REAL-TIME RENDERING DAN KOMPRESI *VIDEO* PARALEL MENGGUNAKAN ALGORITMA *LEMPEL-ZIV-WELCH* (LZW)

Achmad M. Satrio W [1], Firiyani, S.Si,M.T [2], Izzatul Ummah ST, MT [3]

[1,2,3] Jurusan Teknik Informatika Universitas Telkom, Bandung

[1] Bandulan103a@gmail.com, [2] fitriyani.y@gmail.com, [3] izzatul.ummah@gmail.com,

Abstrak

Seiring dengan berkembangnya teknologi, *video* disimpan dalam bentuk file. Namun sering kali *file-file video* tersebut berukuran relatif besar, sehingga menyulitkan dalam hal penyimpanan data (*storage*) dan dalam hal pengiriman data itu sendiri. Salah satu solusi untuk memperkecil ukuran *file* tersebut adalah dengan melakukan kompresi *video*. Dalam tugas akhir ini, proses kompresi dilakukan secara paralel. Dengan itu diharapkan mampu membantu performa dalam proses kompresi, khususnya dalam hal kecepatan. Algortima yang digunakan untuk kompresi *video* adalah algortima LZW.

Kata kunci : kompresi, kompresi *video*, lzw, paralel.

Pendahuluan

Latar Belakang

Pada era modern seperti sekarang hampir semua data bersifat elektronik dan disimpan pada suatu media penyimpanan. Suatu media penyimpanan pun juga memiliki kapasitasnya atau batasnya. Untuk mencegah kejadian tersebut maka dibutuhkan yang namanya kompresi, agar semua data yang kita inginkan dan dianggap penting tetap bisa disimpan.

Umumnya representasi sebuah *video* membutuhkan memori yang besar. Semakin besar ukuran *video* tentu semakin besar pula memori yang dibutuhkannya. Pada sisi lain, kebanyakan *video* mengandung duplikasi data atau redudensi data.

Suatu proses atau cara pemadatan data sehingga memerlukan ruang penyimpanan yang lebih kecil dari sebelumnya, dan mempersingkat waktu dalam proses pertukaran data disebut dengan kompresi. Dalam tugas akhir ini, proses kompresi dilakukan secara paralel. Dengan implementasi secara paralel diharapkan mampu membantu performa dalam proses kompresi, khususnya dalam hal kecepatan dan ketepatan kompresi. Algoritma yang digunakan pada proses kompresi *video* adalah algoritma LZW.

Algoritma LZW merupakan algoritma kompresi yang bersifat *lossless* dan menggunakan metode *dictionary*. Algoritma ini ditemukan oleh Abraham Lempel, Jacob Ziv, dan Welch pada tahun 1984.

Beberapa penelitian membuktikan bahwa proses kompresi yang dilakukan dengan paralel dapat membantu performa dalam proses kompresi, khususnya dalam hal kecepatan kompresi, seperti pada penelitian yang dilakukan Stefan Karlsson, Erik Hansson dengan judul "*Lossless Message Compression*"[12].

Rumusan Masalah

Dengan mengacu pada latar belakang rumusan masalah pada tugas akhir ini adalah :

1. Bagaimana mengimplementasikan *real-time rendering* dan kompresi *video* secara sekuensial dan secara paralel menggunakan algoritma kompresi *Lempel–Ziv–Welch* (LZW)?
2. Bagaimana analisis hasil implementasi *real-time rendering* dan kompresi *video* secara sekuensial dan secara paralel menggunakan algoritma kompresi *Lempel–Ziv–Welch* (LZW) dari segi waktu komputasi dan rasio kompresi?
3. Bagaimana analisis performansi *Lempel–Ziv–Welch* (LZW) secara sekuensial dan paralel dalam segi *speedup, improvement performancy,* dan *efficiency*?

Tujuan

Tujuan yang ingin dicapai dalam tugas akhir ini adalah :

1. Mengimplementasikan *real-time rendering* dan kompresi *video* secara sekuensial dan secara paralel menggunakan algoritma kompresi *Lempel–Ziv–Welch* (LZW).
2. Menganalisis hasil *real-time rendering and video compression* menggunakan algoritma kompresi *Lempel–Ziv–Welch* (LZW) secara sekuensial dan secara paralel dari segi waktu dan rasio kompresi.
3. Menganalisis performansi *Lempel–Ziv–Welch* (LZW) dalam segi *speedup, improvement performancy, efficiency* baik secara sekuensial maupun secara paralel.
4. Mengimplementasi algoritma *Lempel– Ziv–Welch* (LZW) pada proses kompresi *video*.

