IMPLEMENTASI LIFTING SCHEME DARI WAVELET DAUBECHIES 4 DALAM KOMPRESI CITRA LOSSY

SKRIPSI

oleh:
Ash-Shiddiqul Akbar Hidayat

0510963001-96

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

2010

UNIVERSITAS BRAWIJAYA

IMPLEMENTASI LIFTING SCHEME DARI WAVELET DAUBECHIES 4 DALAM KOMPRESI CITRA LOSSY

SKRIPSI

SEBAGAI SALAH SATU SYARAT UNTUK MEMPEROLEH GELAR SARJANA ILMU KOMPUTER

oleh:
Ash-Shiddiqul Akbar Hidayat

0510963001-96

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

2010

LEMBAR PENGESAHAN SKRIPSI

IMPLEMENTASI LIFTING SCHEME DARI WAVELET DAUBECHIES 4 DALAM KOMPRESI LOSSY

Oleh:
Ash-Shiddiqul Akbar Hidayat
0510963001-96

Setelah dipertahankan di depan Majelis Penguji
Pada tanggal 29 Desember 2010
Dan dinyatakan memenuhi syarat untuk memperoleh
gelar Sarjana dalam bidang Ilmu Komputer

Pembimbing I,	Pembimbing II,
Bayu Rahayudi,ST.,MT	**Dewi Yanti Liliana,S.Kom.,M.Kom**
NIP. 197407122006041001	**NIP. 198111162005012004**

Mengetahui,
Ketua Jurusan Matematika
Fakultas MIPA Universitas Brawijaya

Dr. Agus Suryanto, M.Sc
NIP: 196908071994121001

LEMBAR PERNYATAAN

Saya yang bertandatangan di bawah ini:

Nama	**: Ash-Shiddiqul Akbar Hidayat**
NIM	**: 0510963001**
Jurusan	**: Matematika**
Penulis Tugas Akhir	**: Implementasi Lifting Scheme dari Wavelet Daubechies 4 dalam Kompresi Citra Lossy**

Dengan ini menyatakan:

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

Demikian saya buat pernyataan ini dengan segala kesadaran

Penulis,

Ash-Shiddiqul Akbar Hidayat
NIM : 0510963001

IMPLEMENTASI LIFTING SCHEME DARI WAVELET DAUBECHIES 4 DALAM KOMPRESI LOSSY

ABSTRAK

Pada kompresi citra lossy, terjadi pengurangan atau penghilangan informasi. Pengurangan atau pembuangan informasi ini dapat merusak kualitas citra. Tujuan dari kompresi citra lossy adalah memperbesar tingkat kompresi namun dengan tetap menjaga kualitas citra hasil kompresi.

Terdapat beragam metode untuk mencapai tujuan di atas. Dalam penelitian ini, langkah pertama adalah dengan merubah komponen warna RGB menjadi YCbCr. Kemudian citra mengalami proses transformasi menggunakan metode lifting scheme. Tahapan tersebut diikuti oleh kuantisasi skalar berbasis deadzone. Tahapan akhir adalah Run Length Encoding dan Huffman Encoding. Sementara itu untuk mengukur kualitas citra, digunakan rumus Peak Signal-to-Noise Ratio (PSNR). Semakin tinggi nilai PSNR, semakin baik kualitas citra.

Pada penelitian ini, ditemukan bawha peran metode lifting scheme sangat penting dalam mendekorelasi citra ke dalam bentuk yang lebih mudah untuk dikompresi lebih lanjut. Dalam penelitian ini, diperoleh tingkat kompresi antara 43.94 % hingga 95.5 %. Kisaran nilai PSNR adalah 29.26 hingga 47.32. Untuk tingkat kompresi terbaik dengan menjaga nilai PSNR tetap tinggi diperoleh tingkat kompresi sebesar 94.01 % dengan nilai PSNR sebesar 37.19.

Kata Kunci: *Lifting Scheme*, kompresi citra lossy, *Peak Signal-to-Noise Ratio.*

THE IMPLEMENTATION OF LIFTING SCHEME FROM DAUBECHIES 4 WAVELET ON LOSSY IMAGE COMPRESSION

ABSTRACT

In lossy image compression, there is a lost of information on the compressed image. This lost of information would lead to the degradation of quality from the compressed image. The main objecitve of lossy image compression is to maintain balance between high compression rate and image quality.

There are many method to achieve the above objective. In this research, the first step is to transform the image components from RGB to YCbCr. This is continued by transforming the image with lifting scheme. The image is then quantized by using scalar quantization by applying deadzone. Lastly, the image is processed through entropy coding by using Run Length Encoding and Huffman Encoding. To measure the quality of image, Peak Signal-to-Noise Ratio (PSNR) is used.

From this research, it is found that lifting scheme has an important part by tranforming the image onto a representation which allows a great deal of compression rate achieved by the compression steps afterward. It is found that the compression rate ranges from 43.94% up to 95.5%. The PSNR ranges from 29.26 up to 47.32. The highest compression rate achieved by keeping the PSNR level high is 94.01% where the PSNR value is 37.19.

Keywords: Lifting scheme, lossy image compression, Peak Signal-to-Noise Ratio

KATA PENGANTAR

Segala puji dan syukur penulis panjatkan ke hadirat Allah SWT, yang telah melimpahkan Rahmat, Taufik dan Hidayah-Nya sehingga penulis mampu menyelesaikan skripsi yang berjudul "**Implementasi Lifting Scheme dari Wavelet Daubechies 4 dalam Kompresi Citra Lossy"**. Shalawat serta salam tetap tercurahkan kepada Nabi Muhammad SAW.

Dalam penulisan skripsi ini, banyak dukungan dan bantuan yang penulis terima. Oleh sebab itu, penulis ingin berterima kasih kepada:

1. Bayu Rahayudi, S.T, M.T, selaku pembimbing I atas segala bimbingan, dorongan, nasehat, dan sarannya, serta waktu yang telah diberikan kepada penulis selama penulisan tugas akhir ini.
2. Dewi Yanti Liliana, S.Kom, M.Kom, selaku Pembimbing II atas dorongan, kritik dan sarannya, serta waktu yang telah diberikan kepada penulis selama penulisan tugas akhir ini.
3. Dr. Agus Suryanto, M.Sc selaku Ketua Jurusan Matematika atas semua bantuannya dalam kelancaran penulisan skripsi ini.
4. Drs. Marji, MT selaku Ketua Program Studi Ilmu Komputer atas semua bantuannya dalam kelancaran penulisan skripsi ini.
5. Edy Santoso, S.Si, M.Kom, Nurul Hidayat, S.Pd, M.Sc, dan Muhammad Tanzil Furqon, S.Kom, selaku dosen penguji seminar proposal atas segala saran yang diberikan untuk perbaikan skripsi ini.
6. Drs Marji, MT, Lailil Muflikhah, S.Kom, M.Sc dan Nanang Yudi S, ST selaku dosen penguji ujian sarjana atas kritikan, masukan dan saran yang diberikan demi perbaikan skripsi ini.
7. Bapak dan Ibu Dosen Jurusan Matematika umumnya, dan Program Studi Ilmu Komputer khususnya yang telah memberikan bekal ilmu kepada penulis.
8. Orang tua tercinta atas segala doa, kasih sayang, motivasi, didikan dan bimbingannya yang tak pernah kenal lelah beliau berikan kepada penulis secara tulus dan ikhlas.
9. Kharirotul Latifah atas kesabaran, semangat dan motivasi tiada henti yang diberikan selama penulisan skripsi ini.
10. Teman-teman Ijo Item dimanapun engkau berada yang tidak mengenal kata selesai dalam berhijrah.
11. Teman-teman Ilmu Komputer umumnya dan angkatan 2005

khsususnya yang selalu berjuang untuk meningkatkan kualitas bangsa, semoga semangat untuk maju terus membara.

12. Semua pihak yang telah membantu yang dimana penulis tidak dapat menuliskan satu persatu.

Akhirnya, penulisa hanya mampu berdoa agar Allah SWT menyapa semua pihak di atas atas amalan yang sangat mulai tersebut. Penulis menyadari sepenuhnya bahwa dalam penulisan tugas akhir ini tidak lepas dari kesalahan atau kekurangan. Oleh sebab itu, saran dan kritik yang membangun sangat diharapkan oleh penulis.

Akhir kata, semoga tulisan ini bermanfaat bagi penulis khususnya dan semua pihak pada umumnya.

Daftar Isi

Daftar Gambar

Daftar Tabel

Daftar Grafik

Daftar Sourcecode

BAB I
PENDAHULUAN

1.1 Latar Belakang Masalah

Dewasa ini, penggunaan gambar dalam dunia komputer semakin meningkat. Mulai dari bidang kesehatan, pelayanan publik hingga yang saat ini sangat populer adalah penggunaan gambar pada jejaring sosial di internet.

Tren peningkatan penggunaan gambar ini membawa permasalahan terkait dengan media penyimpanan dan media pengiriman. Gambar-gambar yang diunduh dari internet semisal, memiliki ukuran mulai dari yang kecil, sekitar 50 KB, hingga beberapa gambar dengan resolusi tinggi dengan ukuran hingga melebihi 1 MB. Gambar-gambar yang diperoleh dari kamera juga biasanya memiliki ukuran yang besar. Ukuran yang besar ini memerlukan ruang penyimpanan yang besar pula.

Selain mengenai ukuran media penyimpanan, ukuran gambar juga memiliki dampak pada bagaimana gambar tersebut dikirimkan melalui media internet. Sudah bukan hal yang langka saat ini, dimana banyak sekali pengguna komputer saling berbagai gambar yang dimilikinya. Ukuran gambar yang besar, berarti waktu pengiriman atau pengunduhan yang lama.

Untuk mengatasi permasalahan ini, ukuran gambar perlu dimampatkan. Proses kompresi pada dasarnya memanfaatkan kenyataan pada citra bahwa terdapat suatu derajat perulangan data. Perulangan data inilah yang kemudian dimampatkan untuk menghasilkan suatu citra termampatkan.

Kompresi citra memiliki 2 jenis, yaitu *lossless* dan *lossy.* Kompresi dengan metode *lossless* menghasilkan citra yang sudah dimampatkan tanpa mengurangi informasi dari citra asli. Sementara itu, pada metode *lossy*, terjadi pengurangan data pada citra terkompresi dari citra asli. Kompresi dengan menggunakan metode *lossy* memberikan tingkat kompresi yang lebih tinggi bila dibandingkan dengan kompresi *lossles*.

Tahapan yang umum dalam proses kompresi adalah transformasi citra, kuantisasi dan *entropy coding*. Masing-masing tahapan tersebut memiliki peran yang beda-beda namun adalah kesatuan yang penting dalam proses kompresi. Menjadi catatan pula, bahwa dengan adanya proses kuantisasi, maka proses kompresi yang berlangsung berjenis *lossy*.

Pada kompresi *lossy*, tahap transformasi memainkan peran yang penting, karena pada tahapan ini, data-data yang berguna bagi kompresi gambar dikumpulkan ke dalam jumlah data yang lebih sedikit. Hal ini dapat dicapai dengan menggunakan metode *lifting scheme*. Metode ini adalah pengembangan dari metode wavelet. Sehingga metode *lifting scheme* seringkali disebut sebagai wavelet generasi kedua (Daubechies, 1997).

Alasan dikembangkannya metode *lifting scheme* adalah karena kelebihan yang ditawarkan. Keuntungan yang diberikan oleh metode ini adalah kemampuannya untuk menerapkan proses transformasi menjadi lebih cepat. *Lifting scheme* juga memungkinkan adanya perhitungan transformasi wavelet pada gambar dan langsung mengganti data pada gambar asli. Terakhir, proses *inverse transformation* dapat dilakukan hanya dengan cara memundurkan proses kalkulasi yang berjalan pada tahapan *forward transformation* (Sweldens, 1996) Selebihnya, metode *lifting scheme* ini dapat diterapkan pada banyak jenis transformasi wavelet (Daubechies, 1997).

Melihat kelebihan dari metode *lifing scheme*, penelitian ini mengangkat tema "Implementasi Lifting Scheme dari Wavelet Daubechies 4 dalam Kompresi Citra".

1.2 Rumusan Masalah

Berdasarkan uraian latar belakang masalah di atas, maka dalam skripsi ini dapat dirumuskan permasalahan yang akan dibahas sebagai berikut:

1. Bagaimana merancang sistem kompresi citra *lossy* menggunakan metode *Lifting Scheme* dari Wavelet Daubechies 4?
2. Berapa kualitas hasil kompresi citra *lossy* ketika diterapkan metode *Lifting Scheme* dari Wavelet Daubechies 4, dengan menggunakan tolak ukur obyektif?

1.3 Batasan Masalah

Dari permasalahan di atas, berikut ini diberikan batasan masalah untuk menghindari melebarnya masalah yang akan diselesaikan:

1. Berkas citra yang digunakan berjenis .ppm.
2. Citra yang digunakan adalah citra berwarna dengan komponen RGB.
3. Metode kuantisasi yang digunakan adalah metode skalar

kuantisasi menggunakan *deadzone*.

4. Metode *entropy codying* adalah metode *Run Length Encoding* dan *Huffman coding*.
5. Ukuran citra yang digunakan berukuran 256x256 pixel.

1.4 Tujuan Penelitian

Tujuan penelitian yang ingin dicapai adalah:

1. Merancang sistem kompresi citra *lossy* dengan metode *Lifting Scheme* dari Wavelet Daubechies 4.
2. Mengukur tingkat kualitas citra hasil kompresi *lossy* menggunakan metode *Lifting Scheme* dari Wavelet Daubechies 4.
3. Mengetahui pengaruh dari tahapan kompresi keseluruhan terhadap kualitas kompresi citra.

1.5 Manfaat Penelitian

Manfaat yang bisa didapatkan melalui penelitian ini adalah:

1. Mengetahui kualitas kompresi citra *lossy* menggunakan metode *Lifting Scheme* dari Wavelet Daubechies 4.
2. Memperoleh citra dengan ukuran bit yang lebih kecil berdasarkan pengaturan nilai paramater tertentu pada masing-masing tahapan kompresi.

1.6 Metodologi Penelitian

Untuk mencapai tujuan yang dirumuskan sebelumnya, maka metodologi yang digunakan dalam penulisan tugas akhir ini adalah:

1. Studi Literatur
   Mempelajari teori-teori yang berhubungan dengan konsep kompresi citra menggunakan wavelet Daubechies 4, penerapan metode *lifting scheme* dari wavelet Daubechies 4, dan teori kompresi pada umumnya.
2. Pendefinisian dan analisis masalah
   Mendefinisikan dan menganalisis masalah untuk mencari solusi yang tepat.
3. Perancangan dan implementasi sistem
   Membuat rancangan sistem kompresi gambar yang teratur dan efektif. Kemudian dilakukan penerapan rancangan ini ke dalam aplikasi perangkat lunak untuk memampatkan berkas citra.

4. Uji coba dan analisis hasil implementasi
   Menguji perangkat lunak dengan gambar-gambar. Kemudian menganalisa hasil kompresi citra menggunakan rumus perhitungan kualitas citra hasil kompresi.

1.7 Sistematika Penulisan

Skripsi ini disusun berdasarkan sistematika penulisan sebagai berikut:

1. BAB I PENDAHULUAN
   Berisi latar belakang masalah, perumusan masalah, batasan masalah, tujuan penelitian, manfaat penelitian, metodologi penelitian, dan sistematika penulisan.
2. BAB II TINJAUAN PUSTAKA
   Menguraikan teori-teori yang berhubungan dengan penggunaan wavelet pada proses kompresi, metode *lifting scheme*, tahapan kompresi dan ukuran kualitas hasil kompresi.
3. BAB III METODOLOGI PENELITIAN
   Pada bab ini akan dijelaskan mengenai metode-metode yang digunakan dalam kompresi gambar, penerapan metode *lifting scheme* pada wavelet daubechies db4.
4. BAB IV HASIL DAN PEMBAHASAN
   Bab ini menerangkan proses implementasi dari rancangan penelitian yang dijelaskan pada bab III. Implementasi yang dijelaskan yaitu implementasi dari perangkat lunak. Bab ini juga menjelaskan bagimana derajat kualitas dari proses kompresi terhadap citra percobaan menggunakan rumus perhitungan kualitas citra hasil kompresi.
5. BAB V KESIMPULAN DAN SARAN
   Berisi kesimpulan dari seluruh rangkaian penelitian serta saran beserta kemungkinan pengembangannya.

BAB II
TINJAUAN PUSTAKA

2.1 Komponen Warna

Komponen warna dalam citra pada umumnya direpresentasikan menggunakan RGB (*Red Green Blue*). Banyaknya pengunaan sistem ini dalam citra dikarenakan sistem penglihatan manusia melihat warna sebagai kombinasi linear dari 3 warna utama, yaitu merah, hijau dan biru (*Red, Green, Blue*) (Shi, 2008).

Namun, sistem penglihatan manusia memiliki tingkat sensitifitas yang berbeda untuk tiga komponen warna di atas. Sistem penglihatan manusia lebih sensitif terhadap warna hijau daripada merah dan biru. Sehingga jika data dialokasikan pada informasi yang lebih berguna bagi penglihatan manusia, maka dapat diperoleh efisiensi dalam representasi data dalam citra (Shi, 2008).

Biasanya, dalam pemrosesan citra, sistem warna yang digunakan melibatkan informasi mengenai *luminance* dan *chrominance*. *Luminance* menunjukkan intensitas cahaya pada citra. Sedangkan *chrominance* menunjukkan informasi terkait *hue* dan *saturation* pada citra. *Hue* menunjukkan warna dominan yang dipersepsi oleh mata. Sementara itu *saturation* menunjukkan tingkat perpaduan cahaya putih dengan *hue* (Gonzalez, 1993).

