KOMPRESI CITRA DIGITAL MENGGUNAKAN TRANSFORMASI *WAVELET* 9/7 DAN METODE KUANTISASI VEKTOR ADAPTIF DENGAN ALGORITMA *PARTIAL CODEWORD UPDATING*

SKRIPSI

Sebagai salah satu syarat untuk memperoleh gelar Sarjana Komputer
dalam bidang Ilmu Komputer

Oleh:
HANIF ANDANU
0310963009-96

PROGRAM STUDI ILMU KOMPUTER
JURUSAN MATEMATIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS BRAWIJAYA
MALANG

2009


UNIVERSITAS BRAWIJAYA

LEMBAR PENGESAHAN SKRIPSI

KOMPRESI CITRA DIGITAL MENGGUNAKAN TRANSFORMASI *WAVELET* 9/7 DAN METODE KUANTISASI VEKTOR ADAPTIF DENGAN ALGORITMA *PARTIAL CODEWORD UPDATING*

Oleh :
HANIF ANDANU
0310963009-96

Setelah dipertahankan di depan Majelis Penguji
pada tanggal 30 Maret 2009
dan dinyatakan memenuhi syarat untuk memperoleh gelar
Sarjana Komputer dalam bidang Ilmu Komputer

Pembimbing I,	**Pembimbing II,**
Drs. Marji, MT	**Dani Primanita K, ST**
NIP. 131 993 386	**NIP. 132 310 159**

Mengetahui,
Ketua Jurusan Matematika
Fakultas MIPA Universitas Brawijaya

Dr. Agus Suryanto, MSc
NIP. 132 126 049


UNIVERSITAS BRAWIJAYA

LEMBAR PERNYATAAN

Saya yang bertanda tangan di bawah ini :

Nama	**:** Hanif Andanu
NIM	**:** 0310963009-96
Program Studi	**:** Ilmu Komputer
Penulis Skripsi berjudul	**:** Kompresi Citra Digital MenggunakanTransformasi *Wavelet* 9/7 dan Metode Kuantisasi Vektor Adaptif dengan Algoritma *Partial Codeword Updating*

Dengan ini menyatakan bahwa :

1. **Isi dari Skripsi yang saya buat adalah benar-benar karya sendiri dan tidak menjiplak karya orang lain, selain nama-nama yang termaktub di isi dan tertulis di daftar pustaka dalam Skripsi ini.**
2. **Apabila dikemudian hari ternyata Skripsi yang saya tulis terbukti hasil jiplakan, maka saya akan bersedia menanggung segala resiko yang akan saya terima.**

Demikian pernyataan ini dibuat dengan segala kesadaran.

Malang, 30 Maret 2009
Yang menyatakan,

Hanif Andanu
NIM. 0310963009-96

KOMPRESI CITRA DIGITAL MENGGUNAKAN TRANSFORMASI *WAVELET* 9/7 DAN METODE KUANTISASI VEKTOR ADAPTIF DENGAN ALGORITMA *PARTIAL CODEWORD UPDATING*

ABSTRAK

Penyimpanan citra digital dalam bentuk mentah memiliki ukuran data besar sehingga perlu dilakukan penyesuaian dengan kapasitas media penyimpanan yang terbatas. Oleh karena itu muncul ide menjadikan data berukuran lebih kecil dengan kompresi. Metode Transformasi *Wavelet* dan Kuantisasi Vektor Adaptif dapat digunakan untuk kompresi citra digital. Agar ukuran *file* yang dikompres semakin kecil digunakan pengkodean *Entropy* Huffman. Maka permasalahan pada penelitian ini adalah bagaimana merancang dan mengimplementasikan aplikasi kompresi citra digital menggunakan Transformasi *Wavelet* 9/7 dan metode Kuantisasi Vektor Adaptif.

Transformasi *Wavelet* yang digunakan adalah Transformasi *Wavelet* 9/7 dengan skema *lifting*. Setelah proses transformasi, citra dikuantisasi dengan Kuantisasi Vektor Adaptif menggunakan algoritma *Partial Codeword Updating*. Tahapan yang dilakukan utnuk melakukan proses kompresi adalah *blocking* citra, Transformasi *Wavelet* 9/7, Kuantisasi Vektor Adaptif, dan Huffman *Encoder*. Sedangkan tahap untuk melakukan proses dekompresi adalah Huffman *Decoder*, Dekuantisasi Vektor Adaptif, *Invers* Transformasi *Wavelet* 9/7, dan *Deblocking* citra.

Berdasarkan penelitian yang telah dilakukan, didapatkan nilai rata-rata rasio dari ke-18 citra uji 24 bit, berekstensi .bmp dan mempunyai resolusi 256 x 256 pixel adalah sebesar 57,83 %. Pada pengujian tingkat kesalahan yang diukur menggunakan nilai MSE, didapatkan nilai rata-rata MSE citra *low detail* untuk keseluruhan pengujian memiliki nilai rata-rata MSE berada diantara nilai 9,00 sampai 88,78. Dan nilai rata-rata MSE citra *high detail* dan *medium detail* berada diantara nilai 9,00 sampai 113,92.

DIGITAL IMAGE COMPRESSION USING WAVELET 9/7 TRANSFORMATION AND ADAPTIVE VECTOR QUANTIZATION WITH PARTIAL CODEWORD UPDATING ALGORITHM

ABSTRACT

Storing digital images in raw format results in big file sizes which will cause issue with limited media storage capacity. That results in the idea of minimizing the data size with compression. The Wavelet Transformation and Adaptive Vector Quantization can be used for digital image compression. To minimize the file size even further, the Huffman Entropy coding is used. Therefore this research focus is in designing and implementing digital image compression with Wavelet 9/7 Transformation and Adaptive Vector Quantization method.

The Wavelet Transformation used in this research is Wavelet 9/7 Transformation with lifting scheme. After the transformation being process, image is quantitized with Adaptive Vector Quantization with Partial Codeword Updating algorithm. The steps to compress the image are image blocking, Wavelet 9/7 Transformation, Adaptive Vector Quantization, and Huffman Encoder. On the other hand, the steps to decompress the image are Huffman Decoder, Adaptive Vector Dequantization, Wavelet 9/7 Inverse Transformation, and image Deblocking.

Based on the research, the compression ratio from 18 test images, all of them 24 bit .bmp 256 x 256 pixel files, is 57.83 %. In error rate testing using MSE value, the mean MSE for low detail images between 9.00 to 88.78. The mean MSE for high detail and medium detail images is between 9.00 to 113.92.

KATA PENGANTAR

Alhamdulillahi rabbil ‘alamin. Puji syukur penulis panjatkan kehadirat Allah SWT yang telah melimpahkan rahmat, taufik serta hidayah-Nya sehingga Skripsi yang berjudul “Kompresi Citra Digital Menggunakan Transformasi *W*avelet 9/7 dan Metode Kuantisasi Vektor Adaptif dengan Algoritma *Partial Codeword Updating*” dapat diselesaikan.

Skripsi ini disusun dan diajukan sebagai syarat untuk memperoleh gelar sarjana pada program studi Ilmu Komputer, jurusan Matematika, fakultas MIPA, Universitas Brawijaya.

Banyak pihak yang berperan atas terselesaikannya penelitian dan penulisan Skripsi ini. Atas bantuan yang telah diberikan, penulis ingin menyampaikan penghargaan dan ucapan terima kasih yang sedalam-dalamnya kepada:

1. Drs. Marji, MT, selaku dosen pembimbing I yang telah memberikan saran dan bimbingan atas penelitian “Kompresi Citra Digital Menggunakan Transformasi *W*avelet 9/7 dan Metode Kuantisasi Vektor Adaptif dengan Algoritma *Partial Codeword Updating*”.
2. Dani Primanita K, ST., selaku dosen pembimbing II atas bimbingan dalam penulisan laporan.
3. Dr. Agus Suryanto, MSc, selaku ketua jurusan Matematika FMIPA UB Malang.
4. Wayan Firdaus Mahmudy, SSi., MT, selaku Ketua Program Studi Ilmu Komputer UB Malang.
5. Segenap bapak dan ibu dosen yang telah mendidik, dan mengamalkan ilmunya kepada penulis.
6. Segenap staf dan karyawan di Jurusan Matematika FMIPA UB.
7. Bapak, ibu, dan adikku yang telah memberikan semangat dan mendoakan selama belajar di Program Studi Ilmu Komputer FMIPA UB dan dalam mengerjakan skripsi.
8. Prima, Rosi, Boim, Hafiz, Dian Telkom serta teman-teman ilkomers ’03 yang telah memberikan semangat dan dukungan pada penulis.
9. Semua pihak lain yang telah membantu terselesaikannya skripsi ini yang tidak bisa penulis sebutkan satu-persatu.

Penulis menyadari bahwa skripsi ini masih jauh dari kesempurnaan, dan memiliki banyak kekurangan, sehingga dengan segala kerendahan hati penulis mengharapkan kritik dan saran yang membangun dari pembaca. Semoga penulisan skripsi ini bermanfaat bagi pembaca.

Malang, 30 Maret 2009

Penulis

DAFTAR ISI

DAFTAR GAMBAR

Halaman

DAFTAR TABEL

Halaman

DAFTAR *SOURCE CODE*

Halaman

DAFTAR LAMPIRAN

Halaman

BAB I
PENDAHULUAN

1.1 Latar Belakang

Citra dalam bentuk digital sangat diperlukan dalam bidang pendidikan, citra satelit, peta cuaca, dan bidang lainnya. Penyimpanan citra digital dalam bentuk mentah memiliki ukuran data besar sehingga perlu dilakukan penyesuian dengan kapasitas media penyimpanan yang terbatas. Keterbatasan media penyimpanan untuk menyeimbangkan kapasitas data yang terus membesar dari hasil digitalisasi citra menjadi salah satu faktor yang harus dicari solusinya. Saat ini ada cara aplikasi kompresi data yang dilakukan terhadap citra digital dengan tujuan untuk mengurangi redundansi dari data-data yang terdapat dalam citra, karena prinsipnya yang menghilangkan atau mencuplik sebagian informasi pada data akan membuat ukuran data menjadi berkurang sehingga pilihan untuk menghapus citra-citra yang tersimpan sebelumnya guna menyeimbangkan kapasitas media penyimpanan tidak lagi menjadi pilihan utama. Operasi pemampatan citra dapat dilakukan agar citra dapat direpresentasikan dalam bentuk yang lebih kompak sehingga memiliki jumlah bit yang minimal dan ukuran yang tidak terlalu besar.

Terdapat berbagai metode yang dapat digunakan untuk operasi pemampatan citra diantaranya adalah metode transformasi *wavelet* yaitu metode yang akan mengubah citra dari domain spasial ke domain frekuensi. Selain itu ada operasi pemetaan dari daerah intensitas yang lebar menjadi daerah intensitas terbatas yang disebut kuantisasi juga menjadi dasar suatu ilmu kompresi. Sehingga kombinasi transformasi *wavelet* dan operasi kuantisasi bisa menjadi salah satu cara yang cukup baik untuk operasi pemampatan citra. Kompresi citra tidak hanya mengurangi ukuran data, tetapi juga harus dapat menjaga kualitas citra agar mendekati citra asli.

Transformasi *Wavelet* 9/7, jenis *wavelet* ini merupakan *wavelet* biorthogonal yang digunakan sebagai standar untuk JPEG2000. Variasi data hasil transformasi *wavelet* memang sangat tinggi sehingga perlu dilakukan kuantisasi untuk mengurangi variasi data tersebut. Proses ini akan mengubah matrik citra ke vektor-vektor berukuran $1 \times n \times n$ lalu akan dicocokkan dengan vektor

pengkuantisasi yang biasa disebut *codeword*, sehingga ditemukan *codeword* yang tingkat distorsinya minimum. Sebuah algoritma pada metode Kuantisasi Vektor Adaptif atau *Adaptive Vector Quantization* yang ditemukan Kai guo tahun 2007 dimana perubahan *codeword* dilakukan secara *partial*. Menurut Guo (2007) teknik *Partial Codeword Updating* (PCU) ini dapat menghasilkan *rate-distortion* yang optimal daripada menggunakan teknik yang dinamakan *Full Codeword Updating* (FCU). Agar ukuran file yang dikompres semakin kecil digunakan pengkodean *entropy* Huffman yang menggunakan prinsip bahwa nilai derajat intensitas yang sering muncul di dalam citra akan dikodekan dengan jumlah bit yang lebih sedikit sedangkan nilai derajat intensitas yang frekuensi kemunculannya sedikit dikodekan dengan jumlah bit yang lebih panjang. Maka pada penelitian ini akan digunakan Transformasi *Wavelet* 9/7, Kuantisasi Vektor Adaptif dengan algoritma PCU, dan pengkodean *entropy* Huffman.

Dengan adanya permasalahan mengenai keterbatasan media penyimpanan dalam menyimpan data citra digital yang cenderung berukuran besar maka timbul inisiatif untuk merancang sebuah model kompresi citra menggunakan kombinasi antara Transformasi *Wavelet* dan Metode Kuantisasi Vektor Adaptif. Oleh karena itu penelitian ini mengambil judul **Kompresi Citra Digital Menggunakan Transformasi *Wavelet* 9/7 Dan Metode Kuantisasi Vektor Adaptif dengan Algoritma *Partial Codeword Updating*.**

1.2 Rumusan Masalah

Rumusan masalah dalam penelitian ini adalah:

1. Bagaimana merancang aplikasi kompresi citra digital menggunakan Transformasi *Wavelet* 9/7 dan metode Kuantisasi Vektor Adaptif dengan algoritma *partial codeword updating*.
2. Berapa nilai rasio kompresi citra digital yang dihasilkan dengan menggunakan Transformasi *Wavelet* 9/7 dan metode Kuantisasi Vektor Adaptif.
3. Berapa tingkat kesalahan kompresi citra digital menggunakan transformasi *Wavelet* 9/7 dan metode Kuantisasi Vektor Adaptif.

1.3 Batasan Masalah

Batasan masalah dalam penelitian ini adalah sebagai berikut:

1. Inputan citra berwarna 24 bit berukuran 256 x 256 pixel
2. Format citra yang digunakan adalah bitmap (.bmp)
3. Proses dekomposisi transformasi citra hanya 2 level
4. Tidak ada proses pembentukan data *codebook* awal. Data *codebook* awal sudah tersedia
5. Entropy encoding menggunakan Huffman
6. Parameter yang digunakan untuk menganalisis hasil kompresi adalah rasio kompresi dan kualitas citra hasil yang dinilai berdasarkan parameter MSE.

1.4 Tujuan Penelitian

Tujuan penelitian dalam skripsi ini adalah sebagai berikut:

1. Merancang dan mengimplementasikan aplikasi kompresi citra digital dengan Transformasi *Wavelet* 9/7 dan metode Kuantisasi Vektor Adaptif.
2. Mendapatkan dan menganalisis rasio kompresi dari proses kompresi.
3. Menganalisis kualitas citra berdasarkan nilai MSE yang dihasilkan dari proses dekompresi.

1.5 Manfaat Penelitian

Manfaat yang dapat diambil dari penelitian ini adalah:

1. Menyediakan perangkat lunak (*software*) untuk mengkompres dan mendekompresi citra menggunakan Transformasi *Wavelet* 9/7 dan metode Kuantisasi Vektor Adaptif.
2. Menyediakan informasi mengenai rasio pemampatan dan kualitas citra berdasarkan nilai MSE yang dihasilkan dari proses kompresi dengan Transformasi *Wavelet* 9/7 dan metode Kuantisasi Vektor Adaptif.

1.6 Sistematika Penulisan

Pembuatan skripsi ini dilakukan dengan pembagian bab sebagai berikut:

BAB I: PENDAHULUAN

Pada bab ini membahas mengenai latar belakang, rumusan masalah, batasan masalah, tujuan, manfaat, serta sistematika penulisan skripsi.

BAB II: TINJAUAN PUSTAKA

Bab ini menjelaskan tentang pengertian citra digital, kompresi citra digital, *Wavelet*, Kuantisasi, dan pengkodean entropy. Adapun literatur yang digunakan meliputi buku referensi, jurnal penelitian dan dokumentasi internet.

BAB III: METODOLOGI DAN PERANCANGAN

Studi literatur, tahap menambah wawasan dari buku-buku, artikel dan internet, perancangan, penyediaan data uji yang berupa 3 tipe citra digital, yaitu *low detail*, *medium detail*, dan *high detail*, implementasi, menguji dengan berbagai citra digital, menganalisis hasil keluaran proses kompresi dengan transformasi *wavelet* filter 9/7 dan metode kuantisasi vektor adaptif.

.

BAB IV: HASIL DAN PEMBAHASAN

Bab ini menerangkan proses implementasi dari rancangan penelitian yang dijelaskan pada bab III. Selain itu, bab ini juga menjelaskan penerapan aplikasi, analisa hasil percobaan mengenai rasio kompresi dan tingkat kesalahan citra terkompresi yang dikompresi menggunakan Transformasi *Wavelet* 9/7 dan Metode Kuantisasi Vektor Adaptif dengan Algoritma *Partial Codeword Updating*.

BAB V: PENUTUP

Bab lima berisi kesimpulan dari pembahasan dan saran yang diharapkan bermanfaat untuk pengembangan skripsi ini selanjutnya.

BAB II
TINJAUAN PUSTAKA

2.1 Pengertian Citra Digital

Segala hal yang terdapat di bidang dua dimensi dan tertangkap oleh alat optik dapat disebut citra. Citra sebenarnya adalah fungsi kontinu intensitas cahaya f(x,y) pada bidang dua dimensi di titik (x,y). Maksudnya bayangan citra yang sampai di mata dipengaruhi oleh tingkat intensitas cahaya f yang mengenai citra dua dimensi di sembarang titik x dan y tersebut. Agar dapat diolah dengan komputer digital, maka suatu citra harus direpresentasikan secara numerik dengan nilai-nilai diskrit. Representasi citra dari fungsi malar (kontinu) menjadi nilai-nilai diskrit disebut digitalisasi. Citra yang dihasilkan inilah yang disebut citra digital (Munir, 2004). Representasi matrik citra digital seperti yang ditunjukkan pada persamaan 2.1 (Rachmawaty, 2007):

$$f(x,y) = \begin{bmatrix} f(1,1) & f(1,2) & \dots & f(1,M) \\ f(2,1) & f(2,2) & \dots & f(2,M) \\ \dots & & & \dots \\ . & & & \dots \\ . & & & \dots \\ f(N,1) & f(N,2) & \dots & f(N,M) \end{bmatrix} \quad \text{.....(2.1)}$$

Indeks baris (*i*) dan indeks kolom (*j*) menyatakan suatu koordinat titik pada citra, sedangkan $f(i, j)$ merupakan derajat intensitas pada titik (i, j).

Pada citra warna, setiap titik mempunyai warna yang spesifik yang merupakan kombinasi 3 warna dasar, yaitu: merah, hijau, dan biru. Format citra ini disebut sebagai citra RGB (*red-green-blue)*. Setiap warna dasar mempunyai intensitas sendiri dengan nilai maksimum 255 (8 bit), misalnya warna kuning merupakan kombinasi warna merah dan hijau sehingga nilai RGB-nya adalah 255 255 0; sedangkan warna ungu muda nilai RGB-nya adalah 150 0 150. Dengan demikian setiap titik pada citra warna membutuhkan 3 *byte*.

Jumlah kombinasi warna yang mungkin untuk format citra ini adalah 2^{24} atau lebih dari 16 juta warna, dengan demikian bisa dianggap mencakup semua warna yang ada, inilah sebabnya format ini dinamakan *true color* (Achmad, 2004)

2.2 Kompresi Citra Digital

Kompresi citra merupakan suatu usaha untuk membuat ukuran citra menjadi lebih kecil dengan mengurangi duplikasi data-data di citra sehingga menghemat kapasitas media penyimpanan, mempercepat pengiriman data, dan memperkecil kebutuhan bandwidth.

Ada dua jenis hasil kompresi yaitu kompresi bersifat *lossy* dan *loseless*. Dikatakan bersifat *lossy* bilamana proses pengubahan detail dan warna pada citra menjadi lebih sederhana dengan ditandai banyaknya informasi yang hilang tapi masih cukup untuk digunakan sehingga dapat dikenali oleh mata manusia. Hal-hal yang perlu diperhatikan pada kompresi *lossy* yaitu dengan proses kompresi dan dekompresi yang lebih efisien dan ukuran yang lebih kecil tetapi kualitas citra masih dapat dipertahankan. Citra yang biasanya dikompresi secara *lossy* adalah citra-citra biasa yang tidak memerlukan detail dan tidak mengandung informasi yang penting. Sedangkan untuk kebutuhan hasil citra dekompresi yang tepat sama dengan aslinya maka citra akan dikompresi secara *loseless*. Citra akan dikompresi dengan tidak menghilangkan informasi penting di dalamnya. Biasanya digunakan untuk citra medis.

2.3 Algoritma Kompresi / Dekompresi Citra

Algoritma umum untuk kompresi *image* adalah (Anonym, 2006) :

1. Pembagian data *image* ke dalam bagian-bagian tertentu sesuai dengan tingkat kepentingan yang ada (*classifying*). Seperti menggunakan salah satu teknik: DWT (*Discreate Wavelet Transform*) yang akan mencari frekuensi nilai pixel masing-masing, dan menggabungkannya menjadi satu dan mengelompokkannya.

2. Pembagian bit-bit di dalam masing-masing bagian yang ada (*bit allocation*).
3. Lakukan kuantisasi (*quantization*).
   a. **Kuantisasi Skalar** : data-data dikuantisasi sendiri-sendiri
   b. **Kuantisasi Vektor** : data-data dikuantisasi sebagai suatu himpunan nilai-nilai vektor yang diperlakukan sebagai suatu kesatuan.
4. Lakukan pengenkodingan untuk masing-masing bagian yang sudah dikuantisasi tadi dengan menggunakan teknik *entropy coding* (huffman) dan menuliskannya ke dalam file hasil.

Sedangkan algoritma umum dekompresi *image* adalah :

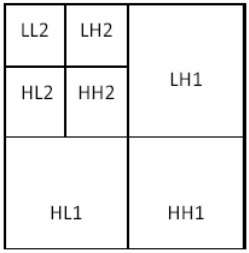
1. Baca data hasil kompresi menggunakan *entropy dekoder*.
2. Dekuantisasi data.
3. *Rebuild image*.

2.4 *Wavelet*

Wavelet merupakan gelombang singkat (*small wave*) yang energinya terkonsentrasi pada suatu selang waktu digunakan untuk menganalisis suatu sinyal (Rachmawaty, 2007).

2.4.1 Transformasi *Wavelet*

Transformasi *Wavelet* akan mengubah citra dari domain spasial ke domain frekuensi sehingga dimungkinkan nilai hasil transformasi tersebut dapat merepresentasikan data yang ukuran filenya lebih kecil dari file aslinya. Transformasi *wavelet* mendekomposisi sinyal input berupa citra menjadi beberapa macam frekuensi dalam *sub-band* citra. Transformasi *wavelet* 2 dimensi dilakukan terhadap baris, kemudian terhadap kolom, atau sebaliknya. Gambar 2.1 menunjukkan pemetaan atau pembagian dekomposisi *wavelet*.



Gambar 2.1 Pemetaan Dekomposisi *Wavelet*
Sumber : Rachmawaty, 2007

Keterangan hasil subband:
HH1=Filtering dengan *high-pass filter* horizontal dan *highpass filter* vertikal
HL1= Filtering dengan *high-pass filter* horizontal dan *lowpass filter* vertikal
LH1= Filtering dengan *low-pass filt*er horizontal dan *highpass filter* vertikal
LL1= Filtering dengan *low-pass filter* horizontal dan *lowpass filter* vertikal

Untuk dekomposisi sampai level dua, maka *sub-band* LL1 yang disebut juga koefisien *approximation* akan didekomposisi lagi menjadi LL2, LH2, HL2, dan HH2. Sedangkan LH, HL dan HH merupakan koefisien detail vertikal, detail horizontal, dan detail diagonal. *Sub-band* yang mengandung keluaran *High* (H) di masing-masing level ditinggalkan tanpa didekomposisi lagi.

Analisis data pada transformasi *wavelet* yang dilakukan dengan membagi (dekomposisi) suatu sinyal ke dalam komponen-komponen frekuensi yang berbeda-beda seperti halnya pada proses *filtering*, dimana sinyal dalam domain waktu dilewatkan ke dalam filter *highpass* dan *lowpass* dan memisahkan komponen frekuensi tinggi dan frekuensi rendah.

2.4.2 Implementasi skema *Lifting* pada *Discreate Wavelet Transform*

Prinsip dasar dari skema *lifting* DWT (*Discreate Wavelet* Transform) adalah merubah koefisien filter *wavelet* yaitu *high-pass* dan *low-pass* menjadi rangkaian koefisien filter yang lebih sederhana. Sehingga skema *lifting* ini memiliki keunggulan dibandingkan dengan skema konvolusi (DWT bentuk lama) yaitu efisiensi komputasi, biasanya *lifting-based DWT* memerlukan komputasi yang lebih singkat hanya 50% dari waktu komputasi yang diperlukan dengan teknik konvolusi (Acharya, 2005).

Penelitian skema *lifting* transformasi *wavelet* yang telah dilakukan oleh Daubechies dan Swelden tahun 1996 menjelaskan pasangan filter *high-pass G(z)* dan *low-pass H(z)* akan direpresentasikan dalam bentuk *polyphase* yaitu suatu model untuk merepresentasikan struktur khusus dari filter. Bentuk *polyphase* filter ditunjukkan pada persamaan 2.2.

$$\begin{aligned} H(z) &= h_0 + h_1 z^{-1} + h_2 z^{-2} + h_3 z^{-3} + h_4 z^{-4} + ... \\ &= (h_0 + h_2 z^{-2} + h_4 z^{-4} + ...) \\ &\quad + z^{-1}(h_1 + h_3 z^{-2} + h_5 z^{-4} + ...) \\ &= H_e(z^2) + z^{-1} H_o(z^2) \end{aligned}$$

$$G(z) = G_e(z^2) + z^{-1} G_o(z^2) \quad \text{.....(2.2)}$$

Dimana:
H_e *dan* G_e : koefisien filter genap
H_o *dan* G_o : koefisien filter ganjil.

Persamaan 2.2 dapat direpresentasikan kedalam bentuk matrik seperti yang ditunjukkan pada persamaan 2.3. (Kotteri, 2004).

$$\begin{bmatrix} V_0(z) \\ V_1(z) \end{bmatrix} = \begin{bmatrix} H_e(z) & H_o(z) \\ G_e(z) & G_o(z) \end{bmatrix} \begin{bmatrix} X_e(z) \\ z^{-1}X_o(z) \end{bmatrix} \quad \text{.....(2.3)}$$

Dimana:
$V_0(z)$: *low-pass subband*
$V_1(z)$: *high-pass subband*

Sehingga didefinisikan *polyphase matrix* seperti yang ditunjukkan pada persamaan 2.4 (Daubechies, 1996):

$$P(z) = \begin{bmatrix} H_e(z) & H_o(z) \\ G_e(z) & G_o(z) \end{bmatrix} \quad \text{.....(2.4)}$$

Berdasarkan penelitian Daubechies dan Swelden tahun 1996 telah dilakukan suatu komputasi terhadap *polyphase matrik* sehingga akan terbentuk faktorisasi dari *polyphase matrix*, seperti yang ditunjukkan pada persamaan 2.5.

$$\tilde{P}(z) = \left\{ \prod_{i=1}^{m} \begin{bmatrix} 1 & \tilde{s}_i(z) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \tilde{t}_i(z) & 1 \end{bmatrix} \right\} \begin{bmatrix} K & 0 \\ 0 & 1/K \end{bmatrix} \quad \text{.....(2.5)}$$

Penelitian Daubechies dan Swelden tahun 1996 telah menemukan koefisien-koefisien *lifting* dari filter Wavelet 9/7 untuk proses transformasi. Filter *low-pass H(z)* dan high-pass *G(z)* Wavelet 9/7 berdasarkan persamaan 2.2 dapat dinyatakan seperti pada persamaan 2.6

$$\begin{aligned} H(z) &= h_0 + h_1(z + z^{-1}) + h_2(z^2 + z^{-2}) + h_3(z^3 + z^{-3}) + h_4(z^4 + z^{-4}) \\ G(z) &= g_0 z + g_1(z^2 + 1) + g_2(z^3 + z^{-1}) + g_3(z^4 + z^{-2}) \end{aligned} \quad \text{.....(2.6)}$$

Dimana masing-masing koefisien filter Wavelet 9/7 yang ditemukan oleh Ingrid Daubechies, sebagai berikut:

h_o = 0.85269867900889 g_o = 0.78848561640558
h_1 = 0.37740285561283 g_1 = -0.41809227322162
h_2 = -0.11062440441844 g_2 = -0.04068941760916
h_3 = -0.02384946501956 g_3 = 0.06453888262870
h_4 = 0.03782845550726

Berdasarkan persamaan 2.4 dapat dibentuk *polyphase matrix* untuk filter *Wavelet* 9/7 yang ditunjukkan pada persamaan 2.7.

$$P(z) = \begin{bmatrix} h_4(z^2+z^{-2}) + h_2(z+z^{-1}) + h_0 & h_3(z^2+z^{-1}) + h_1(z+1) \\ g_1(1+z) + g_3(z^2+z^{-1}) & g_0 z + g_2(z^2+1) \end{bmatrix} \quad ..(2.7)$$

Setelah terbentuk *polyphase matrix,* Daubechies dan Swelden melakukan komputasi untuk mendapatkan faktorisasi *polyphase matrix Wavelet* 9/7 dan hasilnya seperti yang ditunjukan pada persamaan 2.8.

$$P(z) = \begin{bmatrix} 1 & \alpha(1+z^{-1}) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \beta(1+z) & 1 \end{bmatrix} \begin{bmatrix} 1 & \gamma(1+z^{-1}) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \delta(1+z) & 1 \end{bmatrix} \begin{bmatrix} \zeta & 0 \\ 0 & 1/\zeta \end{bmatrix} \quad .....(2.8)$$

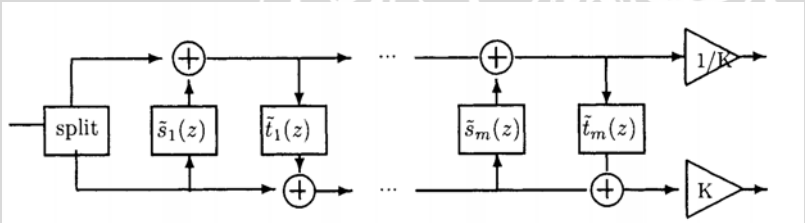
Symbol α, β, γ, δ, dan ζ adalah yang disebut koefisien-koefisien *lifting* yang ditemukan oleh Daubechies dan Swelden dan mempunyai nilai koefisien seperti yang tunjukkan pada tabel 2.1.