Batasan Masalah

Batasan masalah yang digunakan dalam tugas akhir ini adalah :

1. Format *video* yang digunakan adalah .avi tanpa suara atau tanpa *audio*.
2. Perangkat lunak yang digunakan selama proses kompresi adalah Matlab R2014a.
3. Menggunakan *tools* yang sudah tersedia pada Matlab R2014a untuk pengolahan *video*, serta *Parallel Computing Toolbox*.

Dasar Teori

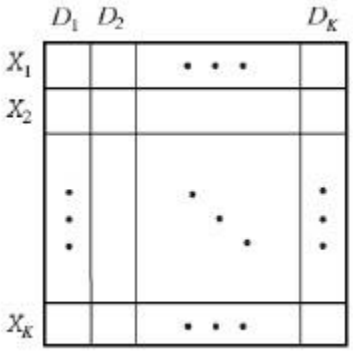
Video

Video merupakan gabungan gambar-gambar mati yang dibaca berurutan dalam suatu waktu dengan kecepatan tertentu. Gambar-gambar yang digabung tersebut dinamakan *frame* dan kecepatan pembacaan gambar disebut dengan *frame rate*, dengan satuan fps (*frame per second*). Karena dimainkan dalam kecepatan yang tinggi maka tercipta ilusi gerak yang halus, semakin besar nilai *frame rate* maka akan semakin halus pergerakan yang ditampilkan.

Karakteristik Video Digital

Resolusi

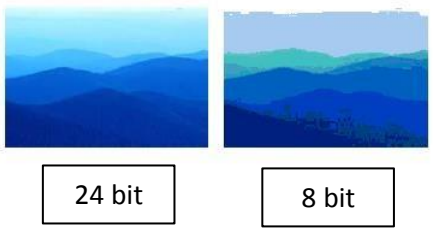
Resolusi atau dimensi *frame* merupakan ukuran sebuah *frame* yang dinyatakan dalam *pixel x pixel*. Semakin tinggi resolusi maka akan semakinbaik tampilan video tersebut. Namun resolusi yang tinggi membutuhkanjumlah bit yang besar pula.



Gambar 2.1 Resolusi suatu frame atau citra

Kedalaman Bit

Kedalaman bit akan menentukan jumlah bit yang digunakan untuk merepresentasikan tiap piksel pada sebuah *frame*. Sama halnya dengan resolusi, semakin besar kedalaman bit yang digunakan akan membutuhkan jumlah bit yang semakin besar.



Gambar 2.2 Kedalaman bit pada suatu frame

Bit Rate

Bit rate merupakan banyaknya data yang dikirimkan / detik (*bits per second* / bps), sangat dipengaruhi oleh *Frame Resolution*, *Bit Depth* dan *FrameRate*. *Bit rate* disebut juga dengan nama *data rate*. *Bit rate* menentukan jumlah data yang ditampilkan saat video dimainkan.

Frame Rate

Jumlah gambar atau *frame* yang terlihat setiap detik disebut dengan *frame rate*. Karakteristik ini berkaitan dengan kehalusan gerakan (*smoothness of motion*) sebuah objek di dalam *video*.

LZW

Algoritma LZW merupakan algoritma kompresi yang bersifat *lossless* dan menggunakan metode *dictionary*.

Encode

1. *Dictionary* diinisialisasi dengan semua karakter dasar yang ada, [„A‟..‟Z‟,‟a‟..‟z‟,‟0‟..‟0‟].
2. **P** adalah karakter pertama dalam *stream* karakter.
3. **Q** adalah karakter berikutnya dalam *stream* karakter.
4. Apakah *string* **(P+Q)** terdapat dalam *dictionary*?
   a. Jika “ya” maka **P=P+Q** (gabungan **P** dan **Q** menjadi *string* baru)
   b. Jika “tidak” maka”
      i. Output sebuah kode untuk menggantikan *string* **P.**
      ii. Tambahkan *string* **(P+Q)** ke dalam *dictionary* dan berikan nomor atau kode berikutnya yang belum digunakan dalam *dictionary* untuk *string* tersebut.
      iii. **P = Q**
5. Apakah masih ada karakter berikutnya dalam *stream* karakter?
   a. Jika “ya” maka kembali pada langkah ke dua.
   b. Jika “tidak” maka *output* kode yang menggantikan *string* **P**, lalu proses berhenti.