Dalam standar JPEG 2000, sistem warna yang digunakan adalah YCbCr (Shi, 2008). Rumus konversi dari sistem RGB menjadi YCbCr dinyatakan oleh rumus 2.1 (Hamilton, 1992).

$$\begin{pmatrix} Y \\ Cb \\ Cr \end{pmatrix} = \begin{pmatrix} 0 \\ 128 \\ 128 \end{pmatrix} + \begin{pmatrix} 0.299 & 0.587 & 0.114 \\ -0.169 & -0.331 & 0.504 \\ 0.5 & -0.419 & -0.081 \end{pmatrix} \cdot \begin{pmatrix} R \\ G \\ B \end{pmatrix} \quad (2.1)$$

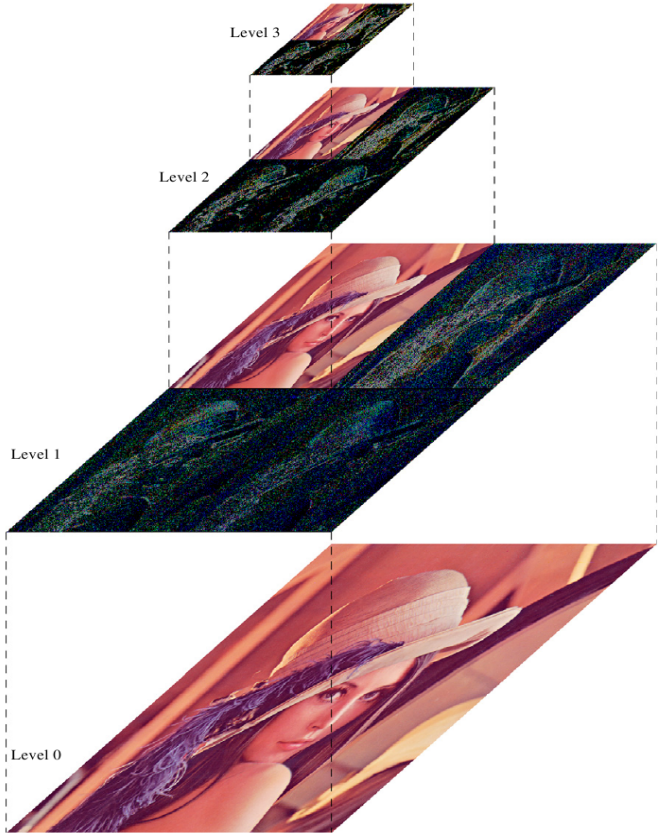
Sementara itu, rumus konversi dari YCbCr menjadi RGB dinyatakan oleh rumus 2.2 (Hamilton,1992):

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1.4 \\ 1 & -0.343 & -0.711 \\ 1 & 1.765 & 0 \end{pmatrix} \cdot \begin{pmatrix} Y \\ Cb-128 \\ Cr-128 \end{pmatrix} \quad (2.2)$$

2.2 Wavelet

Wavelet adalah sebuah alat untuk melakukan dekomposisi sinyal, semisal gambar, ke dalam hirarki dengan tingkat resolusi

yang berbeda. Semakin banyak tingkatan hirarki, maka semakin detail informasi yang diberikan (Xiong, 2009). Gambar 2.1 memperlihatkan contoh dekomposisi gambar Lena dengan 3 tingkat hirarki.



Gambar 2.1 Dekomposisi citra ke dalam hirarki bertingkat 3
Sumber: Xiong, 2009

Kemampuan dekomposisi ini dimungkinkan karena adanya sepasang *waveform*, yaitu *wavelet functions* dan *scaling functions*. *Wavelet functions* berfungsi untuk merepresentasikan bagian pada gambar dengan nilai frekuensi yang tinggi, yang menunjukkan bagian detail dari gambar. Sedangkan *scaling functions*

merepresentasikan bagian halus dari gambar (frekuensi rendah) (Grgic, 2001).

Wavelet functions ini kemudian mengalami proses translasi dan dilasi sepanjang domain waktu dengan posisi dan nilai skala yang berbeda. Translasi menunjukkan proses penempatan wavelet pada lokasi yang berbeda. Sedangkan dilasi menunjukkan proses pengecilan atau pembesaran ukuran dari wavelet yang berarti adanya perubahan pada nilai skala (Polikar, 2001).

Tujuan dari proses translasi dan dilasi ini adalah untuk mentransformasi bagian dari sinyal yang memiliki nilai frekuensi yang berbeda. Bagian dengan nilai frekuensi tinggi ditransformasi dengan nilai skala yang rendah. Sementara itu bagian dengan nilai frekuensi rendah ditransformasi dengan nilai skala tinggi (Grgic, 2001). Dapat dilihat bahwa hubungan antara skala dan frekuensi adalah terbalik (Polikar, 2001).

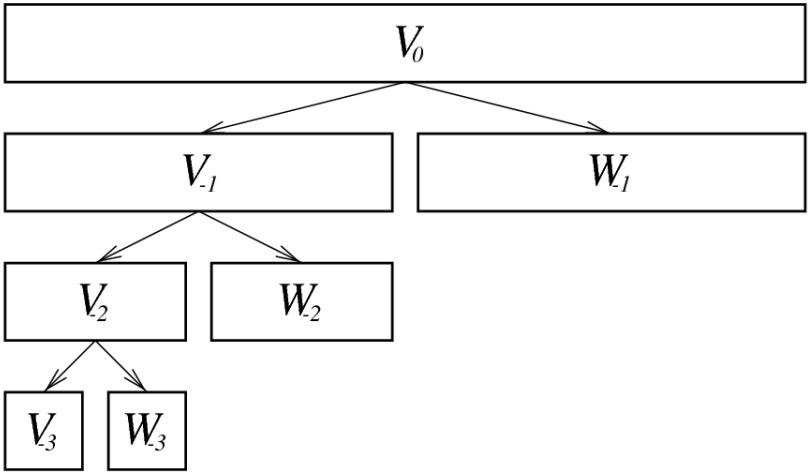
Frekuensi menunjukkan tingkat perubahan pada sinyal. Jika mengambil contoh pada sebuah gambar alam, maka langit memiliki nilai frekuensi yang rendah, karena tingkat perubahan nilai pixel pada objek langit rendah. Sementara itu pada bagian tepi dari sebuah objek terjadi perubahan nilai pixel yang tinggi, berarti nilai frekunsi pun tinggi (Xiong, 2009).

Hasil dari transformasi wavelet ini adalah himpunan koefisien yang mengukur kontribusi dari wavelet pada sinyal dengan lokasi dan skala yang berbeda (Grgic, 2001). Dengan demikian, maka transformasi wavelet ini mengukur kemiripan dari wavelet terhadap sinyal (Polikar, 2001).

Dalam dunia komputer, konsep di atas dapat diterapkan menggunakan Discrete Wavelete Transformation (DWT). Metode DWT dapat dilihat pada gambar 2.2:

Simbol V_n menunjukkan nilai frekuensi rendah pada level n. Sedangkan W_n menyimpan detail pada level n (Uytterhoeven, 1997). Data pada V_n dan W_n diperoleh melalui tahapan *low pass filter* dan *high pass filter.* Proses *low pass filters* membuang frekuensi yang nilainya di atas setengah frekuensi tertinggi. Begitu pula sebaliknya untuk *high pass filters* (Polikar, 2001).

Setelah tahapan *filtering* maka data dapat di*subsample*, atau pengurangan jumlah sampel. Proses subsampling ini menggandakan nilai skala (Polikar, 2001).



Gambar 2.2 Proses dekomposisi citra menggunakan filter bank
Sumber: Uytterhoeven, 1997

Secara keseluruhan, proses *low pass filtering* mengurangi ½ dari resolusi citra. Hal ini dikarenakan ½ dari informasi yang ada pada citra itu terkurangi. Sementara itu, subsampling menggandakan nilai skala. Proses subsampling itu sendiri tidak mengurangi atau menghilangkan informasi yang ada pada citra (Polikar, 2001).

Untuk melakukan proses *low pass filtering* dan *high pass filtering* digunakanlah koefisien-koefisien yang sesuai untuk masing-masing wavelet. Misal rumusan scaling function untuk proses *low pass filtering* pada wavelet Daubechies 4, koefisiennya ditunjukkan oleh rumus 2.3.

$$h0=\frac{1+\sqrt{3}}{4\sqrt{2}} \qquad h1=\frac{3+\sqrt{3}}{4\sqrt{2}}$$
$$h2=\frac{3-\sqrt{3}}{4\sqrt{2}} \qquad h3=\frac{1-\sqrt{3}}{4\sqrt{2}} \qquad (2.3)$$

Sementara itu, untuk high pass filtering, pada Daubechies 4 ditunjukkan oleh rumus 2.4.

g0=h3, g1=-h2, g2=h1, g3=-h0 (2.4)

2.3 Lifting Scheme

Metode transformasi wavelet yang sudah dijelaskan sebelumnya

dikenal sebagai wavelet generasi pertama (Daubechies, 1997). Hal ini dikarenakan metode transformasi yang digunakan berbasis translasi dan dilasi.

Sementara itu, metode *lifting scheme* yang sering diberi nama sebagai wavelet generasi kedua, melakukan proses transformasi pada domain spasial tanpa adanya proses translasi dan dilasi (Daubechies, 1997). Hal ini berbeda dengan wavelet generasi pertama mengingat proses transformasi berjalan pada domain frekuensi.

Keunggulan yang diberikan oleh metode *lifting scheme* adalah (Sweldens, 1995):

1. Memungkinkan adanya implementasi dalam transformasi wavelet secara cepat. Pada wavelet generasi pertama, proses berjalan dengan adanya pembagian sinyal ke dalam bentuk *high pass* dan *low pass.* Perulangan terjadi dimana sinyal *low pass* dipisah kembali menjadi *high pass* dan *low pass*, dan seterusnya. Dalam metode *lifting scheme*, diketahui bahwa terdapat kemiripan antara *high pass* dan *low pass* yang dapat dioptimalkan guna mempercepat proses transformasi.
2. Data pada citra asli dapat langsung ditransformasi dan dirubah tanpa perlu disimpan secara temporer pada lokasi lain.
3. Secara praktikal, proses untuk menemukan *inverse wavelet transform* dapat dilakukan dengan lebih mudah. Hal ini didapatkan dengan hanya merubah urutan operasi dan menjadikan tanda + menjadi – begitu pula sebaliknya.

Tahap dalam *lifting scheme* ada tiga, yaitu *split, predict,* dan *update.* Masing-masing tahap akan dibahas di bawah ini:

1. Split

   Dimisalkan sebuah sinyal x. Tahapan *split* memisah data pada x ke dalam dua subset yang lebih kecil. Untuk mengeksploitasi adanya korelasi yang besar antara pixel yang bersebelahan, maka x dapat dikelompokkan ke dalam kelompok berindeks genap, x_e, dan berindex ganjil, x_o (Daubechies, 1997).

   Proses *split* ini dilakukan karena pada dasarnya, yang ingin dicapai dari proses *lifting scheme* ini adalah citra

aproksimasi terhadap citra asli. Sudah pasti citra aproksimasi ini memiliki ukuran yang lebih kecil dari citra asli.

2. Predict

Pada umumnya, sifat dari pixel yang bersebelahan adalah memiliki korelasi yang tinggi. Mengingat proses split ini membagi ke dalam kelompok berdasarkan indeks ganjil dan genap, maka data pada satu subset dapat diprediksi berdasarkan data pada subset lainnya, semisal data pada x_e dapat digunakan untuk memprediksi data pada x_o (Daubechies, 1997).

Sudah jelas, bahwa prediksi yang dihasilkan belum tentu tepat. Sehingga, perbedaan, d, dari prediksi dengan nilai asli yang ingin diprediksi dapat disimpan menggunakan rumus 2.5(Daubechies, 1997).

$$d = x_o - P(x_e) \quad (2.5)$$

Dari rumus 2.3, dapat dilihat bahwa untuk mendapatkan nilai dari x_o digunakanlah rumus 2.6 (Daubechies, 1997).

$$x_o = P(x_e) + d \quad (2.6)$$

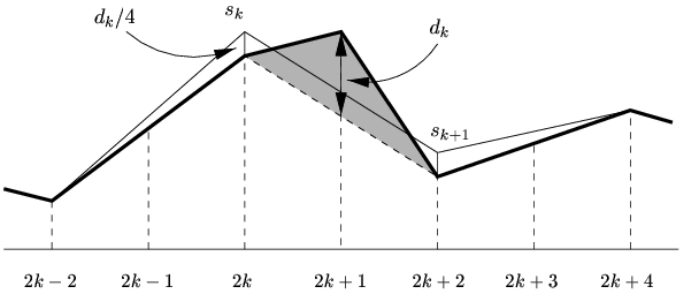
Salah satu bentuk prediksi yang mudah diterapkan untuk contoh kasus di atas adalah dengan mencari nilai tengah antara kedua tetangga x_o. Rumusannya dinyatakan pada rumus 2.7 (Daubechies, 1997).

$$d_k = x_{2x+1} - (x_{2k} + x_{2k+2})/2 \quad (2.7)$$

3. Update

Pada tahapan ini, data asli x, sudah dapat direpresentasikan menggunakan subset (x_e, d). Dengan tahapan prediksi, korelasi yang ada antara pixel dapat dikurangi. Namun terdapat masalah dari segi dekorelasi dalam domain frekuensi.

Dapat dilihat bahwa x_e diperoleh dengan melakukan proses *subsampling*, atau hanya mengambil sebagian data dari citra asli. Akibat dari adanya *subsampling* ini adalah adanya *aliasing* (Daubechies, 1997).



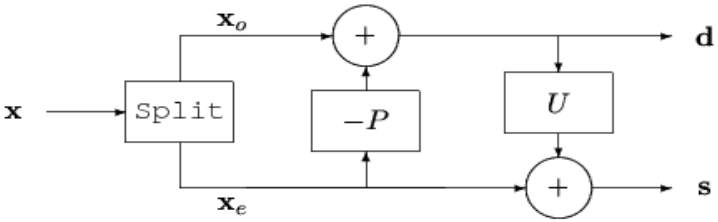
Gambar 2.3 Skenario updating
Sumber: Daubechies, 1997

Wujud adanya *aliasing* dapat dilihat pada gambar 2.3. Pada gambar 2.3, citra asli ditunjukkan oleh garis tebal. Nilai dari d_k diperoleh melalui proses prediksi. Untuk menjaga agar nilai rata-rata dari pixel tetap terjaga (menghindari *aliasing)*, maka bagian pada daerah yang diabukan, dapat dibagikan kepada pixel tetangga yang berindeks genap. Hasil dari pembagian ini adalah garis s_k (Daubechies, 1997).

Dari contoh di atas, maka proses *updating* adalah proses untuk menemukan nilai x_e yang memungkinkan dijaganya kauntitas skalar tertentu yang bersifat global, semisal rata-rata kecerahan citra (Swelden, 1995). Dari contoh di atas, maka rumus *update* dinyatakan pada rumus 2.8 (Daubechies, 1997).

$$s_k = x_{2k} + (d_{k-1} + d_k)/4 \qquad (2.8)$$

Secara keseluruhan, proses *split*, *predict* dan *update* dapat digambarkan seperti pada Gambar 2.3



Gambar 2.4 Diagram proses split, predict dan update
Sumber: Daubechies, 1997

Tiga tahap *lifting* yang sudah dijelaskan sebelumnya, dapat di*invert* dengan hanya menukar posisi dan mengganti tanda + menjadi -, begitu pula sebaliknya. Sehingga, secara keseluruhan, transformasi wavelet yang terjadi dapat dinotasikan sebagai berikut (Sweldens, 1995):

$$\texttt{For j = -1 downto -n:} \quad \begin{cases} \{\lambda_j, \gamma_j\} & := & \texttt{Split}(\lambda_{j+1}) \\ \gamma_j & -= & \mathcal{P}(\lambda_j) \\ \lambda_j & += & \mathcal{U}(\gamma_j). \end{cases}$$

Gambar 2.5 *Forward Transformation*
Sumber: Sweldens, 1995

Untuk tahapan *inverse*, maka notasinya adalah sebagai berikut:

$$\texttt{For j = -n to -1:} \begin{cases} \lambda_j & -= & \mathcal{U}(\gamma_j) \\ \gamma_j & += & \mathcal{P}(\lambda_j) \\ \lambda_{j+1} & := & \texttt{Join}(\lambda_j, \gamma_j). \end{cases}$$

Gambar 2.6 *Inverse Transformation*
Sumber: Sweldens, 1995

Melihat dari tahapan yang sudah dijelaskan sebelumnya, s merepresentasikan rata-rata dari citra. Sedangkan d merepresentasikan perbedaan pada pixel. Sehingga s dan d adalah wujud dari filter *low-pass* (s) dan *high-pass* (d) pada citra (Spires, 2005).

2.4 Kompresi Citra

Kompresi citra adalah sebuah proses dimana jumlah data yang digunakan untuk merepresentasikan sebuah gambar menjadi lebih sedikit. Prinsip dasar dalam kompresi citra adalah pengurangan derajat pengulangan (redundansi) pada data. Maksud dari adanya redundansi pada data adalah data tersebut tidak menyediakan informasi yang berguna atau sekedar mengulang informasi yang sudah dibawakan oleh data lainnya.

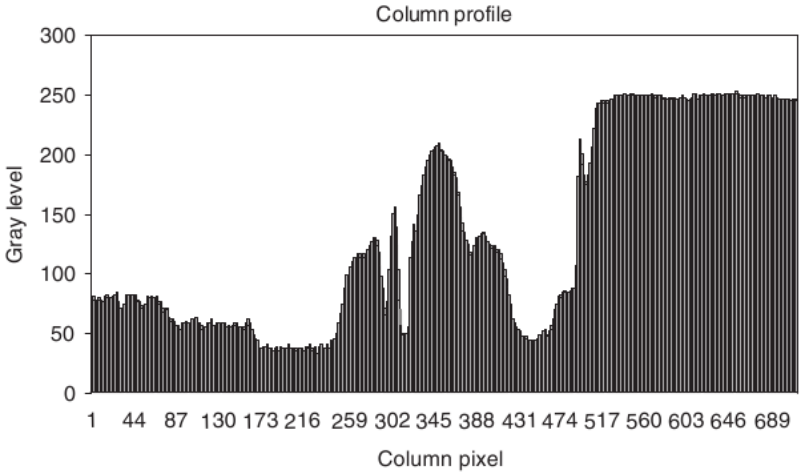
Ada tiga macam redundansi pada citra yang dapat dieksploitasi demi tercapainya kompresi. Redundansi tersebut adalah redundansi

antar pixel (*interpixel redundancy)*, redundansi pada sistem pengkodean (*coding redundancy)* dan redundansi pada sistem psychovisual (*psychovisual redundancy)* (Gonzalez, 1993).

2.4.1. Redundansi antar pixel

Gambar 2.7 adalah sebuah diagram histogram yang menunjukkan nilai pixel pada suatu baris pada sebuah gambar. Dapat dilihat pada gambar, bahwa pada beberapa rentang posisi pixel, terdapat korelasi yang tinggi antar pixel (semisal dalam rentang 517-689 pada bidang horizontal). Korelasi ini menunjukkan adanya kedekatan nilai antar suatu pixel dengan pixel tetangganya. (Shi, 2008).

Pada himpunan pixel yang memiliki tingkat korelasi yang tinggi, maka informasi yang dibawa oleh masing-masing data (pixel) adalah kecil. Sehingga, kontribusi sebuah pixel kepada gambar juga bersifat redundan, atau dalam bahasa lain mengulangi kontribusi yang sudah diberikan oleh pixel lainnya.



Gambar 2.7 Nilai pixel pada suatu baris citra.
Sumber: Shi, 2008

Jenis redundansi seperti yang telah dicontohkan di atas disebut redundansi antar pixel *(interpixel redundancy).* Untuk mengurangi

redundansi ini, maka himpunan pixel yang redundan tadi perlu diubah kedalam bentuk yang lebih efisien.

Salah satu contoh cara untuk melakukan proses pengurangan redundansi di atas adalah dengan merepresentasikan data redundan tadi ke dalam suatu data tunggal yang menyimpan perbedaan nilai antara himpunan data yang redundan tadi. Pada umumnya, proses perubahan data ini disebut pemetaan (*mapping)* (Gonzalez, 1993).

2.4.2. Redundansi pada sistem pengkodean

Redundansi pada sistem pengkodean berhubungan dengan bagaimana sebuah informasi direpresentasikan. Secara singkat, inti dari pengurangan redundansi pada sistem pengkodean adalah bagaimana konstruksi informasi dikodekan secara hemat.