Tabel 2.1. Koefisien *Lifting Wavelet* 9/7

	Irrational	Rational
α	-1.58613434...	-3/2
β	-0.0529801185...	-1/16
γ	0.882911076...	4/5
δ	0.443506852...	15/32
ζ	1.14960439...	$4\sqrt{2}/5$

2.4.2.1 *Lifting-based Forward* Transformasi *Wavelet* 9/7

Pada skema *lifting* ini *sample* sinyal masukan di bagi menjadi dua bagian yaitu *sample* sinyal genap dan *sample* sinyal ganjil lalu masing-masing *sample* sinyal secara bergantian di eksekusi dengan koefisien *lifting* yang sesuai. Untuk *sample* sinyal ganjil di eksekusi koefisien maju *lifting* transformasi *wavelet* 9/7 yang mempunyai nilai a = -1.586134342 dan c = 0.8829110762 sedangkan *sample* sinyal genap koefisien *lifting* transformasi *wavelet*-nya mempunyai nilai b = -0.05298011854 dan d = 0.4435068522 dan hasil keluaran masing-masing *sample* sinyal di skala dengan nilai 1/K dan K (K= 1.149604398) untuk menghasilkan *high-pass sub-band* dan *low-pass sub-band*. Sehingga bisa dikatakan keluaran akhir sampel sinyal ganjil merupakan *high-pass sub-band* dan keluaran sampel sinyal genap merupakan *low-pass sub-band.* Harap menjadi catatan bahwa sampel ganjil dikalkulasi dari sampel genap dan sampel genap dikalkulasi dari hasil sampel ganjil. Agar mempermudah untuk mengerti algoritma komputasi dari skema *lifting-based forward* Transformasi *Wavelet* 9/7 dapat dijelaskan pada gambar 2.2 dan 2.3 berikut ini:

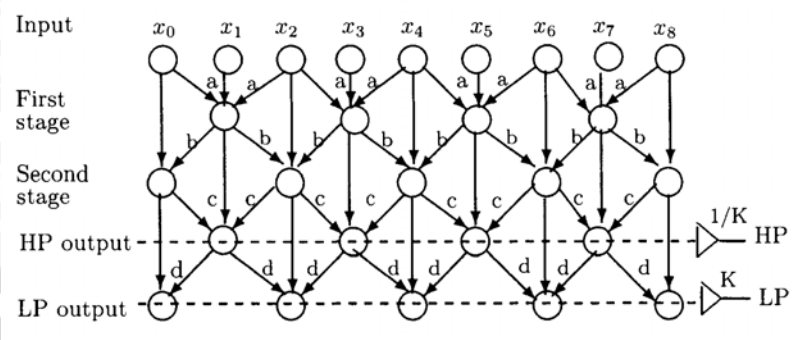


Gambar 2.2 Proses Transformasi Maju
Sumber : Acharya, 2005

Keterangan Simbol Proses Transformasi Maju:

$\tilde{s}_m(z)$ = koefisien *lifting* untuk sampel sinyal ganjil

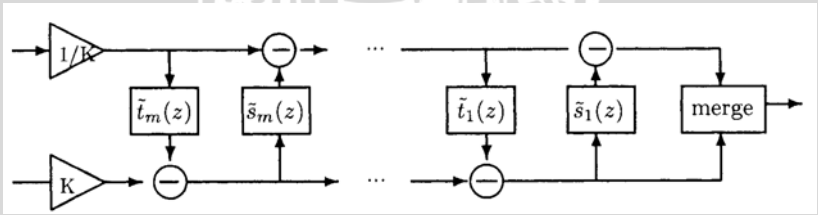
$\tilde{t}_m(z)$ = koefisien *lifting* untuk sampel sinyal genap



Gambar 2.3 Proses komputasi maju *lifting* transformasi 9/7
Sumber : Acharya, 2005

2.4.2.2 *Lifting-based Invers* Transformasi *Wavelet* 9/7

Untuk mengembalikan *sub-bands* yang telah didekomposisi untuk keperluan rekonstruksi, maka dilakukan operasi invers transormasi *wavelet*. Nilai *lowpass-sub-band* dan *high-pass sub-band* masing-masing di skala dengan 1/K dan K (K = 1.149604398) lalu hasil sampel sinyal yang telah diskalakan di eksekusi dengan koefisien balik *lifting* transformasi *wavelet* 9/7 yang sesuai. Untuk sampel sinyal genap eksekusi koefisien balik *lifting* d = -0.4435068522 dan b = 0.05298011854 dan sampel sinyal ganjil eksekusi koefisien balik *lifting* c = -0.8829110762 dan a = 1.586134342. Pada gambar 2.4 dan 2.5 menjelaskan algoritma komputasi dari skema *lifting-based invers* Transformasi *Wavelet* 9/7.

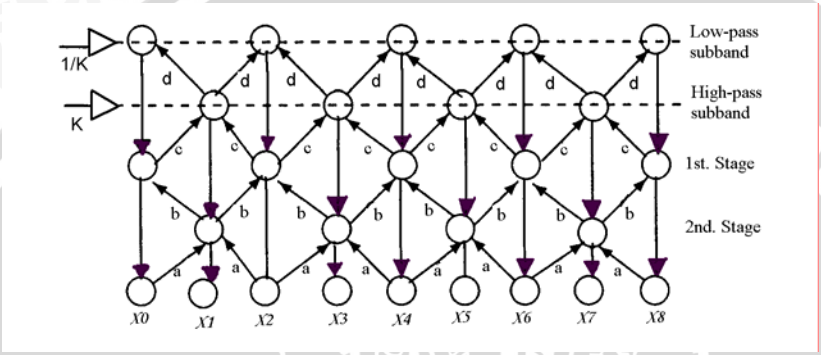


Gambar 2.4 Proses Transformasi balik
Sumber : Acharya, 2005

Keterangan Simbol Proses Transformasi Balik:

$\tilde{s}_m(z)$ = koefisien *lifting* untuk sampel sinyal ganjil

$\tilde{t}_m(z)$ = koefisien *lifting* untuk sampel sinyal genap



Gambar 2.5 Proses komputasi balik *lifting* transformasi 9/7
Sumber : pengamatan

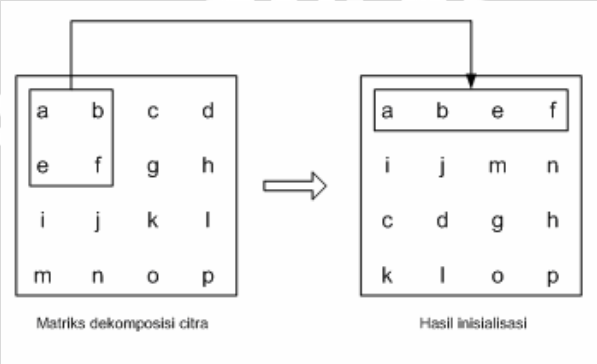
2.5 Kuantisasi

Walaupun sudah diambil sebagian kecil saja, variasi data hasil transformasi *wavelet* memang sangat tinggi sehingga perlu dilakukan kuantisasi untuk mengurangi variasi data tersebut. Kuantisasi adalah proses mencari nilai perwakilan paling tepat dari beberapa nilai numerik sehingga dapat merepresentasikan data di citra. Data yang tadinya direpresentasikan oleh banyak bit di citra dapat diminimalisasi dengan dikonversi ke nilai dengan tingkat presisi lebih rendah sehingga bit representasi citra dapat berkurang. Ilustrasinya, misalnya ada 4 warna, biru muda, biru sangat muda, biru, biru sedikit tua. Agar tidak terlalu banyak representasi data, maka keempat warna itu dinyatakan dengan biru saja (Rachmawaty, 2007).

Ada dua jenis kuantisasi yaitu kuantisasi skalar dan kuantisasi vektor. Kuantisasi skalar akan mengkuantisasi pada setiap nilai diskrit atau tiap-tiap koefisien secara terpisah. Sedangkan kuantisasi vektor akan mengkuantisasi himpunan nilai diskrit yang masih dalam satu vektor. Pada sistem ini, dikatakan satu vektor bilamana pixel

berdekatan pada citra memiliki hubungan. Ada kemungkinan yang sangat besar bahwa pixel yang bertetangga dengan suatu pixel P akan mempunyai nilai yang sama atau hampir sama dengan pixel P.

Dalam kuantisasi vektor awalnya ada beberapa citra uji yang dibagi menjadi vektor-vektor berukuran n x n. Setiap blok n x n pixel akan diubah menjadi satu vektor berdimensi 1 x n x n. Proses matriks n x n menjadi bentuk vektor 1 x n x n ditunjukkan pada gambar 2.6 berikut:



Gambar 2.6 Proses matrik n x n → 1 x n x n
Sumber : Rachmawaty, 2007

Lalu vektor-vektor tersebut dijadikan vektor pengkuantisasi yang disebut *codeword*. Himpunan *codeword* akan membentuk *codebook* yang nantinya dijadikan sebagai pengkuantisasi. Pengkuantisasi ini bisa diartikan sebagai kamus untuk membandingkan antara vektor input dengan *codeword.* Pada sistem ini, *codebook* telah tersedia sehingga tidak ada proses pembentukan *codebook*.

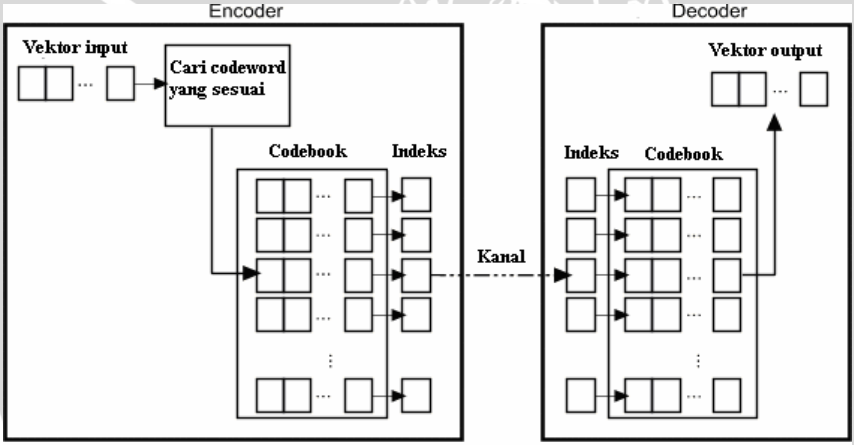
Lalu ada citra input yang menghasilkan vektor input. Vektor input inilah yang akan dibandingkan dengan setiap *codeword* di *codebook*. Hasilnya adalah indek-indek *codeword* atau dengan kata lain vektor input yang dikodekan dengan indek *codeword* yang memiliki jarak terpendek dengan vektor input menurut rumus jarak *Euclidean*. Jarak *Euclidean* didefinisikan pada persamaan 2.9 sebagai berikut (Rachmawaty, 2007):

$$d(x, y_i) = \sqrt{\sum_{j=1}^{k} (x_j - y_{ij})^2} \quad \text{.....(2.9)}$$

dimana:

x_j = komponen ke-j vector input
y_{ij} = komponen ke-j *codeword* yi.

Sebuah pengkuantisasi vektor terdiri dari dua bagian, yaitu: pengkode (*encoder*) dan pendekode (*decoder*). Fungsi pengkode adalah mengambil vektor input dan menghasilkan indeks *codeword* yang memberikan distorsi paling minimum. Indeks *codeword* inilah yang akan disimpan dalam media penyimpanan data digital atau dikirimkan melalui kanal komunikasi. Pendekode akan mengubah indeks *codeword* yang diterimanya dengan *codeword* dari *codebook* ke vektor paling representatif. *Codebook* pada sisi pengkode dan pendekode harus sama. Gambar 2.7 menunjukkan *encoder* dan *decoder* di pengkuantisasi vektor.



Gambar 2.7 *Encoder* dan *Decoder* di Pengkuantisasi Vektor
Sumber : Rachmawaty, 2007

2.5.1 Kuantisasi vektor adaptif

Teori tingkat-distorsi Shannon menyatakan bahwa kuantisasi vektor kurang optimal untuk pengkodean citra input yang berbeda-beda (Wickerhauser, 2002). Untuk mengatasi masalah ini, maka diciptakan algoritma vektor kuantisasi adaptif yang *codebook*-operasionalnya dapat berubah mengikuti perubahan pada citra input

yang berbeda sehingga didapatkan citra rekonstruksi yang hasilnya lebih baik.

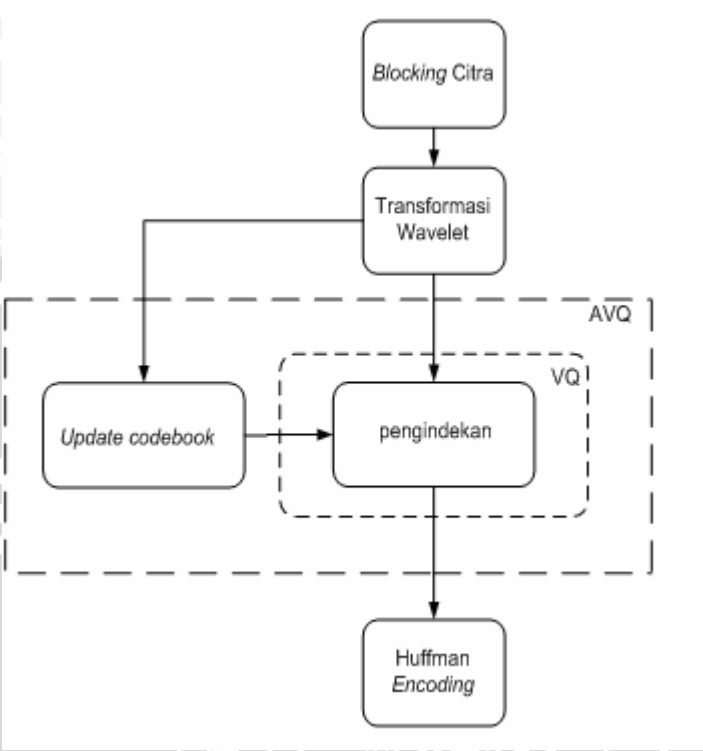
Tabel 2.2. Perbedaan Kuantisasi Vektor dan Kuantisasi Vektor Adaptif:
Sumber : Rachmawaty, 2007

No.	Pembanding	Kuantisasi vektor	Kuantisasi vektor adaptif
1	Citra input	Bersifat *stationary source*, yaitu citra input harus sesuai dengan *codebook*. Bila tidak sesuai, maka citra dekompresi terlihat tidak baik.	Bersifat *non-stationary source*, yaitu citra input dapat bervariasi dan dengan adanya fasilitas *update-codebook* sesuai citra input dengan kondisi tertentu maka citra dekompresi setidaknya terlihat lebih baik di mata pengamat daripada kuantisasi vektor non-adaptif
2	*Codebook*	Bersifat statis	Bersifat dinamis

AVQ sebenarnya merupakan teknik untuk mengubah isi *codebook* pada sistem kuantisasi vektor sehingga *codebook* bersifat dinamis. Ada beberapa macam teknik yang pernah digunakan, yaitu *Mean-adaption, Gain-adaption, Switched-codebook-adaption Finite-state* VQ, *Predictive* VQ, *Variable-dimension* VQ.

Teknik yang digunakan untuk penelitian ini adalah AVQ *algorithm with the partial codeword updating* (PCU) *scheme*.

Ilustrasi sistem AVQ ini dtunjukkan pada gambar 2.8 sebagai berikut:



Gambar 2.8 Hubungan Sistem AVQ dengan VQ
Sumber : Rachmawaty, 2007

2.5.2 AVQ dengan skema *Partial Codeword Updating*

AVQ dengan skema *Partial Codeword Updating* (PCU) hanya mengubah komponen *codeword* yang memiliki kuantisasi *error* (selisih antara vektor input dengan vektor pengkuantisasi) lebih besar dari *threshold T*. Pada proses pengkodean dalam PCU-AVQ ini digunakan kriteria *Rate-Distortion* untuk menentukan *mode* mana yang terbaik diantara *mode* tanpa perubahan, *mode Full Codeword Updating* (FCU), atau *mode* PCU. Lalu ada pengganda *Lagrange* λ yang berfungsi untuk mengatur *rate* dan *distortion* tersebut. Nilai λ yang kecil akan mempertegas nilai distorsi oleh karenanya nilai *threshold T* juga harus kecil untuk mengurangi nilai distorsi dan bila nilai λ *yang* besar akan mempertegas nilai *bit rate* sehingga nilai

threshold harus besar untuk mengontrol *bit rate*. *Rate-Distortion* dapat dinyatakan pada persamaan 2.10 sebagai berikut :

$$J(\mathbf{x}, \mathbf{y}; \lambda) = d(\mathbf{x}, \mathbf{y}) + \lambda \cdot r(\mathbf{y}) \quad \text{.....(2.10)}$$

Dimana x, y adalah dua vektor, $d(\mathbf{x}, \mathbf{y})$ dan $r(\mathbf{y})$ masing-masing mewakili distorsi dan *bit rate*. *Bit rate* adalah sejumlah *bit* per data *sample* yang akan ditransmisikan, sedangkan distorsi adalah varian atau nilai yang merupakan perbedaan antara nilai *input* dan *output* (Babu, 2008). Dengan menetapkan λ dan *threshold T* akan dapat meminimalkan nilai *Rate-distortion* (Guo, 2007).

Algoritma PCU-AVQ dapat dinyatakan sebagai berikut (Guo, 2007):

1. Inisialisasi *codebook P*, pengganda *Lagrange* λ dan *threshold T*
2. Temukan jarak terdekat *codeword p* terhadap vektor input s berdasarkan *Euclidean norm*
3. Bangkitkan *partial updated codeword p(T)* dimana,

$$p_j(T) = \begin{cases} s_j \; jika \;\; |s_j - p_j| > T \\ p_j \; selainnya \end{cases}$$

Buat informasi yang mencatat lokasi perubahan *codeword u(T)* dimana,

$$u_j(T) = \begin{cases} 1, \; jika \;\; |s_j - p_j| > T \\ 0, \; selainnya \end{cases}$$

4. Hitung besar *Rate-Distortion* $J_3 = d(s,p(T)) + \lambda[r(i) + r(p(T)) + r(u(T))]$ (PCU *mode*), $J_1 = d(s,p) + \lambda r(i)$ (*non-updating mode*), $J_2 = \lambda[r(i) + r(s)]$ (FCU *mode*);
   If $J1 = min(J1, J2J3)$, *no updating happens*;
   If $J2 = min(J1, J2J3)$, FCU *mode is selected*;
   If $J3 = min(J1, J2J3)$, PCU *mode is selected*;

2.6 Pengkodean *Entropy* Huffman

Agar ukuran file yang dikompres semakin kecil, maka dapat digunakan metode pengkodean *entropy* Huffman. Teknik pemampatan Huffman merupakan salah satu jenis pengkodean statistik (*statistical encoding*) yang bertujuan untuk mengurangi rata-rata panjang kode yang digunakan untuk merepresentasikan simbol, dalam hal ini pixel pada citra (Rachmawaty, 2007). Teknik ini akan mengkodekan derajat intensitas yang sering muncul dengan jumlah bit yang lebih sedikit dan derajat intensitas yang frekuensi kemunculannya lebih sedikit akan dikodekan dengan jumlah bit yang lebih panjang.

Setelah kode Huffman terbentuk, maka data citra akan disusun ke dalam urutan bit berdasarkan kode Huffman yang bersesuaian. Kode Huffman ini dapat disebut sebagai kode prefiks, yaitu tidak ada kode biner derajat intensitas yang merupakan awalan bagi kode biner derajat intensitas yang lain (Munir, 2004). Adapun algoritma atau prosedur pembuatan kode Huffman pada citra digital adalah (Munir, 2004):

1. Urutkan secara menaik (*ascending order*) atau menurun (*descending order*) nilai-nilai derajat intensitas berdasarkan frekuensi kemunculan (atau berdasarkan peluang kemunculan , p_k, yaitu frekuensi kemunculan (n_k) dibagi dengan jumlah *pixel* di dalam gambar (n)). Setiap nilai derajat intensitas dinyatakan sebagai pohon bersimpul tunggal. Setiap simpul di-*assign* dengan frekuensi kemunculan nilai derajat intensitas tersebut.
2. Gabung dua buah pohon yang mempunyai frekuensi kemunculan paling kecil pada sebuah akar. Akar mempunyai frekuensi yang merupakan jumlah dari frekuensi dua buah pohon penyusunnya.
3. Ulangi langkah 2 sampai tersisa hanya satu buah pohon biner.
4. Beri label setiap sisi pada pohon biner. Sisi kiri dilabeli dengan 0 dan sisi kanan dilabeli dengan 1.
5. Untuk mengkodekan setiap pixel di dalam citra, telusuri pohon biner dari akar ke daun. Barisan label-label sisi dari akar ke daun menyatakan kode Huffman untuk derajat intensitas yang bersesuaian.

   Contoh proses pembentukan kode Huffman dapat dilihat pada lampiran II.

2.7 Rasio kompresi

Rasio kompresi merupakan nilai yang menyatakan berapa persen suatu citra dapat dikompresi. Rasio kompresi didefinisikan seperti pada persamaan 2.11 (Munir, 2004) :

$$Ratio = \left(1 - \left(\frac{ukuran\_citra\_terkompres}{ukuran\_citra\_asli}\right)\right) \times 100\% \quad .....(2.11)$$

2.8 *Mean Square Error*

MSE atau *Mean Square Error* adalah rata-rata kuadrat nilai *error* antara citra asli dengan citra hasil rekonstruksi atau dengan kata lain ketidaksamaan yang terjadi antara citra asli dengan citra hasil dekompresi. MSE secara matematis dapat dirumuskan pada persamaan 2.12.

$$MSE = \frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} [f'(x,y) - f(x,y)]^2 \quad .....(2.12)$$

dimana :

M	= Panjang citra hasil (dalam pixel)
N	= Lebar citra hasil (dalam pixel)
(x,y)	= Koordinat masing-masing pixel
f’	= Nilai pixel citra dekompresi pada koordinat x,y
f	= Nilai pixel citra asli pada koordinat x,y

2.9 Cara menarik kesimpulan Data Hasil

Menurut Walpole tahun 1995, untuk menyelidiki segugus data kuantitatif akan sangat membantu bila kita mendefinisikan ukuran-ukuran numerik yang menjelaskan ciri-ciri data yang penting. Salah satu cara yang dapat ditempuh adalah penggunaan rata-rata.

Bagi orang awam sekumpulan variasi data yang disajikan akan sulit dipahami arti dari data-data tersebut. Sehingga bisa dikatakan sekumpulan data mempunyai arti bila terdapat parameter. Pengertian parameter tersebut adalah sembarang nilai yang menjelaskan ciri

populasi atau data. Dalam hal ini parameter yang dimaksut adalah nilai rata-rata.

Rata-rata digunakan untuk membantu menjelaskan ciri-ciri data sehingga mampu mencerminkan kondisi sebenarnya dari data. Jika kondisi data sudah diketahui maka akan mempermudah untuk menarik kesimpulan terhadap data yang disajikan.

2.10 Standar Deviasi

Standar Deviasi merupakan alat ukur sebaran terbaik, secara umum memiliki pengertian ukuran terpencarnya data amatan dari nilai tengah. Perumpamaan awam untuk standar deviasi ini, adalah seperti melempar batu ke permukaan air yang tenang, riak terluar yang muncul adalah bentuk standar deviasinya. Standar Deviasi secara matematis dapat dirumuskan pada persamaan 2.13.

$$s = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \overline{x})^2} \quad \text{.....(2.13)}$$

dimana :

S = Standar Deviasi
N = Jumlah data sampel
x_i = Nilai data sampel
$\overline{x}$ = Rata – rata data sampel

Sejumlah data sampel jika memiliki nilai Standar Deviasi kecil maka nilai keragaman data yang dimiliki juga kecil, demikian sebaliknya jika nilai Standar Deviasi besar maka nilai keragaman data yang dimiliki besar.

BAB III
METODOLOGI DAN PERANCANGAN

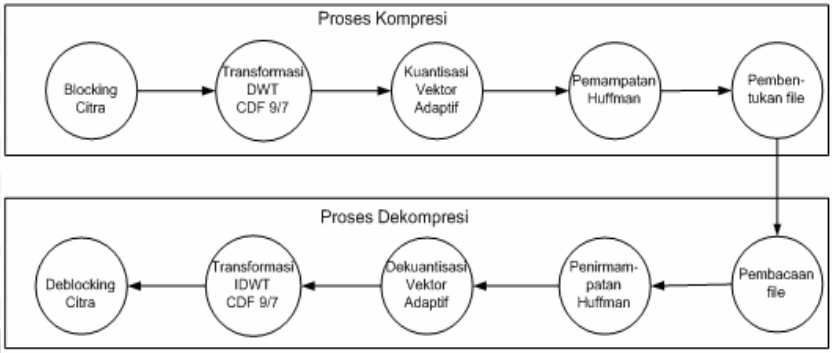
Pada bab metodologi dan perancangan akan dibahas metode, rancangan yang digunakan, serta langkah-langkah yang dilakukan dalam mengkompresi dan mendekompresi citra menggunakan Transformasi *Wavelet* 9/7 Dan Metode Kuantisasi Vektor Adaptif dengan Algoritma Partial Codeword Updating. Langkah-langkah yang dilakukan dalam penelitian ini meliputi:

1. Mempelajari metode yang digunakan dari referensi yang pernah ada, yang telah disinggung pada bab 2.
2. Menganalisa dan merancang perangkat lunak dengan menggunakan penggabungan metode pada penelitian sebelumnya.
3. Membuat perangkat lunak berdasarkan analisis dan perancangan yang dilakukan.
4. Uji coba perangkat lunak dan evaluasi hasil uji coba.

3.1 Analisa Perangkat Lunak

3.1.1 Deskripsi Umum Perangkat Lunak

Sistem atau perangkat lunak yang dibuat dalam penelitian ini terdiri dari dua proses utama yaitu proses kompresi dan dekompresi. Untuk setiap proses utama, terdapat lima subproses lagi seperti yang ditunjukkan pada Gambar 3.1. Citra masukan akan melalui serangkaian proses kompresi hingga dihasilkan *file* citra terkompresi. Sedangkan *input*-an yang dibutuhkan pada proses dekompresi berupa *file* citra terkompresi. *File* ini akan melewati serangkaian proses dekompresi hingga dihasilkan citra rekonstruksi.



Gambar 3.1 Blok kompresi dan dekompresi

Perangkat lunak ini merupakan sebuah program yang akan mengkompresi *file* citra digital masukan user yang mengimplementasikan penggunaan implementasi *lifting wavelet* 9/7 pada saat transformasinya dan menerapkan Kuantisasi Vektor Adaptif dengan skema parsial. Adapun proses-proses yang dilakukan untuk mengkompresi adalah sebagai berikut:

1. *User* memasukkan *input*-an citra berwarna.
2. Dilakukan *blocking* citra.
3. Masing-masing elemen warna (RGB) ditransformasi dengan Transformasi *Wavelet* 9/7.
4. Masing-masing elemen warna (RGB) dikuantisasi menggunakan metode PCU-AVQ.
5. Masing-masing elemen warna (RGB) dikompres menggunakan Pemampatan Huffman yang akan menghasilkan *file* citra terkompresi.

Sedangkan proses-proses yang dilakukan untuk mendekompres langkah-langkahnya adalah sebagai berikut:

1. *File* citra terkompresi akan melalui proses pembacaan file.
2. Masing-masing elemen warna (RGB) akan melalui proses Penirmampatan Huffman lalu Dekuantisasi Vektor.
3. Masing-masing elemen warna (RGB) ditransformasi menggunakan *Invers* Transformasi *Wavelet* 9/7.
4. Masing-masing elemen warna (RGB) disatukan kembali melalui proses *deblocking* citra.

Untuk mengetahui seberapa efektif penggunaan sistem dalam penelitian sebagai salah satu ide solusi untuk kompresi citra maka sistem akan memberikan nilai sebagai bentuk performansi sistem. Proses itu terdiri dari menghitung *Mean Squarerror* (MSE), dan menghitung Rasio Kompresi.

3.1.2 Batasan Perangkat Lunak

Batasan perangkat lunak ini adalah:

1. Inputan citra berwarna 24 bit berukuran 256 x 256 pixel dengan ekstensi bitmap (.bmp).
2. Proses dekomposisi transformasi citra hanya 2 level.
3. Tidak ada proses pembentukan data *codebook* awal. Data *codebook* awal sudah melalui proses optimasi dan tersedia untuk perangkat lunak.

3.2 Perancangan Perangkat Lunak

3.2.1 Perancangan Struktur *File* Kompresi

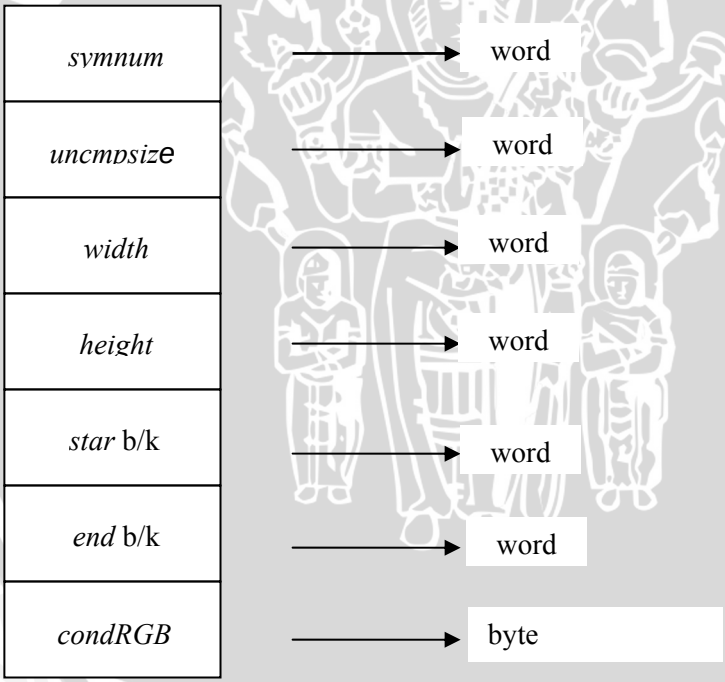
Berkas kompresi terdiri dari dua bagian yaitu *Header file* dan data kompresi. *Header file* berisi informasi mengenai data kompresi. Informasi tersebut berguna untuk proses dekompresi. Sehingga citra rekonstruksi sesuai dengan citra aslinya. Bagian *File* dijelaskan pada Tabel 3.1.

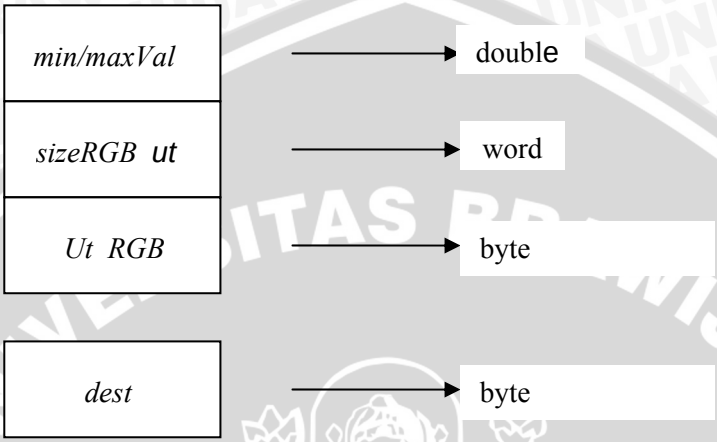
Tabel 3.1 Bagian *File* kompresi

HEADER FILE	
Symnum	*Symbol number*: berisi informasi mengenai jumlah derajat intensitas yang muncul
UncmpSize	*Uncompressed size*: berisi informasi mengenai besar file citra sebelum dimampatkan.
Width	Berisi informasi mengenai lebar citra.
Height	Berisi informasi mengenai tinggi citra.
Start baris-kolom	Berisi informasi mengenai lokasi *codeword* terpilih.
End baris-kolom	

Cond R/G/B	Berisi informasi mengenai *mode* yang digunakan pada proses kuantisasi per subband
minVal / *maxVal*	Berisi informasi mengenai nilai yang digunakan untuk proses *convert to byte* dan *convert to double*
*Size*R/G/B_ut	Berisi informasi mengenai besar faktor **Ut** masing-masing elemen warna
Ut_R/G/B	Berisi informasi lokasi komponen *codeword* yang ter-*update* atau tidak
Data kompresi	
Dest	*Destination*: berisi data kompresi ketiga komponen warna RGB

Struktur *file* kompresi dapat ditunjukkan seperti pada Gambar 3.2. *File* kompresi disimpan dengan ekstensi .huff dan hanya bisa dibuka atau didekompresi menggunakan perangkat lunak ini.