Decode

1. *Dictionary* diinisialisasi dengan semua karakter dasar yang ada, [„A‟..‟Z‟,‟a‟..‟z‟,‟0‟..‟0‟].
2. **Q** adalah kode pertama dari *stream* kode (menunjuk ke salah satu karakter dasar).
3. Lihat *dictionary* dan *output string* dari kode tersebut (*string***.Q**) ke *stream* karakter.
4. **P = Q.**
5. **Q** = kode berikutnya dari *stream* kode.
6. Apakah *string***.Q** terdapat dalam *dictionary*?
   a. Jika ada maka :
      i. *Output string***.Q** ke *stream* karakter.
      ii. $\mathbf{P_1}$ = *string***.P**
      iii. $\mathbf{Q_1}$ **=** karakter pertama dari *string***.Q**
      iv. Tambahkan *string* $\mathbf{(P_1+Q_1)}$ ke dalam *dictionary*.
   b. Jika tidak maka :
      i. $\mathbf{P_1}$ = *string***.P**
      ii. $\mathbf{Q_1}$ **=** karakter pertama dari *string***.P**
      iii. *Output string* $\mathbf{(P_1+Q_1)}$ ke *stream* karakter dan tambahkan *string* tersebut ke dalam *dictionary* (sekarang berkorespondensi dengan **Q**).
7. Apakah terdapat kode ladi di *stream* kode?
   a. Jika “ya” maka kembali pada langkah 4.
   b. Jika “tidak” maka proses berhenti.

Performansi

Untuk mengukur performansi pada tugas akhir ini menggunakan *speedup* dan *efficiency* berdasarkan pada waktu komputasi algoritma sekuensial dan paralel. Dari proses performansi ini dapat diketahui algoritma paralel tersebut, seberapa besar peningkatan peningkatan waktu komputasi algoritma paralel, dan seberapa efisien penggunaan *core processor* pada saat menjalankan proses komputasi.

Speedup

Untuk mengukur seberapa cepat algoritma paralel dibandingkan dengan algoritma sekuensial.

——————— (1)

Efficiency

Untuk mengetahui seberapa efisien atau seberapa baik pemanfaatan *processor* dalam menyelesaikan suatu masalah.

——————————— (2)

Perancangan Dan Implementasi

Sistem tersebut dirancang untuk melakukan proses kompresi *video* dengan menggunakan algoritma *Lempel–Ziv–Welch* (LZW) yang dijalankan secara sekuensial dan secara paralel seperti yang telah digambarkan di atas. Dan terakhir dilakukan analisis terhadap *video* awal dengan *video* hasil uji, dan membandingkan mana yang paling baik, kompresi secara sekuensial atau secara paralel.

- **Proses Pemecahan *Frame***

  Tahap pertama *video* dipecah menjadi *frame-frame*, di mana tiap-tiap *frame* tersebut merupakan suatu *video.*

- **Pembacaan Tiap *Frame***

  Yang kedua adalah pembacaan tiap frame atau *video*. Proses ini merupakan langkah awal sebelum memasuki proses *encode.* Pada proses ini frame atau *video* disusun menjadi sebuah matriks 1 x (m*n).

- ***Encode***

  Pada proses *encode*, matriks diubah menjadi *vector* yang kemudian baru memasuki proses kompresi atau *encode* menggunakan algoritma LZW.

- ***Decode***

  Untuk proses *decode*, *vector* hasil dari proses *encode* disusun kembali menjadi sebuah matriks 1 x (m*n) sesuai dengan semula setelah melalui proses *decode*.

- **Penyusunan *Frame***

  Selanjutnya matriks hasil dari proses *decode* tersebut dikembalikan menjadi sebuah frame atau *video*.

- ***Rendering***

  Kemudian *frame-frame* tersebut dikumpulkan kembali, dan dikembalikan atau disusun kembali menjadi sebuah *video*.

Analisis Dan Hasil

Percobaan ini dilakukan dengan menggunakan data video dengan format .avi, yang diambil melalui proses *live video* dengan program pada percobaan ini. Video yang digunakan adalah video dengan resolusi sebesar 320x240, yang mana 320 merupakan lebar frame, dan 240 merupakan tinggi frame. Untuk detailnya bisa dilihat pada tabel di bawah ini.