Untuk memahami konsep ini, maka diandaikan dua buah sistem pengkodean, Code 1 dan Code 2.

Tabel 2.1 Contoh sistem pengkodean

Simbol	Kemungkinan Kemunculan	Code 1	Code 2
a1	0.1	000	0000
a2	0.2	001	01
a3	0.5	010	1
a4	0.05	011	0001
a5	0.15	100	001

Dapat dilihat dari tabel 2.1, sistem pengkodean pada Code 1 menunjukkan adanya kesamaan dalam merepresentasikan informasi. Tiap simbol direpresentasikan dengan 3 kode. Biasanya sistem pengkodean seperti ini disebut dengan *uniform-length code word assignment.*

Sementara itu, pengkodean pada Code 2 memiliki sistem yang berbeda dengan Code 1. Pada sistem Code 2, representasi informasi tidak sama rata. Pada simbol yang memiliki kemungkinan kemunculan lebih tinggi, pengalokasian simbol lebih sedikit.

Untuk membandingkan mana yang lebih efektif dalam merepresentasikan kode ini, dan yang terkait dengan redundansi pada

sistem pengkodean adalah dengan membandingkan rata-rata panjang kode.

Untuk Code 1, memiliki rata-rata panjang kode 3. Untuk Code 2, perhitungannya adalah:

L_{avg} = (4x0.1)+(2x0.2)+(1x0.5)+(4x0.05)+(3x0.15)=1.95bits/symbol

Dengan contoh di atas, dapat dilihat bahwa pengaturan bagaimana sebuah informasi direpresentasikan dapat meningkatkan kompresi suatu gambar.

2.4.3. Redundansi pada sistem psychovisual

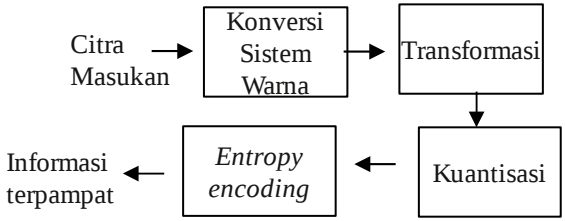
Hipotesis yang terkait mengenai pengurangan redundansi dalam sistem *psychovisual* adalah adanya informasi yang kurang berguna bila dibandingkan dengan informasi lainnya. Keberadaan informasi yang kurang berguna ini dapat dihilangkan tanpa merusak kualitas citra. (Gonzalez, 1993).

Redundansi dalam sistem *psychovisual* terkait dengan bagaimana Sistem Penglihatan Manusia (SPM). Ada beberapa aspek yang memiliki pengaruh terhadap SPM yang dimana aspek-aspek ini dapat dimanfaatkan untuk kompresi citra. Aspek-aspek yang berpengaruh semisal luminasi, tekstur dan frekuensi (Shi, 2008).

2.5 Tahapan kompresi Citra

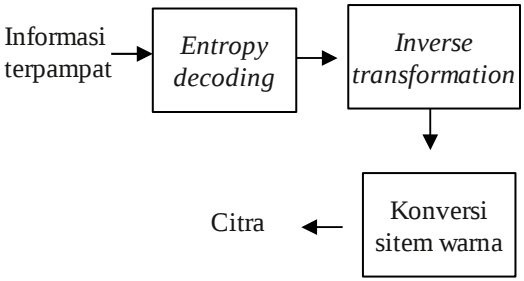
Ketika melakukan proses kompresi citra, tiga katagori redundansi diatas, biasanya digabungkan menjadi sebuah sistem kompresi. Proses kompresi terdiri atas dua tahap, yaitu tahapan *encoding* dan tahap *decoding.*

Tahapan *encoding* terdiri atas 4 langkah yang dapat dilihat pada gambar 2.8:



Gambar 2.8 Tahapan Encoding

Sementara itu, proses decoding adalah seperti yang tergambar pada gambar 2.9:



Gambar 2.9 *Tahapan decoding*

2.5.1. Konversi Sistem Warna

Sistem warna yang bagus untuk kompresi citra adalah yang melibatkan informasi *luminance* dan *chrominance* (Al-Abudi, 2005). Sistem warna RGB dirubah kedalam bentuk YCbCr berdasarkan rumus 2.1. Untuk proses pengembalian dari bentuk YCbCr ke dalam bentuk RGB digunakan rumus 2.2. Pada tahapan selanjutnya, masing-masing komponen warna akan mengalami proses kompresi secara terpisah (Gershikov, 2007).

2.5.2 Transformasi Menggunakan Lifting Scheme pada D4

Secara sederhana, tahapan transformasi merubah nilai pixel dan mengelompokkannya ke dalam koefisien-koefisien transformasi. Tujuan dari perubahan ini adalah untuk memisahkan (dekorelasi) nilai pixel atau memampatkan sebanyak mungkin informasi ke dalam jumlah koefisien yang sesedikit mungkin, dimana koefisien-koefisien tersebut memiliki nilai frekuensi yang rendah (Grgic, 2001). Dengan adanya kompresi koefisien seperti ini, maka koefisien lain yang tidak penting dapat dibuang tanpa menghilangkan terlalu banyak informasi (Xiong, 2001).

Salah satu bentuk transformasi yang efektif untuk kompresi citra adalah wavelet. Wavelet memiliki kemampuan untuk menangkap fenomena frekuensi tinggi (semisal tepi gambar, tekstur) dan frekuensi rendah (semisal langit).

Kemampuan ini menjadi penting untuk kompresi citra, karena

kriteria transformasi yang baik adalah yang mampu mengkonsentrasikan data pada jumlah koefisien sesedikit mungkin. Ditambah lagi, dengan kemampuan untuk mendeteksi frekuensi rendah, maka koefisien-koefisien tadi dikumpulkan pada koefisien berfrekuensi rendah. Hal ini menguntungkan, karena mata manusia lebih sensitif pada fenomena spasial berfrekuensi rendah (Grgic, 2001).

Dengan adanya kompresi data kedalam jumlah koefisien yang sedikit, maka bisa dikatakan bahwa data setelah transformasi bersifat compact jika dibandingkan dengan data pada representasi asli. Jadi, transformasi wavelet memungkinkan adanya aproksimasi terhadap citra asli secara akurat hanya dengan menggunakan sebagian kecil dari koefisien wavelet (Sweldens, 1995).

Seperti yang sudah dituliskan pada bagian 2.2, wavelet mengalami pengembangan dengan ditemukannya metode *lifting scheme* yang kemudian disebut sebagai *wavelet generasi kedua*. Salah satu bentuk yang sering dijumpai adalah wavelet D4 yang termasuk keluarga wavelet Daubechies.

Proses transformasi menggunakan *lifting scheme* pada wavelet D4 adalah sebagai berikut (Latha, 2008):

1. Splitting dengan menggunakan metode pembagian data ke dalam kelompok genap dan ganjil, atau yang dikenal dengan nama *lazy wavelet.*
2. Update 1:

```
For n = 0 to half-1
```

$$S[n] = S[n] + \sqrt{3}\, S[half + n] \tag{2.9}$$

3. Predict :

$$S[half] = S[half] - \frac{\sqrt{3}}{4} S[0] - \frac{\sqrt{3}-2}{4} S[half - 1]$$

```
For n=1 to half-1
```

$$S[half + n] = S[half + n] - \frac{\sqrt{3}}{4} S[n] - \frac{\sqrt{3}-2}{4} S[n-1] \tag{2.10}$$

4. Update 2:

```
For n=0 to half-2
      S[n]=S[n]-S[half+n+1]
S[half-1]=S[half-1]-S[half]
```

(2.11)

5. Normalisasi :

```
For n=0 to half-1
```

$$S[n]=\frac{\sqrt{3}-1}{\sqrt{2}}S[n]$$
$$S[n+half\,]=\frac{\sqrt{3}+1}{\sqrt{2}}S[n+half\,] \quad (2.12)$$

Sementara itu untuk proses *inverse transformation* adalah sebagai berikut:

1. Normalisasi:

```
For n=0 to half-1
```

$$S[n]=\frac{\sqrt{3}+1}{\sqrt{2}}S[n]$$
$$S[n+half\,]=\frac{\sqrt{3}-1}{\sqrt{2}}S[n+half\,] \quad (2.13)$$

2. Update 2:

```
For n=0 to half-2
      S[n]=S[n]+S[half+n+1]
S[half-1]=S[half-1]+S[half]
```
(2.14)

3. Predict:

$$S[half\,]=S[half\,]+\frac{\sqrt{3}}{4}S[0]+\frac{\sqrt{3}-2}{4}S[half-1]$$

```
For n=1 to half-1
```

$$S[half+n]=S[half+n]+\frac{\sqrt{3}}{4}S[n]+\frac{\sqrt{3}-2}{4}S[n-1] \quad (2.15)$$

4. Update 1:

```
For n = 0 to half-1
```

$$S[n]=S[n]-\sqrt{3}\,S[half+n] \quad (2.16)$$

5. Penggabungan data yang terpisah.

2.5.3 Kuantisasi Dead Zone

Setelah citra mengalami proses transformasi, maka hasil transformasi ini perlu untuk di kuantisasi. Proses kuantisasi menjadi bagian penting dari kompresi bermodel *lossy.* Pada tahapan kuantisasi ini, terjadi pengurangan nilai maupun jumlah data unik pada citra (Celebi, 2009).

Secara umum, metode kuantisasi adalah memetakan data yang ada pada gambar pada data-data perwakilan dengan jumlah keunikan

yang lebih sedikit. Dimisalkan himpunan data asli dari gambar adalah C = {c_i, i=1,2,....N}. N adalah jumlah data keseluruhan dari citra I. Maka proses kuantisasi adalah memetakkan data dari C → $\overline{C}$, dimana himpunan $\overline{C}$ adalah {$\bar{c}_j$, i=1,2,...K}. K adalah jumlah kelompok pada citra hasil kuantisasi $\bar{I}$, dimana K < N (Albayrak, 2001). $\bar{c}$ adalah data hasil dari kuantisasi atau *quantization level* (Shi, 2008).

Salah satu metode kuantisasi adalah dengan menggunakan metode *uniform quantizer* dengan *dead zone*. Metode ini adalah salah satu pengembangan dari metode *scalar quantization* yang bersifat *uniformed*.

Scalar Quantization adalah proses kuantisasi pada masing-masing data. Sementara itu, *quantizer* yang bersifat *uniformed*, memiliki ukuran *step size*, atau panjang interval kelompok hasil kuantisasi, yang sama (Shi, 2008).

Dalam kuantisasi, ukuran dari *step size* ini memiliki arti yang penting bagi kualitas dari citra. Jika ukuran *step size besar*, maka tingkat kompresi citra akan semakin bertambah. Namun hal ini berakibat pada berkurangnya kualitas citra, karena adanya nilai eror hasil kuantisasi. Nilai eror hasil kuantisasi ini mengukur berapa perbedaan nilai antar data masukan dan data hasil kuantisasi.

Kuantisasi skalar berbasis *dead zone* dapat diilustrasikan seperti yang tampak pada gambar 2.10. Berdasarkan gambar tersebut, dapat dilihat bahwa data-data yang berada pada suatu interval tertentu pada kordinat x, dipetakan terhadap sebuah nilai y. Interval pada kordinat x ini disebut dengan *step size* (Δ), sementara itu nilai y hasil pemetaan disebut dengan *reconstruction level* (Shi, 2008).

Berdasarkan gambar 2.10, maka rumusan kuantisasi dinyatakan pada rumus 2.17.

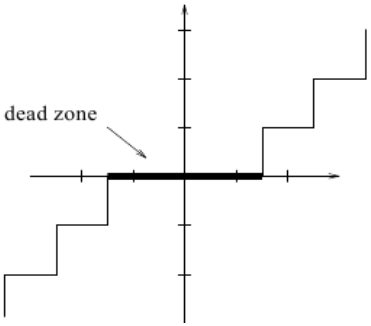
$$y = Q(x) \tag{2.17}$$

Adanya wilayah *dead zone* pada sistem kuantisasi diatas memungkinkan penambahan jumlah data yang memiliki nilai nol. Sehingga, rumusan Q(x) berdasarkan gambar 2.10 dinyatakan pada rumus 2.18.

$$y = \begin{cases} 0 & |x| \le T \\ x/\Delta & |x| > T \end{cases} \tag{2.18}$$

T adalah nilai *threshold*, dimana bila sebuah nilai berada dalam

interval T, maka nilai tersebut akan di-nol-kan. T sendiri bernilai 2 kali lebih besar dari Δ (Chang, 2006).



Gambar 2.10 Kuantisasi Skalar berbasis *dead zone*
Sumber: Strom, 1996

2.5.4 Run Length Encoding

Metode *Run Length Encoding* (RLE) bekerja dengan cara mengelompokkan sebuah sekuens simbol dengan nilai yang sama dimana sekuens ini kemudian diwakilkan oleh sepasang nilai, biasanya nilai yang berurutan ini dan jumlah urutannya (Salomon, 2004). Kemampuan *Run Length Encoding* ini penting bagi sistem kompresi citra menggunakan wavelet mengingat pada level dekomposisi yang lebih rendah, banyak terdapat koefisien dengan nilai yang kecil dan sama. Hal ini misalnya dapat dilihat pada gambar 2.1 dimana pada level dekomposisi yang lebih rendah, pada bagian HH, HL dan LH, warna didominasi oleh warna hitam. Ini menunjukkan bahwa terdapat banyak keserahgaman pada bagian ini.

Dengan memanfaatkan banyaknya keseragaman ini, maka metode RLE dapat mengurangi jumlah simbol. Hal ini memiliki pengaruh terhadap meningkatnya persentase kompresi citra.

Jika diumpamakan terdapat data yaitu 12,12,12,12,12,3,3,10,7,7,2, maka dengan menggunakan metode RLE, data tersebut dipampatkan menjadi 12,12,3,3,3,0,10,7,7,0,2. Pada data asli, terdapat 5 jumlah nilai 12 yang berurutan. Ketika RLE diterapkan, perlu diketahui mana yang nilai asli dan mana nilai yang menunjukkan urutan. Oleh karena itu, untuk nilai 12, penerapan RLE menghasilkan 12,12,3. Hasil tersebut berarti bahwa 12 mengalami pengulangan (sequence), karena muncul 12 dua kali.

Angka 3 berarti ada 3 jumlah 12 yang mengikuti 12, 12 pada data asli.

Dapat dilihat pada contoh di atas, bahwa metode RLE menjadi lebih maksimal jika memang terdapat banyak data yang memiliki nilai yang berurutan. Sehingga hal ini cocok untuk memapatkan koefisien hasil transformasi wavelet pada bagian LH, HL, dan HH pada level dekomposisi yang lebih rendah.

2.5.5 Entropy Coding Menggunakan Huffman Coding

Tahap *entropy coding* berfungsi untuk mengurangi rata-rata penggunaan jumlah bits untuk merepresentasikan data. Sehingga, *entropy coding* mengurangi keberadaan redundansi pada sistem pengkodean.

Metode huffman coding bekerja dengan cara memberikan data yang memiliki tingkat kemunculan lebih tinggi jumlah bits yang lebih sedikit. Sementara data dengan tingkat kemunculan yang lebih sedikit, memperoleh jumlah bits yang lebih banyak. Seperti yang telah diperlihatkan pada bagian 2.4.2, metode ini dapat mengurangi rata-rata jumlah bits yang digunakan oleh data.

Metode ini dimulai dengan pengurutan data berdasarkan tingkat kemunculannya. Kemudian dibangunlah sebuah pohon (*tree*) dimana masing-masing data berada pada daunnya (*leaf*) dengan berurutan dari bawah ke atas. Dua simbol dengan tingkat kemunculan yang paling kecil diambil dan ditambahkan kedalam pohon. Dua data ini kemudian direpresentasikan pada level yang lebih tinggi dengan nilai hasil penjumlahan kedua data tadi. Tahapan ini diulang kembali hingga semua data sudah berada dalam pohon dan semua data diwakili oleh satu data (Salomon, 2004).

Sebagai contoh terdapat data beserta tingkat kemunculannya sebagai berikut:

Data	a1	a2	a3	a4	a5
Kemunculan	0.4	0.2	0.2	0.1	0.1

Maka, langkah-langkahnya adalah sebagai berikut:

1. Ambil dua data dengan tingkat kemuculan terkecil, yaitu a4 dan a5. Gabungkan keduanya dan jumlahkan nilainya. Hasil akhirnya adalah

Data	a1	a2	a3	a4,5
Kemunculan	0.4	0.2	0.2	0.2

2. Data terkecil saat ini adalah a3 dan a4,5. Gabungkan keduanya

Data	a1	a2	a3,4,5
Kemunculan	0.4	0.2	0.4

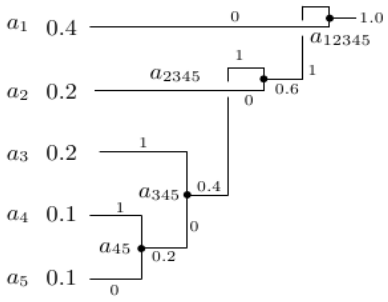
3. Data terkecil beriktunya adalah a3,4,5 dan a2.

Data	a1	a2,3,4,5
Kemunculan	0.4	0.6

4. Terakhir adalah a1 dan a2,3,4,5

Data	a1,2,3,4,5
Kemunculan	1.0

5. Kemudian, pada masing-masing ruas pohon, diberikan nilai 1 untuk ruas dengan nilai kemunculan lebih tinggi, dan 0 untuk yang lebih rendah.
6. Hasil akhir dari tahapan di atas dapat dilihat pada gambar :



Gambar 2.11 Hasil tahapan huffman coding

Sehingga, hasil *encoding* untuk masing-masing data adalah:

Data	a1	a2	a3	a4	a5

Encoding	0	10	111	1101	1100

Dari rata-rata di atas dapat dilihat bahwa rata-rata jumlah bits yang digunakan adalah 2.2 bits/simbol. Sementara itu, nilai entropy atau rata-rata informasi yang dikandung oleh simbol adalah 2.12 bits. Nilai entropy dihitung menggunakan rumus 2.19 (Shi, 2008):

$$-\sum_{i}^{N} P_i \log_2(P_i) \qquad (2.19)$$

Dari hasil di atas, dapat dilihat bahwa nilai rata-rata jumlah bits yang dihasilkan oleh *huffman coding* (2.2 bits/simbol) mendekati nilai optimalnya (2.12 bits).

2.6 Penilaian Kualitas kompresi Citra

Hasil dari proses kompresi citra adalah ukuran yang berkurang. Berkurangnya ukuran ini dapat diketahui berdasarkan nilai rasio kompresinya atau *compression rate* (CR), yang dinyatakan pada rumus 2.20.

CR = ukuran terpampat / ukuran asli (2.20)

Untuk mengukur kualitas citra hasil kompresi, dapat digunakan metode subjektif dan objektif. Metode objektif mengukur kualitas melalui perhitungan eror.