Gambar 3.2 Struktur *file* kompresi

3.2.2 Perancangan Proses Kompresi

3.2.2.1 *Blocking Citra*

Blocking citra dilakukan setelah data citra dibaca, dengan memisahkan kanal citra menjadi 3 kanal yaitu elemen_R(merah), elemen_G (hijau), dan elemen_B (biru). Proses pemisahan kanal dilakukan agar masing-masing komponen dapat ditransformasi secara independen. Meskipun ditransformasi secara terpisah, tetapi proses yang akan diterapkan pada ketiga kanal tersebut sama.

Langkah-langkah yang dilakukan pada tahap ini adalah:

1. Memasukkan *input*-an berupa citra.
2. Melakukan *looping* (perulangan) sepanjang lebar dan tinggi citra.
3. Mengambil nilai RGB citra pada pixel tertentu.

3.2.2.2 *Lifting-based Forward Wavelet* 9/7 *Transform*

Pada tahap ini akan dilakukan transformasi matrik dari *domain spasial* ke *domain* frekuensi. Proses ini akan mendekomposisi citra sampai dua level. Pertama mendekomposisi terhadap baris citra, lalu terhadap kolom citra sehingga terdapat empat frekuensi pembentuk citra yang berbeda, yaitu subband diagonal, subband vertikal, subband horizontal dan residue pelolos rendah. Residue pelolos

rendah didekomposisi lagi sehingga akan ada 7 buah frekuensi pembentuk citra yang berbeda, seperti yang ditunjukkan pada gambar 2.1. Pada akhir transformasi blok n x n pixel di masing-masing subband akan diubah menjadi satu vektor berdimensi 1 x n x n sehingga akan terbentuk 7 vektor input.

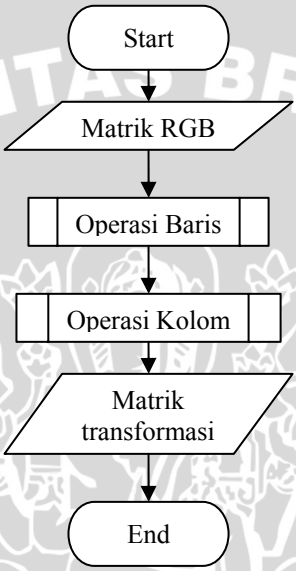
Langkah-langkah yang dilakukan pada tahap ini adalah:

1. Memasukkan matrik elemen_R, elemen_G, dan elemen_B.
2. Untuk masing elemen lakukan proses transformasi terhadap baris.
   *Step*1. Untuk koefisien ganjil: jumlahkan koefisien genap sebelum dan sesudah koefisien ganjil tersebut lalu kalikan dengan nilai a (a = -1,586134342) kemudian tambah dengan koefisien ganjil tersebut
   *Step*2. Untuk koefisien genap: jumlahkan koefisien ganjil hasil dari *step*1 sebelum dan sesudah koefisien genap tersebut lalu kalikan dengan nilai b (b = -0,05298011854) kemudian tambah koefisien genap tersebut
   *Step*3. Untuk koefisien ganjil hasil dari *step* 1: jumlahkan koefisien genap hasil dari *step* 2 sebelum dan sesudah koefisien ganjil tersebut lalu kalikan dengan nilai c (c = 0,8829110762) kemudian tambah koefisien ganjil tersebut
   *Step*4. Untuk koefisien genap hasil dari *step* 2: jumlahkan koefisien ganjil hasil dari *step* 3 sebelum dan sesudah koefisien genap tersebut lalu kalikan dengan nilai d (d = 0,4435068522) kemudian tambah koefisien genap tersebut
   *Step*5. Untuk koefisien ganjil hasil dari *step* 3 kalikan dengan nilai 1/K dan untuk koefisien genap hasil dari *step* 4 kalikan dengan nilai K (K = 1,149604398)
3. Susun koefisien genap yang merupakan koefisien aproksimasi pada setengah jumlah baris pertama matriks lalu susun koefisien ganjil yang merupakan koefisien detail pada setengah jumlah baris kedua matrik.
4. Lakukan proses transformasi terhadap kolom dan lakukan *step* 1 – 5.
5. Susun koefisien genap yang merupakan koefisien aproksimasi pada setengah jumlah kolom pertama matriks lalu susun koefisien

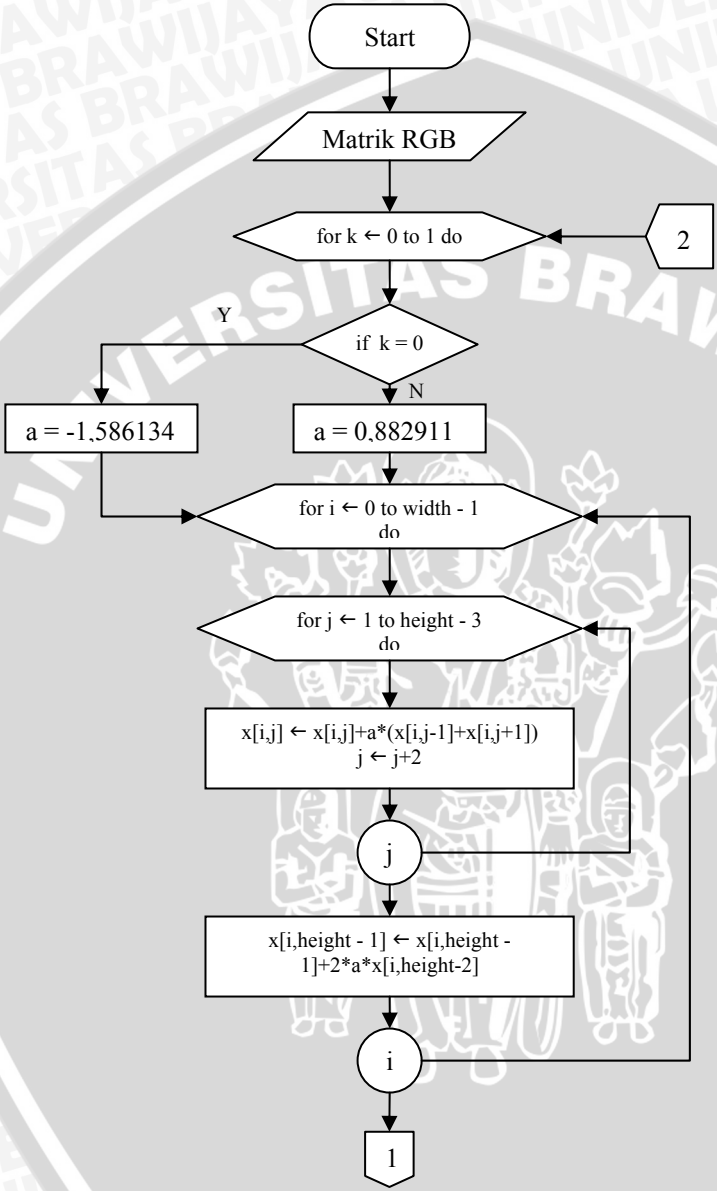
ganjil yang merupakan koefisien detail pada setengah jumlah kolom kedua matrik.

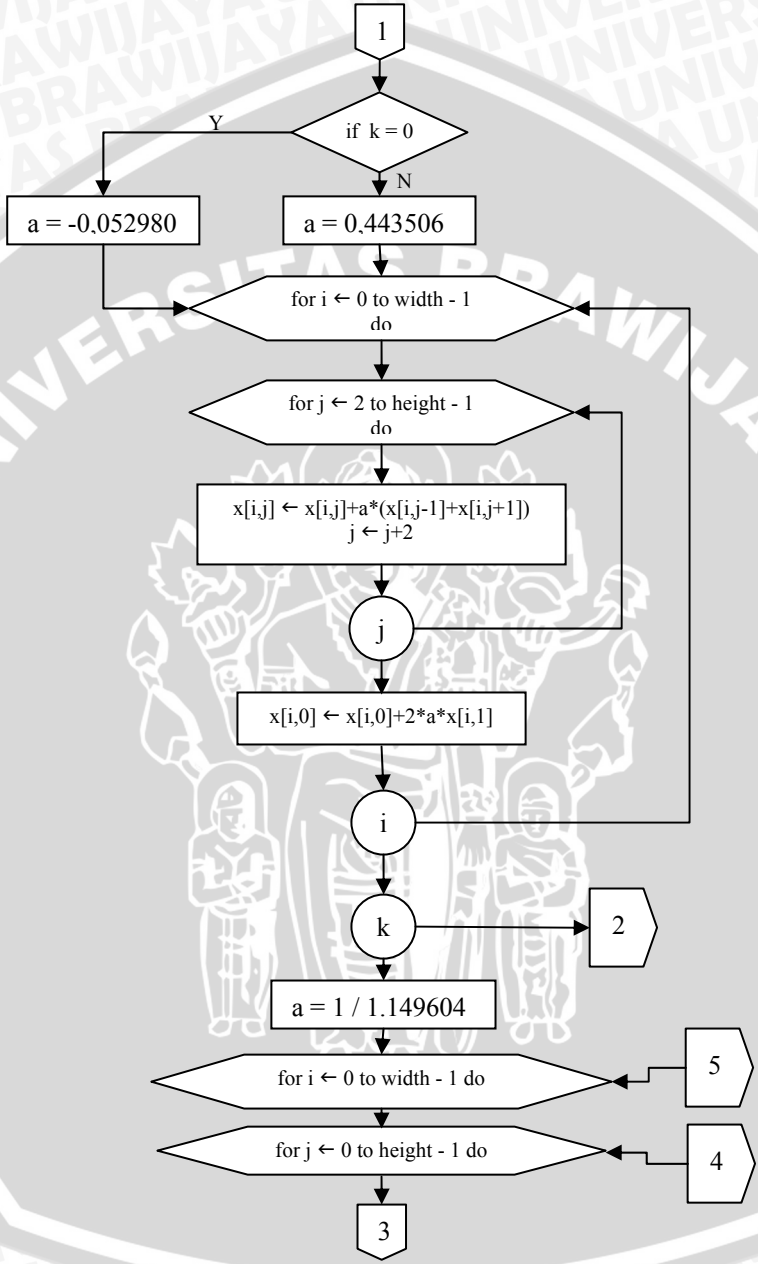
6. Membagi dua ukuran matrik untuk transformasi pada level selanjutnya (kembali ke *point* 2).

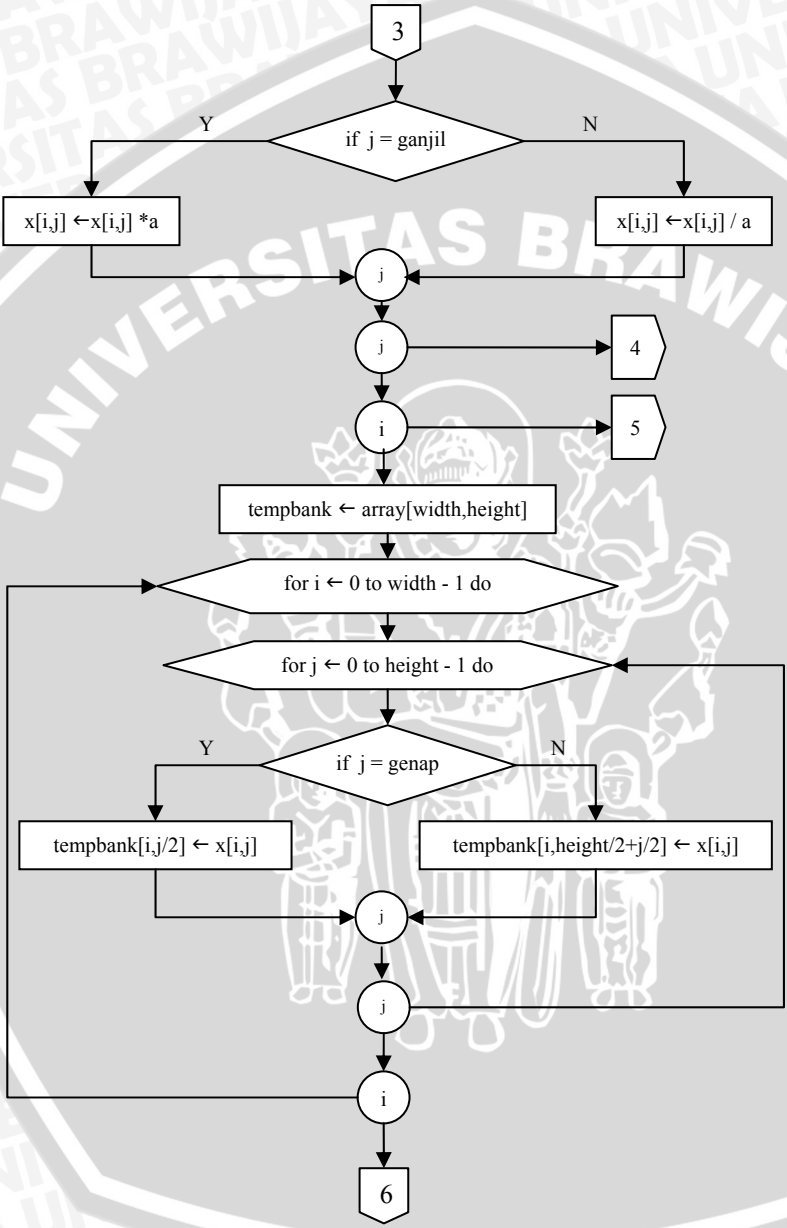
Flowchart untuk keseluruhan proses *Forward Wavelet* 9/7 *Transform* ditunjukkan pada Gambar 3.3, 3.4, dan 3.5.

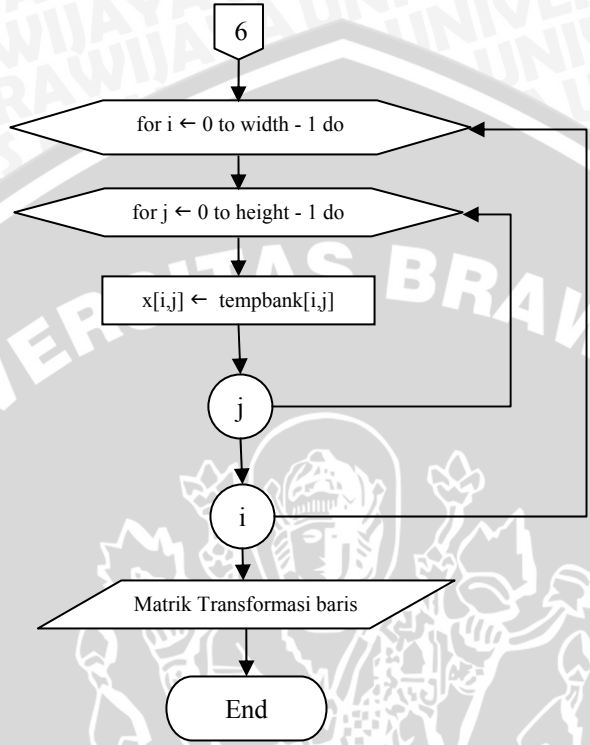


Gambar 3.3 *Flowchart* proses *Forward Wavelet* 9/7 *Transform*

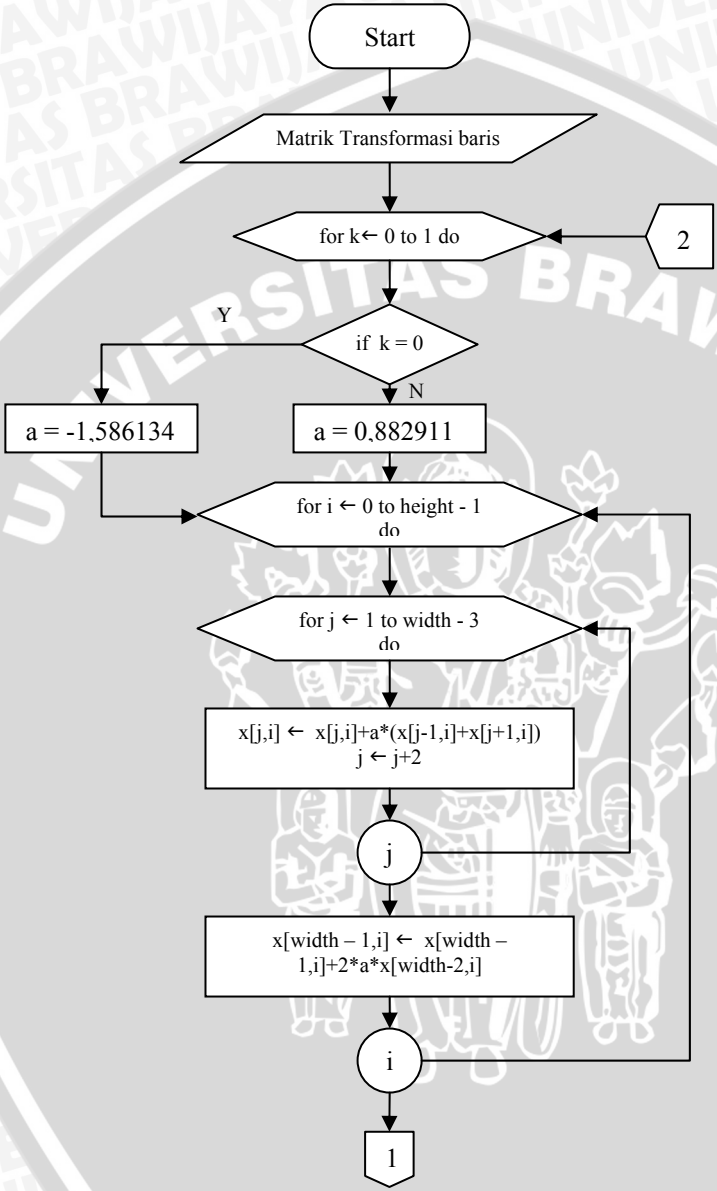

Start
Matrik RGB
for k ← 0 to 1 do
2
Y
if k = 0
N
a = -1,586134
a = 0,882911
for i ← 0 to width - 1 do
for j ← 1 to height - 3 do
x[i,j] ← x[i,j]+a*(x[i,j-1]+x[i,j+1])
j ← j+2
j
x[i,height - 1] ← x[i,height - 1]+2*a*x[i,height-2]
i
1

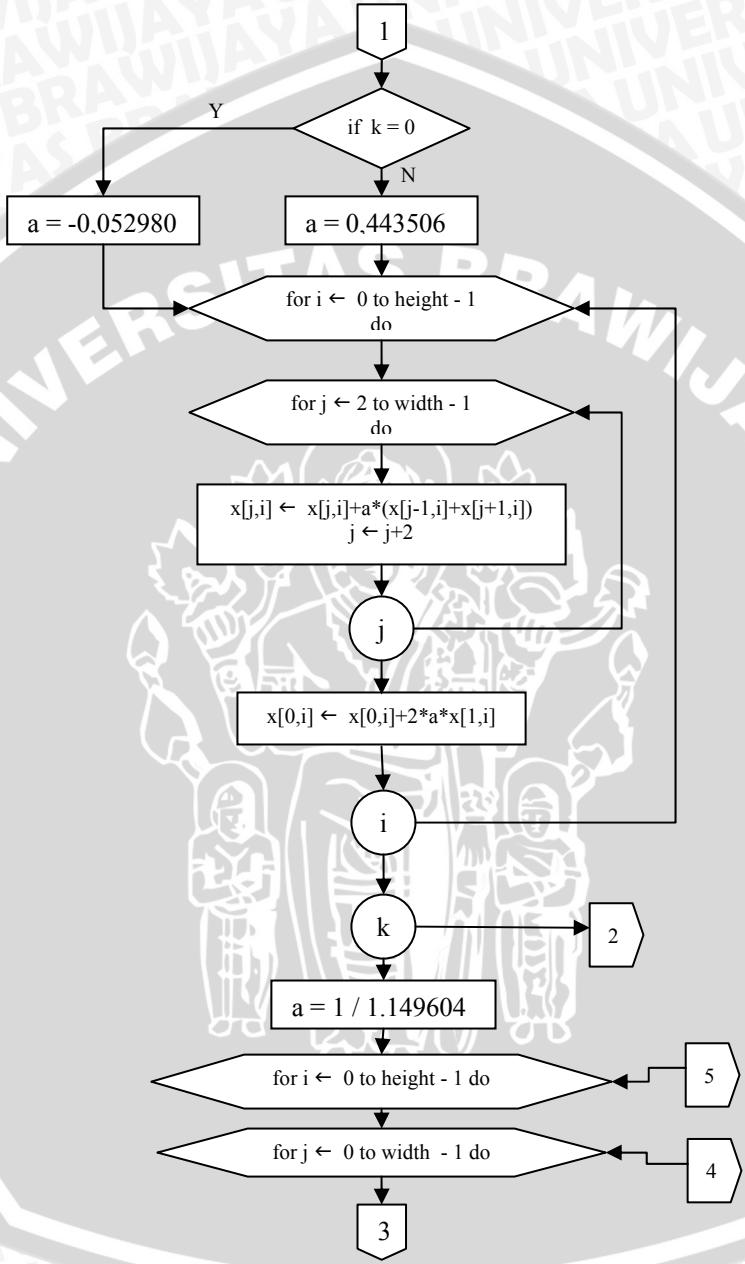

1
if k = 0
Y
N
a = -0,052980
a = 0,443506
for i ← 0 to width - 1 do
for j ← 2 to height - 1 do
x[i,j] ← x[i,j]+a*(x[i,j-1]+x[i,j+1])
j ← j+2
j
x[i,0] ← x[i,0]+2*a*x[i,1]
i
k
2
a = 1 / 1.149604
for i ← 0 to width - 1 do
5
for j ← 0 to height - 1 do
4
3

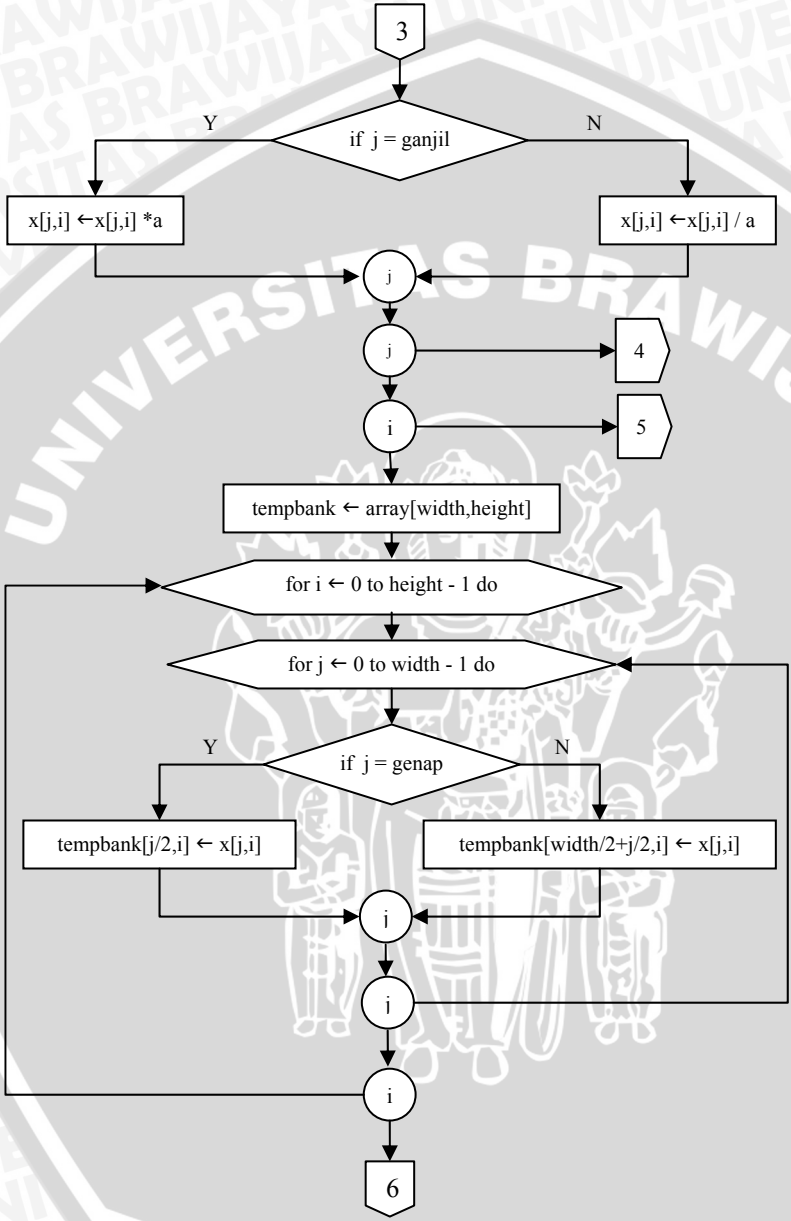

3
if j = ganjil
Y
N
x[i,j] ←x[i,j] *a
x[i,j] ←x[i,j] / a
j
j
4
i
5
tempbank ← array[width,height]
for i ← 0 to width - 1 do
for j ← 0 to height - 1 do
if j = genap
Y
N
tempbank[i,j/2] ← x[i,j]
tempbank[i,height/2+j/2] ← x[i,j]
j
j
i
6

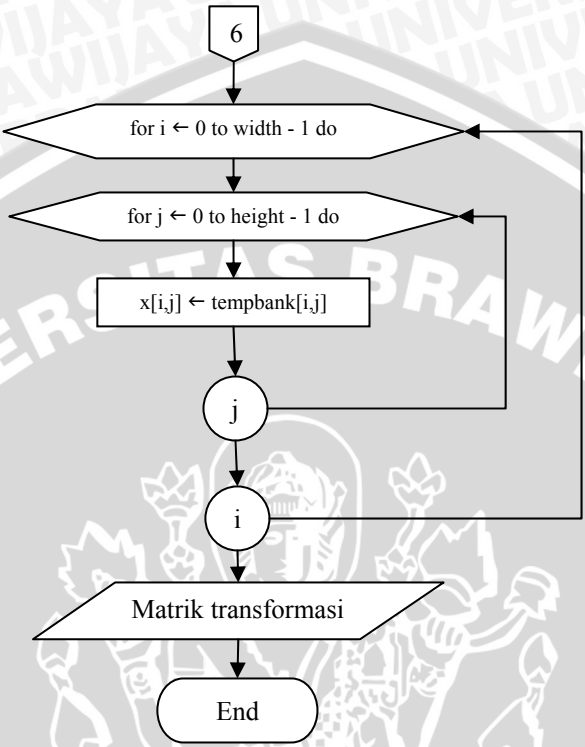


Gambar 3.4 *Flowchart* proses *Forward Transform* operasi baris


Start
Matrik Transformasi baris
for k← 0 to 1 do
2
Y
if k = 0
N
a = -1,586134
a = 0,882911
for i ← 0 to height - 1 do
for j ← 1 to width - 3 do
x[j,i] ← x[j,i]+a*(x[j-1,i]+x[j+1,i])
j ← j+2
j
x[width – 1,i] ← x[width – 1,i]+2*a*x[width-2,i]
i
1


1
if k = 0
Y
N
a = -0,052980
a = 0,443506
for i ← 0 to height - 1 do
for j ← 2 to width - 1 do
x[j,i] ← x[j,i]+a*(x[j-1,i]+x[j+1,i])
j ← j+2
j
x[0,i] ← x[0,i]+2*a*x[1,i]
i
k
2
a = 1 / 1.149604
for i ← 0 to height - 1 do
5
for j ← 0 to width - 1 do
4
3


3
Y
if j = ganjil
N
x[j,i] ←x[j,i] *a
x[j,i] ←x[j,i] / a
j
j
4
i
5
tempbank ← array[width,height]
for i ← 0 to height - 1 do
for j ← 0 to width - 1 do
Y
if j = genap
N
tempbank[j/2,i] ← x[j,i]
tempbank[width/2+j/2,i] ← x[j,i]
i
i
i
6



Gambar 3.5 *Flowchart* proses *Forward Transform* operasi kolom

3.2.2.3 Kuantisasi Vektor

Proses ini akan mengubah masing-masing *subband* matrik transformasi citra ke vektor berukuran *1 x n x n*. Vektor-vektor ini akan dicocokkan dengan *codeword* sehingga ditemukan *codeword* yang tingkat distorsinya paling kecil lalu terjadi proses pengubahan *codebook* operasional jika syarat terpenuhi. Akan dikodekan dengan indek *codeword* terpilih masing-masing vektor tersebut. Langkah-langkah yang dilakukan pada tahap ini adalah:

1. Memasukkan *input*-an berupa vektor masing-masing *subband* hasil transformasi (7 vektor *input*)
2. Memasukkan nilai *Lagrange* λ.
3. Memasukkan nilai *Threshold T*.