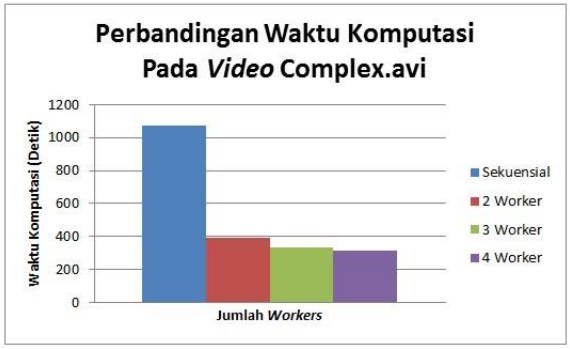
Tabel 4.1 Data percobaan

No	Nama *Video*	Resolusi	Jumlah *frames*	Ukuran
1	Sederhana	320x240	45	11,2 MB
2	Complex	320x240	45	9,89 MB

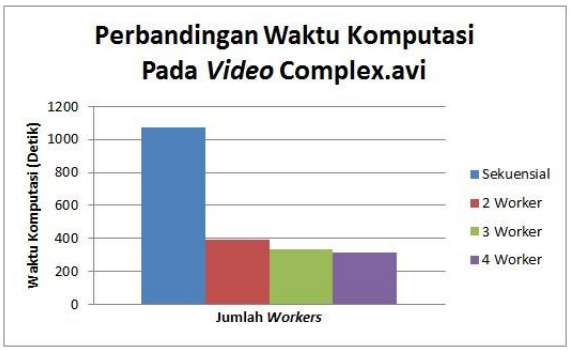
Gambar 4. 1 Contoh salah satu frame pada *video* Sederhana.avi

Gambar 4. 2 Contoh salah satu frame pada *video* Complex.avi

Analisis Waktu



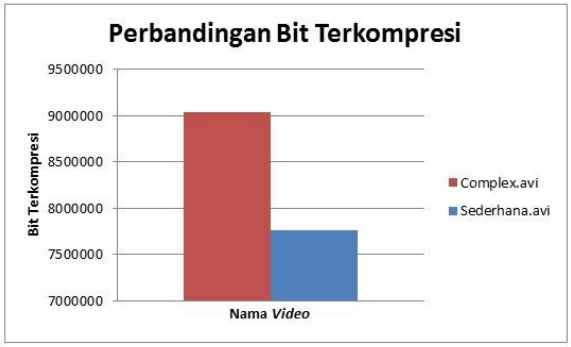
Gambar 4. 5 Grafik Perbandingan Waktu Koputasi complex.avi



Gambar 4. 5 Grafik Perbandingan Waktu Koputasi complex.avi

Mengacu pada hasil dan grafik di atas, terbukti bahwa kompresi secara paralel dapat mempercepat proses kompresi dengan signifikan. Semakin banyak *workers* yang digunakan, maka akan semakin cepat pula proses kompresi tersebut.

Analisis Bit Terkompresi



Gambar 4. 6 Grafik Perbandingan Bit Terkompresi

Video dengan warna yang lebih banyak atau lebih kompleks akan memiliki jumlah bit yang lebih banyak dibandingkan dengan video dengan warna yang lebih sederhana.

Berdasarkan pada grafik di atas, *video* Complex.avi memiliki nilai bit terkompresi paling tinggi dibandingkan dengan *video* Sederhana.avi, karena pada *video* Complex.avi memiliki lebih banyak warna atau warna yang lebih kompleks, sedangkan pada *video* Sederhana.avi tidak memiliki kerumitan warna atau memiliki warna yang sederhana. Karena tidak memiliki kerumitan warna, *video* memiliki redudensi data yang besar dibandingkan dengan *video* Complex.avi. Itulah yang menyebabkan *video* Sederhana.avi dapat terkompresi dengan baik.

Performansi

Tabel 4. 6 Tabel Hasil Evaluasi Performansi

Nama Video	Rata-Rata Waktu Komputasi Sekuensial	Jumlah Worker	Rata-Rata Waktu Komputasi Paralel	*Speedup*	*Efficiency*
Sederhana.avi	968.24	2	387.65	2.5	1.25
		3	327.84	2.95	0.98
		4	307.61	3.15	0.79
Complex.avi	1073.33	2	393.17	2.72	1.4
		3	336.03	3.2	1.1
		4	315.9	3.4	0.85

Berdasarkan penelitian pada *video* Sederhana.avi dan *video* Complex.avi seperti gambar di atas, dihasilkan bahwa rata-rata waktu *speedup* tertinggi dicapai saat menggunakan 4 *worker*.