Jika sinyal masukan didefinisikan sebagai *f(x,y)*, maka *f(x,y)* akan memasuki sistem kompresi, sesuai dengan tahapan yang sudah dijelaskan sebelumnya. Keluaran dari proses ini didefinisikan sebagai *g(x,y)*. Maka, tingkat eror, *e(x,y)*,dari proses ini dapat dirumuskan sebagai perbedaan antara sinyal asli dan sinyal keluaran: $e(x,y) = f(x,y) - g(x,y)$

Dari rumusan di atas, maka nilai dari *Mean Square Error*, E_{ms}, dihitung menggunakan rumus 2.21.

$$E_{\text{ms}} = \frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} e(x,y)^2 \qquad (2.21)$$

M dan N menunjuk pada dimensi dari citra, M menunjuk pada dimensi horizontal dan N menunjuk pada dimensi vertikal. Nilai dari *Root Mean Square*, E_{rms}, didapatkan menggunakan rumus 2.22.

$$E_{\text{rms}} = \sqrt{E_{\text{ms}}}. \quad (2.22)$$

Ukuran yang sering digunakan dalam mengukur tingkat eror dari proses kompresi citra adalah *Peak Signal to Noise Ratio*, *PSNR* (Shi, 2008). Dengan menggunakan rumus *Mean Square Error* sebelumnya, rumusnya ditunjukkan oleh rumus 2.23.

$$PSNR = 10 \log_{10} \left(\frac{255^2}{E_{\text{ms}}} \right) \quad (2.23)$$

Interpretasi dari hasil perhitungan PSNR adalah semakin besar nilai PSNR maka semakin baik kualitas dari citra hasil kompresi, *g(x,y)* (Shi, 2008). PSNR diukur menggunakan satuan decibels (dB) (Grgic, 2001).

BAB III
METODOLOGI DAN PERANCANGAN SISTEM

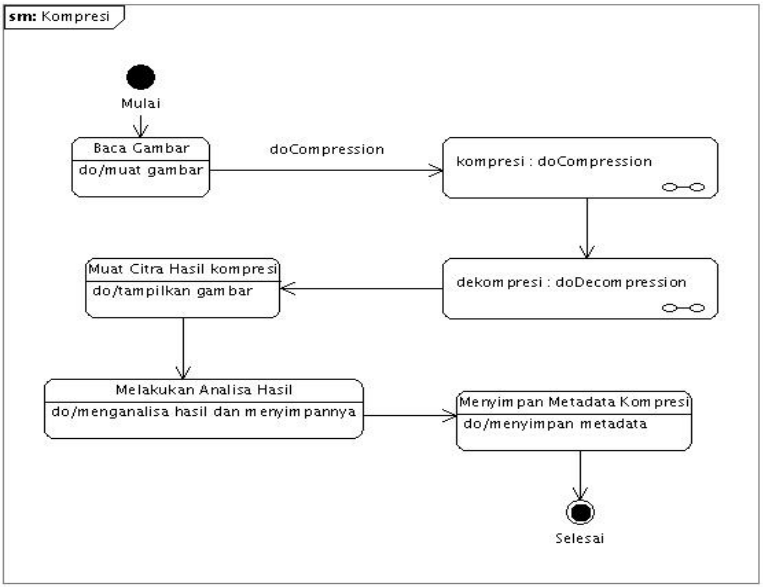
Pada bab Metode dan Perancangan Sistem ini, akan dibahas mengenai perancangan sistem dan penelitian kompresi data menerapkan metode *Lifting Scheme* dari wavelet D4.

3.1 Analisis Perangkat Lunak

Dalam subbab Analisis Perangkat Lunak ini, akan dijelaskan mengenai pemodelan dan perancangan sistem kompresi data yang diterapkan pada skripsi ini. Dalam subbab ini, akan digunakan diagram-diagram UML (*Unified Modelling Language*) untuk menjelaskan bagian-bagian dari perancangan.

3.1.1 Rancangan Alur Sistem

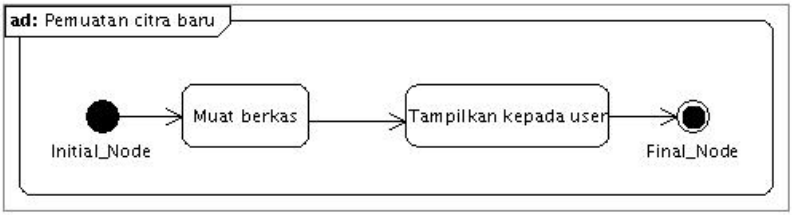
Sistem kompresi citra terdiri atas dua proses besar. Proses tersebut adalah kompresi dan dekompresi. Proses kompresi secara keseluruhan ditunjukkan oleh gambar 3.1.



Gambar 3.1 Sistem Kompresi Citra

3.1.1.1 Membaca Citra

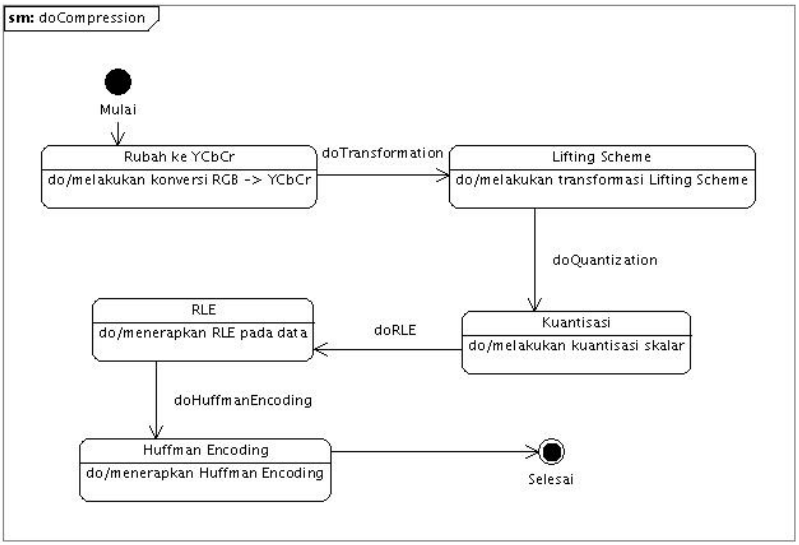
Gambar 3.2 menunjukkan *activity diagram* untuk proses pemuatan gambar yang akan dipampatkan. Proses yang terjadi adalah `class MainWindow` menerima perintah dari user untuk memuat citra dan kemudian menampilkan citra tersebut.



Gambar 3.2 Membaca Citra

3.1.1.2 Kompresi

Langkah terperinci untuk proses kompresi ditunjukkan oleh diagram 3.3. Proses ini adalah rincian dari proses yang digambarkan pada gambar 3.1.



Gambar 3.3 Rincian Proses Kompresi

Untuk memperjelas proses kompresi, maka berikut adalah *pseudocode* sesuai dengan urutan proses kompresi.

3.1.1.2.1 Konversi dari RGB Menjadi YcbCr

Proses konversi dari komponen RGB menjadi YCbCr, dilakukan dengan melakukan iterasi terhadap seluruh data merah (R[]), hijau (G[]), biru (B[]). Masing-masing nilai yang tersimpan pada ketiga komponen tersebut dirubah menjadi komponen Y (Y[]), Cb (Cb[]), dan Cr (Cr[]) menggunakan rumusan 2.1.

```
1) R[n],G[n],B[n]=komponen merah, hijau, dan
   Biru  pada data pada index n
2) Y[n],Cb[n],Cr[n]=komponen Y, Cb, Cr secara
   berurutan pada data pada index n

3) pada tiap baris:
4) mulai dari n=0 hingga panjang citra:
5)   Y[n]=0.299*R[n]+0.587*G[n]
        +0.114*B[n]
6)   Cb[n]=(-0.1687)*R[n]-0.3313*G[n]
        +0.5*B[n]+128
7)   Cr[n]=0.5*R[n]-0.4187*G[n]-
        0.0813*B[n]+128
```

3.1.1.2.2 Lifting Scheme Maju

Proses lifting scheme diterapkan menggunakan rumus 2.9 hingga 2.12. Proses *lifting scheme* pertama kali dikakukan pada semua baris citra dengan lebar yang sama dengan lebar asli. Setelah semua baris dikenakan transformasi *lifting scheme*, maka transformasi diterapkan pada semua kolom citra dengan tinggi sesuai dengan tinggi citra asli.

Langkah di atas kemudian di ulang kembali sejumlah banyak dekomposisi yang dikehendaki. Namun lebar dan tinggi dari data yang terkena proses transformasi berkurang ½ dari ukuran lebar dan tinggi pada tahapan sebelumnya.

```
1) d_level=nilai dekomposisi level
```

```
2) selama level < d_level:
3)lakukan pada tiap baris:
4)   pisahkan data menjadi dua, index genap di
       sebelah kiri, dan index ganjil di
       sebelah kanan
5)   tengah=setengah panjang pada tiap level

6)   mulai dari n=0 hingga tengah:
       data[tengah]=data[tengah]+sqrt(3)
         *data[tengah+n]

7)   data[tengah]=data[tengah]-(sqrt(3)/4
       *data[0]-(((sqrt(3)-2)/4)*data[tengah-1]
       )

8)   mulai dari n=1 hingga tengah:
       data[tengah+n]=data[tengah+n]-(sqrt(3
         )/4*data[n]-(((sqrt(3)-2)/4)
         *data[n-1])

9)   mulai dari n=0 hingga tengah-1:
       data[n] = data[n] - data[tengah+n+1]
     data[tengah-1]=data[tengah-1]-data[tengah

10)  mulai dari n=0 hingga tengah:
       data[n]=((sqrt(3)-1)/sqrt(2))*data[n]
         data[n+tengah]=((sqrt3+1.0)/sqrt2)
         *S[n+half]
11) Lakukan langkah 2-10 untuk data kolom
```

3.1.1.2.3 Kuantisasi

Proses kuantisasi membaca data pada cira satu-satu. Kemudian masing-masing data dikenakan kuantisasi berbasis deadzone.

```
1) step_size = ukuran step size

2) Jika sumber belum habis:
3)   jika |data[i]| < step_size:
4)      data[i] = 0
5)   else
6)      data[i] = data[i] / step_size
7)   i = i + 1
```

3.1.1.2.4 Run Length Encoding pada Proses Kompresi

Proses Run Length Encoding (RLE), melakukan pembacaan data satu-per-satu. Ketika membaca sebuah data, RLE melihat pada data beriktunya apakah kedua data tersebut bernilai sama. Jika sama, maka RLE melakukan penelusuran hingga data keberapa terjadi keseragaman nilai. Pada akhirnya, data yang tertulis pada struktu data penampung hasil encoding adalah (nilai,nilai,jumlah berurut-2). Namun jika tidak terjadi, maka data yang tertulis adalah nilai dari data itu saja.

```
1) enc=penampung data hasil encoding

2) Selama data belum habis terbaca:
3)   jika data[i+1] == data[i]:
4)     n=i+2
5)     Selama benar:
6)       jika data[n]!=data[i]:
7)           break
8)       else
9)           n = n + 1

10)    tuliskan “data[i],data[i],(n-i-2)” pada
           enc
11)    i = n

12)  else:
13)    tuliskan “data[i]” pada enc
```

3.1.1.2.5 Huffman Encoding

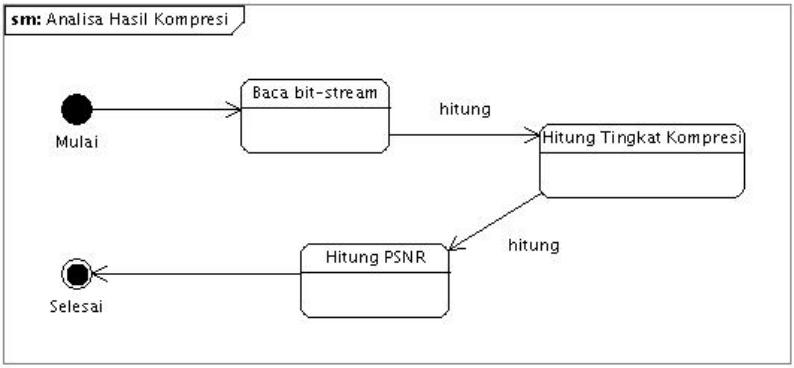
Untuk melakukan proses huffman, data perlu diurutkan terlebih dahulu berdasarkan tingkat kemunculanya. Kemudian dibangunlah pohon huffman. Dari bentukan pohon huffman inilah *code* untuk masing-masing data diperoleh. Kemudian seluruh data dirubah representasi bit-nya menggunakan code baru bentukan dari pohon huffman.

```
1) Baca data
2)hitung frekuensi seluruh data
```

3) urutkan data dari kecil ke besar sesuai frekuensi
4) bangun pohon huffman berdasarkan urutan frekuensi data
5) Berikan code untuk masing-masing data berdasarkan hasil pohon huffman.

3.1.1.3 Melakukan Analisa Hasil Kompresi

Proses menganalisa hasil kompresi dapat dilihat pada gambar 3.4. Yang pertama dilakukan adalah dengan membaca *bit-stream* hasil kompresi. Dari data ini dapat dihitung tingkat kompresi dan nilai *Peak Signal to Noise Ratio*. Kedua hasil perhitungan inipun disimpan.

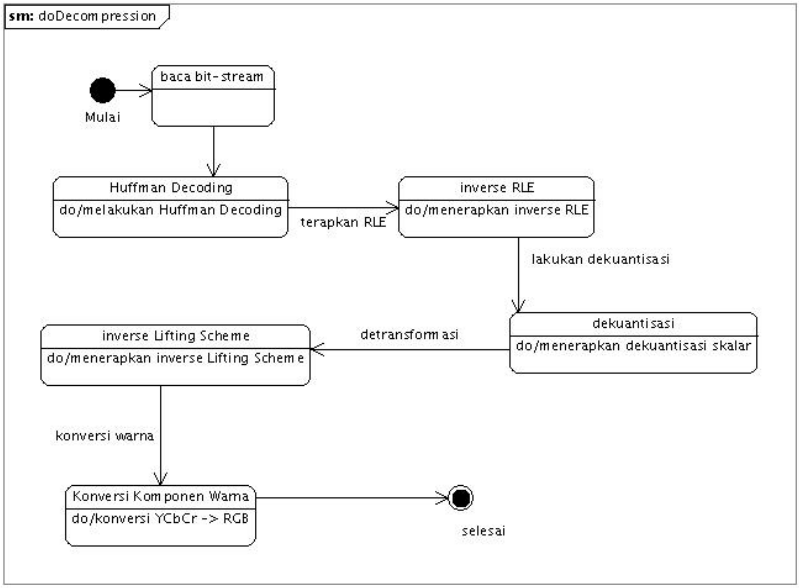


Gambar 3.4 Menganalisa Hasil Kompresi

3.1.1.4 Dekompresi

Proses dekompresi adalah proses pembacaan data hasil kompresi. Yang pertama kali dilakukan adalah membaca data berupa *bit-stream*. Pada proses pembacaan ini, data dimasukkan ke dalam struktur data gambar. Struktur data gambar inilah yang kemudian diproses oleh proses dekompresi berikutnya, yaitu *huffman decoding*, RLE mundur, dekuantisasi skalar, lifting scheme mundur, dan konversi warna dari YCbCr menjadi RGB.

Untuk memperjelas proses yang terjadi pada tahapan *huffman encoding* hingga konversi warna, *pseudocode* untuk masing-masing proses tersebut dituliskan pada subbab 3.1.1.4.1 hingga 3.1.1.4.5.



Gambar 3.5 Proses Dekompresi

3.1.1.4.1 Huffman Decoding

Huffman decoding melakukan proses mundur dari huffman encoding. Data hasil encoding dicocokkan dengan code encoding yang telah disimpan. Dari langkah ini diperoleh representasi bit asli dari masing-masing data.

```
1) Baca data bit-stream
2) Tentukan representasi bit pada data sesuai
   dengan nilai encoding yang telah disimpan
```

3.1.1.4.2 Run Length Encoding Pada Proses Dekompresi

Proses Run Length Decoding (RLD)membaca data hasil dari proses *huffman decoding*. RLD membaca dua data yang bersebelahan sekaligus (data[n], data[n+1])). Jika kedua data tersebut bernilai sama, maka dibaca data berikutnya (data[n+2]) yang dimana data ini menunjukkan berapa jumlah data berniali sama yang akan dituliskan pada struktur data citra. Jika data[n] dan data[n+1] tidak sama, maka yang ditulis adalah data[n] saja.

```
1) dec = variabel penampung hasil decoding
2) i = index data pada posisi pertama
3) Baca sumber

4) Jika sumber belum habis:
5)   baca data pada index i
6)   baca data pada index i + 1

7)    jika data[i] = data[i + 1]:
8)      n =  i + 2
9)      tuliskan data pada i sebanyak n + 2
kali    pada dec
10)   else
        tulis data pada index i
```

3.1.1.4.3 Dekuantisasi

Proses dekuantisasi melakukan pembacaan terhadap semua data. Jika data yang dibaca bernilai tidak sama dengan 0, maka kalikan nilai ini dengan nilai stepsize. Jika data bernilai 0, maka nilai dari data tidak berubah.

```
1) step_size = ukuran step size
2) Baca sumber

3) Jika sumber belum habis:
4)    jika |data[i]| !=0:
5)      data[i] = data[i]*step_size
6)    i = i + 1
```

3.1.1.4.4 Lifting Scheme Mundur

Proses lifting scheme mundur menerapkan rumus 2.13 hingga 2.16. Tahapan pertama adalah melakukan transformasi mundur pada semua baris. Lebar baris yang dikenakan proses transformasi sesuai dengan lebar pada level dekomposisi tertinggi. Setelah itu, proses transformasi diterapkan pada semua kolom. Tinggi kolom berukuran sama dengan lebar baris.

Proses di atas dilanjutkan dengan menambah lebar dan tinggi jangkauan data yang terkena proses transformasi sebanyak 2 kali lipat.

```
1)  Mulai dari level dekomposisi tertinggi:

2)  Lakukan proses berikut pada tiap baris:
3)    tengah=setengah panjang pada level

4)    mulai dari n=0 hingga tengah:
5)      data[n]=((sqrt(3)+1)/sqrt(2))*data[n]
        data[n+tengah]=((sqrt3-1.0)/sqrt2)
          *S[n+half]

6)    mulai dari n=0 hingga tengah-1:
7)      data[n] = data[n] + data[tengah+n+1]
8)    data[tengah-1]=data[tengah-1]+data[tengah

9)    data[tengah]=data[tengah]+(sqrt(3)/4
        *data[0]+(((sqrt(3)-2)/4)*data[tengah-1
        ])
10)   mulai dari n=0 hingga tengah:
11)     data[tengah+n]=data[tengah+n]+(sqrt(3
          )/4*data[n]+(((sqrt(3)-2)/4)
          *data[n-1])

12)   mulai dari n=0 hingga tengah:
13)     data[tengah]=data[tengah]-sqrt(3)
        *data[tengah+n]

14)   gabungkan data pada bagian kiri dan kanan
15)     menjadi satu sesuai dengan urutannya

16)   Lakukan proses 3-15 pada tiap   kolom
17)   Ulangi langkah di atas pada level yang
lebih      rendah
```

3.1.1.6.5 Konversi YCbCr menjadi RGB

Proses perubahan komponen warna dari YcbCr menjadi RGB mengikuti rumus 2.2. Masing-masing data dibaca dan masing-masing komponen dari data tersebut yang bertipe Y, Cb, dan Cr dirubah menjadi RGB.

```
1) R[n],G[n],B[n]=unsur merah, hijau, dan Biru
```

```
   pada data pada index n
2) Y[n],Cb[n],Cr[n]=unsur Y, Cb, Cr secara
   berurutan pada data pada index n

3) pada tiap baris:
4)    mulai dari n=0 hingga panjang citra:
5)       R[n]=Y[n]+1.402*Cr[n]-128
6)       G[n]=Y[n]-0.34414*(Cb[n]-128)-
               0.71414*(Cr[n]-128)
7)       B[n]=Y[n]+1.772*(Cb[n]-128)
```

3.1.2 Class Diagram

Untuk menerapkan perancangan yang dijelaskan pada subbab 3.1.1, ditentukanlah *class diagram*nya. Penentuan ini dilakukan untuk mempermudah pembuatan sistem melalui pemrograman. Class diagram dapat dilihat pada gambar 3.6.