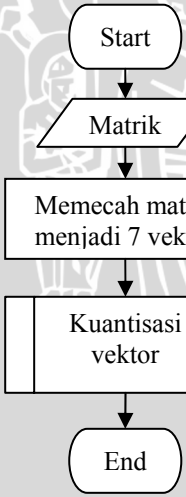
4. Vektor input akan bersesuaian dengan vektor pengkuantisasi (*codeword*) pada *codebook* yang memiliki jarak terdekat.
5. Bangkitkan sebuah vektor *partial update codeword p(T)* dengan komponen *codeword*-nya yang berubah dan catat pada vektor u(*T*) berdasarkan syarat:

$$p_j(T) = \begin{cases} s_j \; jika \; |s_j - p_j| > T \\ p_j \; selainnya \end{cases}$$

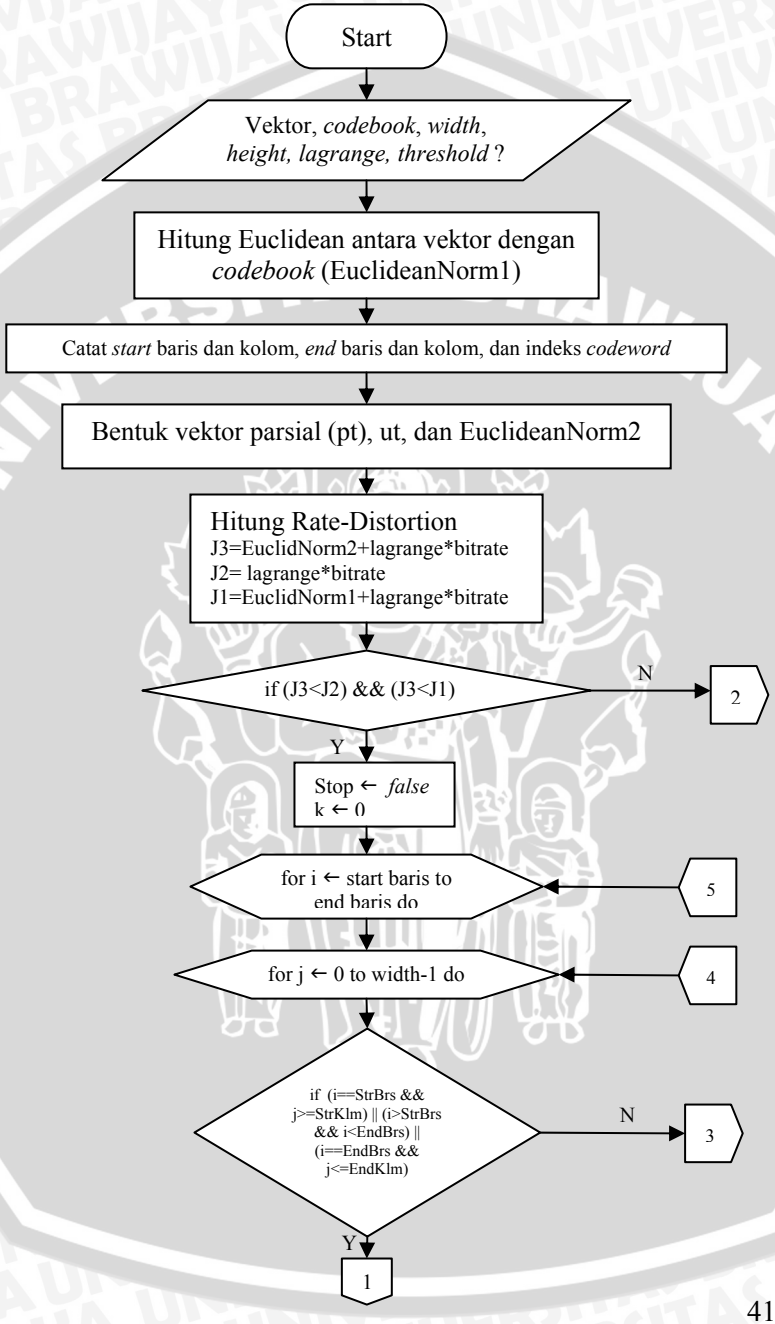
$$u_j(T) = \begin{cases} 1, \; jika \; |s_j - p_j| > T \\ 0, \; selainnya \end{cases}$$

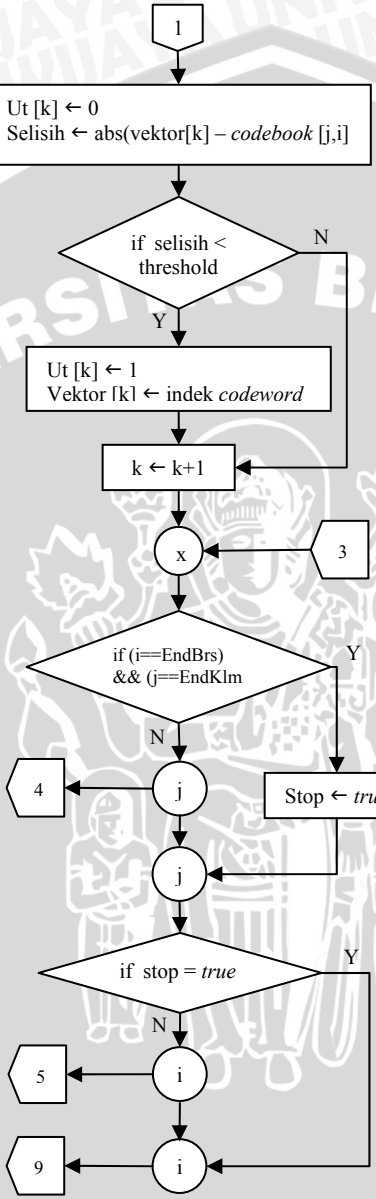
6. Hitung *Rate-Distortion* untuk menentukan *mode* mana yang tepat untuk melakukan proses kuantisasi.
7. Jika mode PCU, maka komponen vektor input yang selisihnya dengan komponen *codeword* kurang dari *threshold* dikodekan dengan indek *codeword* terpilih.
8. Jika mode FCU, maka seluruh komponen vektor input tidak dikodekan dengan indek *codeword* terpilih.
9. Jika mode *No Update*, maka seluruh komponen vektor input dikodekan dengan indek *codeword* terpilih.

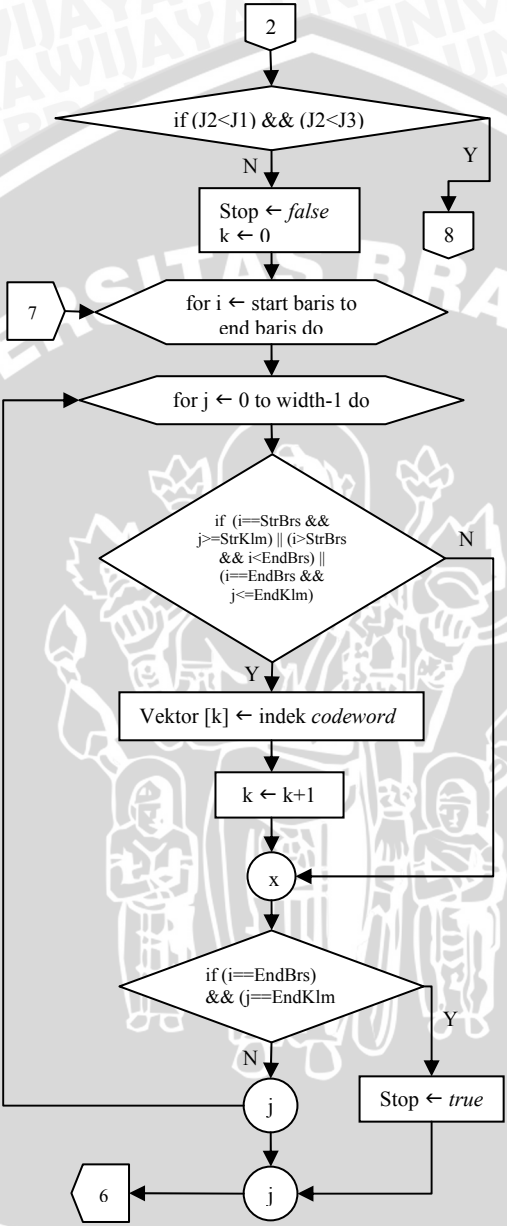
Flowchart proses kuantisasi ditunjukkan pada Gambar 3.6 dan 3.7

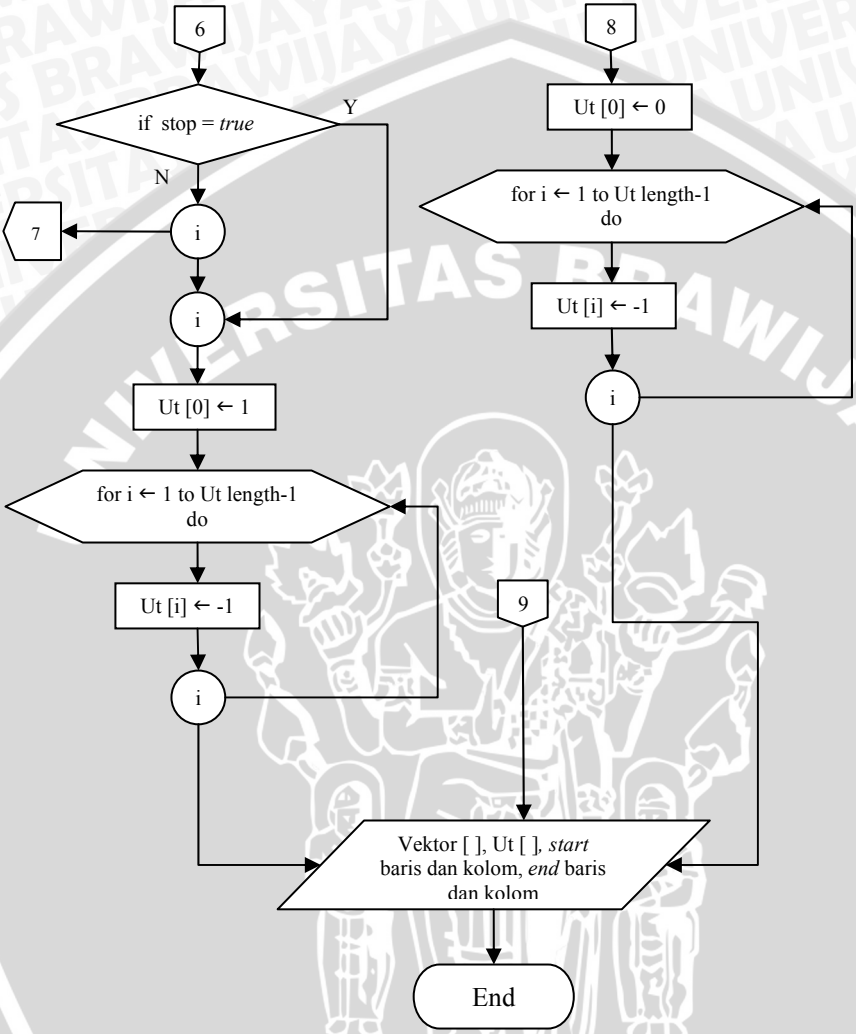


Gambar 3.6 *Flowchart* proses *predefined* Kuantisasi


Start
Vektor, *codebook*, *width*, *height, lagrange, threshold* ?
Hitung Euclidean antara vektor dengan *codebook* (EuclideanNorm1)
Catat *start* baris dan kolom, *end* baris dan kolom, dan indeks *codeword*
Bentuk vektor parsial (pt), ut, dan EuclideanNorm2
Hitung Rate-Distortion
J3=EuclidNorm2+lagrange*bitrate
J2= lagrange*bitrate
J1=EuclidNorm1+lagrange*bitrate
if (J3<J2) && (J3<J1)
N
2
Y
Stop ← *false*
k ← 0
for i ← start baris to end baris do
5
for j ← 0 to width-1 do
4
if (i==StrBrs && j>=StrKlm) || (i>StrBrs && i<EndBrs) || (i==EndBrs && j<=EndKlm)
N
3
Y
1


1
Ut [k] ← 0
Selisih ← abs(vektor[k] – *codebook* [j,i]
if selisih < threshold
N
Y
Ut [k] ← 1
Vektor [k] ← indek *codeword*
k ← k+1
x
3
if (i==EndBrs) && (j==EndKlm
Y
N
4
j
Stop ← *true*
j
if stop = *true*
Y
N
5
i
9
i


2
if (J2<J1) && (J2<J3)
N
Y
Stop ← *false*
k ← 0
8
7
for i ← start baris to
end baris do
for j ← 0 to width-1 do
if (i==StrBrs &&
j>=StrKlm) || (i>StrBrs
&& i<EndBrs) ||
(i==EndBrs &&
j<=EndKlm)
N
Y
Vektor [k] ← indek *codeword*
k ← k+1
x
if (i==EndBrs)
&& (j==EndKlm
Y
N
j
Stop ← *true*
6
j



Gambar 3.7 *Flowchart* proses Kuantisasi

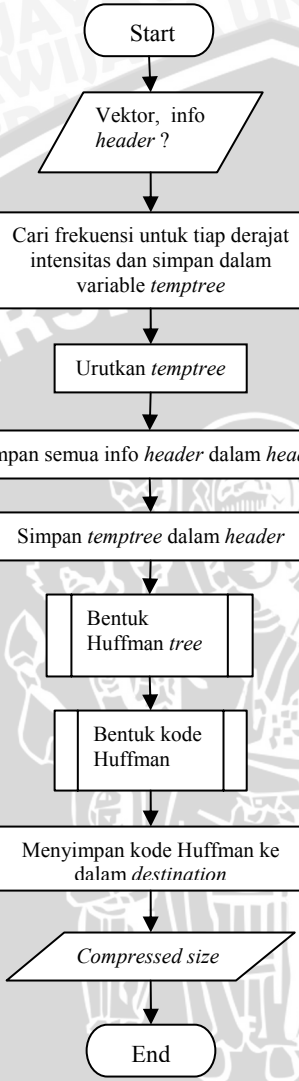
3.2.2.4 Huffman *Encoder*

Pada tahap ini, 7 vektor yang telah dikuantisasi (dari masing-masing elemen RGB) akan digabungkan dalam 1 *array* dan dikompresi menggunakan metode Huffman. Sebelum menuliskan data kompresi akan dibentuk *header file* mengenai berbagai informasi yang nantinya akan digunakan dalam proses dekompresi. Sistem akan membentuk *tree* untuk mendapatkan kode Huffman. Setelah semua koefisien *array* dikodekan, maka terbentuk *file* kompresi hasil proses pengkodean.

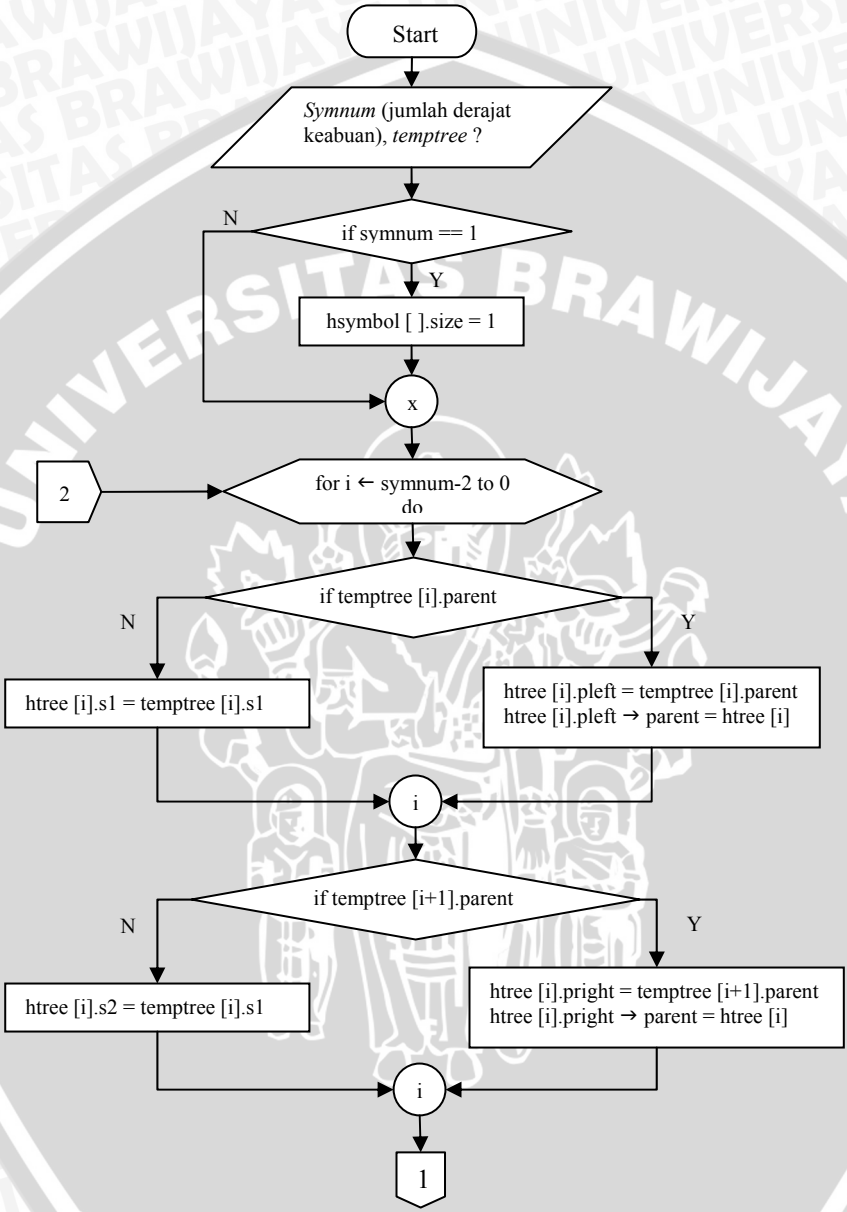
Langkah-langkah yang dilakukan pada tahap ini adalah:

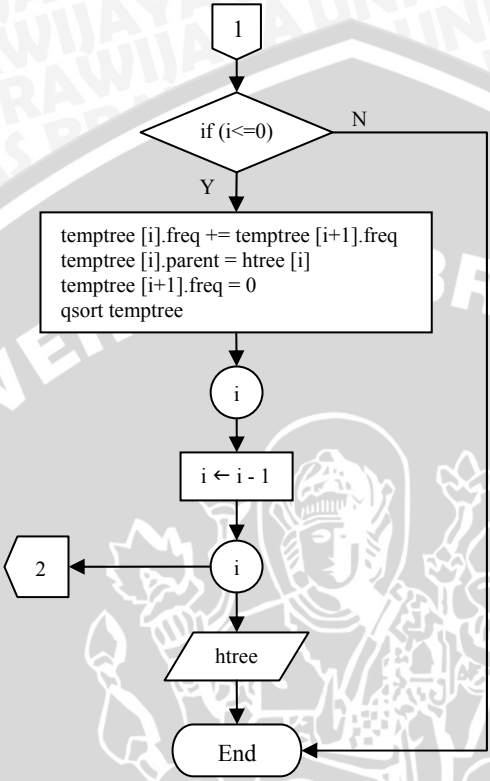
1. Memasukkan *input*-an berupa *array* 7 vektor kuantisasi dan info *header*.
2. Lakukan *looping* untuk inisialisasi nilai derajat intensitas dan mendapatkan frekuensi kemunculannya. Simpan dalam variabel *temptree*.
3. Urutkan menaik *temptree*.
4. Masukan semua informasi yang dibutuhkan untuk proses dekompresi pada *header file*.
5. Simpan *temptree* dalam *header*.
6. Bentuk Huffman *tree*.
7. Bentuk kode Huffman dari Huffman *tree*.
8. Tulis kode Huffman ke *destination*.

Flowchart untuk keseluruhan proses Huffman Encoder ditunjukkan pada Gambar 3.8, 3.9, dan 3.10.

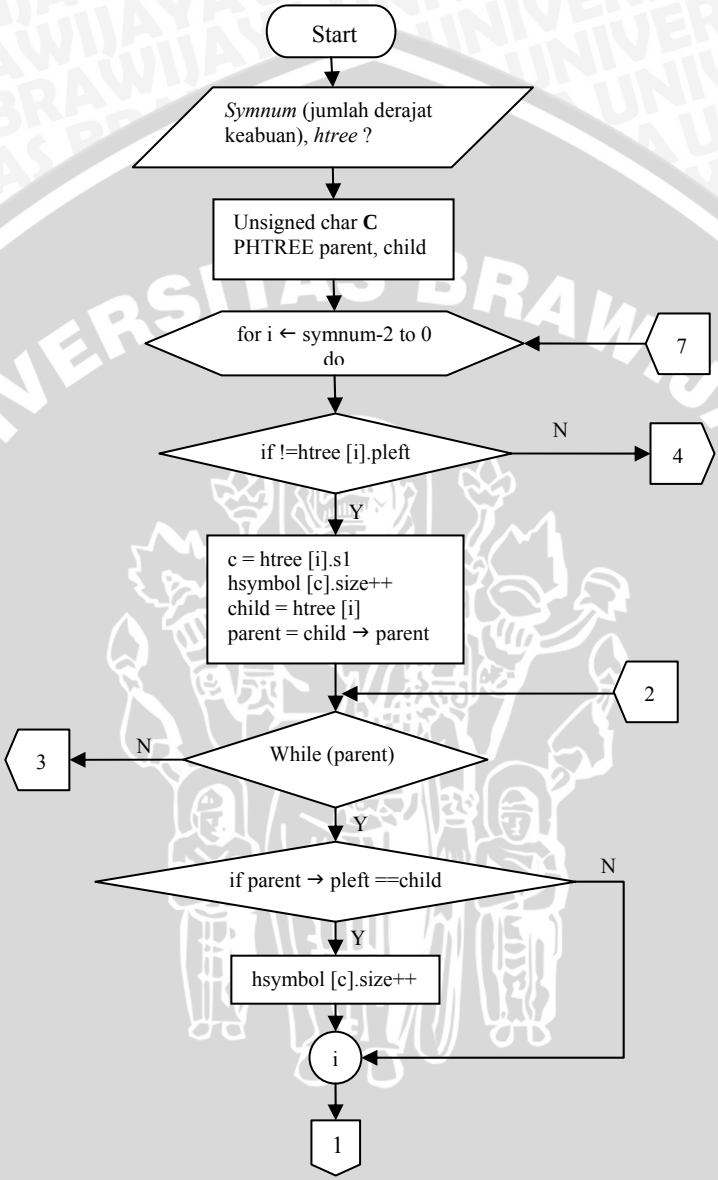


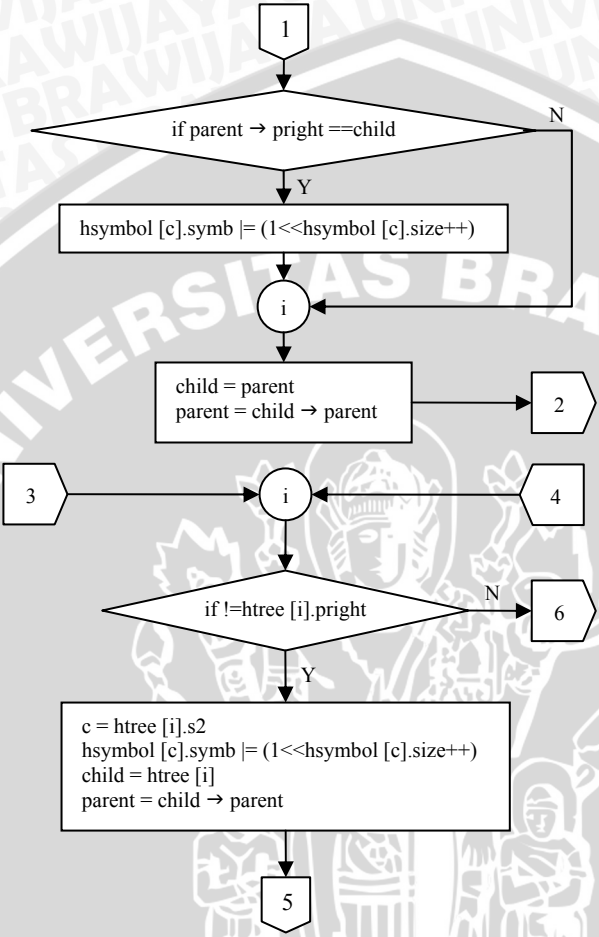
Gambar 3.8 *Flowchart* proses Huffman *Encoder*

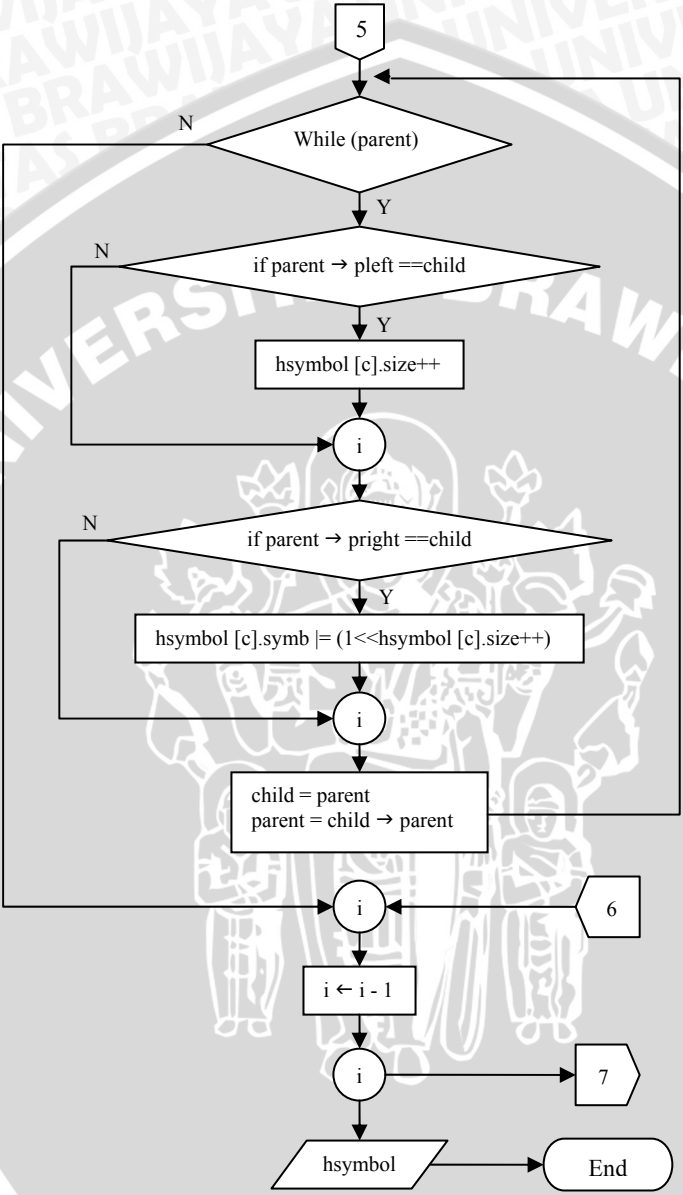

Start
Symnum (jumlah derajat keabuan), *temptree* ?
if symnum == 1
N
Y
hsymbol [ ].size = 1
x
2
for i ← symnum-2 to 0 do
if temptree [i].parent
N
Y
htree [i].s1 = temptree [i].s1
htree [i].pleft = temptree [i].parent
htree [i].pleft → parent = htree [i]
i
if temptree [i+1].parent
N
Y
htree [i].s2 = temptree [i].s1
htree [i].pright = temptree [i+1].parent
htree [i].pright → parent = htree [i]
i
1



Gambar 3.9 *Flowchart* proses Bentuk Huffman *tree*


Start
Symnum (jumlah derajat keabuan), *htree* ?
Unsigned char **C**
PHTREE parent, child
for i ← symnum-2 to 0 do
7
if !=htree [i].pleft
N
4
Y
c = htree [i].s1
hsymbol [c].size++
child = htree [i]
parent = child → parent
2
While (parent)
N
3
Y
if parent → pleft ==child
N
Y
hsymbol [c].size++
i
1


1
if parent → pright ==child
N
Y
hsymbol [c].symb |= (1<<hsymbol [c].size++)
i
child = parent
parent = child → parent
2
3
i
4
if !=htree [i].pright
N
6
Y
c = htree [i].s2
hsymbol [c].symb |= (1<<hsymbol [c].size++)
child = htree [i]
parent = child → parent
5



Gambar 3.10 *Flowchart* proses Bentuk Kode Huffman

3.2.3 Perancangan Proses Dekompresi

3.2.3.1 Huffman *Decoder*

Pada tahap ini sistem akan melakukan rekonstruksi / penyusunan ulang array 7 vektor kuantisasi yang sebelumnya telah dikodekan oleh kode Huffman. Pertama sistem akan membaca berkas kompresi untuk mengetahui informasi yang terdapat pada *header file* yang dibutuhkan untuk proses dekompresi. Sistem akan mendapatkan informasi *temptree* yang digunakan untuk membentuk Huffman *tree*. Setelah terbentuk Huffman *tree* sistem akan mendekodekan kode Huffman menjadi nilai derajat intensitas kembali, sehingga terbentuk *array* dengan komponen 7 vektor kuantisasi kembali.

Langkah-langkah yang dilakukan pada tahap ini adalah:

1. Membaca *header file* untuk mengetahui semua informasi yang dibutuhkan untuk proses dekompresi.
2. Dari informasi yang di dapat, bentuk *Huffman tree* yang digunakan untuk membaca kode huffman menjadi nilai derajat intensitas kembali.
3. Mengisikan hasil pendekodean pada *destination*.

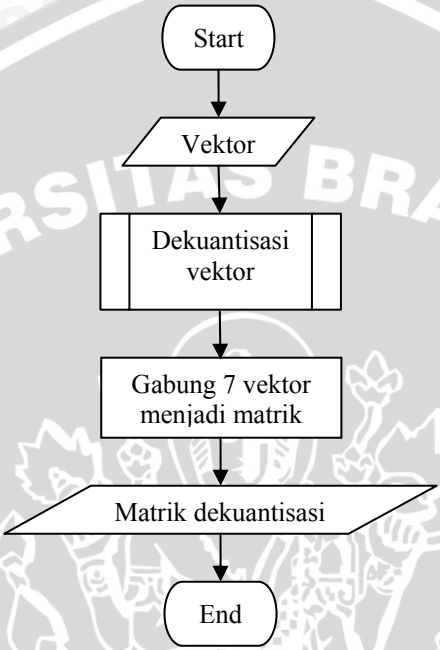
3.2.3.2 Dekuantisasi Vektor

Proses ini akan mencari *codeword* di *codebook* berdasarkan informasi lokasi *codeword* terpilih (*star* baris dan *star* kolom serta *end* baris dan *end* kolom) lalu nilai indek yang dimiliki oleh komponen-komponen yang ada pada array 7 vektor kuantisasi ditukar dengan nilai *codeword*. Setelah itu sistem akan merubah vektor tersebut dari bentuk *1 x n x n* ke bentuk *subband n x n*. Akan kembali terbentuk *subband-subband* matrik citra untuk masing-masing elemen RGB yang akan di *invers*.