Pada tabel di atas nilai *efficiency* tertinggi yang di dapat adalah saat menggunakan 2 *worker*. Yang menandakan bahwa hanya dengan menggunakan 2 *worker* saja sudah cukup dalam melakukan proses kompresi. Hal ini dikarenakan hanya dengan menggunakan jumlah *worker* 2, rata-rata waktu komputasi paralel telah lebih baik dari rata-rata waktu komputasi sekuensial.

Kesimpulan

Kesimpulan yang bisa didapat dari percobaan di atas adalah :

1. Proses kompresi dengan algoritma *Lempel–Ziv–Welch* (LZW) secara paralel terbukti dapat mempercepat proses kompresi.
2. Proses kompresi pada *video* Sederhana.avi dengan algoritma *Lempel–Ziv–Welch* (LZW) 25,17% lebih kecil ukurannya dari semula, sedangkan pada complex.avi 12,85% lebih kecil ukurannya dari semula.
3. Setelah melakukan proses evaluasi performansi, nilai *speedup*, dan nilai *efficiency* tertinggi dicapai pada saat menggunakan 2 *worker*.
4. Hasil dari kompresi dengan algoritma *Lempel–Ziv–Welch* (LZW) yaitu *lossless* atau kompresi tanpa rugi (sama seperti aslinya).

Saran

Adapun saran untuk pengembangan selanjutnya adalah :

1. Dapat mengimplementasikan kompresi menggunakan algoritma dengan jumlah data yang lebih besar lagi.
2. Dibutuhkan adanya *upgrade* perangkat keras maupun perangkat lunak apabila digunakan untuk penelitian lebih lanjut

Daftar Pustaka

[1] T. Sutoyo, Edy Mulyanto, Vincent Suhartono, Oky Dwi Nurhayati, Wijanarto. 2009. *Teori Pengolahan Citra Digital*. Yogyakarta: ANDI

[2] Darma Putra. 2010. *Pengolahan Citra Digital*. Yogyakarta: ANDI

[3] Gardiansyah Prayosa. 2014. *Analisis Implementasi Ant Colony Optimization Untuk Deteksi Tepi Pada GPU.* Bandung: Universitas Telkom.

[4] (2014).[Online]. Tersedia: http://id.wikipedia.org/wiki/Kompresi_data [diakses pada 28 Oktober 2014]

[5] (2014).[Online]. Tersedia: http://en.wikipedia.org/wiki/Lossless_compression [diakses pada 28 Oktober 2014]

[6] (2014).[Online]. Tersedia: http://en.wikipedia.org/wiki/Lossy_compression [diakses pada 28 Oktober 2014]

[7] (2014).[Online]. Tersedia: http://cs.ipb.ac.id/~yeni/files/ppcd/Kuliah%2013%20Kompresi%20*Citra*%202%20edit.pdf [diakses pada 28 Oktober 2014]

[8] (2014).[Online]. Tersedia: http://en.wikipedia.org/wiki/Lempel%E2%80%93Ziv%E2%80%93Welch [diakses pada 28 Oktober 2014]

[9] (2014).[Online]. Tersedia: http://id.wikipedia.org/wiki/Graphics_Interchange_Format [diakses pada 28 Oktober 2014]

[10] (2014).[Online]. Tersedia: http://id.wikipedia.org/wiki/Portable_Network_Graphics [diakses pada 28 Oktober 2014]

[11] (2014).[Online]. Tersedia: http://id.wikipedia.org/wiki/Unit_pemroses_grafis [diakses pada 28 Oktober 2014]

[12] Stefan Karlsson, Erik Hansson. 2013. *Lossless Message Compression*. Sweden: Mälardalen University.

[13] Marvin Ch. Wijaya, Prijono Agus. 2007. *Pengolahan Citra Digital Menggunakan Matlab.* Bandung: Informatika

[14] Jung W. Suh, Youngmin Kim. 2014. *Accelerating Matlab With GPU Computing*. USA: Morgan Kaufman.

[15] Sinaga Marlando. 2005. *Kompresi Citra Menggunakan Lempel-Ziv-Welch (LZW) Dan Arithmetic Coding (AC).* Bandung : STT Telkom.

[16] Hao, Hu. 2005. *Lossless Compression*.

[17] Ahmad Firdaus Ahmad Fadzil, Noor Elaiza Abdul Khalid, dan Mazani Manaf. (2013), *Scaling Performance of Task-Intensive Applications via Mapreduce Parallel Processing*, Faculty of Computer and Mathematical Science, UiTM Shah Alam Selangor, Malaysia.