Dari proses penentuan ini, dapat ditentukan *class* sebagai berikut:

•`MainWindow`

Class ini memiliki tugas sebagai berikut:

1. Mengatur interaksi user dengan *class* lainnya.
2. Menghitung kualitas kompresi
3. Mengatur hubungan aplikasi dan database.

•`Compression`

Class ini memiliki tugas sebagai berikut:

1. Melakukan proses kompresi dan dekompresi sesuai dengan alurnya. Untuk memenuhi alur ini, maka dibuat pula *class* sebagai berikut:
   - `ColorComponentTransformation`
   1. Melakukan proses perubahan warna dari tipe *RGB* ke dalam format YCbCr.

   - `D4LiftingScheme`
   1. Melakukan proses transformasi maju dan mundur citra menggunakan metode *Lifting Scheme* berdasarkan *wavelet* Daubechies 4.

- `ScalarQuantization`

1. Melakukan proses kuantisasi dan dekuantisasi menggunakan metode *scalar quantization* dengan menerapkan wilayah *dead-zone.*

- `EntropyCoding`

1. Mengatur proses entropy encoding dan decoding
2. Menyiapkan dan meyimpan struktur data untuk proses entropy encoding dan decoding

- `RLE`

1. Melakukan proses pengurangan dan penambahan citra menggunakan metode *Run Length Encoding.*

- `HuffmanCoding`

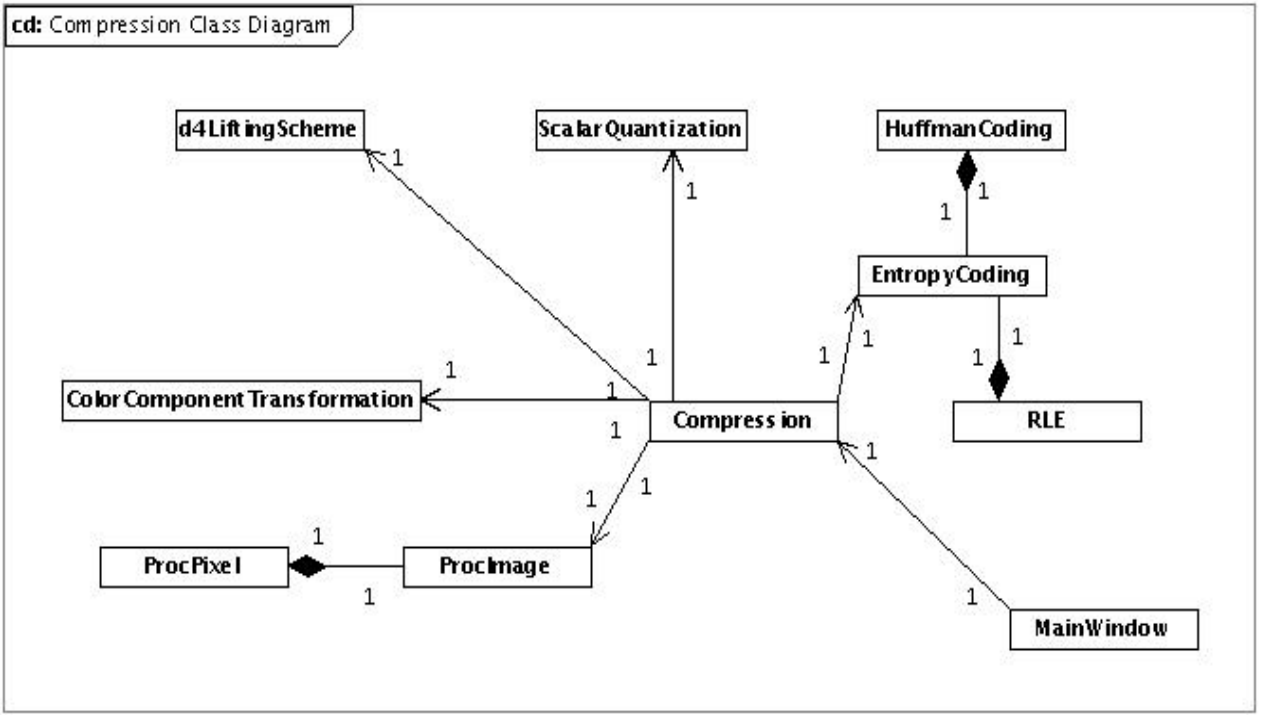
1. Melakukan proses perubahan sistem pengkodean data pada citra menggunakan metode *Huffman Coding*.

- Gambar (dalam gambar 3.6 bernama `ProcImage`)

1. Merubah struktur data citra agar sesuai dengan kebutuhan kompresi. Proses ini dibantu oleh `class UserInterface`.
2. Merubah struktur data citra untuk dikembalikan ke dalam struktur data citra asli. Proses ini dibantu oleh `class UserInterface`.
3. Mengelola perubahan pada struktur data citra.
4. Mengetahui *metadata* terkait proses kompresi yang sedang berlangsung.

- `ProcPixel`

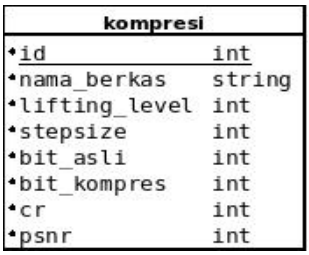
1. Menampung komponen warna



Gambar 3.6 Class Diagram untuk sistem kompresi

3.1.3 Rancangan Database

Pada penelitian ini, metadata terkait dengan proses kompresi perlu disimpan ke dalam sebuah database. Hal ini dirancanang sedemikian rupa untuk memunginkan penggunaan metadata ini dalam proses analisa hasil kompresi terhadap seluruh data. Rancangan database dapat dilihat pada gambar 3.7.

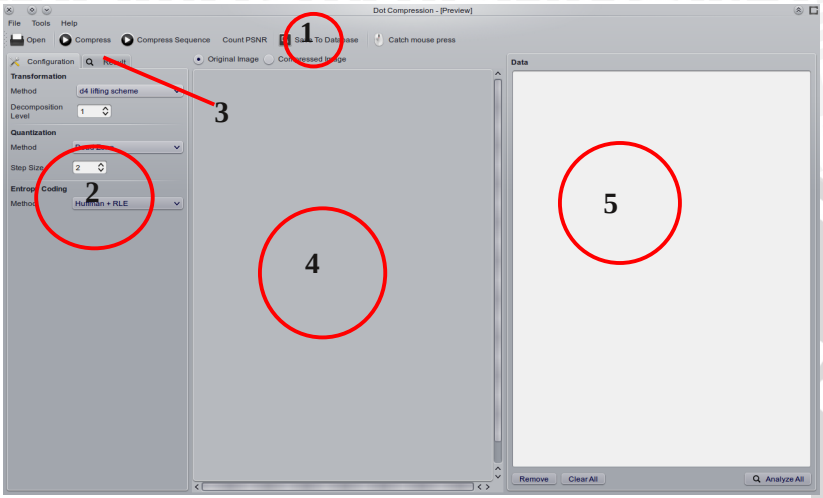


Gambar 3.7 Skema Database Sistem

Dari gambar 3.7 dapat dilihat bahwa primary key untuk tabel ada *field* id. Field nama_berkas menyimpan nama berkas yang digunakan dalam proses kompresi. *Field* lifting_level menyimpan berapa nilai dekomposisi *lifting scheme* yang digunakan. *Field* stepsize menyimpan nilai *stepsize* yang digunakan untuk proses kuantisasi skalar. *Field* bit_asli menyimpan berapa jumlah bit citra asli, sementara itu bit_kompres menyimpan berapa jumlah bit citra hasil kompresi. *Field* cr menyimpan rasio kompresi. *Field* psnr meyimpan nilai PSNR (Peak Signal-to-Noise Ratio) antara citra asli dan citra hasil kompresi.

3.1.4 Rancangan Antarmuka

Rancangan antarmuka secara keseluruhan dapat dilihat pada gambar 3.8. Pada rancangan antarmuka tersebut, terdapat 5 bagian. Bagian 1 menunjukkan wilayah toolbar untuk melakukan operasi-operasi utama. Bagian 2 berguna untuk memasukkan parameter yang terkait dengan ujicoba kompresi. Bagian 3 menampilkan hasil untuk proses kompresi. Bagian 4 memungkinkan pengguna untuk melihat citra asli dan citra kompresi. Bagian 5 menampilkan *metadata* mengenai proses kompresi yang disimpan di dalam database.



Gambar 3.8 Rancangan antarmuka utama

3.2 Pengumpulan Data

Data citra uji coba yang digunakan adalah data uji coba yang diperoleh dari http://www.imagecompression.info/ sebanyak 12 buah data. Data citra uji dari tempat tersebut bertipe ppm.

3.3 Rancangan Penelitian

Penelitian ini memiliki tiga proses besar, yaitu proses *encoding*, *decoding*, dan analisa hasil. Berikut adalah rincian dari tiga proses besar ini.

3.3.1 Encoding

Proses *encoding* adalah proses dimana citra asli dipampatkan ke dalam ukuran yang lebih kecil. Citra masukan berupa citra ppm. Tahapan *encoding* adalah:

1. Merubah data ke dalam struktur data yang sesuai untuk proses kompresi.
2. Merubah komponen warna dari RGB ke bentuk YCbCr.
3. Melakukan proses *forward transformation lifting scheme* pada wavelet d4. Jumlah dekomposisi yang diujikan adalah 1,2,4,6 dan 8.
4. Melakukan proses kuantisasi menggunakan metode *scalar*

quantization dengan menerapkan *dead zone*. Ukuran *dead zone* yang digunakan adalah 2 x ukuran *step size*. Sementara itu ukuran step size yang diujikan adalah 2, 4, 6 dan 8.

5. Menjalankan algoritma encoding RLE.
6. Menghitung tingkat kemunculan masing-masing simbol
7. Menjalankan proses encoding huffman coding.
8. Menyimpan metadata terkait proses kompresi ini ke dalam *database*.

3.3.2 Decoding

Proses *decoding* adalah proses dimana citra hasil kompresi ditampilkan seperti ketika citra tersebut belum terpampat. Karena proses kompresi bersifat *lossy*, maka hasil *decoding* tidak sama persis dengan sebelum citra dipampatkan. Proses *decoding* adalah sebagai berikut:

1. Melakukan proses decoding huffman coding.
2. Melakukan proses decoding RLE.
3. Melakukan proses dekuantisasi.
4. Melakukan proses *inverse transform* pada metode d4 *lifting scheme*.
5. Merubah komponen warna dari YCbCr ke bentuk RGB.
6. Merubah bentuk data ke bentuk citra agar dapat dilihat oleh pengguna.

3.3.3 Rancangan Uji Coba

Proses analisa hasil adalah proses dimana hasil kompresi secara keseluruhan dihitung kualitasnya berdasarkan kriteria uji coba dan parameter perhitungan kualitas.

Jadi, dari 10 citra uji coba, akan diproses ke dalam sistem kompresi dengan mengujikan variabel-variabel yang berbeda pada masing-masing tahapan, dan kemudian mengukur nilai PSNR dan rasio kompresi (C.R). Hasil dari ujicoba ini akan direpresentasikan ke dalam tabel dan diplotkan ke dalam grafik.

Tabel 3.2 Contoh tabel perbandingan hasil berdasarkan nilai PSNR

	Ukuran stepsize = N				
	L.D = 2	L.D = 4	L.D = 6	L.D = 8	

Citra 1	*Nilai PSNR*	*Nilai PSNR*	*Nilai PSNR*	*Nilai PSNR*	
Citra 2	*Nilai PSNR*	*Nilai PSNR*	*Nilai PSNR*	*Nilai PSNR*	
Citra 3	*Nilai PSNR*	*Nilai PSNR*	*Nilai PSNR*	*Nilai PSNR*	

Tabel 3.3 Contoh tabel perbandingan hasil berdasarkan rasio kompresi.

	Ukuran stepsize = N				
	L.D = 2	L.D = 4	L.D = 6	L.D = 8	
Citra 1	*CR%*	*CR%*	*CR%*	*CR%*	
Citra 2	*CR%*	*CR%*	*CR%*	*CR%*	
Citra 3	*CR%*	*CR%*	*CR%*	*CR%*	

Bentuk tabel dapat dilihat pada tabel 3.2 dan tabel 3.3.

3.4 Contoh Perhitungan Manual

Dimisalkan, data adalah sebuah array S dengan panjang 16. Maka S adalah:

20	23	24	25	21	15	16	26	27	30	28	29	21	24	25	19

Representasi biner untuk nilai di atas adalah:

Tabel 3.4 Representasi Bit dari Data Contoh

S	Biner	S	Biner
20	10100	27	11011
23	10111	30	11110
24	11000	28	11100
25	11001	29	11101
21	10101	19	10011
15	01111		

16	10000		
26	11010		

Demi mempermudah contoh ini, nilai pada S disimpan pada data bertipe 5 bit. Hal ini diambil karena pada contoh di atas, bilangan tertinggi, 30, berjumlah 5 bit. Dalam contoh ini akan diperlihatkan bagaimana nilai asli S akan dipampatkan.

Dimisalkan pula bahwa nilai pada S merepresentasikan citra *greyscale*. Dengan demikian S tidak perlu ditransformasi ke dalam bentuk YCbCr. Pendekatan ini diambil untuk memudahkan contoh perhitungan manual sistem kompresi citra pada penelitian ini.

3.4.1 Transformasi Wavelet

Tahapan dalam transformasi wavelet d4 menggunakan metode lifting scheme adalah splitting → update 1 → predict → update 2 → normalisasi.

3.4.1.1 Splitting

Array S mengalami proses transformasi wavelet d4 menggunakan metode *lifting scheme*. Maka proses pertama adalah memisahkan data pada index ganjil dan genap (data yang diwarnai pada contoh dibawah) pada S, dimana data dengan index genap berada disebelah kiri dan yang ganjil berada disebelah kanan. Hasilnya adalah:

20	23	24	25	21	15	16	26	27	30	28	29	21	24	25	19

20	24	21	16	27	28	21	25	23	25	15	26	30	29	24	19

3.4.1.2 Update 1

Proses update 1 berjalan sesuai dengan rumus 2.9. Hasil kalkulasi rumus 2.9 terhadap array S adalah:

59	67	46	61	78	78	62	57	23	25	15	26	30	29	24	19

3.4.1.3 Predict

Proses predict berjalan sesuai dengan rumus 2.10. Hasil kalkulasi rumus 2.10 terhadap array S adalah:

59	67	46	61	78	78	62	57	-23	0	0	2	0	0	2	-1

3.4.1.4 Update 2

Proses udpate 2 berjalan sesuai dengan rumus 2.11. Hasil kalkulasi S dengan rumus 2.11 adalah:

59	67	44	61	78	76	63	80	-23	0	0	2	0	0	2	-1

3.4.1.5 Normalisasi

Proses akhir dari *forward transformation* adalah normalisasi. Tahapan ini berjalan sesuai dengan rumus 2.12. Hasil kalkulasi S dengan rumus 2.12 adalah:

30	34	22	31	40	39	32	41	-44	0	0	3	0	0	3	-1

3.4.2 Kuantisasi Skalar

Pada tahapan ini, dimisalkan ukuran dari *step size* adalah 4, maka, nilai dari *threshold* adalah antara -4 dan 4. Jika ini diterapkan pada S menggunakan rumusan:

$$y = \begin{cases} 0 & |x| \le 4 \\ x/\Delta & |x| > 4 \end{cases}$$

maka hasilnya adalah:

7	8	5	7	10	9	8	10	-11	0	0	0	0	0	0	0

3.4.3 RLE

Dengan menerapkan metode RLE pada S, maka hasilnya adalah:

7	8	5	7	10	9	8	10	-11	0	0	5

3.4.4 Huffman Coding

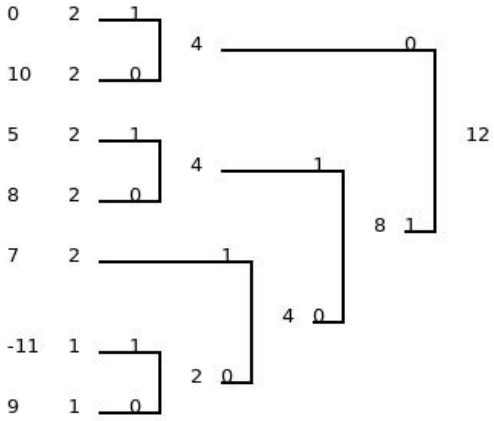
Metode *huffman coding* dimulai dengan menghitung tingkat kemunculan masing-masing simbol. Hasilnya adalah:

Simbol	7	8	5	10	9	-11	0
Kemunculan	2	2	2	2	1	1	2

Jika diurutkan maka:

Simbol	9	-11	7	8	5	10	0
Kemunculan	1	1	2	2	2	2	2

Langkah selanjutnya adalah dengan menerapkan metode *huffman coding.* Hasilnya adalah seperti tampak pada gambat 3.9.



Gambar 3.9 Hasil *Huffman Encoding*

3.4.5 Hasil Akhir Kompresi

Dari proses perhitungan *huffman coding*, didapatkan hasil sebagai beriktu:

Data	9	-11	7	8	5	10	0
Encoding	1000	1001	101	110	111	00	01

Sehingga, hasil akhir dari keseluruhan proses kompresi citra untuk data S adalah 101.110.111.101.00.1000.110.00.1001.01.01.111. Jumlah bits yang digunakan adalah 34 bits. Dari hasil ini, tingkat kompresi adalah

sebesar 57.5% $\left(\frac{80-34}{80} * 100\right)$.