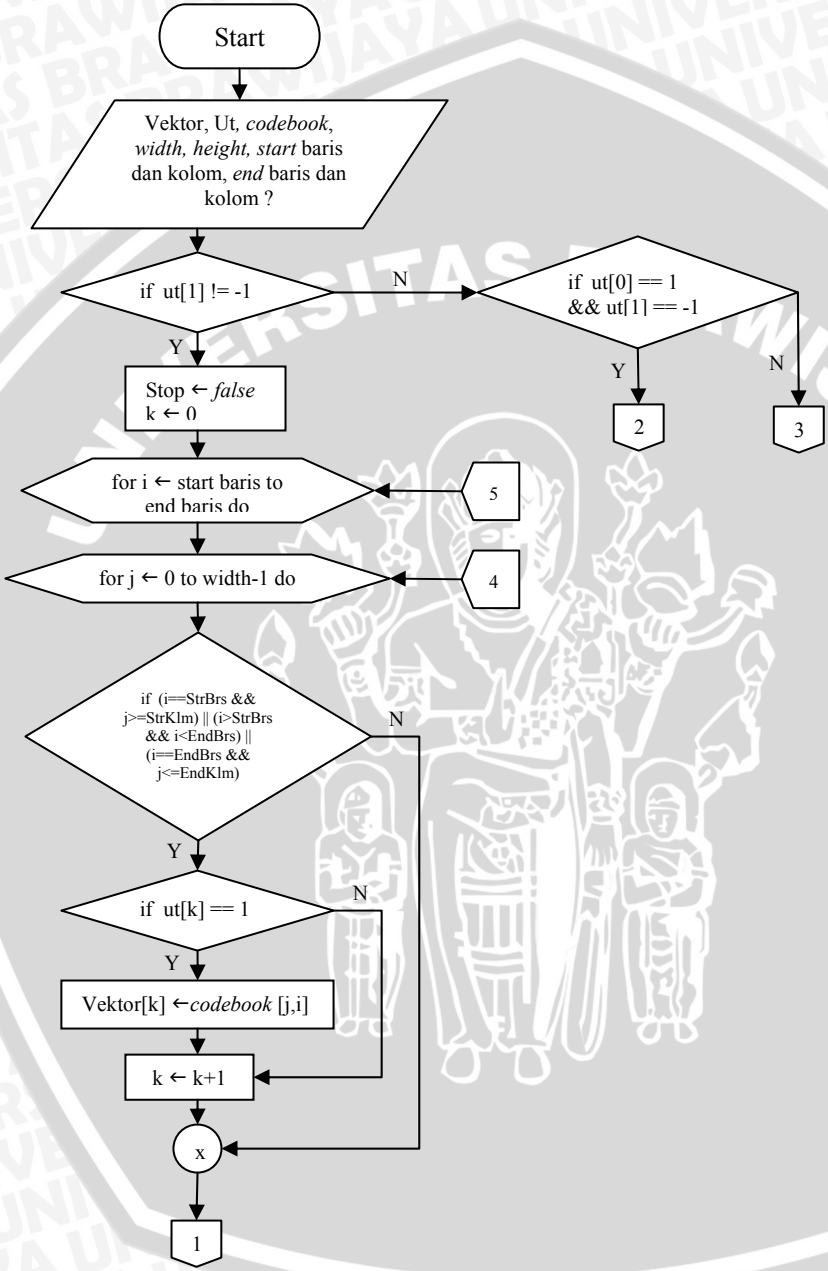
Langkah-langkah yang dilakukan pada tahap ini adalah:

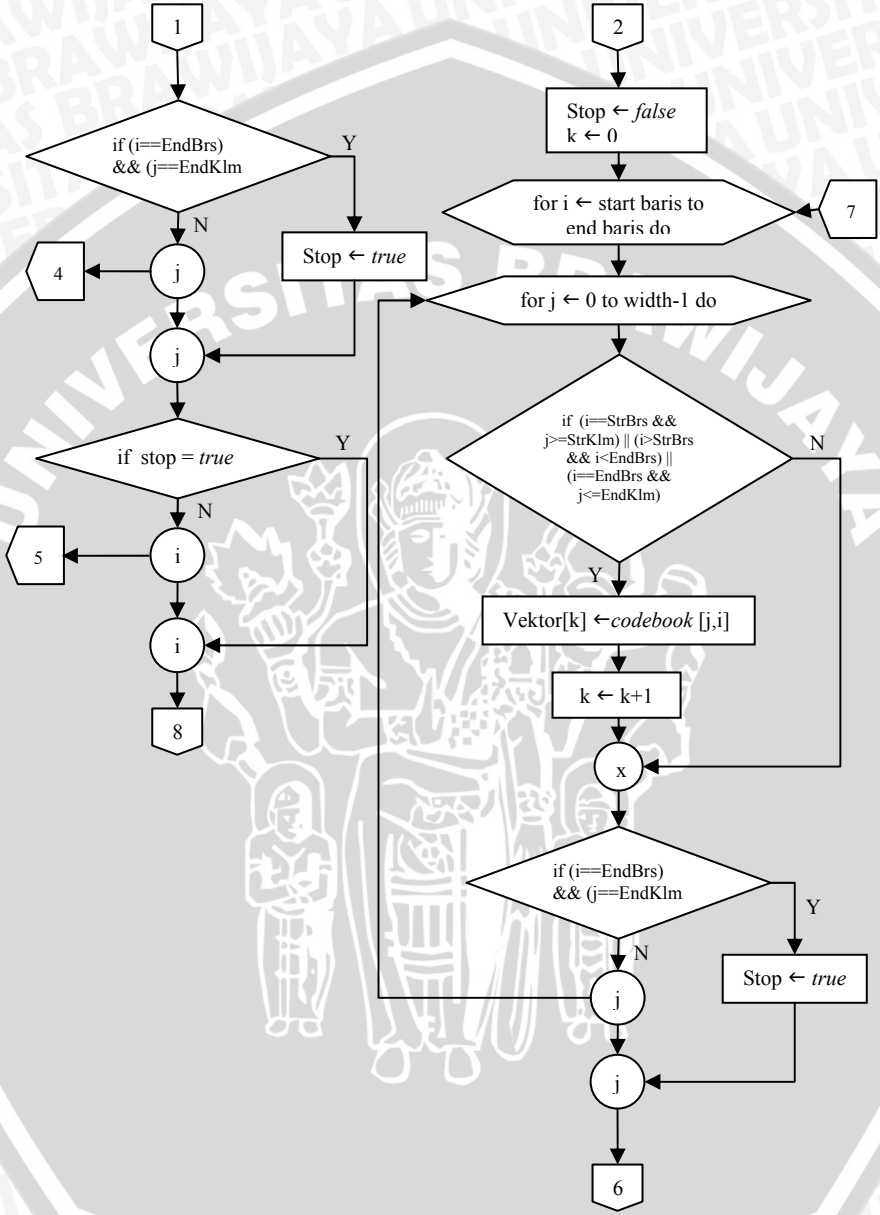
1. Cari *codeword* yang sesuai berdasarkan informasi lokasi *codeword* terpilih.
2. Ganti komponen Vektor dengan komponen *codeword* berdasarkan informasi vektor Ut.
3. Merubah 7 vektor kuantisasi (7 *codeword*) masing-masing ke bentuk *subband n x n*.
4. Gabungkan *subband n x n* sehingga terbentuk matrik yang berisi koefisien *subband* matrik citra.

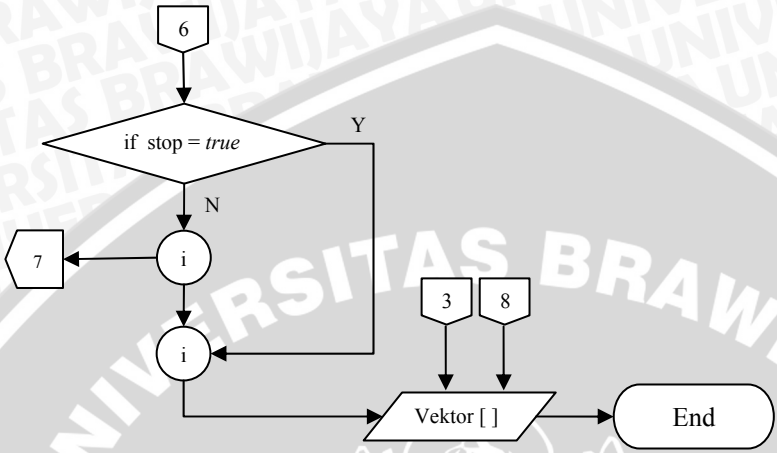
Flowchart proses dekuantisasi ditunjukkan pada Gambar 3.11 dan 3.12



Gambar 3.11 *Flowchart* proses *predifened* Dekuantisasi


Start
Vektor, Ut, *codebook*, *width, height, start* baris dan kolom, *end* baris dan kolom ?
if ut[1] != -1
N
if ut[0] == 1 && ut[1] == -1
Y
N
Y
2
3
Stop ← *false*
k ← 0
for i ← start baris to end baris do
5
for j ← 0 to width-1 do
4
if (i==StrBrs && j>=StrKlm) || (i>StrBrs && i<EndBrs) || (i==EndBrs && j<=EndKlm)
N
Y
if ut[k] == 1
N
Y
Vektor[k] ←*codebook* [j,i]
k ← k+1
x
1


1
if (i==EndBrs)
&& (j==EndKlm
Y
N
Stop ← true
4
j
j
if stop = true
Y
N
5
i
i
8
2
Stop ← false
k ← 0
for i ← start baris to
end baris do
7
for j ← 0 to width-1 do
if (i==StrBrs &&
j>=StrKlm) || (i>StrBrs
&& i<EndBrs) ||
(i==EndBrs &&
j<=EndKlm)
N
Y
Vektor[k] ←codebook [j,i]
k ← k+1
x
if (i==EndBrs)
&& (j==EndKlm
Y
N
Stop ← true
j
j
6



Gambar 3.12 *Flowchart* proses Dekuantisasi

3.2.3.3 *Lifting-based Invers Wavelet* 9/7 *Transform*

Proses ini merupakan kebalikan dari proses *Lifting-based forward* Transformasi *Wavelet* 9/7, akan dilakukan transformasi matrik dari *domain* frekuensi ke *domain spasial*. Langkah-langkah yang dilakukan pada tahap ini adalah:

1. Memasukkan matrik elemen_R, elemen_G, dan elemen_B.
2. Mendefinisikan ukuran matrik pada level terdalam.
3. Koefisien-koefisien genap yang merupakan koefisien aproksimasi yang terletak pada setengah jumlah kolom pertama matriks dan koefisien-koefisien ganjil yang merupakan koefisien detail yang terletak pada setengah jumlah kolom kedua matrik diubah susunannya menjadi 1 koefisien genap pada kolom pertama lalu 1 koefisien ganjil pada kolom kedua dan begitu seterusnya saling bergantian untuk tiap-tiap koefisien genap dan ganjil.
4. Untuk masing elemen lakukan proses transformasi terhadap kolom.
   *Step*1. Untuk koefisien ganjil kalikan dengan nilai K dan untuk koefisien genap kalikan dengan nilai 1/K (K = 1,149604398)
   *Step*2. Untuk koefisien genap hasil dari *step*1: jumlahkan koefisien ganjil hasil dari *step*1 sebelum dan sesudah

koefisien genap tersebut lalu kalikan dengan nilai d (d = -0,4435068522) kemudian tambah dengan koefisien genap tersebut.

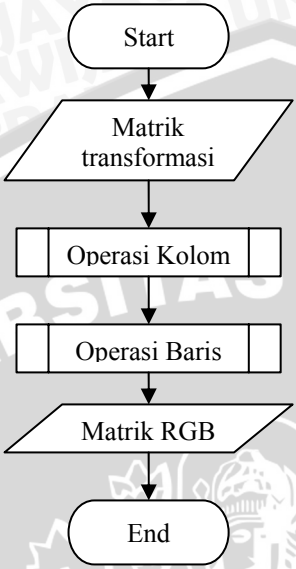
*Step*3. Untuk koefisien ganjil hasil dari step1: jumlahkan koefisien genap hasil dari step2 sebelum dan sesudah koefisien ganjil tersebut lalu kalikan dengan nilai c (c = -0,8829110762) kemudian tambah dengan koefisien ganjil tersebut.

*Step*4. Untuk koefisien genap hasil dari *step* 2: jumlahkan koefisien ganjil hasil dari *step* 3 sebelum dan sesudah koefisien genap tersebut lalu kalikan dengan nilai b (b = 0,05298011854) kemudian tambah dengan koefisien genap tersebut.

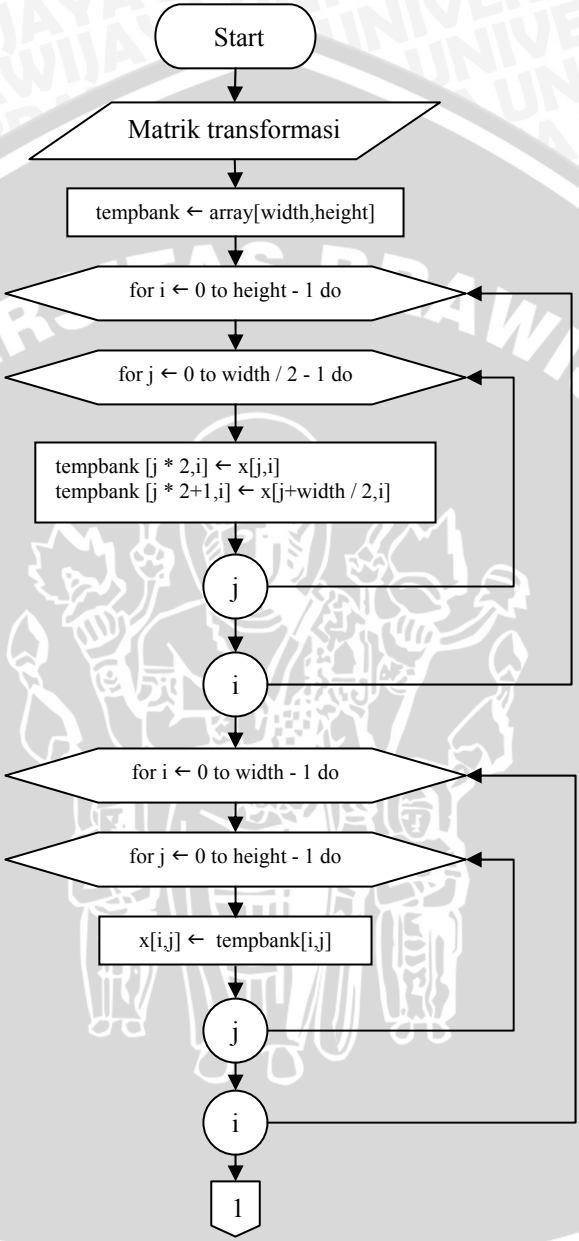
*Step*5. Untuk koefisien ganjil hasil dari *step* 3: jumlahkan koefisien genap hasil dari *step* 4 sebelum dan sesudah koefisien genap tersebut lalu kalikan dengan nilai a (a = 1,586134342) kemudian tambah dengan koefisien ganjil tersebut.

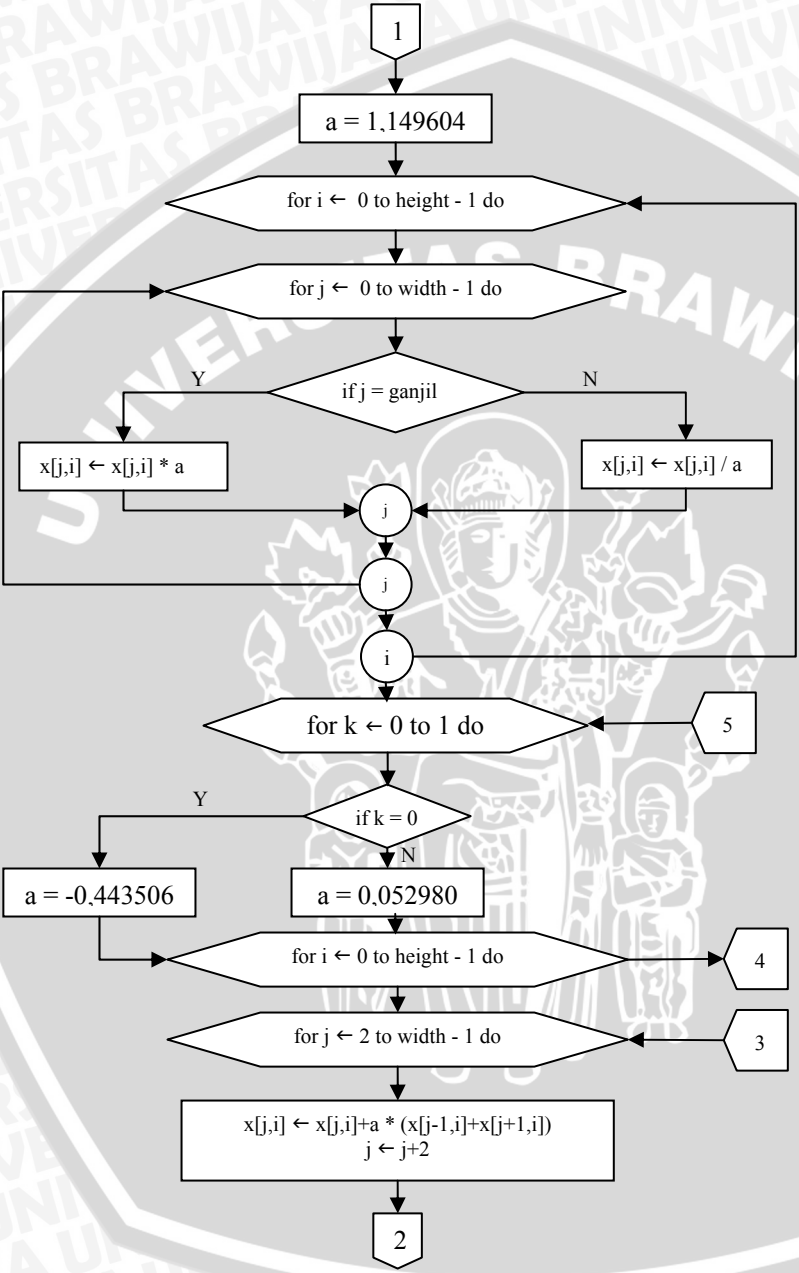
5. Koefisien-koefisien genap yang merupakan koefisien aproksimasi yang terletak pada setengah jumlah baris pertama matriks dan koefisien-koefisien ganjil yang merupakan koefisien detail yang terletak pada setengah jumlah baris kedua matrik diubah susunannya menjadi 1 koefisien genap pada baris pertama lalu 1 koefisien ganjil pada baris kedua dan begitu seterusnya saling bergantian untuk tiap-tiap koefisien genap dan ganjil.
6. Lakukan proses transformasi terhadap baris dan lakukan *step*1 – 5.
7. Mengalikan dua ukuran matrik untuk transformasi pada level selanjutnya (kembali ke *point* 2).

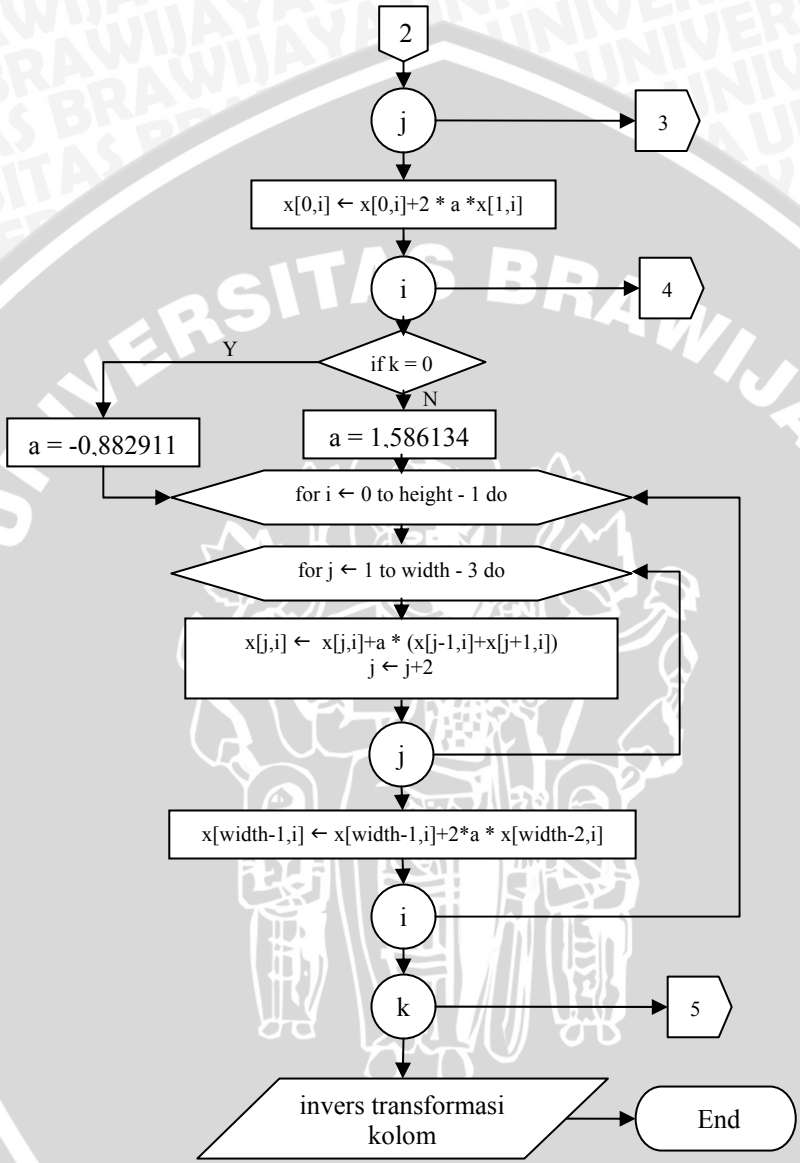
Flowchart untuk keseluruhan proses *Invers Wavelet* 9/7 *Transform* ditunjukkan pada Gambar 3.13, 3.14, dan 3.15.



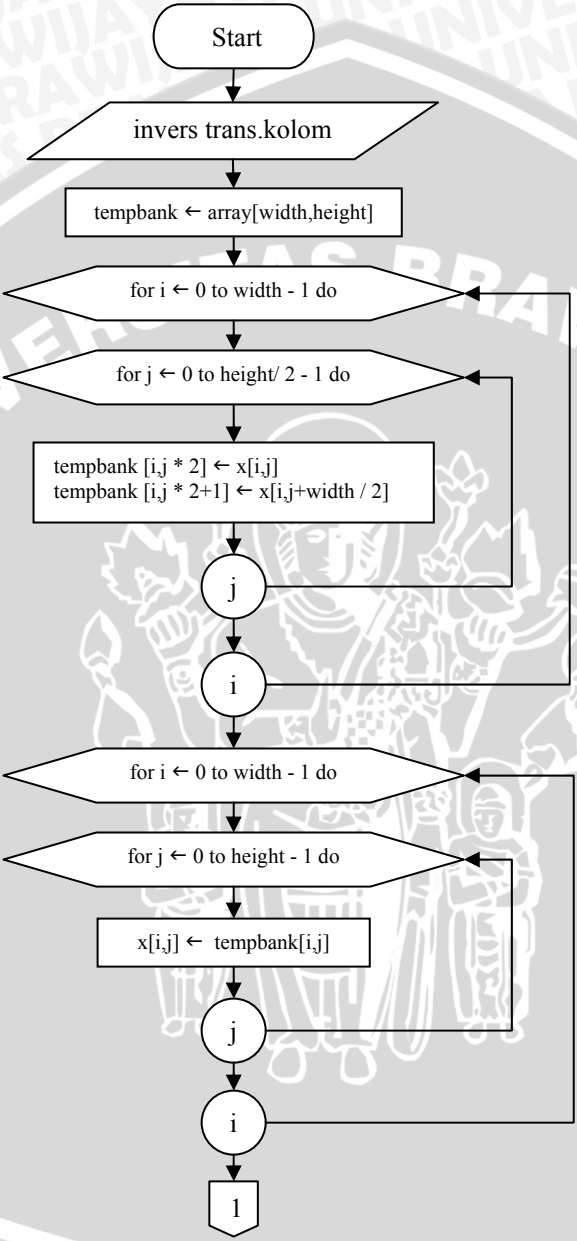
Gambar 3.13 *Flowchart* proses *Invers Wavelet* 9/7 *Transform*

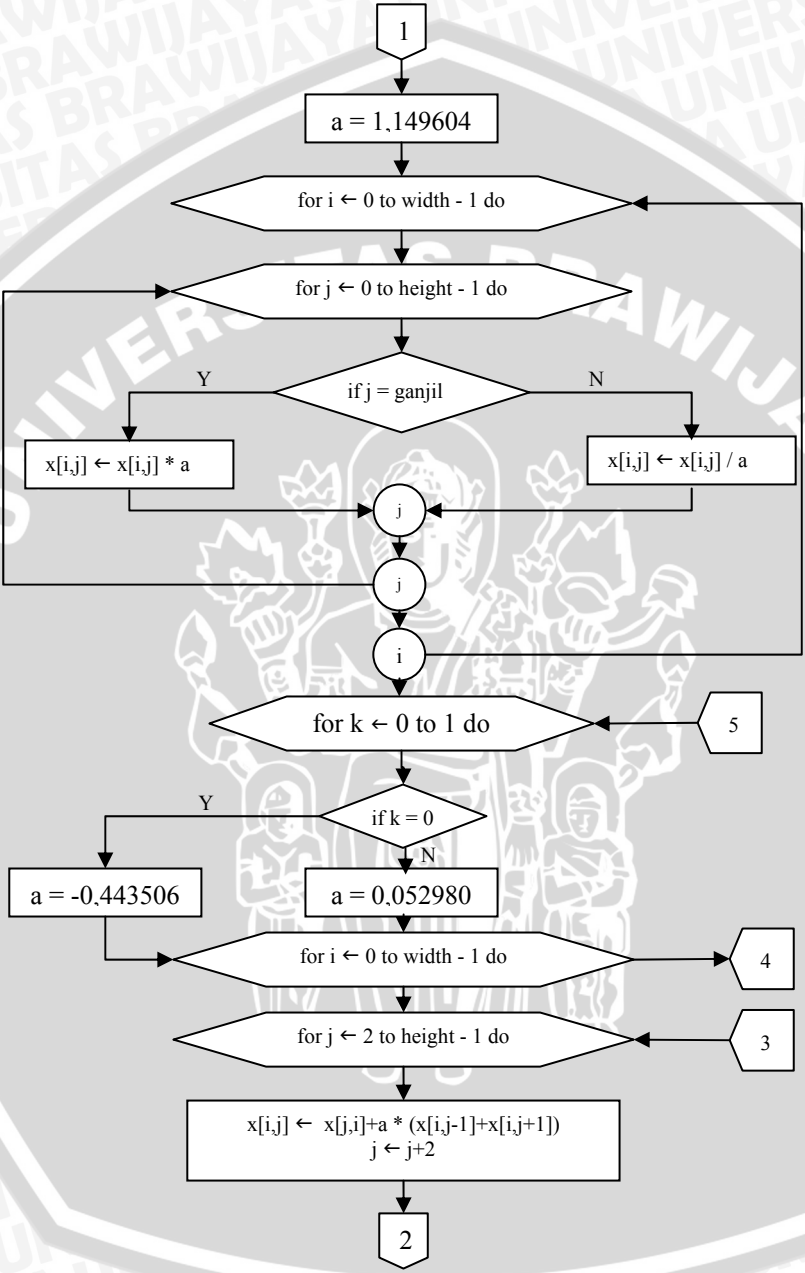

Start
Matrik transformasi
tempbank ← array[width,height]
for i ← 0 to height - 1 do
for j ← 0 to width / 2 - 1 do
tempbank [j * 2,i] ← x[j,i]
tempbank [j * 2+1,i] ← x[j+width / 2,i]
j
i
for i ← 0 to width - 1 do
for j ← 0 to height - 1 do
x[i,j] ← tempbank[i,j]
j
i
1

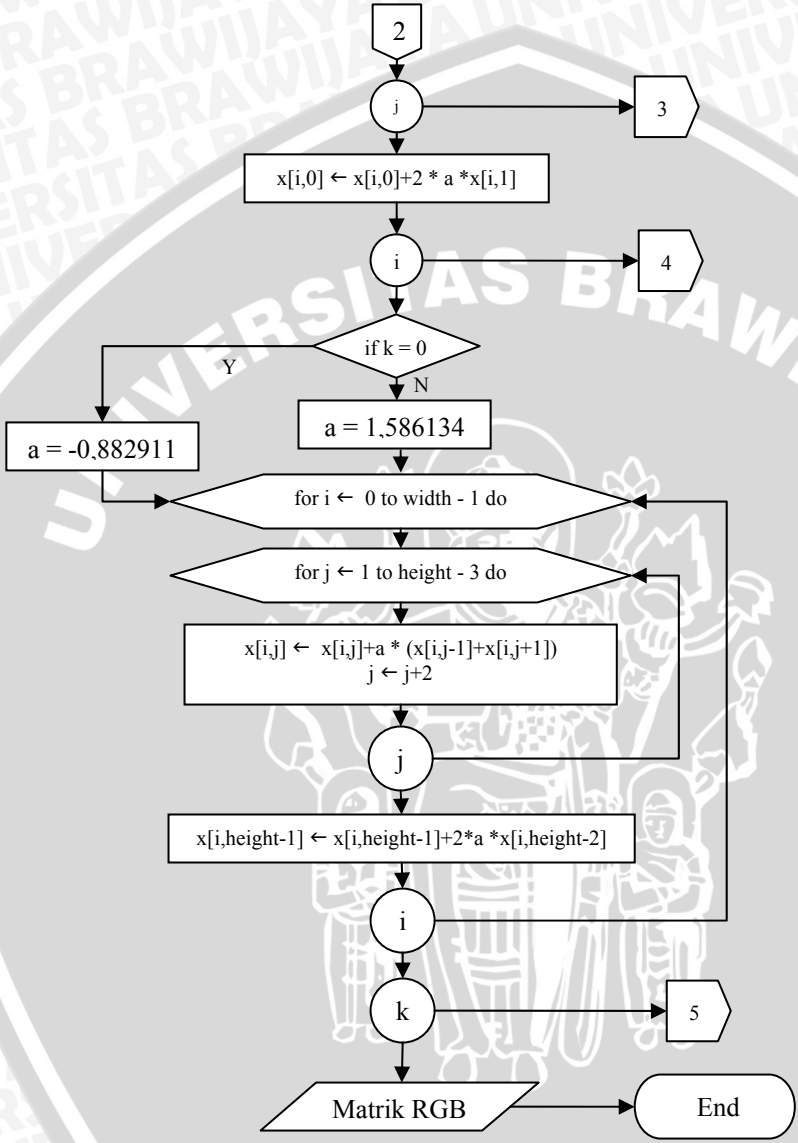

1
a = 1,149604
for i ← 0 to height - 1 do
for j ← 0 to width - 1 do
Y
if j = ganjil
N
x[j,i] ← x[j,i] * a
x[j,i] ← x[j,i] / a
j
j
i
for k ← 0 to 1 do
5
Y
if k = 0
N
a = -0,443506
a = 0,052980
for i ← 0 to height - 1 do
4
for j ← 2 to width - 1 do
3
x[j,i] ← x[j,i]+a * (x[j-1,i]+x[j+1,i])
j ← j+2
2



Gambar 3.14 *Flowchart* proses *Invers Transform* operasi kolom


Start
invers trans.kolom
tempbank ← array[width,height]
for i ← 0 to width - 1 do
for j ← 0 to height/ 2 - 1 do
tempbank [i,j * 2] ← x[i,j]
tempbank [i,j * 2+1] ← x[i,j+width / 2]
j
i
for i ← 0 to width - 1 do
for j ← 0 to height - 1 do
x[i,j] ← tempbank[i,j]
j
i
1


1
a = 1,149604
for i ← 0 to width - 1 do
for j ← 0 to height - 1 do
if j = ganjil
Y
N
x[i,j] ← x[i,j] * a
x[i,j] ← x[i,j] / a
j
j
i
for k ← 0 to 1 do
5
if k = 0
Y
N
a = -0,443506
a = 0,052980
for i ← 0 to width - 1 do
4
for j ← 2 to height - 1 do
3
x[i,j] ← x[j,i]+a * (x[i,j-1]+x[i,j+1])
j ← j+2
2



Gambar 3.15 *Flowchart* proses *Invers Transform* operasi baris

3.2.3.4 *Deblocking* Citra

Pada tahap ini akan dilakukan proses penggabungan tiga elemen RGB hasil transformasi balik sehingga citra rekonstruksi akan tampak mendekati citra asli.

3.3 Perancangan Pengujian

Pengujian yang akan dilakukan pada perangkat lunak ini adalah pengujian rasio dan pengujian tingkat kesalahan (*error rate)* terhadap jenis citra uji.

Citra uji yang digunakan pada penelitian ini terdiri dari tiga kriteria berdasarkan jenis detail citra, yang meliputi *high detail*, *medium detail*, dan *low detail*. Suatu citra dikatakan *low detail* bila keragaman warna sedikit. Citra dikatakan *high detail* bila keragaman warna banyak. Bila tampilan citra berada di antara *high detail* dan *low detail*, maka citra termasuk dalam kriteria citra *medium detail.* Untuk mendapatkan kriteria jenis detail citra dari beberapa citra uji dilakukan penghitungan nilai Standar Deviasi untuk masing-masing citra uji berdasarkan persamaan 2.13. Hasil penghitungan Standar Deviasi untuk masing-masing citra uji dapat dilihat pada lampiran III.

Pengujian dilakukan sebanyak tiga kali dengan kondisi nilai *lagrange* λ yang divariasikan ($\lambda = 50$, $\lambda = 100$, $\lambda = 150$).

3.3.1 Citra Uji

Citra yang diuji adalah citra yang memiliki karakteristik sebagai berikut:

1. 18 citra dengan ekstensi bitmap (.bmp).
2. Citra warna 24 bit.
3. Citra berukuran 256 x 256.
4. Citra uji didapatkan dari penelitian Rahadiany Rachmawaty tahun 2007.

Daftar citra yang akan di uji seperti yang ditunjukkan pada tabel 3.2 yang terbagi kedalam 3 jenis detail citra.

Tabel 3.2. Tabel Citra Uji dan Jenis Detailnya

Jenis Detail Citra	Nama citra
High	kupu.bmp
	hely.bmp
	doggie.bmp
	monkey.bmp
	air terjun.bmp
	bunga.bmp
Medium	danau.bmp
	city.bmp
	city_paint.bmp
	taman.bmp
	tower.bmp
	lena.bmp
Low	hutan.bmp
	jet.bmp
	tulip.bmp
	kota_malam.bmp
	eiffel.bmp
	burung.bmp

3.3.2 Pengujian Rasio Kompresi (Nisbah pemampatan)

Rasio kompresi menunjukkan berapa persen suatu citra dapat dikompresi. Dalam penelitian ini akan dianalisis pengaruh pengubahan nilai *Threshold* T dengan 3 kondisi yang divariasikan nilai *Lagrange* λ-nya terhadap ratio kompresi. Pengujian ini menggunakan persamaan 2.5. Tabel yang digunakan untuk menganalisis seperti yang ditunjukkan pada tabel 3.3.

