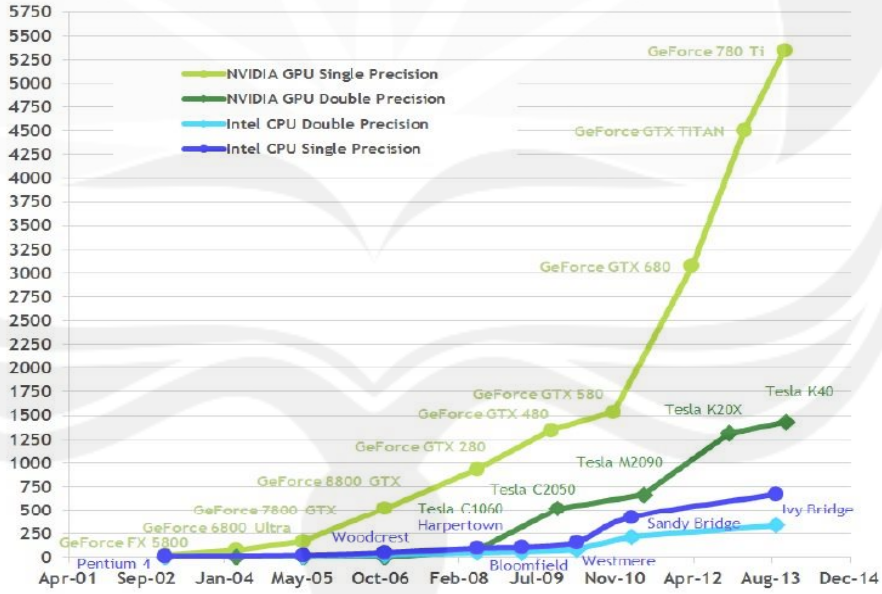
III.7. Komputasi Paralel

Komputasi paralel merupakan komputasi yang dilakukan pada komputer dengan mendistribusikan beban komputasi. Beban komputasi akan disitribusikan pada setiap individual prosesor. Sebuah *single computer* hanya dapat melakukan satu hal pada satu waktu secara sekuensial sedangakan *parallel computer* dapat melakukan banyak komputasi pada waktu bersamaan dengan membagi beban komputasi kepada setiap komputer.

Sejak tahun 2003 terdapat 2 arsitektur dalam mengembangkan mikroprosesor, yaitu *multicore* dan *manycore*. Arsitektur *multicore* bermula dari prosesor berinti dua dan terus berkembang hingga saat ini sudah menjadi delapan inti pada satu

mikroprosesor. Arsiterktur *multicore* banyak digunakan pada CPU. Arsitektur ini digunakan untuk memberikan performa yang baik pada program yang didesain untuk berjalan pada banyak inti, dan juga tetap menjaga performa kecepatan eksekusi yang didesain secara sekuensial. Arsitektur *many-core* memiliki lebih banyak unit pemroses daripada arsitektur *multicore* dan arsitektur ini biasanya digunakan pada GPU.

Pada gambar (3.2) dapat dikatakan bahwa proses komputasi yang dilakukan oleh GPU lebih cepat dibandingkan dengan CPU karena GPU lebih difokuskan untuk perhitungan intensif dan paralel. Meskipun kecepatan komputasi GPU relatif cepat, tetapi GPU tidak dapat menggantikan CPU sepenuhnya karena GPU hanya didesain untuk melakukan perhitungan numerik dan melakukan proses komputasi secara paralel saja.



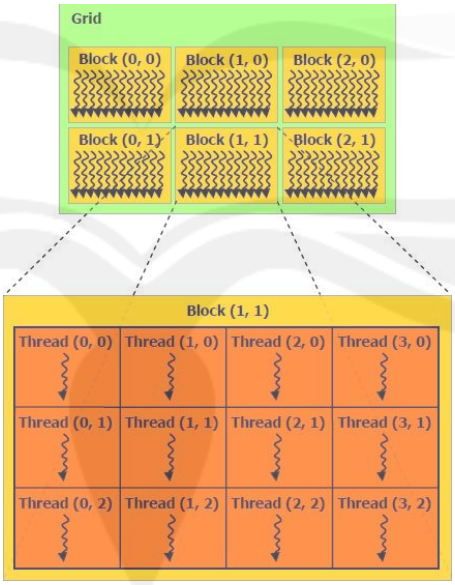
Gambar 3. 2 Perbandingan Performa CPU dan GPU *(Panca, 2015)*