3.4.6 Decoding Huffman coding dan RLE

Karena huffman coding dan RLE bersifat *lossles*, atau tidak ada data yang hilang, maka hasil decodingnya adalah sama seperti nilai asli sebelum encoding. Yaitu:

- 101 → 7
- 110 → 8
- 111 → 5
- 101 → 7
- 00 → 10
- 1000 → 9
- 110 → 8
- 00 → 10
- 1001 → -11
- 01 → 0
- 01 → 0
- 111 → 5

Kemudian hasil ini pun mengalami proses decode berdasarkan metode RLE. Data hasil decode huffman coding di atas, dapat dilihat bahwa simbol 0 muncul 7 kali (0 0 5) dan simbol lainnya muncul satu kali. Sehingga hasil akhirnya adalah:

7	8	5	7	10	9	8	10	-11	0	0	0	0	0	0	0

3.4.7 Proses Dequantisasi

Proses dequantisasi berjalan berdasarkan rumusan:

$$x = \begin{cases} 0 & |y| = 0 \\ v * \Delta & v\,! = 0 \end{cases}$$

Dari rumusan tersebut, maka hasil dequantisasi adalah:

28	32	20	28	40	36	32	40	-44	0	0	0	0	0	0	0

3.4.8 Inverse wavelet – normalisasi

Proses *inverse wavelet transformation* berjalan secara terbalik

terhadap *forward transformation*. Tahapan pada inverse transformation adalah normalisasi → update 2 → prediction → update 1 → merge.

Proses normalisasi pada data S menggunakan rumus 2.11 adalah

54	61	38	54	77	69	61	77	-22	0	0	0	0	0	0	0

3.4.9 Inverse wavelet – update 2

Menerapkan rumus 2.12 pada data S akan merubah data S menjadi:

54	61	38	54	77	69	61	55	-22	0	0	0	0	0	0	0

3.4.10 Inverse wavelet – predict

Menerapkan rumus 2.13 pada data S, akan merubah data S menjadi:

54	61	38	54	77	69	61	55	21	22	12	20	29	24	21	19

3.4.11 Inverse wavelet – update 1

Pada tahapan ini, data pada S mengalami perubahan berdasarkan rumus 2.14. Sehingga data S menjadi:

17	22	17	19	26	27	24	22	21	22	12	20	29	24	21	19

3.4.12 Inverse wavelet – merge

Pada tahapan akhir dari proses *inverse wavelet transformation*, data pada S perlu digabungkan atau mengalami proses *merging*. Perlu diingat bahwa data pada bagian tengah kiri pada S adalah data yang menempati indeks genap pada data S asli. Sementara itu data pada tengah kanan pada S adalah data yang menempati indeks ganjil pada data S asli. Proses penggabungan ini bertujuan untuk menempatkan kembali data-data S pada posisi semula.

Hasil dari proses penggabungan ini adalah:

17	21	22	22	17	12	19	20	26	29	27	24	24	21	22	19

3.4.11 Pengamatan terakhir

Dengan membandingkan data S asli dan data S hasil kompresi, dapat dilihat adanya perbedaan. Data S asli adalah:

20	23	24	25	21	15	16	26	27	30	28	29	21	24	25	19

Sementara itu data S hasil kompresi adalah:

17	21	22	22	17	12	19	20	26	29	27	24	24	21	22	19

Perbedaan pada data S asli dengan data S hasil kompresi menunjukkan bahwa sistem kompresi bersifat *lossy.* Sehungga terdapat informasi yang hilang demi mendapatkan tingkat kompresi yang besar.

Untuk mengukur kualitas kompresi, dapat digunakan rumus *Peak Signal to Noise Ratio (PSNR)* berdasarkan rumus 2.22. Hasil PSNR untuk kompresi data S adalah 38.38 db.

BAB IV
HASIL DAN PEMBAHASAN

4.1 Deskripsi Sistem

Aplikasi kompresi pada penelitian ini melakukan proses kompresi terhadap citra dengan ukuran 256x256 pixel. Ukuran citra yang digunakan harus bernilai 2^N, dimana N adalah jumlah dekomposisi maksimal yang dapat diterapkan pada citra. Proses kompresi melakukan pengurangan terhadap jumlah bit yang digunakan oleh citra dengan menjaga agar kualitas citra tidak rusak. Penelitian ini mempunyai fokus terhadap kompresi berjenis *lossy.* Dengan kompresi *lossy*, maka terdapat pengurangan informasi pada citra hasil kompresi jika dibandingkan dengan citra asli. Sehingga, salah satu objektif dari kompresi citra berbasis *lossy* adalah pengurangan jumlah bit sebesar-besarnya dengan menjaga kualitas citra berdasarkan ukuran nilai PSNR (*Peak Signal to Noise Ratio*).

4.2 Struktur Data

Dalam penelitian ini, dibuat 4 struktur data untuk menampung data kompresi. Untuk struktur data citra dapat dilihat pada gambar 4.1. Nilai pixel disimpan ke dalam ProcPixel. ProcPixel sendiri adalah sebuah *class* yang memiliki *attribute* berupa *red*, *green*, *blue* yang ketiganya bertipe *float*.

Untuk merepresentasikan sebuah citra, ProcPixel ditampung oleh kontainer Vector. Vector ini merepresentasikan nilai pixel pada satu baris pada gambar. Kemudian vektor pada masing-masing baris pun ditampung oleh vektor. Sehingga, kombinasi vektor ini pun membentuk vektor 2 dimensi. Panjang untuk masing-masing baris pada vektor sama dengan lebar dari gambar. Dan tinggi dari vektor sama dengan tinggi pada gambar.

Susunan struktur data untuk menampung citra dapat dilihat pada tabel 4.1. Atribut transformationType menunjukkan jenis transformasi yang akan dipakai. Dalam penelitian ini, atribut ini menyimpan nilai 1 yang berarti tipe transformasi yang digunakan adalah *lifting scheme* dari wavelet Daubechies 4. Atribut decompositionLevel menyimpan jumlah dekomposisi yang digunakan. Atribut quantizationType menyimpan nilai 1 yang menunjukkan bahwa tipe kuantisasi yang digunakan adalah

kuantisasi skalar. Atribut qSize menyimpan berapa nilai *stepsize* yang digunakan. Atribut usedRLE menunjukkan apakah metode *Run Length Encoding* (RLE) digunakan, dan dalam penelitian ini, atribut tersebut bernilai *true*. Atribut EntropyCoding menyimpan nilai 1 yang berarti metode *entropy encoding* yang digunakan adalah RLE dan huffman. Atribut filename menyimpan nama file yang menjadi citra uji. Atribut height dan width menyimpan dimensi dari citra. Sementara itu atribut image adalah vektor dalam vektor. Vektor ini bertipe QVector yang merupakan library vektor dari framework Qt. Ilustrasi atribut image dapat dilihat pada gambar 4.1.

Tabel 4.1 Bentuk struktur data penampung nilai citra

1	Int transformationType
2	int decompositionLevel
3	int quantizationType
4	int qSize
5	bool usedRLE
6	int entropyCoding
7	QString filename
8	int height
9	int width
10	Qvector<QVector<ProcPixel> > image

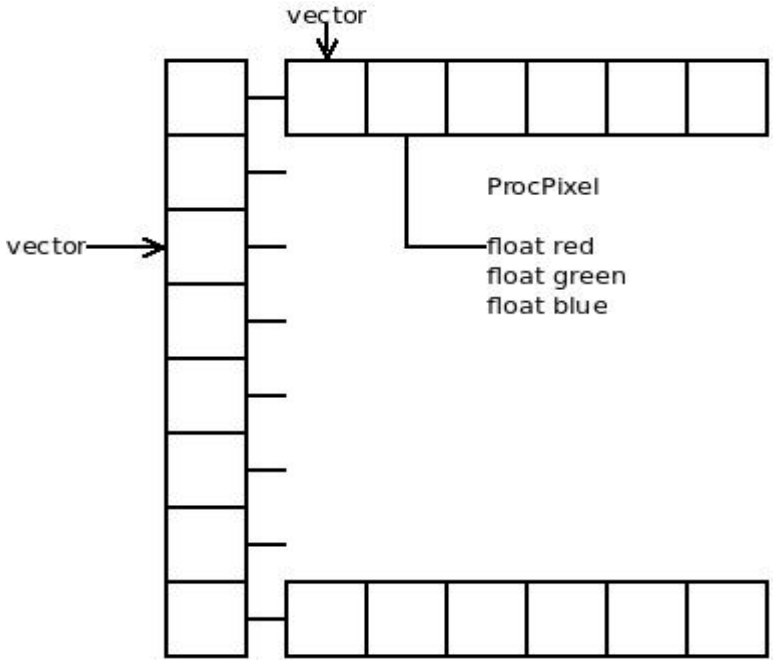
Sementara itu, bentuk struktur data untuk class ProcPixel dapat dilihat pada tabel 4.2. ProcPixel memiliki atribut berupa red, green dan blue yang dimana masing-masing menampung nilai merah, hijau dan biru. Namun dalam proses kompresi, atribut red, green dan blue secara berurutan menampung nilai Y, Cb dan Cr.

Tabel 4.2 Bentuk struktur data class ProcPixel

1	Float red
2	float green
3	float blue

Proses Encoding melakukan proses pembuangan pengulangan (*redundancy*) dari data. Pada proses ini, panjang sebuah baris pada struktur data vektor di atas dapat berkurang. Pada struktur data

vektor, penghapusan atau penambahan data ditengah-tengah struktur data merupakan proses yang lambat. Sehingga pembuatan struktur data baru menjadi hal yang praktis untuk menampung data hasil pengkodean entropy.



Gambar 4.1 Struktur Data Citra

Struktur data pada tahapan pengkodean entropy ada tiga. Struktur data pertama untuk menampung frekuensi kemunculan symbol. Struktur data kedua adalah untuk menampung symbol hasil proses *Run Length Encoding*. Struktur data ketiga adalah untuk menampung *bit-stream* hasil akhir dari keseluruhan proses kompresi.

Struktur data frekuensi digunakan untuk menampung tingkat kemunculan data. Data mengenai frekuensi penting dalam proses *huffman encoding*. Struktur data frekuensi berupa vektor yang memiliki panjang sama dengan jumlah symbol unik pada data. Vektor ini menampung *class* SymbolData. Atribut dari *class* tersebut dapat dilihat pada tabel 4.1.

Tabel 4.3 Atribut SymbolData

1	Int data;
2	int frequency;
3	bool isLeft;
4	bool isRight;
5	bool hasChildren;
6	SymbolData *left;
7	SymbolData *right;
8	SymbolData *parent;
9	QbitArray bitData;

Sesuai dengan tabel 4.3, int data menyimpan nilai symbol dalam bentuk integer. Tingkat kemunculan symbol ditunjukkan oleh frequency. Atribut isLeft, isRight dan hasChildren adalah petunjuk yang digunakan pada proses pembangunan pohon huffman. Atribut isLeft menunjukkan apakah simbol ini berada di *node* sebelah kiri. Jika tidak, maka isRight bernilai benar. Sementara itu, hasChildren bernilai benar jika symbol ini adalah sebuah anak dalam struktur pohon.

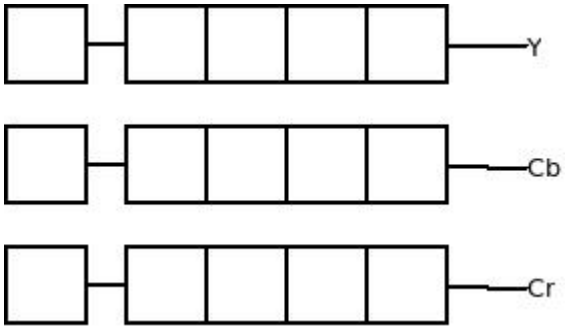
Pointer left, right, parent berguna untuk menunjukkan (secara berurut) anak pada sebelah kiri, sebelah kanan dan orang tua dari simbol. Pada inisialisasi, pointer ini bernilai *NULL*.

Sementara itu bitData adalah variabel bertipe QbitArray yang menampung nilai bit sesuai dengan pembentukan pohon huffman. Atribut bitData akan digunakan dalam proses huffman encoding.

Data hasil pengkodean RLE disimpan pada struktur data berupa SymbolStream. SymbolStream ini adalah sebuah vektor yang memiliki panjang 3. Pada baris pertama vektor SymbolStream, vektor menampung semua komponen pertama (komponen Y) hasil pengkodean RLE. Baris kedua menampung semua komponen Cb, sedangkan baris ketiga menampung komponen Cr. Masing-masing data dari ketiga komponen tersebut disimpan dalam vektor yang kemudian mengisi ketiga ruang pada vektor SymbolStream. Tipe data yang disimpan pada vektor di masing-masing baris adalah integer. Untuk lebih jelasnya, struktur data SymbolStream dapat

dilihat pada gambar 4.2.

Hasil akhir dari proses entropy coding adalah vektor yang menampung nilai bit hasil pengkodean huffman. Struktur data ini bernama CodeStream. CodeStream adalah vektor yang pada masing-masing ruang menampung data bertipe QbitArray. Panjang dari CodeStream adalah jumlah seluruh simbol hasil dari pengkodean entropy.



Gambar 4.2 Struktur SymbolStream

Struktur data *frequency*, *SymbolStream* dan *CodeStream* adalah bagian dari *class EntopyCoding.* Deklarasi struktur data tersebut dapat dilihat pada tabel 4.4.

Tabel 4.4 Deklarasi struktur data pada *class EntopyCoding*

1	QVector<SymbolStream> frequency
2	QVector<QBitArray> codeStream
3	QVector<QVector<int> > symbolStream

4.3 Kompresi dan Dekompresi Citra

4.3.1. Konversi komponen warna menjadi YcbCr

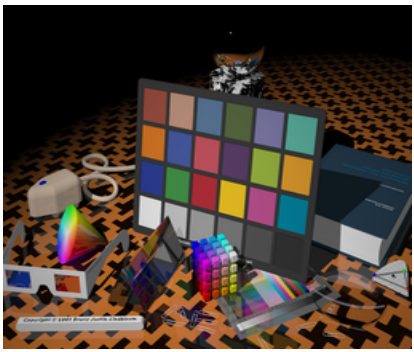
Tahap pertama dalam proses kompresi adalah merubah komponen warna dari RGB menjadi YCbCr. Proses ini ditunjukkan oleh *sourcecode* 4.1.

```
1  void
2  colorComponentTransformation::RgbToYCbCr(Proc
3  Image &image)
4  {
```

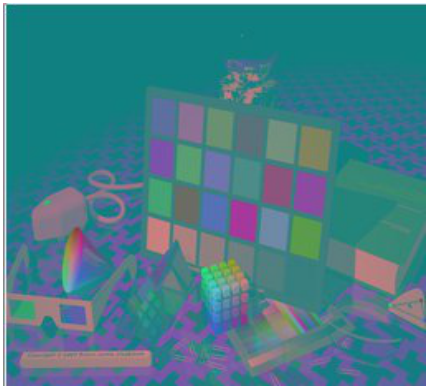
```
5     int width = image.getWidth();
6     int height = image.getHeight();
7     float r,g,b;
8     float Y,Cb,Cr;
9     ProcPixel *line = image.scanline(0);
10    for (int y = 0; y < height; y++)
11    {
12      line = image.scanline(y);
13      for (int x = 0; x < width; x++)
14      {
15        r = line[x].getRed();
16        g = line[x].getGreen();
17        b = line[x].getBlue();
18        Y = (66*r+129*g+25*b+128)/256+16;
19        Cb = (-38*r-74*g+112*b+128)/256+128;
20        Cr = (112*r-94*g-18*b+128)/256+128;
21        line[x].setCurrentPixel(
22          (Y<0?0:(Y>255?255:Y)),
23          (Cb<0?0:(Cb>255?255:Cb)),
24          (Cr<0?0:(Cr>255?255:Cr))
25          );
26       }
27    }
28  }
```

Sourcecode 4.1 Konversi RGB menjadi YCbCr

Hasil dari konversi warna dapat dilihat pada gambar 4.4. Gambar asli sebelum terjadi proses konversi dapat dilihat pada gambar 4.3.



Gambar 4.3 Citra asli



Gambar 4.4 Citra setelah konversi YCbCr

4.3.2. Lifting Scheme

Tahapan berikutnya dari proses kompresi adalah transformasi menggunakan lifting scheme. Proses transformasi dilakukan sebanyak nilai level dekomposisi. Pada *sourcecode* 4.2, nilai level dekomposisi disimpan pada variabel *decompositionLevel.*

```
1   void
2   D4LiftingScheme::forwardTransformation(ProcIm
3   age& image)
4   {
5     int decompositionLevel
6         =image.getDecompositionLevel();
7     int width = image.getWidth();
8     int height = image.getHeight();
9     int HHalf = width >> 1;
10    int VHalf = height >> 1;
11    collumnPixel collumn;
12    for (int i=0;i<decompositionLevel;i++){
13      for (int y=0;y<height;y++)
14        doForwardTransformation(image.scanline(
15           y),HHalf);
16      for (int x=0;x<width;x++){
17        image.scanCollumn(x,collumn);
```

```
18          doForwardTransformation(collumn.data(),
19            VHalf);
20          image.replaceCollumn(x,collumn);
21          }
22          width = HHalf;
23          height = VHalf;
24          HHalf = HHalf >> 1;
25          VHalf = VHalf >> 1;
26        }
27    }
28
```

Sourcecode 4.2 *Pengaturan transfomasi maju*

Pada *sourcecode* 4.2, proses transformasi lifting scheme dilakukan pertama-tama pada semua baris. Kemudian proses transformasi diterapkan pada semua kolom. Proses transf itu sendiri terjadi pada *sourcecode* 4.3.

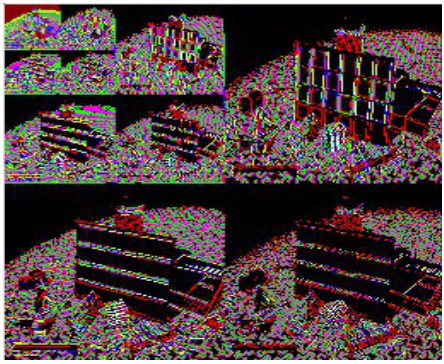
```
1   void
2   D4LiftingScheme::doForwardTransformation(Proc
3   Pixel *line, const int &half)
4   {
5     int start=1;
6     int end=(half*2)-1;
7     ProcPixel temp(0);
8     while (start<end){  // proses split
9       for (int n=start;n<end;n=n+2){
10        temp = line[n];
11        line[n]=line[n+1];
12        line[n+1]=temp;
13        }
14      start++;
15      end--;
16    }
17    // update 1
18    for(int n=0;n<half;n++)
19      line[n]=line[n]+sqrt3*line[n+half];
20    // predict
21    line[half]=line[half]-(sqrt3/4.0)*line[0]-
22        (((sqrt3-2)/4.0)*line[half-1]);
23    for (int n=1;n<half;n++)
```

```
24      line[half+n]=line[half+n]-(sqrt3/4.0)
25        *line[n]-(((sqrt3-2)/4.0)*line[n-1]);
26      // update 2
27      for (int n=0;n<half-1;n++)
28        line[n]=line[n]-line[half+n+1];
29      line[half-1]=line[half-1]-line[half];
30      // normalize
31      for (int n=0;n<half;n++){
32        line[n]=((sqrt3-1.0)/sqrt2)*line[n];
33        line[n+half]=((sqrt3+1.0)/sqrt2)
34          *line[n+half];
35      }
36  }
```

Sourcecode 4.3 *Lifting scheme*

Hasil dari proses transformasi di atas adalah citra pada bagian berlevel tinggi akan didominasi oleh nilai pixel rendah, dan pada bagian berlevel rendah menjadi bagian yang mewakili nilai global dari citra asli. Citra pada bagian level tinggi menyimpan nilai detail dari tiap tahapan transformasi.