Tabel 3.3. Tabel Rasio Kompresi

No.	Nama Citra	Ratio Kompresi (%) λ = (50 / 100 / 150)			
		T = 25	T = 50	T = 75	T = 100
Rata-rata					

3.3.3 Pengujian Tingkat Kesalahan (*error rate*)

Ada hubungan antara jenis detail citra dan naik turunnya nilai tingkat kesalahan atau MSE, maka akan dilakukan analisis MSE terhadap jenis detail citra uji yang meliputi *low* detail, *medium* detail, dan *high* detail. Pengujian ini akan menunjukkan jenis detail citra mana yang memberikan nilai MSE terendah dengan kata lain ketepatan metode dekompresi dalam menyusun kembali citra asli telah memberikan hasil terbaik. Pengujian ini menggunakan persamaan 2.6. Tabel yang digunakan untuk menganalisis seperti yang ditunjukkan pada tabel 3.4.

Tabel 3.4. Tabel MSE

No.	Nama Citra	MSE λ = (50 / 100 / 150)				
		T = 25	T = 50	T = 75	T = 100	Rata-rata
Rata-rata						

3.4 Contoh Perhitungan

3.4.1 Proses Dekomposisi Citra

Dimisalkan citra berukuran 4 x 4 dengan nilai seperti yang ditunjukkan pada tabel 3.5.

Tabel 3.5. Citra 4 x 4

	Kolom0	Kolom1	Kolom2	Kolom3
Baris0	47	73	47	30
Baris1	67	69	46	42
Baris2	72	40	59	36
Baris3	67	35	72	52

Langkah-langkah dekomposisi *Lifting-based Forward* Transformasi *Wavelet* 9/7 sesuai yang telah dijelaskan pada subbab 3.2.2.2 terhadap potongan citra tersebut adalah:

1. Memasukkan citra input
2. *step*1 : (bar0+bar2)*-1.586134342+bar1 = -121.75

(bar2+bar2)*-1.586134342+bar3 = -161.403
dan seterusnya untuk koefisien ganjil.

*step*2 : (bar1+bar3)*-0.05298011854+bar2 = 87.0015
(bar1+bar1)*-0.05298011854+bar0 = 59.9006
dan seterusnya untuk koefisien genap.

step3 : (bar0+bar2)*0.8829110762+bar1 = 7.95155
(bar2+bar2)*0.8829110762+bar3 = -7.7741
dan seterusnya untuk koefisien ganjil.

step4 : (bar1+bar3)*0.4435068522+bar2 = 87.0801
(bar1+bar1)*0.4435068522+bar0 = 66.9537
dan seterusnya untuk koefisien genap.

*step*5 : bar0*1.149604398 = 76.9703
bar1/1.149604398 = 6.91677
dan seterusnya untuk koefisien genap dan ganjil

Hasil dari proses *step*1 hingga *step*5 ditunjukkan pada tabel 3.6, tabel 3.7 dan tabel 3.8.

Tabel 3.6. Hasil proses *step*1 dan *step*2 terhadap baris

	Kolom0	Kolom1	Kolom2	Kolom3
Baris0	59.90066	84.68033	59.94095	36.6421
Baris1	-121.75	-110.233	-122.13	-62.685
Baris2	87.0015	50.70855	71.57187	42.6165
Baris3	-161.403	-91.8907	-115.164	-62.202

Tabel 3.7. Hasil proses *step*3 dan *step*4 terhadap baris

	Kolom0	Kolom1	Kolom2	Kolom3
Baris0	66.95379	92.93237	54.60457	43.11148
Baris1	7.951553	9.303164	-6.01612	7.293435
Baris2	87.08017	53.79301	73.87953	51.63962
Baris3	-7.77417	-2.34847	11.21934	13.05149

Tabel 3.8. Hasil proses *step*5 terhadap baris

76.97038	106.8355	62.77366	49.56115
6.916773	8.092492	-5.23321	6.3443
100.1077	61.84068	84.93224	59.36513
-6.76248	-2.04285	9.759305	11.35303

3. Susun koefisien genap yang merupakan koefisien aproksimasi pada setengah jumlah baris pertama matriks lalu susun koefisien ganjil yang merupakan koefisien detail pada setengah jumlah baris kedua matrik. Seperti yang ditunjukkan pada tabel 3.9.

Tabel 3.9. Matrik *pack* terhadap baris

76.97038	106.8355	62.77366	49.56115
100.1077	61.84068	84.93224	59.36513
6.916773	8.092492	-5.23321	6.3443
-6.76248	-2.04285	9.759305	11.35303

4. Lakukan proses transformasi terhadap kolom dan lakukan *step* 1 – 5. Hasil dari proses *step*1 hingga *step*5 ditunjukkan pada tabel 3.10, tabel 3.11 dan tabel 3.12

Tabel 3.10. Hasil proses *step*1 dan *step*2 terhadap kolom

89.13645	-114.817	76.78113	-149.574
124.6542	-231.658	108.3346	-210.063
6.342243	5.422128	-6.73612	22.94543
-6.04235	-6.79622	11.1581	-19.6061

Tabel 3.11. Hasil proses *step*3 dan *step*4 terhadap kolom

117.2309	31.67312	84.62285	-13.9919
101.637	-25.9491	88.50447	-18.763
10.84328	5.074367	0.415429	11.05064
-8.06426	-2.27946	10.19022	0.097119

Tabel 3.12. Hasil proses *step*5 terhadap kolom

134.7692	27.55132	97.2828	-12.1711
116.8423	-22.5722	101.7451	-16.3213
12.46548	4.414012	0.477579	9.612559
-9.27071	-1.98282	11.71472	0.08448

5. Susun koefisien genap yang merupakan koefisien aproksimasi pada setengah jumlah kolom pertama matriks lalu susun koefisien ganjil yang merupakan koefisien detail pada setengah jumlah kolom kedua matrik. Seperti yang ditunjukkan pada tabel 3.13.

Tabel 3.13. Matrik *pack* terhadap kolom

134.7692	97.2828	27.55132	-12.1711
116.8423	101.7451	-22.5722	-16.3213
12.46548	0.477579	4.414012	9.612559
-9.27071	11.71472	-1.98282	0.08448

6. Membagi dua ukuran matrik untuk transformasi pada level selanjutnya (kembali ke *point* 2).

Transformasi dilakukan hingga level ke n, dimana $N=2^n$, dengan N adalah ukuran dimensi citra. Oleh karena ukuran cita = 4, maka iterasi dilakukan hingga level 2 ($4 = 2^2$). Hasil akhir transformasi citra ini ditunjukkan pada tabel 3.14.

Tabel 3.14. Transformasi citra 2 level

225.3197	-26.2918	27.55132	-12.1711
-6.73227	11.1945	-22.5722	-16.3213
12.46548	0.477579	4.414012	9.612559
-9.27071	11.71472	-1.98282	0.08448

3.4.2 Proses *Invers* Transformasi Citra

Langkah-langkah proses *Lifting-based invers* Transformasi *Wavelet* 9/7 sesuai yang telah dijelaskan pada subbab 3.2.3.3 terhadap potongan citra tersebut adalah:

1. Memasukkan matrik *input*-an.
2. Mendefinisikan ukuran matrik pada level terdalam. Untuk proses yang ditunjukkan berikut ini sudah pada level 1.
3. Koefisien-koefisien genap yang merupakan koefisien aproksimasi yang terletak pada setengah jumlah kolom pertama matriks dan koefisien-koefisien ganjil yang merupakan koefisien

detail yang terletak pada setengah jumlah kolom kedua matrik diubah susunannya menjadi 1 koefisien genap pada kolom pertama lalu 1 koefisien ganjil pada kolom kedua dan begitu seterusnya saling bergantian untuk tiap-tiap koefisien genap dan ganjil. Matriks yang terbentuk seperti yang ditunjukkan pada tabel 3.15.

Tabel 3.15. Matrik *Unpack* terhadap kolom

	Kolom0	Kolom1	Kolom2	Kolom3
Baris0	134.7692	27.55132	97.2828	-12.1711
Baris1	116.8423	-22.5722	101.7451	-16.3213
Baris2	12.46548	4.414012	0.477579	9.612559
Baris3	-9.27071	-1.98282	11.71472	0.08448

4. *step*1 : kol1*1.149604398 = 101.637
   kol0/1.149604398 = 117.2309
   dan seterusnya untuk koefisien genap dan ganjil.
   *step*2 : (kol1+kol3)*-0.4435068522+kol2 = 76.7811
   (kol1+kol1)*-0.4435068522+kol0 = 89.1364
   dan seterusnya untuk koefisien genap.
   *step*3 : (kol0+kol2)*-0.8829110762+kol1 = -114.817
   (kol2+kol2)*-0.8829110762+kol3 = -149.574
   dan seterusnya untuk koefisien ganjil.
   *step*4 : (kol1+kol3)*0.05298011854+kol2 = 62.7736
   (kol1+kol1)*0.05298011854+kol0 = 76.9703
   dan seterusnya untuk koefisien genap.
   *step*5 : (kol0+kol2)*1.586134342+ko1 = 106.8355
   (kol2+kol2)*1.586134342+kol3 = 62.77366
   dan seterusnya untuk koefisien ganjil.

Hasil dari proses *step*1 hingga *step*5 ditunjukkan pada tabel 3.16, tabel 3.17 dan tabel 3.18.

Tabel 3.16. Hasil proses *step*1 terhadap kolom

	Kolom0	Kolom1	Kolom2	Kolom3
Baris0	117.2309	31.67312	84.62285	-13.9919
Baris1	101.637	-25.9491	88.50447	-18.763
Baris2	10.84328	5.074367	0.415429	11.05064

Baris3	-8.06426	-2.27946	10.19022	0.097119

Tabel 3.17. Hasil proses *step*2 dan *step*3 terhadap kolom

	Kolom0	Kolom1	Kolom2	Kolom3
Baris0	89.13645	-114.817	76.78113	-149.574
Baris1	124.6542	-231.658	108.3346	-210.063
Baris2	6.342243	5.422128	-6.73612	22.94543
Baris3	-6.04235	-6.79622	11.1581	-19.6061

Tabel 3.18. Hasil proses *step*4 dan *step*5 terhadap kolom

	Kolom0	Kolom1	Kolom2	Kolom3
Baris0	76.97038	106.8355	62.77366	49.56115
Baris1	100.1077	61.84068	84.93224	59.36513
Baris2	6.916773	8.092492	-5.23321	6.3443
Baris3	-6.76248	-2.04285	9.759305	11.35303

5. Koefisien-koefisien genap yang merupakan koefisien aproksimasi yang terletak pada setengah jumlah baris pertama matriks dan koefisien-koefisien ganjil yang merupakan koefisien detail yang terletak pada setengah jumlah baris kedua matrik diubah susunannya menjadi 1 koefisien genap pada baris pertama lalu 1 koefisien ganjil pada baris kedua dan begitu seterusnya saling bergantian untuk tiap-tiap koefisien genap dan ganjil. Matriks yang terbentuk seperti yang ditunjukkan pada tabel 3.19.

Tabel 3.19. Matrik *Unpack* terhadap baris

76.97038	106.8355	62.77366	49.56115
6.916773	8.092492	-5.23321	6.3443
100.1077	61.84068	84.93224	59.36513
-6.76248	-2.04285	9.759305	11.35303

6. Lakukan proses transformasi terhadap baris dan lakukan *step*1 – 5. Hasil dari proses *step*1 hingga *step*5 ditunjukkan pada tabel 3.20, tabel 3.21 dan tabel 3.22.

Tabel 3.20. Hasil proses step1 terhadap baris

66.95379	92.93237	54.60457	43.11148
7.951553	9.303164	-6.01612	7.293435
87.08017	53.79301	73.87953	51.63962
-7.77417	-2.34847	11.21934	13.05149

Tabel 3.21. Hasil proses *step*2 dan *step*3 terhadap baris

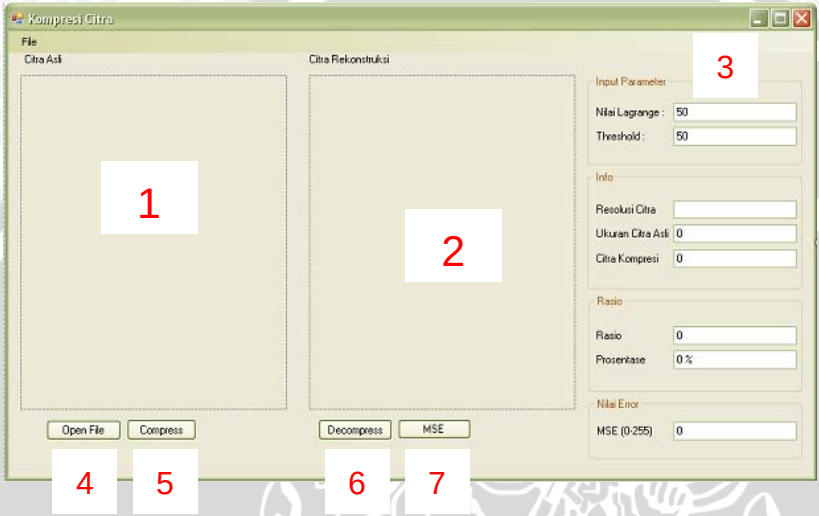
59.90066	84.68033	59.94095	36.6421
-121.75	-110.233	-122.13	-62.684
87.0015	50.70855	71.57187	42.6165
-161.403	-91.8907	-115.164	-62.201

Tabel 3.22. Hasil proses *step*4 dan *step*5 terhadap baris

47	73	47	30
67	69	46	42
72	40	59	36
67	35	72	52

3.5 Perancangan Antarmuka

Perancangan antarmuka software yang akan dibuat, ditunjukkan pada Gambar 3.16.



Gambar 3.16 Perancangan Antarmuka

Keterangan:

1. *Picture box* untuk menampilkan citra asli.
2. *Picture box* untuk menampilkan citra rekonstruksi.
3. *Group box* yang berisi informasi antara lain input parameter, info tentang resolusi citra, ukuran citra asli, ukuran citra kompresi, besar rasio kompresi, dan nilai *error* citra rekonstruksi.
4. Tombol untuk membuka file citra inputan.
5. Tombol untuk proses kompresi.
6. Tombol untuk proses dekompresi.
7. Tombol untuk mendapatkan nilai MSE.

BAB IV
IMPLEMENTASI DAN PEMBAHASAN

4.1 Lingkungan Implementasi

Lingkungan implementasi yang akan dijelaskan dalam subbab ini adalah lingkungan implementasi perangkat keras dan perangkat lunak.

4.1.1 Lingkungan Implementasi Perangkat Keras

Perangkat keras yang digunakan dalam pengembangan perangkat lunak kompresi citra digital ini adalah:

1. Prosesor AMD Turion 64 2.0GHz
2. Memori 512 MB
3. Hardisk dengan kapasitas 80GB
4. Monitor 15”
5. Keybord
6. Mouse

4.1.2 Lingkungan Implementasi Perangkat Lunak

Perangkat lunak yang digunakan dalam pengembangan perangkat lunak kompresi citra digital ini adalah:

1. Sistem operasi yang digunakan adalah Microsoft Windows XP Profesional.
2. *Software* yang digunakan dalam mengembangkan sistem adalah Visual C++ 2005.
3. *Software* yang digunakan untuk mengolah data adalah MS Excel 2003.

4.2 Implementasi Perangkat Lunak

Berdasarkan analisa dan perancangan proses yang terdapat pada subbab 3.2, maka pada subbab ini akan dijelaskan implementasi proses-proses tersebut.

4.2.1 Struktur Data

Struktur data yang digunakan pada perangkat lunak ini direpresentasikan pada *SourceCode* 4.1.

```
Bitmap^ img = dynamic_cast<Bitmap^>
            (pictureBoxCitraAsli->Image);

Red   = gcnew array<double,2>(img->Width,img->Height);
Green = gcnew array<double,2>(img->Width,img->Height);
Blue  = gcnew array<double,2>(img->Width,img->Height);

array<double,1>^ vektor1 = gcnew array<double,1>
                          (img->Height/4*img->Width/4);
array<double,1>^ vektor2 = gcnew array<double,1>
                          (img->Height/4*img->Width/4);
array<double,1>^ vektor3 = gcnew array<double,1>
                          (img->Height/4*img->Width/4);
array<double,1>^ vektor4 = gcnew array<double,1>
                          (img->Height/4*img->Width/4);
array<double,1>^ vektor5 = gcnew array<double,1>
                          (img->Height/2*img->Width/2);
array<double,1>^ vektor6 = gcnew array<double,1>
                          (img->Height/2*img->Width/2);
array<double,1>^ vektor7 = gcnew array<double,1>
                          (img->Height/2*img->Width/2);


array<int,2>^ ut1 = gcnew array<int,2>
                    (3,img->Height/4*img->Width/4);
array<int,2>^ ut2 = gcnew array<int,2>
                    (3,img->Height/4*img->Width/4);
array<int,2>^ ut3 = gcnew array<int,2>
                    (3,img->Height/4*img->Width/4);
array<int,2>^ ut4 = gcnew array<int,2>
                    (3,img->Height/4*img->Width/4);
array<int,2>^ ut5 = gcnew array<int,2>
                    (3,img->Height/2*img->Width/2);
array<int,2>^ ut6 = gcnew array<int,2>
                    (3,img->Height/2*img->Width/2);
array<int,2>^ ut7 = gcnew array<int,2>
                    (3,img->Height/2*img->Width/2);
```

```
struct sideinfo
{
        int startbaris;
        int startkolom;
        int endbaris;
        int endkolom;
};

/*       huffman file headers              */
typedef struct huffheader {
       char magic[4];
       int symnum;
       int uncmpsize;
       int width;
       int height;
       double maxValue[7];
       double minValue[7];
       sideinfo red[7];
       sideinfo green[7];
       sideinfo blue[7];
       unsigned char conditionRed[7];
       unsigned char conditionGreen[7];
       unsigned char conditionBlue[7];
       int lengthUtRed;
       int lengthUtGreen;
       int lengthUtBlue;
} *PHUFFHDR;
```

***SourceCode* 4.1.** Struktur Data

Keterangan struktur data pada *SourceCode* 4.1 ditunjukkan pada Tabel 4.1.

Tabel 4.1 Keterangan Struktur Data

Struktur Data	Keterangan

`array<double,2>^ Red;`
`array<double,2>^ Green;`
`array<double,2>^ Blue;`
	Matrik yang digunakan untuk menampung nilai komponen warna R, G, dan B.

`array<double,1>^ vektor;`	Array vektor yang digunakan untuk menampung vektor hasil kuantisasi pada tiap subband
`array<int,2>^ ut`	Matrik yang digunakan untuk

	menampung vektor informasi Ut masing-masing elemen warna RGB pada tiap-tiap subband
`*PHUFFHDR;`	*Pointer* huffman header yang digunakan menunjuk *Header file* kompresi. *Header* ini berisi beberapa informasi mengenai *file* kompresi dan dekompresi.

4.2.2 Implementasi Kompresi

Untuk melakukan proses kompresi langkah pertama yang dilakukan *User* adalah memasukkan citra inputan, menentukan nilai *Lagrage*, dan menentukan nilai *threshold*. Setelah itu akan dilakukan proses seperti yang telah dijelaskan pada subbab 3.2.2.

4.2.2.1 *Blocking* Citra

Tahap awal dari proses kompresi ini adalah melakukan proses mengambil masing-masing elemen warna RGB pada tiap pixel yang terdapat di citra. Fungsi untuk mendapatkan elemen warna RGB pada tiap pixel ditunjukkan pada *SourceCode* 4.2.

```
Point size = PixelSize(img);

LockBitmap(img);

for (int y = 0; y < size.Y; y++)
{
      PixelData* pPixel = PixelAt(0, y);
      for (int x = 0; x < size.X; x++)
      {
            Red[x,y]   = (double) pPixel->red;
            Green[x,y] = (double) pPixel->green;
            Blue[x,y]  = (double) pPixel->blue;

            pPixel++;
      }

}

UnlockBitmap(img);
```

```
public:
	Point PixelSize(Bitmap^ bmp)
	{
		GraphicsUnit unit = GraphicsUnit::Pixel;
		RectangleF bounds = bmp->GetBounds(unit);
			Point results((int) bounds.Width,
			(int)  bounds.Height);
		return results;

	}

public:
	void UnlockBitmap(Bitmap^ bmp)
	{
		bmp->UnlockBits(bitmapData);
		bitmapData = nullptr;
		pBase = nullptr;
	}

public:
	void LockBitmap(Bitmap^ bmp)
	{
		GraphicsUnit unit = GraphicsUnit::Pixel;
		RectangleF boundsF = bmp->GetBounds(unit);
		Rectangle bounds((int) boundsF.X,
		(int) boundsF.Y,
		(int) boundsF.Width,
		(int) boundsF.Height);

			width = (int) boundsF.Width *
			sizeof(PixelData);
		if (width % 4 != 0)
		{
			 width = 4 * (width / 4 + 1);
		}

			bitmapData = bmp->LockBits(bounds,
			ImageLockMode::ReadWrite,
			PixelFormat::Format24bppRgb);

			pBase = (unsigned char*) bitmapData-
			>Scan0.ToPointer();
	}
```

```
private:
      PixelImage::PixelData* PixelAt(int x, int y)
      {
            return (PixelData*) (pBase + y * width +
            x* sizeof(PixelData));
       }
```

***SourceCode* 4.2.** fungsi PixelAt / GetPixel

Fungsi PixelAt merupakan fungsi modifikasi untuk menggantikan fungsi GetPixel yang merupakan fungsi asli dari Visual C++. Fungsi PixelAt mampu memberikan waktu proses yang lebih cepat daripada fungsi GetPixel. Fungsi PixelAt akan menghasilkan matrik masing-masing elemen warna RGB.

4.2.2.2 *Lifting-based Forward Wavelet* 9/7 *Transform*

Setelah proses *blocking* citra selesai dilakukan, langkah selanjutnya adalah melakukan transformasi *Wavelet* 9/7 pada masing-masing elemen warna RGB. Fungsi untuk melakukan FWT 9/7 ditunjukkan pada *SourceCode* 4.3.

```
void fwt97(array<double,2>^ x, int height, int
           width)
{
 double a; int i,j;

#pragma region operasi baris

// Predict 1
  a=-1.586134342;
  for (i=0; i<width; i++)
  {
      for (j=1; j<height-2; j+=2)
      {
        x->SetValue((Double)(x->GetValue(i,j)) +
              a * ((Double)(x->GetValue(i,j-1)) +
              (Double)(x->GetValue(i,j+1))),i,j);
      }

      x->SetValue((Double)(x->GetValue(i,height-1))+
```

```
        2 * a * height- 1))+ 2 * a * (Double)(x->
                GetValue(i, height- 2)),i,height-1);

      }
// Update 1
   a=-0.05298011854;
      for (i=0; i<width; i++)
      {
       for (j=2; j<height; j+=2)
       {
         x->SetValue((Double)(x->GetValue(i,j)) +
                a * ((Double)(x->GetValue(i,j-1)) +
                (Double)(x->GetValue(i,j+1))),i,j);
       }
       x->SetValue((Double)(x->GetValue(i,0))+ 2 *
                a * (Double)(x->GetValue(i,1)),i,0);
      }


// Predict 2
   a=0.8829110762;
      for (i=0; i<width; i++)
      {
       for (j=1; j<height-2; j+=2)
       {
         x->SetValue((Double)(x->GetValue(i,j))+
                a * ((Double)(x->GetValue(i,j-1))+
                (Double)(x->GetValue(i,j+1))),i,j);
       }
       x->SetValue((Double)(x->GetValue(i,height-
           1))+ 2 * a * (Double)(x->GetValue(i,height-
           2)),i,height-1);
      }

// Update 2
   a = 0.4435068522;
      for (i=0; i<width; i++)
      {
       for (j=2; j<height; j+=2)
       {
         x->SetValue((Double)(x->GetValue(i,j)) +
                a * ((Double)(x->GetValue(i,j-1)) +
                (Double)(x->GetValue(i,j+1))),i,j);
       }
```

```
        x->SetValue((Double)(x->GetValue(i,0))+ 2 *
                a * (Double)(x->GetValue(i,1)),i,0);
}


// Scale
  a=1/1.149604398;
  for (i=0; i<width; i++)
  {
      for (j=0; j<height; j++)
      {
        if (j%2) x->SetValue((Double)(x->
              GetValue(i,j))*  a,i,j);
        else x->SetValue((Double)(x->GetValue(i,j))/
                                a,i,j);
      }
  }

// Pack
  tempbank = gcnew array<double,2>(width,height);
  for (i=0; i<width; i++)
  {
      for (j=0; j<height; j++)
      {
        if (j%2==0) tempbank[i,j/2] = x[i,j];
        else tempbank[i,height/2 + j/2] = x[i,j];
      }
  }


  for (i=0; i<width; i++)
  {
      for (j=0; j<height; j++)
      {
            x[i,j] = tempbank[i,j];
      }
  }

#pragma endregion operasi baris

#pragma region operasi kolom

  // Predict 1
  a=-1.586134342;
  for (i=0; i<height; i++)
```

```
{
    for (j=1; j<width-2; j+=2)
       {
        x->SetValue((Double)(x->GetValue(j,i)) + a *
                    ((Double)(x->GetValue(j-1,i)) +
                  (Double)(x->GetValue(j+1,i))),j,i);
       }
        x->SetValue((Double)(x->GetValue(width-1,i))+
        2 * a * (Double)(x->GetValue(width-2,i)),width-
                      1,i);
       }

// Update 1
   a=-0.05298011854;
      for (i=0; i<height; i++)
        {
          for (j=2; j<width; j+=2)
           {
              x->SetValue((Double)(x->GetValue(j,i)) +
                  a * ((Double)(x->GetValue(j-1,i)) +
                    (Double)(x-
     >GetValue(j+1,i))),j,i);
           }
             x->SetValue((Double)(x->GetValue(0,i))+ 2
                 *a * (Double)(x->GetValue(1,i)),0,i);
        }

// Predict 2
   a=0.8829110762;
       for (i=0; i<height; i++)
        {
          for (j=1; j<width-2; j+=2)
           {
             x->SetValue((Double)(x->GetValue(j,i))+
                  a * ((Double)(x->GetValue(j-1,i))+
                  (Double)(x->GetValue(j+1,i))),j,i);
           }
             x->SetValue((Double)(x->GetValue(width-
                 1,i))+ 2 * a * (Double)(x->
                    GetValue(width-2,i)),width-1,i);
        }

// Update 2
        a = 0.4435068522;
        for (i=0; i<height; i++)
```

```
    {
       for (j=2; j<width; j+=2)
       {
           x->SetValue((Double)(x->GetValue(j,i)) + a *
           ((Double)(x->GetValue(j-1,i)) + (Double)(x->
             GetValue(j+1,i))),j,i);
         }

       x->SetValue((Double)(x->GetValue(0,i))+ 2 * a *
                     (Double)(x->GetValue(1,i)),0,i);
    }

// Scale
    a=1/1.149604398;
    for (i=0; i<height; i++)
    {
       for (j=0; j<width; j++)
       {
         if (j%2) x->SetValue((Double)(x->
               GetValue(j,i))* a,j,i);
           else x->SetValue((Double)(x->GetValue(j,i))
                             / a,j,i);
       }
    }

 // Pack
    tempbank = gcnew array<double,2>(width,height);
    for (i=0; i<height; i++)
    {
       for (j=0; j<width; j++)
       {
              if (j%2==0) tempbank[j/2,i] = x[j,i];
              else tempbank[width/2+j/2,i] = x[j,i];
       }
    }


    for (i=0; i<width; i++)
    {
       for (j=0; j<height; j++)
       {
              x[i,j] = tempbank[i,j];
       }
    }
  #pragma endregion operasi kolom }
```

***SourceCode* 4.3.** FWT 9/7

Fungsi FWT 9/7 menerima masukan dari matrik masing-masing elemen warna RGB dan parameter yang menunjukkan tingkat kedalaman level transformasi. Pada perangkat lunak tingkat kedalaman ditentukan hingga level 2 dan menghasilkan 7 subband yang mempunyai nilai koefisien-koefisien yang berbeda.

4.2.2.3 Kuantisasi Vektor

Setelah proses transformasi selesai dilakukan, langkah selanjutnya adalah melakukan kuantisasi tetapi sebelum proses kuantisasi dilakukan nilai-nilai koefisien yang merupakan hasil dari transformasi di konversi ke *range* nilai *byte* (0 – 255) terlebih dahulu. Hal ini dilakukan untuk mempermudah proses kuantisasi sekaligus menghemat ruang penyimpanan.