III.8. NVIDIA CUDA

CUDA (Compute Unified Device Architecture) adalah *platform* dari komputasi paralel. Kita dapat mengimplementasikan sebuah algoritma paralel dengan

mudah seperti menulis Bahasa C dengan menggunakan CUDA. Pada tahun 2006, NVIDIA mengeluarkan *library* yang bernama CUDA dan *library* tersebut digunakan untuk melakukan komputasi non-grafis atau yang lebih dikenal dengan GPGPU (*General Purpose Graphic Processing Unit*). Cuda adalah ekstensi dari bahasa pemrograman C yang ditambahkan dengan beberapa sintaks untuk bekerja dengan GPU (Jackson, 2009).

CUDA memiliki 2 bagian utama, yaitu: *Thread dan Block (Grid)*. *Thread* berbentuk sebuah *block* dan dapat digunakan dalam bentuk 1D, 2D, dan 3D. *Thread* memiliki *memory* sendiri dan setiap *thread* akan mengerjakan proses yang sama secara bersamaan sehingga dapat mepersingkat waktu komputasi. *Block (Grid)* terdiri atas bebrapa *thread* di dalamnya. *Block* dapat digunakan dalam skema 1D dan 2D. e 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 3. 3 Struktur Unit Pemroses pada CUDA *(Panca, 2015)*

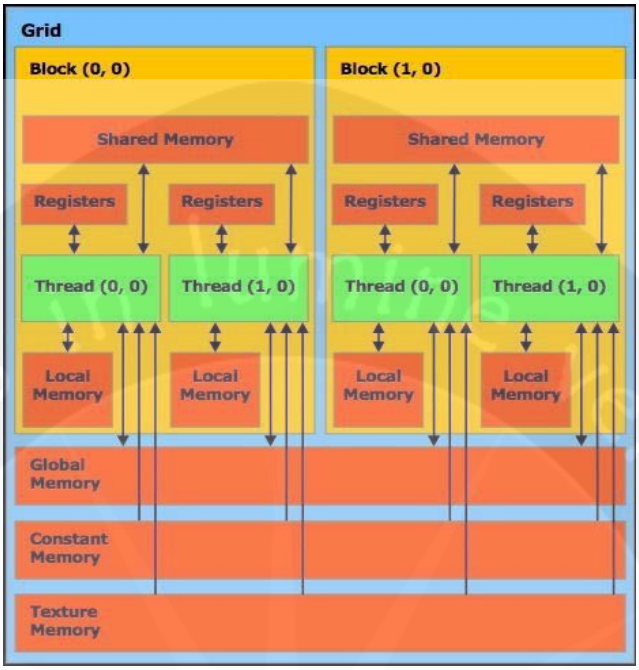
Cuda memiliki bebrapa macam *memory* yang dapat digunakan untuk proses komputasi, yaitu:

1. *Global Memory*

   Global memory merupakan *memory* yang dapat melakukan proses *read* dan *write*. Ukuran *global memory* cukup besar tetapi memiliki kecepatan yang lebih lambat dibandingkan dengan *memory* yang lain.
2. *Shared Memory*

   Semua *thread* yang ada pada sebuah *block* dapat menggunakan *shared memory* untuk melakukan proses *read* dan *write*. *Shared memory* dapat bekerja lebih cepat apabila membaca data pada *thread* yang sama dengan lokasi *shared memory* dan semua indeks *shared array* diperoleh dari permutasi.
3. *Texture Memory*

   Texture memory hanya dapat melakukan proses *read*. *Texture memory* memiliki proses komputasi yang cepat tetapi hanya dapat mengerjakan proses dalam bentuk 2D.
4. *Register*
5. *Local Memory*
6. *Constant Memory*



Gambar 3. 4 Struktur Memori pada CUDA *(Panca, 2015)*