Contoh hasil transformasi *lifting scheme* dengan level dekomposisi sebanyak 3 dapat dilihat pada gambar 4.5. Citra sebelum transformasi dapat dilihat pada gambar 4.4.



Gambar 4.5 *Contoh hasil transformasi berlevel 3*

4.3.3. Kuantisasi Skalar

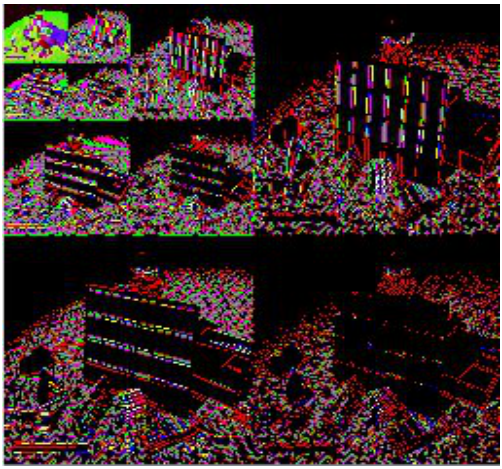
Proses kuantisasi mencoba untuk mengurangi jumlah data

unik dengan cara menyeragamkan nilai. Penyeragaman ini dilakukan dengan mendasarkan kalkulasi kuantisasi berdasarkan nilai *stepsize*. Nilai-nilai yang setelah diabsolutkan bernilai lebih kecil dari stepsize, menjadi 0. Nilai-nilai lainnya disesuaikan sesuai dengan rumus kuantisasi. Code untuk proses kuantisasi skalar ini dapat dilihat pada sourcecode 4.4.

```
1   void
2   ScalarQuantization::doDeadZoneQuantization(Pr
3   ocImage &image)
4   {
5     int stepSize = image.getQSize();
6     int height = image.getHeight();
7     int width = image.getWidth();
8     ProcPixel *data = image.scanline(0);
9     for (int y=0;y<height;y++){
10      data = image.scanline(y);
11      for (int x=0;x<width;x++){
12      if (data[x].getRed() > stepSize
13          || data[x].getRed() < (-1*stepSize))
14        data[x].setRed(floor(
15          data[x].getRed()/stepSize));
16      else
17        data[x].setRed(0);
18
19      if (data[x].getGreen() > stepSize ||
20           data[x].getGreen() < (-1*stepSize))
21        data[x].setGreen(floor(
22          data[x].getGreen()/stepSize));
23      else
24        data[x].setGreen(0);
25      if (data[x].getBlue() > stepSize ||
26          data[x].getBlue() < (-1*stepSize))
27        data[x].setBlue(floor(
28           data[x].getBlue()/stepSize));
29      else
30        data[x].setBlue(0);
31      }
32    }
33  }
```

Soucecode 4.4 Code kuantisasi skalar

Citra hasil kuantisasi skalar dapat dilihat pada gambar 4.6. Pada gambar tersebut, nilai stepsize adalah 4.



Gambar 4.6 Contoh hasil kuantisasi skalar

4.3.4. RLE

Run Length Encoding (RLE) menandakan dimulainya tahapan pengkodean entropy. Pada tahapan RLE, jika terdapat data dengan nilai yang sama muncul secara berurutan, maka urutan data ini akan diringkas dan diwakilkan oleh beberapa data saja. Proses ini dapat dilihat pada *sourcecode* 4.5.

```
1   void RLE::doEncoding(ProcImage &image,
2   EntropyCoding &e_Coding)
3   {
4     int imageHeight = image.getHeight();
5     int imageWidth = image.getWidth();
6     int n=0;
7     const ProcPixel *pixelData =
8       image.scanlineConst(0);
9     for (int y=0;y<imageHeight;y++) {
10      pixelData = image.scanlineConst(y);
11      n=0;
12      while (n<imageWidth) {   // do on red (Y)
13        if ((n+1)<imageWidth) {// ada run
```

```
14          if (pixelData[n].getRed()
15               ==pixelData[n+1].getRed())
16            n=doRun(e_Coding,
17              image.scanlineConst(y),
18              "red",n,imageWidth);
19          else // tidak ada run
20            e_Coding.updateStream(
21              pixelData[n].getRed(),"red");
22        } else // tidak ada run
23           e_Coding.updateStream(
24              pixelData[n].getRed(),"red");
25           n++;
26      }
```

Sourcecode 4.5 Proses Run Length Encoding

Sourcecode 4.5 menunjukkan proses run length encoding pada komponen warna Y. Komponen warna lainnya menjalani proses yang sama. Berdasarkan *sourcecode* 4.5, dapat dilihat bahwa proses pemampatan data yang muncul berurutan terjadi pada fungsi doRun. Fungsi doRun ditampilkan pada *sourcecode* 4.6.

```
1   int RLE::doRun(EntropyCoding &e_Coding, const
2   ProcPixel *pixelData, const QString &color,
3   const int &fromIndex, const int &width)
4   {
5     int count = fromIndex+2;
6     bool finish = false;
7     if (color=="red") {
8       while (finish==false) {
9         if (count >= width ||
10             pixelData[fromIndex].getRed()!=
11             pixelData[count].getRed()) {
12          e_Coding.updateStream(
13             pixelData[fromIndex].getRed(),
14              (count-fromIndex-2),color);
15           finish = true;
16        }
17        count++;
18      }
```

```
19    }
20    else if (color=="green") {
21      while (finish==false) {
22        if (count >= width ||
23             pixelData[fromIndex].getGreen()!
24             =pixelData[count].getGreen()) {
25          e_Coding.updateStream(
26             pixelData[fromIndex].getGreen(),
27             (count-fromIndex-2),color);
28          finish = true;
29        }
30      count++;
31      }
32    }
33    else if (color=="blue") {
34      while (finish==false) {
35        if (count >= width ||
36             pixelData[fromIndex].getBlue()!
37             =pixelData[count].getBlue()) {
38          e_Coding.updateStream(
39             pixelData[fromIndex].getBlue(),
40             (count-fromIndex-2),color);
41          finish = true;
42        }
43        count++;
44      }
45    }
46    return (count-2);
47  }
```

Sourcecode 4.6 Fungsi doRun

Pada sourcecode 4.5 dan 4.6, terdapat fungsi updateStream. Fungsi ini berguna untuk menambahkan data hasil encoding RLE ke dalam struktur data SymbolStream. Ada dua macam updateStream. Fungsi updateStream pertama adalah untuk menambahkan satu data saja ke dalam SymbolStream. Dan fungsi kedua meng-*overload* fungsi pertama, karena pada fungsi kedua, updateStream melakukan penambahan data berupa data yang muncul berurutan. Kedua fungsi tersebut dapat dilihat pada sourcecode 4.7.

```
1   void   EntropyCoding::updateStream(const   int
2   &value, const QString &color)
3   {
4     int row = 0;
5     if (color=="green") {
6       row = 1;
7     } else if (color=="blue") {
8       row = 2;
9     }
10    symbolStream[row].append(value);
11    updateFrequency(value);
12  }
13
14  void EntropyCoding::updateStream(const int
15  &value, const int &runCount, const QString
16  &color)
17  {
18    int row = 0;
19    if (color=="green") {
20      row = 1;
21    } else if (color=="blue"){
22      row = 2;
23    }
24    updateFrequency(value,runCount);
25    symbolStream[row].append(value);
26    symbolStream[row].append(value);
27    symbolStream[row].append(runCount);
28  }
```

Sourcecode 4.7 Fungsi updateStream

Pada fungsi updateStream terdapat fungsi updateFrequency. Fungsi ini berguna untuk melakukan penambahan nilai terhadap tingkat kemunculan simbol yang ditambahkan pada fungsi updateStream. Fungsi updateFrequency dapat dilihat pada *sourcecode* 4.8.

```
1   void EntropyCoding::updateFrequency(const int
2   &key)
3   {
```

```
4     if (!frequency.contains(key))
5       frequency.append(SymbolData(key,1));
6     else // tambahkan nilai frekuensi
7       frequency.data()[
8         frequency.indexOf(key)].
9         IncreaseDataFrequency(1);
10  }
11
12  void EntropyCoding::updateFrequency(const int
13  &key,const int &runCount)
14  {
15      if (!frequency.contains(key))
16          frequency.append(SymbolData(key,2));
17      else // tambahkan frekuensi untuk key ini
18       frequency.data()[frequency.indexOf(key)]
19          .increaseDataFrequency(2);
20
21      if (!frequency.contains(runCount))
22       frequency.append(SymbolData(runCount,1));
23      else // tambahkan frekuensi untuk key ini
24       frequency.data()
25          [frequency.indexOf(runCount)]
26          .increaseDataFrequency(1);
27  }
```

Sourcecode 4.8 Fungsi updateFrequency

Simbol yang diberikan kepada fungsi updateFrequency, *key*, akan dimasukkan pada struktur data frequency. Struktur data tersebut menyimpan semua simbol unik dalam bentuk class SymbolData (lihat 4.2).

Setelah proses RLE selesai, maka simbol pada struktur data frequency perlu diurutkan dari simbol dengan nilai frekuensi terkecil hingga terbesar. Untuk melakukan ini, maka pada class SymbolData perli melakukan *operator overloading* terhadap operator “>”.

4.3.5 Huffman Coding

Tahapan pengkodean entropy terakhir adalah *Huffman encoding*. Proses ini memanfaatkan data yang sudah terurut pada struktur data frequency untuk membangun pohon

Huffman. Namun pada penelitian ini, pohon Huffman tidak digunakan. Untuk membangun representasi huffman dibuatlah *array* tambahan, *huffmanArray*, yang panjangnya sama dengan panjang struktur data frequency. Pada array baru inilah proses *Huffman Encoding* berjalan selayaknya pohon Huffman.

Data pada *huffmanArray* adalah selayaknya *parent* dalam struktur data pohon (*tree*). Data pada masing-masing array tambahan tersebut memiliki pointer kepada data pada struktur data frequency dan data lain pada *huffmanArray*. Pointer ini memanfaatkan atribut pointer yang ada pada *class SymbolData*. Jadi data pada *huffmanArray* adalah objek dari *class SymbolData*. Untuk melihat proses pembangunan *huffmanArray* dapat dilihat pada *sourcecode* 1 pada lampiran 1.

Setelah *huffmanArray* ini dibangun, maka tahapan selanjutnya adalah menentukan representasi bit untuk masing-masing simbol. Proses ini brelangsung di dalam fungsi setBitData. Tiap simbol dalam struktur data frequency menelusuri pointer *parent* hingga mencapai *root*. Jika simbol sementara berada di *node* sebalah kiri, maka representasi pada simbol ini ditambahkan nilai 0, dan sebaliknya bila simbol ini berada pada node sebelah kanan, maka simbol ini memperoleh tambahan bit 1. *Sourcecode* untuk proses ini dapat dilihat pada *sourcecode* 4.9.

```
1   void SymbolData::setBitData()
2   {
3     bitData.resize(1);
4     int count = 0;
5
6     if (isThisLeft()) {
7       bitData.setBit(count,false);
8     } else
9       bitData.setBit(count,true);
10
11    const SymbolData *temp = parent;
12    while (temp->hasParent()) {
13      count++;
14      bitData.resize(count+1);
15
```

```
16      if (temp->isThisLeft())
17        bitData.setBit(count,false);
18      else {
19        bitData.setBit(count,true);
20        temp = temp->parent;
21      }
22  }
```

Sourcecode 4.9 Proses penentuan representasi bit

Langkah terakhir dari seluruh proses kompresi adalah merubah masing-masing simbol ke dalam bentuk bit. Proses ini dapat dilihat pada *sourcecode* 4.10.

```
1   // membangun bitstream
2
3   // bagian red
4   for (int i=0;i<size;i++)
5     codestream.append(
6       frequency[frequency.indexOf(
7       symbolStream.at(0).at(i))].getBitData());
8
9   // bagian green
10  size = symbolStream.at(1).size();
11  for (int i=0;i<size;i++)
12    codestream.append(
13      frequency[frequency.indexOf(
14      symbolStream.at(1).at(i))].getBitData());
15
16  // bagian blue
17  size = symbolStream.at(2).size();
18  for (int i=0;i<size;i++)
19    codestream.append(
20      frequency[frequency.indexOf(
21        symbolStream.at(2).at(i))]
22        .getBitData());
```

Sourcecode 4.10 Proses merubah data ke dalam *bitstream*

4.3.6 Huffman Decoding

Untuk melakukan proses dekompresi, maka data pertama

yang dibaca adalah data pada struktur data codeStream. Setiap representasi bit dalam codeStream dicari padanannya dari representasi bit yang tersimpan dalam objek symbolData. Proses dekompresi memulai proses tanpa mengingat proses yang terjadi selama proses kompresi, kecuali data pada codeStream,struktur data frequency dan huffmanArray tetap digunakan sebagai data awal.

Proses pertama dalam dekompresi adalah *huffman decoding*. Proses ini dapat dilihat pada *sourcecode* 4.11.

```
1   void
2   HuffmanCoding::doHuffmanDecoding(EntropyCoding
3   &eCoding)
4   {
5     // untuk frequency
6     const SymbolData* temp =
7       huffmanArray.data()+huffmanArray.size()-1;
8     // codestream
9     // tempStream menyimpan data QBitArray dari
10    // codestream
11    QBitArray tempStream =
12      eCoding.getCodeStream(0);
13    int posOnTempStream=0;
14    int pos=0;  // posisi pada codestream
15    // symbolStream
16    int redSize =
17      eCoding.getRedSymbolStreamSize();
18    int greenSize =
19      eCoding.getGreenSymbolStreamSize();
20    int blueSize =
21      eCoding.getBlueSymbolStreamSize();
22
23    // lakukan untuk merah
24    for (int n=0;n<redSize;n++) {
25      temp = huffmanArray.data()+
26        huffmanArray.size()-1;
27      tempStream = eCoding.getCodeStream(pos);
28      posOnTempStream = tempStream.size()-1;
29      while(temp->hasChildren()) {
30        if (tempStream.at(posOnTempStream))
```

```
31          temp = temp->getRight();
32        else
33          temp = temp->getLeft();
34        posOnTempStream--;
35        }
36      eCoding.addSymbolStream(
37        temp->getData(),0,n);
38      pos++;
39    }
40
41    // lakukan untuk hijau
42    for (int n=0;n<greenSize;n++) {
43       temp = huffmanArray.data()+
44         huffmanArray.size()-1;
45       tempStream = eCoding.getCodeStream(pos);
46       posOnTempStream = tempStream.size()-1;
47       while(temp->hasChildren()) {
48         if (tempStream.at(posOnTempStream))
49           temp = temp->getRight();
50         else
51           temp = temp->getLeft();
52         posOnTempStream--;
53       }
54       eCoding.addSymbolStream(
55         temp->getData(),1,n);
56       pos++;
57    }
58
59    // lakukan untuk biru
60    for (int n=0;n<blueSize;n++) {
61      temp = huffmanArray.data()+
62         huffmanArray.size()-1;
63      tempStream = eCoding.getCodeStream(pos);
64      posOnTempStream = tempStream.size()-1;
65      while(temp->hasChildren()) {
66        if (tempStream.at(posOnTempStream))
67          temp = temp->getRight();
68        else
69          temp = temp->getLeft();
70        posOnTempStream--;
71      }
72      eCoding.addSymbolStream(
```

```
73          temp->getData(),2,n);
74        pos++;
75      }
76    }
```

Sourcecode 4.11 Proses *huffman decoding*

4.3.7 Run Length Decoding

Data hasil *huffman decoding* di *decode* kembali menggunakan metode *Run Length Decoding*. Metode ini membentuk kembali struktur data citra dengan ukuran asli gambar yang dikompres dan struktur data tersebut diisi dengan data hasil *Run Length Decoding*. *Sourcecodenya* dapat dibaca pada *sourcecode* 4.12. Pada *sourcecode* tersebut ditampilkan proses decoding hanya pada komponen warna Y. Komponen warna lainnya, mengalami proses yang sama.

```
1   void RLE::doDecoding(ProcImage
2   &image,EntropyCoding &e_Coding)
3   {
4     int width = image.getWidth();
5     int height = image.getHeight();
6     int posOnImage = 0;
7     int pos = 0;
8     int currentHeight = 0;
9     int range = 0;
10    int hold=0;
11    int value=0;
12
13    // lakukan pada stream merah
14    int size =
15      e_Coding.getRedSymbolStreamSize();
16    ProcPixel *line =
17      image.scanline(currentHeight);
18    while(pos < size && currentHeight<height) {
19      posOnImage = 0;
20      line = image.scanline(currentHeight);
21      while(posOnImage<width && (pos+1)<size) {
22       value =
23         e_Coding.getRedSymbolStreamValue(pos);
```

```
24
25      if ((posOnImage+1)<width &&
26          e_Coding.getRedSymbolStreamValue(
27            pos)==
28            e_Coding.getRedSymbolStreamValue(
29            pos+1)) {
30        range =
31          e_Coding.getRedSymbolStreamValue(
32          pos+2);
33        hold = posOnImage;
34        for (;posOnImage<(hold+range+2);
35            posOnImage++)
36          line[posOnImage].setRed(value);
37          pos+=3;
38      }else {
39        line[posOnImage].setRed(value);
40        pos++;
41        posOnImage++;
42        }
43      }
44     currentHeight++;
45   }
```

Sourcecode 4.12 Proses Run Length Decoding

4.3.8 Dekuantisasi

Proses dekompresi setelah *entropy decoding* adalah dekuantisasi. Karena kuantisasi adalah proses yang *lossy*, maka data hasil dekuantisasi tidak sama dengan data sebelum kuantisasi. Proses dekuantisasi dapat dilihat pada *sourcecode* 4.13.

```
1   void
2   ScalarQuantization::doDeadZoneDeQuantization(P
3   rocImage &image)
4   {
5     int stepSize = image.getQSize();
6     int height = image.getHeight();
7     int width = image.getWidth();
8     ProcPixel *data = image.scanline(0);
```

```
9     for (int y=0;y<height;y++){
10      data = image.scanline(y);
11
12      for (int x=0;x<width;x++){
13        if (data[x].getRed() == 0)
14           data[x].setRed(0);
15        else
16           data[x].setRed(
17             data[x].getRed()*stepSize);
18
19        if (data[x].getGreen() == 0)
20           data[x].setGreen(0);
21        else
22           data[x].setGreen(
23             data[x].getGreen()*stepSize);
24
25        if (data[x].getBlue() == 0)
26           data[x].setBlue(0);
27        else
28           data[x].setBlue(
29             data[x].getBlue()*stepSize);
30      }
31    }
32  }
```

Sourcecode 4.13 Proses dekuantisasi skalar

4.3.9 Transformasi mundur lifting scheme

Setelah proses dekuantisasi selesai, maka proses berikutnya adalah transformasi mundur menggunakan lifting scheme. Proses transformasi mundur ini mengandalkan informasi mengenai berapa kali dekomposisi dilakukan selama proses kompresi. Sehingga proses transformasi mundur ini dilakukan pertama kali pada tingkat terendah hasil dekomposisi. Kemudian proses transformasi mundur bergerak secara bertahap membangun citra dengan memanfaatkan nilai-nilai detail yang disimpan pada bagian tingkat tinggi hasil dekomposisi citra. Proses transformasi mundur ini dapat dibaca pada *sourcecode* 4.14.

```
1   void
2   D4LiftingScheme::doInverseTransformation(Proc
3   Pixel *line, const int &half)
4   {
5     // normalize
6     for (int n=0;n<half;n++){
7       line[n]=((sqrt3+1)/sqrt2)*line[n];
8       line[n+half]=((sqrt3-1)/sqrt2)
9         *line[n+half];
10    }
11
12    // update 2
13    for (int n=0;n<half-1;n++)
14      line[n]=line[n]+line[n+half+1];
15    line[half-1]=line[half-1]+line[half];
16
17    // prediction
18    line[half]=line[half]+(sqrt3/4)*line[0]+
19         ((sqrt3-2)/4.0)*line[half-1];
20    for(int n=1;n<half;n++)
21      line[half+n]=line[half+n]+(sqrt3/4.0)
22        *line[n]+((sqrt3-2)/4.0)*line[n-1];
23
24    // update 1
25    for(int n=0;n<half;n++)
26      line[n]=line[n]-sqrt3*line[half+n];
27
28    int start = half-1;
29    int end = half;
30    ProcPixel temp(0);
31    while(start>0){ // merge
32      for(int n=start;n<end;n+=2){
33       temp = line[n];
34       line[n]=line[n+1];
35       line[n+1]=temp;
36     }
37     start--;
38     end++;
39    }
40  }
```

Sourcecode 4.14 Proses transformasi mundur

Dapat dilihat dari sourcecode 4.14, proses transformasi mundur adalah tahap kebalikan dari proses transformasi maju (sourcecode 4.3).