Konversi pertama kali dilakukan dengan cara mencari nilai minimum dan maksimum elemen RGB dalam masing-masing subband, setelah medapatkan nilai minimum dan maksimum lakukan proses perhitungan untuk mendapatkan nilai *byte*. Nilai minimum dan maksimum tersebut akan disimpan pada *header file* untuk proses dekompresi. *SourceCode* proses mendapatkan nilai minimum dan maksimum serta proses perhitungan untuk mendapatkan nilai *byte* ditunjukkan pada *SourceCode* 4.4.

```
private:
       System::Void Form1::FindMinMax(array<double,1>^
vektor, array<double,1>^ min, array<double,1>^ max, int
index)
{
  double minValues = min[index];
  double maxValues = max[index];

     for(int i = 0; i < vektor->Length; i++)
        {
          if (minValues > vektor[i])
              minValues = vektor[i];
          if (maxValues < vektor[i])
              maxValues = vektor[i];
        }
          if (minValues < min[index])
              min[index] = minValues;
          if (maxValues > max[index])
              max[index] = maxValues; }
```

```
private:
       System::Void Form1::convertToByte(array<double,2>^
matriks,array<double,1>^ min, array<double,1>^ max, int
indexSubband, int x, int y, int width, int height)
{
  double maxValues = max[indexSubband];
  double minValues = min[indexSubband];

  if ((minValues < 0))
   {
      maxValues = maxValues - minValues;
   }

  for (int i = y; i < height; i++)
    {
      for (int j = x; j < width; j++)
       {
         // perhitungan mendapatkan nilai byte
         matriks[j,i] = Math::Round((matriks[j,i] –
                        minValues) * 255 / maxValues);
       }
    }
  }
```

***SourceCode* 4.4** konversi ke *Byte*

Setelah proses konversi selesai dilakukan, lakukan proses kuantisasi pada masing-masing elemen warna RGB untuk tiap-tiap subband. Selain vektor input dari masing-masing subband, proses ini juga membutuhkan matrik *codebook*, matrik Ut serta *star* baris-kolom dan *end* baris-kolom yang memberikan nilai kembalian. Fungsi untuk melakukan kuantisasi ditunjukkan pada *SourceCode* 4.5.

```
void prosesKuantisasi(array<double,2>^ codebook,
      array<double,1>^ vektor, array<int,2>^ ut,
      int RGB,int width, int height, int
      &startbaris, int &startkolom, int &endbaris,
      int &endkolom)
       {
         int indexVektor = 0;
         double euclideanNorm = 0.0;
```

```
double euclideanNormMin1 = double::MaxValue,
       euclideanNormMin2 = double::MaxValue;

bool stop = false;
int startbaris1 = 0;
int startkolom1 = 0;
int endbaris1 = 0;
int endkolom1 = 0;
int indexCodeword = 0;
int a = 0; // buat membantu mencari indexCodeword

// Mencari jarak euclidean terdekat

for (int i = 0; i < height; i++)
   {
      for (int j = 0; j < width; j++)
       {
          euclideanNorm = euclideanNorm +
          Math::Pow(((double)vektor[indexVektor] –
                    (double)codebook[j,i]),2);
          indexVektor++;
         if (indexVektor == vektor->Length)
             {
             euclideanNorm =
                 Math::Sqrt(euclideanNorm);
                   indexVektor = 0;
                   endbaris1 = i;
                   endkolom1 = j;

                   a++;

             if (euclideanNormMin1 > euclideanNorm)
                {
                   indexCodeword = a;
                   euclideanNormMin1=euclideanNorm;
                   startbaris = startbaris1;
                   startkolom = startkolom1;
                   endbaris = endbaris1;
                   endkolom = endkolom1;
               }

      if (j == width-1)

        {
            startbaris1 = i + 1;
            startkolom1 = 0;
         }
```

```
      else if (j < width-1)
         {
            startbaris1 = i;
             startkolom1 = j+1;
         }
             euclideanNorm = 0;

     int sisa=(height - i - 1)* width + (width - j - 1);
     if (sisa < vektor->Length)
     {
       stop = true;
       break;
     }
  }
}

  if (stop == true)
     break;
}

//membentuk vektor parsial
stop = false;
indexVektor = 0;
array<double,1>^ pt = gcnew array<double,1>
                      (vektor->Length);

for (int i = startbaris; i <= endbaris; i++)
{
  for (int j = 0; j < width; j ++)
  {
     if ((i == startbaris && j >= startkolom) ||
        (i > startbaris && i < endbaris) ||
        (i == endbaris && j <= endkolom))
      {
        double selisih =
               Math::Abs((double)vektor[indexVektor] –
                        (double)codebook[j,i]);



       if (selisih > threshold)
        {
          pt[indexVektor] = vektor[indexVektor];
          ut[RGB,indexVektor] = 1;
        }
        else
```

```
        {
          pt[indexVektor] = codebook[j,i];
          ut[RGB,indexVektor] = 0;
        }
       indexVektor++;
     }

   if ((i == endbaris) && (j == endkolom))
     {
       stop = true;
       break;
     }

   }
   if (stop == true)
   break;
 }

// Mencari jarak euclidean vektor parsial

euclideanNorm = 0.0;
for (int i = 0; i < vektor->Length; i++)
{
  euclideanNorm = euclideanNorm +
  Math::Pow(((double)vektor[i] - (double)pt[i]),2);
}

euclideanNorm = Math::Sqrt(euclideanNorm);
euclideanNormMin2 = euclideanNorm;

//Menetukan mode update codebook
double rPt = 0.0;    //bitrate vektor parsial
double rUt = 0.0;    //bitrate vektor side information
double ri = 8;       //bitrate vektor index (codeword)
double rs = 8;       //bitrate vektor input




int temp1 = 0;
for (int i = 0; i < ut->GetLength(1); i++)
{
    if (ut[RGB,i] == 1)
       temp1++;
}
```

```
rPt = 8 * temp1 / vektor->Length;
rUt = 1 * temp1 / vektor->Length;

double J3 = euclideanNormMin2 + lagrange * (ri +
            rPt + rUt);
double J2 = lagrange * (ri + rs);
double J1 = euclideanNormMin1 + lagrange * ri;

if ((J3 < J2) && (J3 < J1)) // PCU J3
{
  stop = false;
  indexVektor = 0;
  for (int i = startbaris; i <= endbaris; i++)
   {
     for (int j = 0; j < width; j ++)
      {
        if ((i == startbaris && j >= startkolom) ||
            (i > startbaris && i < endbaris) ||
            (i == endbaris && j <= endkolom))
        {
           ut[RGB,indexVektor] = 0;
           double selisih =
           Math::Abs((double)vektor[indexVektor]-
                    (double)codebook[j,i]);
           if (selisih < threshold)
           {
             ut[RGB,indexVektor] = 1;
             vektor[indexVektor]= (double)indexCodeword;
           }
           indexVektor++;

        if ((i == endbaris) && (j == endkolom))
        {
             stop = true;
             break;
        }
      }


   if (stop == true)
   break;
  }
}

else if ((J2 < J1) && (J2 < J3)) // FCU J2
   {
```

```
        ut[RGB,0] = 0;
        for (int i = 1; i < ut->GetLength(1); i++)
             ut[RGB,i] = -1;

    }
else // no update J1
stop = false;
indexVektor = 0;
for (int i = startbaris; i <= endbaris; i++)
{
  for (int j = 0; j < width; j ++)
  {
     if ((i == startbaris && j >= startkolom) ||
         (i > startbaris && i < endbaris) ||
         (i == endbaris && j <= endkolom))
     {
       vektor[indexVektor] =(double)indexCodeword;
       indexVektor++;
     }
     if ((i == endbaris) && (j == endkolom))
     {
       stop = true;
       break;
     }
  }

 if (stop == true)
    break;
}

     ut[RGB,0] = 1;
     for (int i = 1; i < ut->GetLength(1); i++)
             ut[RGB,i] = -1;

    }
}
```

***SourceCode* 4.5.** Kuantisasi

Fungsi ini menghasilkan *output* berupa vektor yang berisi indeks *codeword* dan atau nilai koefisien vektor input. Nilai matrik Ut, *star* baris-kolom, dan *end* baris-kolom akan disimpan pada *header file* untuk proses dekompresi.

4.2.2.4 Huffman *Encoder*

Tahap akhir dari proses kompresi adalah pengkodean Huffman. Vektor hasil dari proses kuantisasi untuk seluruh elemen warna RGB dijadikan dalam satu vektor yang akan dikompresi menggunakan metode Huffman. Potongan kode untuk membentuk vektor input Huffman ditunjukkan pada *SourceCode* 4.6.

```
//Membentuk vektor input huffman
unsigned char* source = new unsigned char
                        [3*img->Width*img->Height];
int k = 0;
for (int i = 0; i < img->Height; i++)
{
  for (int j = 0; j < img->Width; j++)
  {
       int val = (int)Red[j,i];
       source[k] = (unsigned char) val;
       k++;
  }
}
for (int i = 0; i < img->Height; i++)
{
  for (int j = 0; j < img->Width; j++)
  {
       int val = (int)Green[j,i];
       source[k] = (unsigned char)val;
       k++;
  }
}
for (int i = 0; i < img->Height; i++)
{
  for (int j = 0; j < img->Width; j++)
  {
       int val = (int)Blue[j,i];
       source[k] = (unsigned char)val;
       k++;
  }
}
```

***SourceCode* 4.6** Vektor input Huffman

Pada tahap ini juga akan dilakukan penulisan *Header file* yang berisi informasi seperti yang telah dijelaskan pada subbab 3.2.1. Potongan kode untuk inisialisasi *header file* ditunjukkan pada *SourceCode* 4.7.

```
/*                headers init                                    */
memcpy(pheader->magic, "HUFF", 4);
pheader->symnum = symnum;
pheader->uncmpsize = filesize;
pheader->width = widthImage;
pheader->height = heightImage;

for (int i = 0; i < 7; i++)
{
pheader->red[i].startbaris = infoRed[i].startbaris;
pheader->red[i].startkolom = infoRed[i].startkolom;
pheader->red[i].endbaris = infoRed[i].endbaris;
pheader->red[i].endkolom = infoRed[i].endkolom;
pheader->green[i].startbaris = infoGreen[i].startbaris;
pheader->green[i].startkolom = infoGreen[i].startkolom;
pheader->green[i].endbaris = infoGreen[i].endbaris;
pheader->green[i].endkolom = infoGreen[i].endkolom;
pheader->blue[i].startbaris = infoBlue[i].startbaris;
pheader->blue[i].startkolom = infoBlue[i].startkolom;
pheader->blue[i].endbaris = infoBlue[i].endbaris;
pheader->blue[i].endkolom = infoBlue[i].endkolom;
pheader->conditionRed[i] = condRed[i];
pheader->conditionGreen[i] = condGreen[i];
pheader->conditionBlue[i] = condBlue[i];
pheader->minValue[i] = minVal[i];
pheader->maxValue[i] = maxVal[i];
}

pheader->lengthUtRed = sizeR_Ut;
pheader->lengthUtGreen = sizeG_Ut;
pheader->lengthUtBlue = sizeB_Ut;

pdest += sizeof(huffheader);
bitslast = 8;

*pdest = 0;
if (sizeR_Ut > 0)
  {
```

```
  for (int j = 0; j < sizeR_Ut; j++)
     {
       *pdest = *pdest | (utR[j] << (--bitslast));

       if (bitslast == 0)
         {
           pdest += sizeof(unsigned char);
           *(pdest) = 0;
           bitslast = 8;
         }
       }

  }

if (sizeG_Ut > 0)
  {
    for (int j = 0; j < sizeG_Ut; j++)
       {
        *pdest = *pdest | (utG[j] << (--bitslast));

         if (bitslast == 0)
           {
             pdest += sizeof(unsigned char);
             *(pdest) = 0;
             bitslast = 8;
           }
       }
  }

if (sizeB_Ut > 0)
  {
    for (int j = 0; j < sizeB_Ut; j++)
       {
         *pdest = *pdest | (utB[j] << (--bitslast));

          if (bitslast == 0)
            {
              pdest += sizeof(unsigned char);
              *(pdest) = 0;
              bitslast = 8;
             }
       }
   }
```

***SourceCode* 4.7.** Inisialisasi *Header File*

Setelah menuliskan *Header file*, langkah selanjutnya adalah melakukan *Huffman encoding* terhadap vektor input Huffman. Dari vektor input Huffman dicari frekuensi banyaknya masing-masing koefisien lalu dibentuk *tree* lalu kodekan koefisien tersebut dengan kode huffman yang terbentuk. Potongan kode untuk melakukan proses ini ditunjukkan pada *SourceCode* 4.8.

```
void Huffman::encode(unsigned char *dest, int &csize,
unsigned char *sour, int lengthSour, int usize)
{
  pheader = (PHUFFHDR)dest;     //huff header
  psour = sour;                 //sour buffer
  pdest = dest;                 //dest buffer
  lengthSource = lengthSour;
  filesize = usize; //uncompressed file size
  csize = 0;                    //fault protection
  symnum = 0;
  bitslast = 8;

  storetree();        //get freqs and store headers
  buildtree();                  //build htree
  buildbook();                  //build codebook
  for (int i = 0; i < lengthSource; i++)
               writesymb(*psour++);

  csize = int((pdest - dest) + 1);
}

void Huffman::storetree(void)
{
  /*            memory init                          */
    memset(tmass1, 0, 256*sizeof(hufftree));
    memset(tmass2, 0, 256*sizeof(hufftree));
    memset(smass, 0, 256*sizeof(huffsymbol));

    temptree = (PHTREE)tmass1;
    htree = (PHTREE)tmass2;
    hsymbol = (PSYMB)smass;

    //initilize symbols
    for (int i = 0; i < 256; i++) temptree[i].s1 = i;

    for (int i = 0; i < lengthSource; i++)
     temptree[psour[i]].freq++;   //get symb frequencies
```

```
  for (int i = 0; i < 256; i++)
    if (temptree[i].freq) symnum++; //get total symbols

  sort ();  //sort temptree

/*   make [  freq  ][C] pairs    */
for (int k = 0; k < symnum; k++)
{
  psheader = (PSYMBHDR)pdest;
  psheader->symb = temptree[k].s1;
  psheader->freq = temptree[k].freq;

  pdest += sizeof(unsigned char) + sizeof(unsigned
                                          int);
 }

   *pdest = 0;
    bitslast = 8;
}

void Huffman::sort()
{
 for (int i = 0; i < 255; i++)
 {
   for (int j = i + 1; j < 256; j++)
   {
     if (temptree[i].freq < temptree[j].freq)
     {
       hufftree temp = temptree[i];
       temptree[i] = temptree[j];
       temptree[j] = temp;
     }
   }
  }
}

/*         build htree from temptree                */
void Huffman::buildtree()
{
 /*         special case symnum = 1    */
    if (symnum == 1)
       {
       hsymbol[psour[0]].size = 1;
                    return;
        }
```

```
  /*        build 1 & 2 trees          */
for (int i = symnum - 2; i >= 0; i--) {
          if (temptree[i].parent) {
            htree[i].pleft = temptree[i].parent;
            htree[i].pleft->parent = &htree[i];
          } else
            htree[i].s1 = temptree[i].s1;
          if (temptree[i+1].parent) {
            htree[i].pright = temptree[i+1].parent;
            htree[i].pright->parent = &htree[i];
          } else
            htree[i].s2 = temptree[i+1].s1;
         if (i) {
            temptree[i].freq += temptree[i+1].freq;
            temptree[i].parent = &htree[i];
            temptree[i+1].freq = 0;

            sort();
                }
        }
}

/*         build codebook from htree        */
void Huffman::buildbook()
{
 unsigned char c;
 PHTREE parent, child;

       /*     build symbols from htree    */
for (int i = symnum - 2; i >= 0; i--) {
   if (!htree[i].pleft) {                //0 bit symbol
         c = htree[i].s1;
         hsymbol[c].size++;
         child = &htree[i];
         parent = child->parent;
         while (parent) {
             //add 0 to symb code
             if (parent->pleft == child)
                       hsymbol[c].size++;
             //add 1 to symb code
             if (parent->pright == child)
                    hsymbol[c].symb |= __int64(1 <<
                                  hsymbol[c].size++);

             child = parent;  //child becomes
                                       a parent
           parent = child->parent; //parent
```

```
                                becomes 'childs' parent
            }
        }

    if (!htree[i].pright) {           //1bit symbol
        c = htree[i].s2;
        hsymbol[c].symb |= __int64(1 <<
                                hsymbol[c].size++);

        child = &htree[i];
        parent = child->parent;
        while (parent) {
             //add 0 to symb code
             if (parent->pleft == child)
                    hsymbol[c].size++;
             //add 1 to symb code
             if (parent->pright == child)
                    hsymbol[c].symb |= __int64(1 <<
                                hsymbol[c].size++);

             child = parent; //child becomes
                             a parent
             parent = child->parent;
                   //parent becomes 'childs' parent
             }
          }
        }

/*          write codebook symbol to dest     */
void Huffman::writesymb(int c)
{
     for (unsigned int i = 1; i <= hsymbol[c].size; i++)
       {
         *pdest |= ((0x1 & (hsymbol[c].symb >>

               (hsymbol[c].size - i))) << (--bitslast));

             if (bitslast == 0) {
                  *(++pdest) = 0;
                 bitslast = 8;
               }
        }
}
```

***SourceCode* 4.8.** Huffman *Encoder*

Proses Huffman Encoder akan menghasilkan data *dest* dan header file yang akan dituliskan dalam satu file kompresi.

4.2.3 Implementasi Dekompresi

Langkah pertama yang dilakukan *user* untuk melakukan proses dekompresi adalah memasukkan *file* kompresi. Setelah itu akan dilakukan proses seperti yang telah dijelaskan pada subbab 3.2.3.

4.2.3.1 Huffman *Decoder*

Tahap awal dari proses dekompresi adalah membaca *header file*. Kode yang digunakan untuk membaca *header file* ditunjukkan pada *SourceCode* 4.9.

```
/*          read huff headers                    */
void Huffman::readtree()
{
        /*          memory init                  */
   memset(tmass1, 0, 256*sizeof(hufftree));
   memset(tmass2, 0, 256*sizeof(hufftree));
   memset(smass, 0, 256*sizeof(huffsymbol));

   temptree = (PHTREE)tmass1;
   htree = (PHTREE)tmass2;
   hsymbol = (PSYMB)smass;

filesize = pheader->uncmpsize;
symnum = pheader->symnum;
widthImage = pheader->width;
heightImage = pheader->height;

for (int i = 0; i < 7; i++)
{
infoRed[i].startbaris = pheader->red[i].startbaris;
infoRed[i].startkolom = pheader->red[i].startkolom;
infoRed[i].endbaris = pheader->red[i].endbaris;
infoRed[i].endkolom = pheader->red[i].endkolom;
infoGreen[i].startbaris = pheader->
                          green[i].startbaris;
infoGreen[i].startkolom = pheader->
                          green[i].startkolom;
infoGreen[i].endbaris = pheader->green[i].endbaris;

infoGreen[i].endkolom = pheader->green[i].endkolom;
```

```
infoBlue[i].startbaris = pheader->
                         blue[i].startbaris;
infoBlue[i].startkolom = pheader->
                         blue[i].startkolom;
infoBlue[i].endbaris = pheader->blue[i].endbaris;
infoBlue[i].endkolom = pheader->blue[i].endkolom;

condRed[i] = pheader->conditionRed[i];
condGreen[i] = pheader->conditionGreen[i];
condBlue[i] = pheader->conditionBlue[i];

minVal[i] = pheader->minValue[i];
maxVal[i] = pheader->maxValue[i];
}

sizeR_Ut = pheader->lengthUtRed;
sizeG_Ut = pheader->lengthUtGreen;
sizeB_Ut = pheader->lengthUtBlue;

psour += sizeof(huffheader);
bitslast = 8;

utR = new unsigned char[sizeR_Ut];
utG = new unsigned char[sizeG_Ut];
utB = new unsigned char[sizeB_Ut];


if (sizeR_Ut > 0)
{
  for (int j = 0; j < sizeR_Ut; j++)
    {
       int bits =  0x1 & (*psour >> (--bitslast));
       utR[j] = bits;

       if (bitslast == 0)
         {
           psour += sizeof(unsigned char);
           bitslast = 8;
         }

      }
}
else if (sizeR_Ut == 0)
{
  *utR = 0;
}
```

```
if (sizeG_Ut > 0)
{
  for (int j = 0; j < sizeG_Ut; j++)
    {
       int bits =  0x1 & (*psour >> (--bitslast));
       utG[j] = bits;

       if (bitslast == 0)
         {
           psour += sizeof(unsigned char);
           bitslast = 8;
         }
    }
}
else if (sizeG_Ut == 0)
{
  *utG = 0;
}

if (sizeB_Ut > 0)
{
  for (int j = 0; j < sizeB_Ut; j++)
     {
        int bits =  0x1 & (*psour >> (--bitslast));
        utB[j] = bits;

        if (bitslast == 0)
          {
            psour += sizeof(unsigned char);
            bitslast = 8;
          }
      }
}
else if (sizeB_Ut == 0)
{
  *utB = 0;
}

psour += sizeof(unsigned char);
bitslast = 8;
```

***SourceCode* 4.9** Membaca *Header File*

Setelah membaca *header file*, langkah selanjutnya adalah melakukan proses pendekodean kode huffman menjadi koefisien

vektor seperti semula. Potongan fungsi untuk melakukan pendekodean kode huffman ditunjukkan pada *SourceCode* 4.10.

```
void Huffman::decode(unsigned char *dest, int &usize,
unsigned char *sour)
{
        pheader = (PHUFFHDR)sour;     //huff header
        psour = sour;                 //sour buffer
        pdest = dest;                 //dest buffer
        usize = 0;                    //fault protection
        bitslast = 8;

        readtree();         //read headers and get freqs
        buildtree();        //build tree

       lengthSource = 3 * widthImage * heightImage;
        while (lengthSource > 0)
       {
                *pdest++ = readsymb();
                lengthSource--;
        }

        usize = int(pdest - dest);
}

       /*       get freqs       */
 for (int k = 0; k < symnum; k++) {
       psheader = (PSYMBHDR)psour;
       temptree[k].freq = psheader->freq;
       temptree[k].s1 = psheader->symb;

       psour += 5;                }


/*         build htree from temptree                */
void Huffman::buildtree()
{
 /*         special case symnum = 1   */
    if (symnum == 1)
       {
       hsymbol[psour[0]].size = 1;
                    return;
        }

  /*        build 1 & 2 trees        */
for (int i = symnum - 2; i >= 0; i--) {
         if (temptree[i].parent) {
```

```
            htree[i].pleft = temptree[i].parent;
            htree[i].pleft->parent = &htree[i];
          } else
            htree[i].s1 = temptree[i].s1;
          if (temptree[i+1].parent) {
            htree[i].pright = temptree[i+1].parent;
            htree[i].pright->parent = &htree[i];
          } else
            htree[i].s2 = temptree[i+1].s1;
        if (i) {
            temptree[i].freq += temptree[i+1].freq;
            temptree[i].parent = &htree[i];
            temptree[i+1].freq = 0;

            sort ()
                }
        }
}

/*              get bits from sour                */
inline int Huffman::getnextbit()
{
       if (bitslast)
       {
             return 0x1 & (*psour >> (--bitslast));
       }
       else
       {
             psour++;
             bitslast = 8;
             return 0x1 & (*psour >> (--bitslast));
    }
}
/*         decode huff symbol by bit from sour        */
unsigned char Huffman::readsymb()
{
    PHTREE node = &htree[0];
        while (1)
        {
          if (getnextbit())       //next right node
            {
                if (node->pright)
                    node = node->pright;
                else
                    return node->s2;
             }
      else                  //next left node
```

```
        {
          if (node->pleft)
                  node = node->pleft;
          else

              return node->s1;
        }
    }
}
```

***SourceCode* 4.10.** Huffman *Decoder*

Output dari fungsi yang ditunjukkan Gambar 4.10 akan diisikan ke dalam array untuk masing-masing elemen RGB

4.2.3.2 Dekuantisasi Vektor

Array hasil dari proses dekode huffman merupakan vektor input unuk proses dekuantisasi. Pada tahap ini proses dekuantisasi akan mengganti nilai vektor input yang berupa indek dengan koefisien codeword. Potongan fungsi untuk melakukan proses dekuantisasi vektor ditunjukkan pada *SourceCode* 4.11.

```
void prosesDekuantisasi(array<double,2>^ codebook,
array<double,1>^ vektor, array<int,2>^ ut, int RGB, int
width, int height,int startbaris, int startkolom, int
endbaris, int endkolom)
{
  int indexVektor = 0;
  bool stop = false;

  int condition; // 0 = PCU, 1 = FCU, 2 = no update

      if ((ut[RGB,0] == 0) && (ut[RGB,1] == -1))
            condition = 1;
      else if ((ut[RGB,0] == 1) && (ut[RGB,1] == -1))
            condition = 2;
      else if ((ut[RGB,1] != -1))
            condition = 0;
```

```
if (condition == 0) // PCU
{
  for (int i = startbaris; i <= endbaris; i++)
     {
       for (int j = 0; j < width; j ++)
          {
            if ((i == startbaris && j >= startkolom) ||
                (i > startbaris && i < endbaris) ||
                (i == endbaris && j <= endkolom))
              {
                if (ut[RGB,indexVektor] == 1)
                  {
                    vektor[indexVektor] = codebook[j,i];
                  }
                    indexVektor++;
              }

             if (i == endbaris && j == endkolom)
              {
                stop = true;
                break;
              }
            }
              if (stop == true)
                break;
       }
}
else if (condition == 2)
{
  for (int i = startbaris; i <= endbaris; i++)
     {
       for (int j = 0; j < width; j ++)
         {
           if ((i == startbaris && j >= startkolom) ||
                   (i > startbaris && i < endbaris) ||
                     (i == endbaris && j <= endkolom))
              { vektor[indexVektor] = codebook[j,i];
                indexVektor++;}

            if (i == endbaris && j == endkolom)
              {stop = true;
                break;}
           }
              if (stop == true)break;
       }
```

***SourceCode* 4.11.** Dekuantisasi Vektor

Fungsi ini akan menghasilkan matrik elemen warna RGB, namun nilai matrik-matirk tersebut masih ada dalam *range* nilai *byte* (0 – 255) oleh karena itu sebelum dilakukan proses selanjutnya yaitu transformasi balik maka perlu dikonversi kembali menjadi range nilai *double*. Kode yang digunakan untuk mengkonversi ke nilai *double* ditunjukkan pada *SourceCode* 4.12.

```
private:
System::Void Form1::convertToDouble(array<double,2>^
matriks,array<double,1>^ min, array<double,1>^ max, int
indexSubband,int x, int y, int width, int height)
{
  double maxValues = max[indexSubband];
  double minValues = min[indexSubband];

  if ((minValues < 0))
  {
    maxValues = maxValues - minValues;
  }
  for (int i = y; i < height; i++)
  {
    for (int j = x; j < width; j++)
       {
         matriks[j,i] = matriks[j,i] / 255 * maxValues +
                        minValues;
       }
   }
}
```

***SourceCode* 4.12.** Konversi ke *Double*

Dengan menggunakan informasi nilai minimum dan maksimum yang disimpan pada *header file* maka proses konversi ke double dapat menghasilkan nilai yang sebenarnya untuk proses transformasi balik.

4.2.3.3 *Lifting-based Invers Wavelet* 9/7 *Transform*

Tahap selanjutnya akan dilakukan proses transformasi balik seperti yang telah dijelaskan pada subbab 3.2.3.3. Fungsi untuk melakukan IWT 9/7 ditunjukkan pada *SourceCode* 4.13.

```
void iwt97(array<double,2>^ x, int height, int width)
{
  double a;
  int i,j;

#pragma region operasi balik kolom

// Unpack
  tempbank = gcnew array<double,2>(width,height);
  for (i=0; i<height; i++)
  {
    for (j=0; j<width/2;j++)
       {
          tempbank[j*2,i] = x[j,i];
          tempbank[j*2+1,i] = x[j+width/2,i];
       }

  }

  for (i=0; i<width; i++)
  {
    for (j=0; j<height; j++)
       {
          x[i,j]=tempbank[i,j];
       }
  }

// Undo scale
a=1.149604398;
for (i=0; i<height; i++)
{
  for (j=0; j<width; j++)
     {
       if (j%2) x->SetValue((Double)(x->GetValue(j,i))*
                    a,j,i);
       else  x->SetValue((Double)(x->GetValue(j,i)) /
                    a,j,i);
     }
 }

// Undo update 2
a=-0.4435068522;
for (i=0; i<height; i++)
{
  for (j=2; j<width; j+=2)
  {
     x->SetValue((Double)(x->GetValue(j,i)) + a *
```

```
            ((Double)(x->GetValue(j-1,i)) + (Double)
                        (x->GetValue(j+1,i))),j,i);
        }
            x->SetValue((Double)(x->GetValue(0,i))+ 2 * a
                    * (Double)(x->GetValue(1,i)),0,i);

}
// Undo predict 2
a=-0.8829110762;
for (i=0; i<height; i++)
{
  for (j=1; j<width-2; j+=2)
      {
        x->SetValue((Double)(x->GetValue(j,i))+ a *
                    ((Double)(x->GetValue(j-1,i))+
                    (Double)(x->GetValue(j+1,i))),j,i);
      }
       x->SetValue((Double)(x->GetValue(width-1,i))+ 2 *
        a * (Double)(x->GetValue(width-2,i)),width-1,i);

}
// Undo update 1
a=0.05298011854;
for (i=0; i<height; i++)
{
  for (j=2; j<width; j+=2)
      {
        x->SetValue((Double)(x->GetValue(j,i)) + a *
                    ((Double)(x->GetValue(j-1,i)) +
                    (Double)(x->GetValue(j+1,i))),j,i);
      }
        x->SetValue((Double)(x->GetValue(0,i))+ 2 *
                a * (Double)(x->GetValue(1,i)),0,i);
      }

// Undo predict 1
a=1.586134342;
for (i=0; i<height; i++)
{
  for (j=1; j<width-2; j+=2)
      {
        x->SetValue((Double)(x->GetValue(j,i)) + a *
                    ((Double)(x->GetValue(j-1,i)) +
                    (Double)(x->GetValue(j+1,i))),j,i);
      }
        x->SetValue((Double)(x->GetValue(width-1,i))+ 2 *
        a * (Double)(x->GetValue(width-2,i)),width-1,i);}
```

```
#pragma endregion operasi balik kolom

#pragma region operasi balik baris

// Unpack
tempbank = gcnew array<double,2>(width,height);
for (i=0; i<width; i++)
{
  for (j=0; j<height/2;j++)
      {
        tempbank[i,j*2] = x[i,j];
       tempbank[i,j*2+1] = x[i,j+height/2];
      }
}

for (i=0; i<width; i++)
{
  for (j=0; j<height; j++)
      {
        x[i,j]=tempbank[i,j];
      }
}

// Undo scale
a=1.149604398;
for (i=0; i<width; i++)
{
  for (j=0; j<height; j++)
      {
        if (j%2) x->SetValue((Double)(x->GetValue(i,j))*
                                  a,i,j);
        else  x->SetValue((Double)(x->GetValue(i,j)) /
                                  a,i,j);
      }
}

// Undo update 2
a=-0.4435068522;
for (i=0; i<width; i++)
{
  for (j=2; j<height; j+=2)
      {
        x->SetValue((Double)(x->GetValue(i,j)) + a *
                    ((Double)(x->GetValue(i,j-1)) +
                    (Double)(x->GetValue(i,j+1))),i,j);}
```

```
            x->SetValue((Double)(x->GetValue(i,0))+ 2 *
                        a  *  (Double)(x->GetValue(i,1)),i,0);

          }
// Undo predict 2
a=-0.8829110762;
for (i=0; i<width; i++)
{
  for (j=1; j<height-2; j+=2)
     {
       x->SetValue((Double)(x->GetValue(i,j))+ a *
                   ((Double)(x->GetValue(i,j-1))+
                   (Double)(x->GetValue(i,j+1))),i,j);
      }
       x->SetValue((Double)(x->GetValue(i,height-1))+ 2
    * a * (Double)(x->GetValue(i,height-2)),i,height-1);

}
// Undo update 1
a=0.05298011854;
for (i=0; i<width; i++)
{
  for (j=2; j<height; j+=2)
     {
       x->SetValue((Double)(x->GetValue(i,j)) + a *
                   ((Double)(x->GetValue(i,j-1)) +
                   (Double)(x->GetValue(i,j+1))),i,j);

      }
       x->SetValue((Double)(x->GetValue(i,0))+ 2 * a *
                   (Double)(x->GetValue(i,1)),i,0);

}
// Undo predict 1
a=1.586134342;
for (i=0; i<width; i++)
{
  for (j=1; j<height-2; j+=2)
     {
       x->SetValue((Double)(x->GetValue(i,j)) + a *
                  ((Double)(x->GetValue(i,j-1)) +
                  (Double)(x->GetValue(i,j+1))),i,j);

     }
```

```
        x->SetValue((Double)(x->GetValue(i,height-1))+ 2 *
        a * (Double)(x->GetValue(i,height-2)),i,height-1);
    }
#pragma endregion operasi balik baris
}
```

***SourceCode* 4.13.** IWT 9/7

Fungsi ini membutuhkan masukkan berupa matrik elemen RGB dan informasi dimensi matrik (*width* dan *height*). Fungsi menghasilkan matrik elemen RGB yang telah ditransformasi balik.

4.2.3.4 *Deblocking* Citra

Tahap akhir proses dekompresi adalah proses menggabungkan elemen warna RGB sehingga merepresentasikan nilai pixel pada citra. Proses ini membutuhkan inputan berupa 3 matrik elemen RGB. Proses ini menghasilkan nilai RGB yang akan langsung ditampilkan sebagai citra rekonstruksi. Fungsi untuk menggabungkan elemen warna RGB pada tiap pixel ditunjukkan pada *SourceCode* 4.14.

```
Bitmap^imageOutput=gcnew Bitmap(widthImage,heightImage);

Point size = PixelSize(imageOutput);

LockBitmap(imageOutput);

for (int y = 0; y < size.Y; y++)
{
  PixelData* pPixel = PixelAt(0, y);
  for (int x = 0; x < size.X; x++)
  {
    int valRed = (int)Math::Round(Red[x,y]);
    int valGreen = (int)Math::Round(Green[x,y]);
    int valBlue = (int)Math::Round(Blue[x,y]);
    if (valRed > 255)
        valRed = 255;
    else if (valRed < 0)
        valRed = 0;
    if (valGreen > 255)
        valGreen = 255;
    else if (valGreen < 0)
        valGreen = 0;
            if (valBlue > 255)
```

```
                valBlue = 255;
            else if (valBlue < 0)
                valBlue = 0;
            pPixel->red = valRed;
            pPixel->green = valGreen;
            pPixel->blue = valBlue;
            pPixel++;
        }
}

UnlockBitmap(imageOutput);
pictureBoxCitraDecompress->Image = imageOutput;
```

***SourceCode* 4.14.** fungsi PixelAt / SetPixel

4.2.4 Implementasi Rasio Kompresi

Proses rasio digunakan untuk melakukan pengujian rasio. Untuk itu, proses ini membutuhkan inputan berupa ukuran file kompresi *compressedSize* dan ukuran citra asli *uncompressedSize*. Proses ini menghasilkan nilai kompresi rasio beserta prosentasenya. Kode untuk menghitung rasio kompresi ditunjukkan pada *SourceCode* 4.15.

```
fil = gcnew FileInfo(citraAsliPath);
uncompressedSize = (int)fil->Length;

double rasio = Math::Round(1.0 -
                  (double)compressedSize /
                  (double)uncompressedSize, 4);
textBoxRasio->Text = rasio.ToString();

double prosentaseRasio = rasio * 100;
tmp=String::Format("{0}%",Math::Round(prosentaseRasio));
textBoxProsentase->Text = tmp;
```

***SourceCode* 4.15.** Rasio Kompresi

4.2.5 Implementasi MSE

Proses MSE digunakan untuk menghitung tingkat kesalahan antara citra asli dengan citra rekonstruksi. Oleh karena itu, proses ini membutuhkan inputan berupa dimensi citra, citra asli dan citra

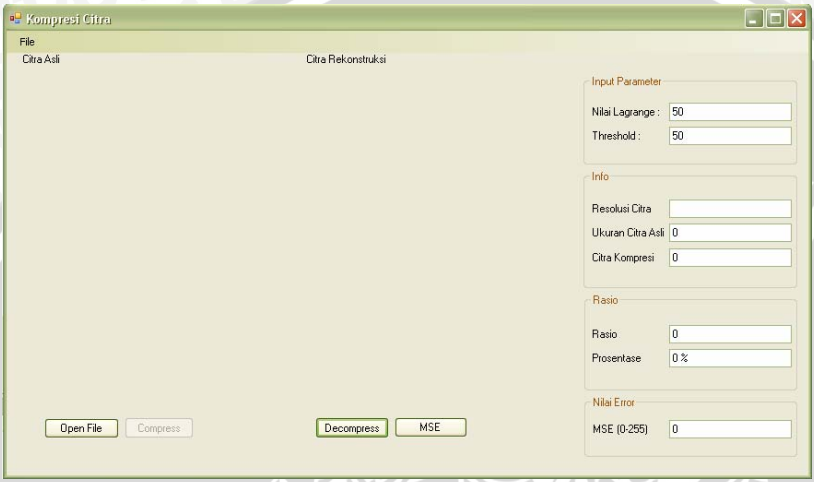
rekonstruksi. Proses ini menghasilkan nilai MSE. Fungsi untuk melakukan proses MSE ditunjukkan pada *SourceCode* 4.16.

```
private:
System::Void buttonMSE_Click(System::Object^  sender,
System::EventArgs^  e)
{
  if (pictureBoxCitraAsli->Image == nullptr ||
      pictureBoxCitraDecompress == nullptr)
      return;
Bitmap^ citraAsli =
dynamic_cast<Bitmap^>(pictureBoxCitraAsli->Image);
Bitmap^ citraRekonstruksi =
dynamic_cast<Bitmap^>(pictureBoxCitraDecompress->Image);

if (citraAsli->Width != citraRekonstruksi->Width ||
    citraAsli->Height != citraRekonstruksi->Height)
    return;

double totalR = 0.0, totalG = 0.0, totalB = 0.0;
for (int i=0; i<citraRekonstruksi->Height; i++)
{
  for (int j=0; j<citraRekonstruksi->Width; j++)
     {
       System::Drawing::Color^ colorAsli =
                        citraAsli->GetPixel(j,i);
       System::Drawing::Color^ colorRekonstruksi =
                       citraRekonstruksi->GetPixel(j,i);

        totalR += Math::Pow((double) colorAsli->R –
                        (double) colorRekonstruksi->R,2);
       totalG += Math::Pow((double) colorAsli->G –
                        (double) colorRekonstruksi->G,2);
       totalB += Math::Pow((double) colorAsli->B –
                        (double) colorRekonstruksi->B,2);
     }
}

double MSE = (totalR + totalG + totalB) / 3;
MSE = MSE / (citraAsli->Width * citraRekonstruksi-
>Height);

textBoxMSE->Text = MSE.ToString();
```

***SourceCode* 4.16.** MSE

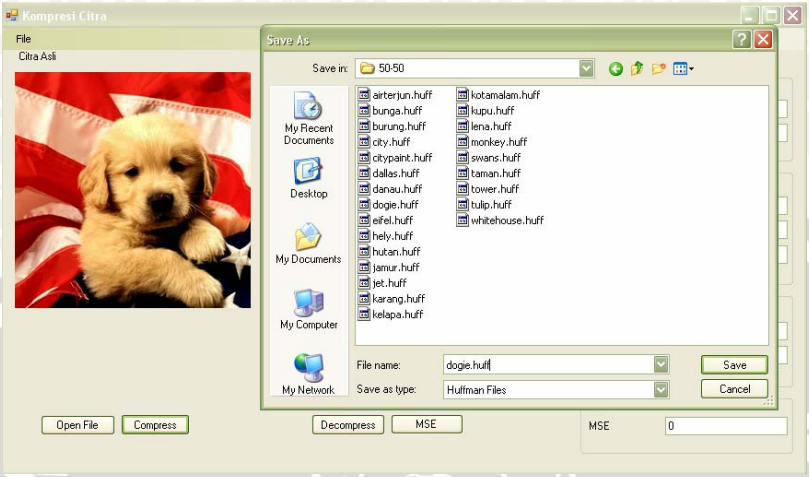
4.3. Implementasi Antarmuka (*interface*)

Tampilan utama dari aplikasi kompresi citra berdasarkan rancangan antarmuka pada subbab 3.5 ditunjukkan pada Gambar 4.1.



Gambar 4.1 Tampilan Utama

Pada proses kompresi, *user* perlu menginputkan citra inputan, nilai parameter lagrange dan nilai *threshold*. Tampilan pada saat proses kompresi ditunjukkan pada Gambar 4.2. Untuk proses dekompresi akan muncul citra rekonstruksi dan untuk mengetahui nilai error (MSE) dari citra rekonstruksi maka *user* perlu juga membuka citra asli setelah itu tekan tombol MSE. Tampilan proses dekompresi ditunjukkan pada Gambar 4.3.



Gambar 4.2 Tampilan Proses Kompresi

Setelah proses kompresi selesai akan muncul jendela dialog untuk menamakan file hasil kompresi beserta lokasi direktori penyimpanannya.

Gambar 4.3 Tampilan Proses Dekompresi

4.4 Analisa Hasil

4.4.1 Analisa Hasil terhadap Rasio Kompresi

Nilai rasio kompresi merupakan besarnya ukuran citra yang dapat dikompresi. Dengan kondisi yang divariasikan, kemudian hasil pengujian dapat dilihat pada tabel 4.2. Data mengenai hasil lengkap uji rasio untuk 18 citra uji dapat dilihat pada lampiran I.

Tabel 4.2. Hasil pengujian tehadap rasio kompresi

Percobaan ke-	Keterangan	Rasio (dalam persen)		
		Min	Max	Rata-rata
1.	λ = 50, T = 100	18,76	63,8	49,68
2.	λ = 100, T = 100	41,85	67,06	62,42
3.	λ = 150, T = 100	54,57	70,04	65,77

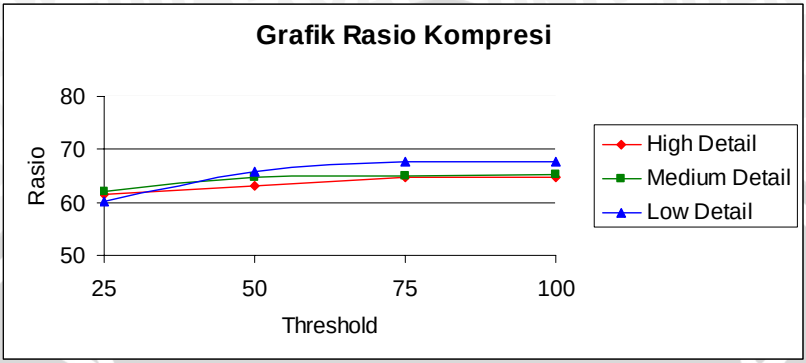
Semakin tinggi nilai rasio kompresi maka ukuran file citra akan semakin kecil. Dari tiga pengujian yang telah dilakukan dengan beberapa kombinasi parameter, maka dapat dilihat bahwa rata-rata rasio kompresi paling tinggi didapatkan saat pengujian ke-3, yaitu pengujian dengan nilai 150 untuk *lagrange* λ dan 100 untuk ambang batas pergantian *codeword* (*threshold* T). Rasio rata-rata untuk semua pengujian yang telah dilakukan yaitu 57,83 %.

4.4.2 Analisa Perubahan *Threshold* terhadap Rasio Kompresi

Nilai *threshold* adalah inputan *user*. Pada pengujian kali ini akan dilihat bagaimana pengaruh pengubahan nilai *threshold* terhadap ratio kompresi yang didapatkan. Data nilai rasio kompresi yang disajikan pada tabel 4.3 diambil dari pengujian ke-3. Grafik yang didapatkan dari pengujian tersebut dapat dilihat pada Gambar 4.4.

Tabel 4.3. Hasil Pengujian Perubahan *Threshold* terhadap Rasio Kompresi

Jenis Detail	Citra Uji	25	50	75	100
High detail	Dogie.bmp	61,48	63,09	64,67	64,67
Medium detail	City paint.bmp	62,12	64,68	64,94	65,2
Low detail	Kota_mlm.bmp	60,21	65,85	67,7	67,7



Gambar 4.4 Grafik perubahan Threshold terhadap Rasio Kompresi

Dari gambar 4.4 dapat dilihat bahwa ratio kompresi dapat terpengaruh oleh pengubahan *threshold*. Meskipun perubahan yang terlihat tidak nampak signifikan, tapi dari tabel 4.3 dapat diketahui bahwa ada perubahan di sana.

Dari pengujian-pengujian yang telah dilakukan, maka dapat diketahui bahwa ada pengaruh perubahan nilai *threshold* terhadap ratio kompresi. Semakin tinggi nilai *threshold,* maka semakin tinggi pula ratio kompresinya. Hal ini dapat disebabkan oleh *threshold* yang tinggi menyebabkan kecilnya peluang proses *update-codeword,* sehingga akan ada beberapa indeks yang sama di beberapa bagian citra yang menghasilkan nilai bit yang kecil pula pada saat *encoding* Huffman.

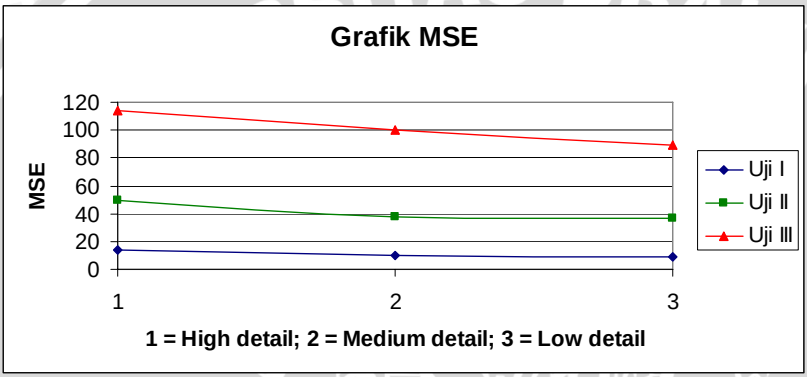
Berdasarkan data hasil penelitian, nilai optimal *threshold* berada pada nilai 75. Hampir semua citra uji yang menggunakan nilai *threshold* diatas 75 (T = 100) menghasilkan nilai rasio kompresi yang sama ketika menggunakan nilai *threshold* 75.

4.4.3 Analisis MSE terhadap Jenis Citra Uji

Setelah beberapa kali mengamati dan menghitung MSE yang dihasilkan dari setiap citra yang diuji maka ada hubungan antara jenis detail citra dan naik turunnya nilai MSE. Untuk itu dilakukan analisis MSE terhadap jenis citra uji yang meliputi *high detail*, *medium*

detail, dan *low detail*. Data mengenai hasil lengkap uji MSE untuk 18 citra uji dapat dilihat pada lampiran I.

Data nilai MSE yang disajikan dalam bentuk grafik seperti ditunjukkan pada Gambar 4.5 adalah rata-rata nilai MSE untuk masing-masing jenis *detail* citra, yaitu no.1-3 secara berurutan adalah *high detail*, *medium detail, low detail* pada pengujian pertama, kedua, dan ketiga dengan kondisi-kondisi yang telah dipaparkan di subbab 3.3.



Gambar 4.5 Grafik nilai MSE Citra Uji

Pada gambar 4.5, terlihat bahwa rata-rata nilai MSE citra tersebar di antara nilai 9,00 – 113,92. Didapatkan rata-rata nilai MSE untuk citra *high detail*, *medium detail*, dan *low detail* pada pengujian pertama, yaitu 14,34; 9,89; dan 9,35. Rata-rata nilai MSE untuk citra *high detail*, *medium detail*, dan *low detail* pada pengujian kedua, yaitu 49,78; 37,76; dan 37,03. Rata-rata nilai MSE untuk citra *high detail*, *medium detail*, dan *low detail* pada pengujian ketiga, yaitu 113,92; 99,97; 88,78 Hal ini menunjukkan dalam rata-rata nilai MSE citra *low detail* untuk keseluruhan pengujian memiliki rata-rata nilai MSE paling rendah berada diantara nilai 9,00 sampai 88,78 dibandingkan citra *high detail* dan *medium detail*.

Hal ini dapat disebabkan oleh proses kuantisasi yang dialami citra. Maksudnya ketika proses pengindekan, ada kemungkinan tingkat error citra yang lebih tinggi untuk citra *high detail* dan *medium detail* tetapi ternyata *codeword* tersebut dipilih untuk

mempresentasikan vektor input. Akumulasi error citra di beberapa titik (x,y) menyebabkan nilai MSE yang lebih tinggi untuk citra *high detail* dan *medium detail*. Namun untuk citra *low detail*, *codeword* yang terpilih untuk mempresentasikan vektor input memiliki tingkat error yang kecil. Sehingga dapat dikatakan bahwa jenis detail citra dapat mempengaruhi tingkat MSE yang dihasilkan citra dekompresi.

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Berdasarkan penelitian yang telah dilakukan, kesimpulan yang diperoleh antara lain:

1. Telah dirancang aplikasi kompresi citra digital dengan cara memisahkan elemen warna RGB pada citra input berekstensi .bmp lalu masing-masing elemen warna tersebut ditransformasi dengan transformasi *W*avelet 9/7 yang kemudian akan dikuantisasi dengan metode Kuantisasi Vektor Adaptif setelah itu hasil kuantisasi akan dimampatkan dengan metode pemampatan Huffman sehingga akan terbentuk file citra kompresi berekstensi .huff
2. Berdasarkan pengujian yang telah dilakukan, didapatkan nilai rata-rata rasio kompresi paling tinggi didapatkan saat pengujian ke-3, yaitu pengujian dengan nilai 150 untuk *lagrange* dan 100 untuk ambang batas pergantian *codeword*. Nilai optimal *threshold* berada pada nilai 75. Hampir semua citra uji yang menggunakan nilai *threshold* diatas 75 (T = 100) menghasilkan nilai rasio kompresi yang sama ketika menggunakan nilai *threshold* 75. Rata-rata rasio untuk semua pengujian yang telah dilakukan yaitu 57,83 %.
3. Pada pengujian tingkat kesalahan yang diukur menggunakan nilai MSE, didapatkan nilai rata-rata MSE citra *low detail* untuk keseluruhan pengujian memiliki nilai rata-rata MSE paling rendah berada diantara nilai 9,00 sampai 88,78 dibandingkan citra *high detail* dan *medium detail* yang berada diantara nilai 9,00 sampai 113,92.

5.2 Saran

Saran yang diberikan untuk mengembangkan penelitian ini antara lain:

1. Dapat dicoba transformasi citra menggunakan metode wavelet yang lain dan membandingkannya. Misalnya Wavelet 5/3.
2. Dapat dicoba proses dekomposisi citra hingga level terkecil.

3. Sebaiknya menerapkan algoritma optimasi *codebook* untuk proses kuantisasi vektor adaptif, seperti *pairwise Nearest Neighbor* (PNN), *Simulated Annealing, Maximum Descent* (MD), dan *Frequency-Sensitive Competitive Learning* (FSCL).
4. Menggunakan metode kompresi lossless yang lain. Misalnya RLE, Shannon, LZW.

DAFTAR PUSTAKA

Acharya, Tinku. 2005. *JPEG 2000 Standard for Image Compression*. A John Wiley & Sons, Inc., Publication : New York

Achmad, Balza. 2004. *Teknik Pengolahan Citra Digital Menggunakan Delphi.* Ardi Publishing : Yogyakarta.

Anonym. 2006. *Multimedia*. Fakultas Teknik Informatika Universitas Kristen Duta Wacana

Babu, Jagadish. 2008. *Rate Distortion Theory*. Aplied Physics and Electronics Department : Umea University

Daubechies, Ingrid. 1996. *Factoring Wavelet Transforms Into Lifting Step.* J. Fourier Anal. Appl., vol. 4, no. 3.

Guo, Kai. 2007. *A New Partial Codeword Updating Scheme Based On Rate-Distortion Optimization For Adaptive Vector Quantization*. Department of Electronic Engineering : City University of Hong Kong.

Kotteri, Kishore. 2004. *Optimal, Multiplierless Implementations of the Discrete Wavelet Transform for Image Compression Applications*. Virginia Polytechnic Institute and State University : USA

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

Rachmawaty, Rahadiany. 2007. *Kompresi Citra Digital Menggunakan Transformasi Wavelet dan Metode Kuantisasi Vektor Adaptif* . Jurusan Teknik Informatika Sekolah Tinggi Teknologi Telkom : Bandung.

Walpole, Ronald E. 1995. *Pengantar Statistika Edisi Ketiga*. Gramedia Pustaka Utama : Jakarta.

Wickerhauser,M.Victor.2002. *A Survey of Wavelet Algorithms and Applications*. Washington University : USA http://www.math.wustl.edu/~victor Diakses pada tanggal 18 Maret 2008

Lampiran I. Data Hasil Pengujian

No.	Nama Citra	Ratio Kompresi (%) λ = 50			
		T = 25	T = 50	T = 75	T = 100
1	Kupu	51.77	55.07	55.33	55.33
2	Hely	62.25	62.25	62.65	62.65
3	Dogie	53.98	54.65	55.95	55.95
4	Monkey	55.05	62.94	63.46	63.46
5	Air terjun	26.8	27.98	28.5	28.5
6	Bunga	22.61	25.79	26.57	26.57
7	Danau	51.7	55.94	56.2	56.2
8	City	18.9	19.25	19.95	19.95
9	City paint	53.9	55.74	56	56.26
10	Taman	36.7	40.29	41.33	41.33
11	Tower	63.17	63.25	63.51	63.51
12	Lena	59.44	63.64	63.64	63.64
13	Hutan	18.63	18.76	18.76	18.76
14	Jet	63.45	63.46	63.72	63.72
15	Tulip	60.63	63.8	63.8	63.8
16	Kota malam	47.25	51.19	51.45	51.45
17	Eiffel	47.49	51.15	51.15	51.15
18	Burung	51.41	51.7	51.96	51.96
Rata-rata		46.95	49.27	49.66	49.68

No.	Nama Citra	Ratio Kompresi (%) λ = 100			
		T = 25	T = 50	T = 75	T = 100
1	Kupu	61.19	65.45	64.52	64.52
2	Hely	65.26	65.26	65.26	65.26
3	Dogie	59.33	61.78	63.35	63.35
4	Monkey	56.09	64.35	64.88	64.88
5	Air terjun	51.49	58.91	62.09	62.09
6	Bunga	52	59.88	63.28	63.28
7	Danau	57.49	63.83	64.35	64.35

8	City	33.59	41.59	41.85	41.85
9	City paint	61.5	64.68	64.94	65.2
10	Taman	57.84	62.87	66.8	67.06
11	Tower	63.17	64.27	64.79	64.79
12	Lena	62.17	64.84	64.84	64.84
13	Hutan	41.64	43.82	46.43	46.43
14	Jet	64.75	64.76	65.02	65.02
15	Tulip	61.96	65.17	65.17	65.17
16	Kota malam	60.21	63.64	64.95	64.95
17	Eiffel	57.8	63.58	64.92	64.92
18	Burung	61.08	63.02	65.57	65.57
Rata-rata		57.14	61.21	62.39	62.42

No.	Nama Citra	Ratio Kompresi (%) $\lambda = 150$			
		T = 25	T = 50	T = 75	T = 100
1	Kupu	61.19	66.89	67.63	67.63
2	Hely	67.09	67.41	68.48	68.48
3	Dogie	61.48	63.09	64.67	64.67
4	Monkey	57.33	64.35	64.88	64.88
5	Air terjun	52.73	59.98	62.67	62.09
6	Bunga	52.83	60.94	63.28	63.28
7	Danau	57.58	63.83	64.35	64.35
8	City	50.21	52.22	54.57	54.57
9	City paint	62.12	64.68	64.94	65.2
10	Taman	57.84	65.14	69.21	70.04
11	Tower	63.59	64.27	64.79	64.79
12	Lena	62.17	67.7	67.7	67.7
13	Hutan	54.29	62.66	67.69	67.69
14	Jet	66.29	67.17	67.7	67.7
15	Tulip	63.61	67.7	67.7	67.7
16	Kota malam	60.21	65.85	67.7	67.7
17	Eiffel	59.47	65.77	67.7	67.7
18	Burung	62.66	64.79	67.7	67.7
Rata-rata		59.59	64.14	65.74	65.77

Rata-rata Rasio Kompresi seluruh pengujian = (rata-rata Rasio Kompresi pengujian ke-1 + rata-rata Rasio Kompresi pengujian ke-2 + rata-rata Rasio Kompresi pengujian ke-3) / 12 = 57,83 %

No.	Nama Citra	MSE λ = 50				
		T = 25	T = 50	T = 75	T = 100	Rata-rata
1.	Kupu	19.67	13.1	13.15	13.15	14.77
2.	Hely	2.43	2.43	2.43	2.43	2.43
3.	Dogie	15.83	14.36	14.7	14.7	14.89
4.	Monkey	24.15	27.49	28.15	28.15	26.99
5.	Air terjun	11.47	12.8	13.22	13.22	12.68
6.	Bunga	12.11	14.78	15.12	15.12	14.28
Rata-rata						14.34
7.	Danau	9.24	7.29	7.34	7.34	7.80
8.	City	11.12	11.38	11.38	11.38	11.31
9.	City paint	9.06	10.87	11.11	11.15	10.55
10.	Taman	18.78	12.12	12.12	12.12	13.79
11.	Tower	11.15	9.28	9.32	9.32	9.77
12.	Lena	6.03	6.16	6.16	6.16	6.13
Rata-rata						9.89
13.	Hutan	14.15	13.28	13.28	13.28	13.49
14.	Jet	4.94	5.1	5.13	5.13	5.08
15.	Tulip	8.48	5.77	5.77	5.77	6.45
16.	Kota malam	10.94	11.66	11.83	11.83	11.57
17.	Eiffel	12.2	10.74	10.74	10.74	11.11
18.	Burung	7.95	8.2	8.68	8.68	8.38
Rata-rata						9.35

No.	Nama Citra	MSE λ = 100				
		T = 25	T = 50	T = 75	T = 100	Rata-rata
1.	Kupu	95.15	112.58	29.95	29.95	66.91
2.	Hely	40.86	40.86	40.86	40.86	40.86
3.	Dogie	16.7	20.1	21.36	21.36	19.88

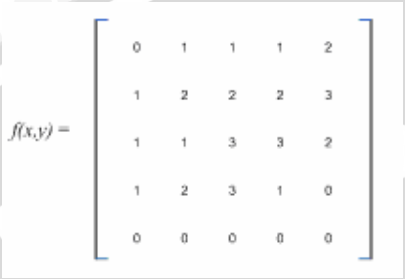
4.	Monkey	56.27	72.91	73.57	73.57	69.08
5.	Air terjun	60.19	55.74	20.18	20.18	39.07
6.	Bunga	87.4	54.21	54.93	54.93	62.87
Rata-rata						49.78
7.	Danau	9.78	12.25	12.48	12.48	11.75
8.	City	40.19	48.1	48.12	48.12	46.13
9.	City paint	42.71	44.98	45.23	45.26	44.55
10.	Taman	80.12	69.74	71.64	71.84	73.34
11.	Tower	11.15	14.01	14.25	14.25	13.43
12.	Lena	62.29	29.06	29.06	29.06	37.37
Rata-rata						37.76
13.	Hutan	59.03	61.84	62.29	62.25	61.35
14.	Jet	32.22	32.38	32.41	32.41	32.36
15.	Tulip	25.26	25.55	25.55	25.55	25.48
16.	Kota malam	25.42	28.7	28.88	28.88	27.97
17.	Eiffel	33.09	38.52	39.62	39.62	37.71
18.	Burung	36.32	37.3	37.85	37.85	37.33
Rata-rata						37.03

No.	Nama Citra	MSE $\lambda = 150$				
		T = 25	T = 50	T = 75	T = 100	Rata-rata
1.	Kupu	95.15	211.97	188.68	188.68	171.12
2.	Hely	101.88	164.94	168.89	168.89	151.15
3.	Dogie	110.84	96.12	97.38	97.38	100.43
4.	Monkey	96.6	72.91	73.57	73.57	79.16
5.	Air terjun	96.96	126.17	107.17	20.18	87.62
6.	Bunga	116.75	149.62	54.93	54.93	94.058
Rata-rata						113.92
7.	Danau	35.68	42.25	42.48	42.48	40.72
8.	City	55.94	65.38	65.6	65.6	63.13
9.	City paint	63.04	64.98	65.23	65.26	64.63
10.	Taman	132.12	144.21	164.91	175.13	154.09
11.	Tower	151.7	154.01	154.25	154.25	153.55
12.	Lena	120.29	124.79	124.79	124.79	123.67

Rata-rata						99.97
13.	Hutan	66.53	76.3	76.91	76.91	74.16
14.	Jet	69.45	138.71	138.74	138.74	121.41
15.	Kota malam	70.42	87.12	88.87	88.87	83.82
16.	Tulip	79.48	101.09	101.09	101.09	95.69
17.	Burung	46.72	49.32	50.22	50.22	49.12
18.	Eiffel	85.91	113.66	117.15	117.15	108.47
Rata-rata						88.78

Lampiran II. Contoh pembentukan Kode Huffman

Misalnya pemakaian pemampatan Huffman menggunakan matrik yang ditunjukkan pada gambar sebagai berikut:



Gambar Matrik Pemampatan Huffman 5 x 5
Sumber : pengamatan

Pada dasarnya, prosedur pembuatan kode Huffman pada citra digital terdiri dari 3 proses utama, yaitu :

1. Pembentukan tabel distribusi frekuensi sesuai dengan frekuensi kemunculan dari masing-masing level warna pembentuk citra,ditunjukkan pada tabel dibawah ini

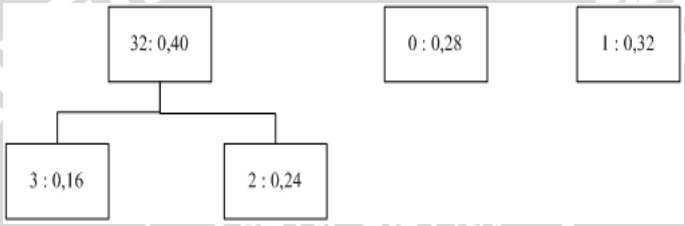
Tabel distribusi frekuensi Derajat Intensitas

Derajat Intensitas (k)	Jumlah pixel (n)	Peluang (n/total pixel)
0	7	0.28
1	8	0.32
2	6	0.24
3	4	0.16

2. Pembentukan pohon Huffman berdasarkan distribusi frekuensi yang dihasilkan.
   Langkah-langkahnya yaitu:
   a. Urutkan data berdasarkan peluang terkecil sampai terbesar, seperti yang ditunjukkan pada berikut ini:

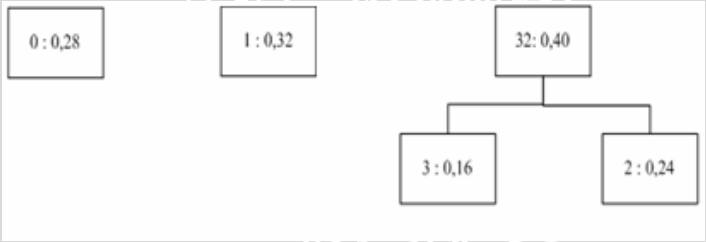
Gambar data peluang terkecil sampai peluang terbesar
Sumber : pengamatan

b. Gabungkan dua data terkecil dan jumlahkan peluangnya. Ditunjukkan pada gambar dibawah ini:



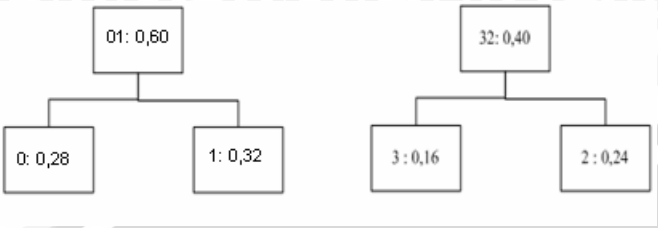
Gambar gabungan dua data terkecil I
Sumber : pengamatan

c. Bandingkan data peluang berikutnya dengan jumlah dua akar terkecil tadi. Bila lebih kecil, pindahkan ke kiri. Bila lebih besar, tidak dipindah. Ditunjukkan pada gambar dibawah ini:



Gambar data kecil dipindahkan ke kiri
Sumber : pengamatan

d. Lalu gabungkan lagi. Ditunjukkan pada gambar sebagai berikut:



Gambar gabungan dua data terkecil II
Sumber : pengamatan

Demikian seterusnya hingga selesai.

3. Pembuatan kode Huffman sesuai dengan alur pada pohon Huffman.
   Langkah-langkahnya yaitu:
   a. Beri kode 0 dan 1 seperti gambar dibawah berikut untuk setiap akar

Gambar pemberian kode 0 dan 1 pada tiap daun
Sumber : pengamatan

   b. Pembentukan kode, sehingga menghasilkan derajat intensitas dengan jumlah paling banyak dikodekan dengan panjang kode lebih pendek, seperti yang ditunjukkan pada table dibawah berikut:

Tabel pembentukan kode

Derajat Intensitas (k)	Kode
0	10
1	11
2	01
3	00

Lampiran III. Hasil penghitungan Standart Deviasi

Jenis Detail Citra	Nama citra	Standart Deviasi
High	kupu.bmp	80,39
	hely.bmp	77,56
	doggie.bmp	76,97
	monkey.bmp	76,14
	air terjun.bmp	71,41
	bunga.bmp	67,43
Medium	danau.bmp	66,37
	city.bmp	61,44
	city_paint.bmp	54,98
	taman.bmp	54,56
	tower.bmp	52,66
	lena.bmp	45,41
Low	hutan.bmp	43,33
	jet.bmp	40,81
	tulip.bmp	38,24
	kota_malam.bmp	36,55
	eiffel.bmp	34,60
	burung.bmp	33,38


UNIVERSITAS BRAWIJAYA