4.3.10 Transformasi komponen warna dari YcbCr menjadi RGB

Tahapan terakhir dari proses dekompresi adalah tahapan transformasi warna. Selama proses kompresi, komponen warna yang digunakan adalah dalam bentuk YCbCr. Agar gambar dapat dikenali seperti gambar asli sebelum di kompresi, maka citra warna perlu dirubah ke dalam bentuk RGB. Proses ini dapat dilihat pada sourcecode 4.15.

```
1   void
2   colorComponentTransformation::YCbCrToRgb(ProcI
3   mage &image)
4   {
5     int width = image.getWidth();
6     int height = image.getHeight();
7     float Y,Cb,Cr;
8     float r,g,b;
9     ProcPixel *line = image.scanline(0);
10    for (int y = 0; y < height; y++){
11      line = image.scanline(y);
12      for (int x = 0; x < width; x++){
13        Y = line[x].getRed()-16;
14        Cb = line[x].getGreen()-128;
15        Cr = line[x].getBlue()-128;
16
17        r = (298*Y+409*Cr+128)/256;
18        g = (298*Y-100*Cb-208*Cr+128)/256;
19        b = (298*Y+516*Cb+128)/256;
20
21        line[x].setCurrentPixel(
22          (r<0?0:(r>255?255:r)),
23          (g<0?0:(g>255?255:g)),
24          (b<0?0:(b>255?255:b))
25        );
26      }
27    }
28  }
```

Sourcecode 4.15 Proses Transformasi warna menjadi RGB

4.3.11 Perhitungan Nilai PSNR

Perhitungan nilai PSNR berguna untuk mengukur secara objektif kualitas citra hasil kompresi. *Sourcecode* dapat dilihat pada *sourcecode* 4.16. Pada sourcecode tersebut, perhitungan ems tidak lupa membagi hasil kumulasi tiap pixel dengan dimensi citra * 3. Pengalian 3 ini dilakukan karena dalam tiap pixel terdapat 3 komponen warna.

```
1   double ems = 0;
2   int width = originalImage.width();
3   int height = originalImage.height();
4   QRgb* original = (QRgb*)
5   originalImage.constScanLine(0);
6   QRgb* compressed = (Qrgb*)
7
8   compressedImage.constScanLine(0);
9
10  for(int y=0;y<height;y++) {
11    original = (Qrgb*)
12      originalImage.constScanLine(y);
13    compressed = (Qrgb*)
14      compressedImage.constScanLine(y);
15    for(int x=0;x<width;x++)
16      ems += pow(qRed(original[x])-
17       qRed(compressed[x]),2)+
18       pow(qGreen(original[x])-
19       qGreen(compressed[x]),2)+
20       pow(qBlue(original[x])-
21       qBlue(compressed[x]),2);
22  }
23  ems = ems / (originalImage.width()
24    *originalImage.height()*3);
25  psnr = 10 * log10(pow(255,2)/ems);
```

Sourcecode 4.16 Perhitungan PSNR

4.4 Pembahasan Hasil

Hasil uji coba terhadap 12 citra berdimensi 256x256 dapat dilihat pada tabel 4.5 hingga tabel 4.9. Masing-masing tabel menampilkan pengaruh dari jumlah dekomposisi yang terjadi pada tahapan lifting scheme terhadap tingkat kompresi,

compression rate (CR), dan kualitas citra hasil kompresi menggunakan tolak ukur *Peak Signal-to-Noise Ratio* (psnr).

Dari tabel 4.5 dapat dilihat bahwa semakin bertambah jumlah dekomposisi yang digunakan (kolom *lifting_level*) semakin bertambah nilai CR. Penambahan nilai CR diiringi pula dengan berkurangnya nilai PSNR.

Tabel 4.5 Hasil uji data menggunakan nilai stepsize = 2

CITRA	LIFTING_LEVEL	CR	PSNR
artificial	1	59.78	44.28
artificial	2	64.21	42.44
artificial	4	66.90	41.56
artificial	6	67.11	41.31
artificial	8	67.12	41.23
big_building	1	53.90	43.23
big_building	2	62.78	41.89
big_building	4	66.75	40.87
big_building	6	67.06	40.65
big_building	8	67.08	40.58
big_tree	1	48.36	42.49
big_tree	2	57.20	41.24
big_tree	4	61.57	40.35
big_tree	6	61.92	40.15
big_tree	8	61.93	40.10
bridge	1	55.13	43.12
bridge	2	64.55	41.89
bridge	4	69.00	40.94
bridge	6	69.33	40.71
bridge	8	69.34	40.65

cathedral	1	59.52	43.29
cathedral	2	69.04	41.99
cathedral	4	73.43	41.01
cathedral	6	73.76	40.77
cathedral	8	73.78	40.70
deer	1	69.72	45.29
deer	2	79.07	43.70
deer	4	83.24	42.36
deer	6	83.56	42.05
deer	8	83.57	41.97
fireworks	1	79.88	47.32
fireworks	2	83.67	45.03
fireworks	4	86.09	43.73
fireworks	6	86.39	43.18
fireworks	8	86.41	43.07
flower_foveon	1	65.24	45.27
flower_foveon	2	75.97	43.80
flower_foveon	4	80.50	42.42
flower_foveon	6	80.74	42.11
flower_foveon	8	80.75	42.05
hdr	1	64.35	44.92
hdr	2	74.07	43.36
hdr	4	78.49	42.07
hdr	6	78.76	41.79
hdr	8	78.77	41.72
leaves_iso_200	1	43.94	42.12
leaves_iso_200	2	51.72	40.89
leaves_iso_200	4	55.69	40.05
leaves_iso_200	6	56.03	39.86
leaves_iso_200	8	56.05	39.82
nightshot_iso_100	1	64.89	45.01
nightshot_iso_100	2	73.58	43.50
nightshot_iso_100	4	77.97	42.17
nightshot_iso_100	6	78.32	41.89
nightshot_iso_100	8	78.33	41.80
spider_web	1	70.67	45.99
spider_web	2	80.19	44.27
spider_web	4	84.17	42.74
spider_web	6	84.42	42.40
spider_web	8	84.43	42.32

Tabel 4.6 Hasil uji data menggunakan nilai stepsize = 4

CITRA	LIFTING_LEVEL	CR	PSNR
artificial	1	69.11	38.91
artificial	2	73.08	38.31
artificial	4	75.36	37.47
artificial	6	75.60	37.27
artificial	8	75.61	37.23
big_building	1	66.35	37.93
big_building	2	73.69	37.67
big_building	4	77.23	36.97
big_building	6	77.57	36.77
big_building	8	77.58	36.73
big_tree	1	62.36	37.37
big_tree	2	69.69	37.00
big_tree	4	73.57	36.39
big_tree	6	73.93	36.23
big_tree	8	73.94	36.19
bridge	1	68.85	38.22
bridge	2	75.45	38.07
bridge	4	78.98	37.38
bridge	6	79.32	37.16
bridge	8	79.33	37.12
cathedral	1	72.56	38.62
cathedral	2	80.47	38.52
cathedral	4	84.28	37.77
cathedral	6	84.61	37.54
cathedral	8	84.63	37.49
deer	1	78.97	40.43
deer	2	86.16	41.00
deer	4	89.53	39.85
deer	6	89.84	39.49
deer	8	89.86	39.42
fireworks	1	86.74	42.84
fireworks	2	89.14	42.51
fireworks	4	91.00	41.10
fireworks	6	91.26	40.65
fireworks	8	91.28	40.49
flower_foveon	1	74.75	39.89
flower_foveon	2	83.12	40.19
flower_foveon	4	87.06	39.14
flower_foveon	6	87.33	38.87
flower_foveon	8	87.34	38.80

hdr	1	74.82	39.69
hdr	2	82.22	39.81
hdr	4	86.10	38.80
hdr	6	86.39	38.54
hdr	8	86.40	38.47
leaves_iso_200	1	57.63	36.79
leaves_iso_200	2	64.69	36.38
leaves_iso_200	4	68.35	35.82
leaves_iso_200	6	68.70	35.67
leaves_iso_200	8	68.72	35.63
nightshot_iso_100	1	75.56	39.89
nightshot_iso_100	2	81.86	39.91
nightshot_iso_100	4	85.49	38.95
nightshot_iso_100	6	85.86	38.65
nightshot_iso_100	8	85.87	38.57
spider_web	1	79.42	40.69
spider_web	2	86.68	41.19
spider_web	4	89.99	39.91
spider_web	6	90.25	39.55
spider_web	8	90.27	39.49

Pola pada tabel 4.6 menunjukkan kejadian yang sama dengan tabel 4.5. Tingkat kompresi mengalami peningkatan seiring meningkatnya jumlah dekomposisi yang digunakan. Nilai PSNR secara umum pun mengalami penurunan seiring dengan meningkatnya nilai CR. Namun pada citra deer, flower_foveon, hdr, nightshot_iso_100 dan spider_web, terjadi peningkatan nilai PSNR ketika jumlah dekomposisi meningkat dari 1 menjadi 2. Namun nilai PSNR menurun pada jumlah dekomposisi beriktunya.

Tabel 4.7 Hasil uji data menggunakan nilai stepsize = 6

CITRA	LIFTING_LEVEL	CR	PSNR
artificial	1	74.36	35.32
artificial	2	77.79	35.43
artificial	4	80.11	34.85
artificial	6	80.34	34.66
artificial	8	80.35	34.62
big_building	1	72.76	34.59
big_building	2	79.35	35.02

big_building	4	82.66	34.53
big_building	6	82.99	34.36
big_building	8	83.01	34.31
big_tree	1	69.50	34.15
big_tree	2	75.92	34.33
big_tree	4	79.55	33.90
big_tree	6	79.92	33.77
big_tree	8	79.93	33.73
bridge	1	75.31	35.03
bridge	2	80.83	35.55
bridge	4	83.85	35.09
bridge	6	84.18	34.91
bridge	8	84.19	34.87
cathedral	1	78.57	35.51
cathedral	2	85.07	36.21
cathedral	4	88.56	35.72
cathedral	6	88.89	35.51
cathedral	8	88.91	35.47
deer	1	83.71	36.80
deer	2	89.05	38.82
deer	4	92.01	38.12
deer	6	92.31	37.78
deer	8	92.32	37.69
fireworks	1	89.40	39.84
fireworks	2	91.56	40.18
fireworks	4	93.20	39.27
fireworks	6	93.45	38.82
fireworks	8	93.46	38.71
flower_foveon	1	79.49	36.40
flower_foveon	2	86.63	37.70
flower_foveon	4	90.19	37.02
flower_foveon	6	90.45	36.76
flower_foveon	8	90.46	36.70
hdr	1	80.21	36.21
hdr	2	86.16	37.35
hdr	4	89.61	36.67
hdr	6	89.89	36.42
hdr	8	89.91	36.36

leaves_iso_200	1	64.18	33.58
leaves_iso_200	2	71.15	33.54
leaves_iso_200	4	74.61	33.15
leaves_iso_200	6	74.94	33.03
leaves_iso_200	8	74.96	33.00
nightshot_iso_100	1	80.93	36.75
nightshot_iso_100	2	85.93	37.44
nightshot_iso_100	4	89.01	36.82
nightshot_iso_100	6	89.35	36.56
nightshot_iso_100	8	89.36	36.49
spider_web	1	83.89	36.84
spider_web	2	89.68	38.65
spider_web	4	92.56	37.87
spider_web	6	92.81	37.54
spider_web	8	92.83	37.47

Apa yang terjadi pada tabel 4.7 tidak jauh berbeda dengan apa yang terjadi pada tabel 4.6. Namun, kali ini semua citra, kecuali leaves_iso_200, mengalami peningkatan PSNR ketika jumlah dekomposisi meningkat dari 1 menjadi 2. Nilai PSNR ini kemudian menurun seiring dengan meningkatnya jumlah dekomposisi.

Tabel 4.8 Hasil uji data menggunakan nilai stepsize = 8

CITRA	LIFTING_LEVEL	CR	PSNR
artificial	1	77.98	33.45
artificial	2	81.18	33.48
artificial	4	83.34	32.98
artificial	6	83.58	32.82
artificial	8	83.59	32.78
big_building	1	77.37	32.46
big_building	2	82.98	33.20
big_building	4	86.06	32.87
big_building	6	86.36	32.73
big_building	8	86.38	32.69
big_tree	1	74.07	31.89
big_tree	2	79.79	32.45
big_tree	4	83.27	32.17
big_tree	6	83.62	32.04
big_tree	8	83.64	32.02

bridge	1	79.64	32.82
bridge	2	84.06	33.77
bridge	4	86.87	33.45
bridge	6	87.18	33.27
bridge	8	87.20	33.23
cathedral	1	82.44	33.19
cathedral	2	87.82	34.47
cathedral	4	91.03	34.21
cathedral	6	91.32	34.02
cathedral	8	91.35	33.97
deer	1	85.68	34.25
deer	2	90.79	37.01
deer	4	93.51	36.77
deer	6	93.78	36.45
deer	8	93.80	36.38
fireworks	1	91.30	37.96
fireworks	2	93.12	38.54
fireworks	4	94.50	37.85
fireworks	6	94.73	37.48
fireworks	8	94.75	37.32
flower_foveon	1	82.68	33.74
flower_foveon	2	88.83	35.91
flower_foveon	4	92.07	35.53
flower_foveon	6	92.31	35.29
flower_foveon	8	92.32	35.23
hdr	1	83.49	33.66
hdr	2	88.50	35.54
hdr	4	91.63	35.15
hdr	6	91.89	34.91
hdr	8	91.90	34.85
leaves_iso_200	1	69.71	31.48
leaves_iso_200	2	75.62	31.59
leaves_iso_200	4	78.83	31.32
leaves_iso_200	6	79.17	31.20
leaves_iso_200	8	79.20	31.18
nightshot_iso_100	1	84.08	33.87
nightshot_iso_100	2	88.34	35.60
nightshot_iso_100	4	91.17	35.28
nightshot_iso_100	6	91.48	35.04
nightshot_iso_100	8	91.49	34.96
spider_web	1	86.57	34.21
spider_web	2	91.60	36.76
spider_web	4	94.14	36.39
spider_web	6	94.39	36.08
spider_web	8	94.40	36.02

Pada tabel 4.8, semua citra mengalami peningkatan nilai PSRN ketika jumlah dekomposisi ditingkatkan dari 1 menjadi 2. Setelah itu nilai PSNR berkurang seiring dengan bertambahnya jumlah dekomposisi. Nilai CR pun mengalami peningkatan ketika jumlah dekomposisi ditingkatkan.

Tabel 4.9 Hasil uji data menggunakan nilai stepsize = 10

CITRA	LIFTING_LEVEL	CR	PSNR
artificial	1	80.86	31.34
artificial	2	83.77	31.99
artificial	4	85.81	31.56
artificial	6	86.01	31.41
artificial	8	86.02	31.36
big_building	1	80.45	30.63
big_building	2	85.40	31.78
big_building	4	88.32	31.60
big_building	6	88.61	31.47
big_building	8	88.63	31.43
big_tree	1	77.42	30.10
big_tree	2	82.51	30.96
big_tree	4	85.85	30.81
big_tree	6	86.20	30.69
big_tree	8	86.21	30.67
bridge	1	82.32	30.94
bridge	2	86.24	32.32
bridge	4	88.92	32.19
bridge	6	89.22	32.03
bridge	8	89.23	31.98
cathedral	1	85.19	31.53
cathedral	2	89.77	33.10
cathedral	4	92.70	33.07
cathedral	6	92.98	32.91
cathedral	8	93.00	32.86
deer	1	88.34	32.05
deer	2	92.01	35.59
deer	4	94.51	35.73
deer	6	94.78	35.41
deer	8	94.79	35.35

fireworks	1	92.57	35.94
fireworks	2	94.01	37.19
fireworks	4	95.32	36.73
fireworks	6	95.53	36.34
fireworks	8	95.55	36.21
flower_foveon	1	84.85	31.93
flower_foveon	2	90.26	34.40
flower_foveon	4	93.33	34.35
flower_foveon	6	93.56	34.13
flower_foveon	8	93.57	34.05
hdr	1	85.90	31.72
hdr	2	90.17	34.07
hdr	4	93.01	33.94
hdr	6	93.27	33.73
hdr	8	93.28	33.67
leaves_iso_200	1	74.10	29.26
leaves_iso_200	2	78.95	30.04
leaves_iso_200	4	81.91	29.87
leaves_iso_200	6	82.25	29.76
leaves_iso_200	8	82.28	29.74
nightshot_iso_100	1	85.95	32.23
nightshot_iso_100	2	89.99	34.05
nightshot_iso_100	4	92.60	34.09
nightshot_iso_100	6	92.89	33.86
nightshot_iso_100	8	92.90	33.78
spider_web	1	88.22	32.31
spider_web	2	92.80	35.26
spider_web	4	95.16	35.27
spider_web	6	95.39	34.99
spider_web	8	95.41	34.93

Hasil yang dicapai pada tabel 4.9 tidak jauh berbeda dengan hasil pada tabel 4.8. Semua citra mengalami peningkatan nilai PSNR ketika jumlah dekomposisi meningkat dari 1 ke 2. Nilai PSNR kemudian menurun ketika jumlah dekomposisi terus ditingkatkan. Sementara itu, nilai CR mengalami peningkatan ketika jumlah dekomposisi bertambah.

Dapat dilihat dari tabel 4.5 hingga 4.9, pada masing-masing stepsize, citra dengan tingkat data berfrekuensi rendah lebih banyak memiliki tingkat kompresi yang lebih tinggi dibandingkan

dengan data yang memiliki lebih banyak unsur berfrekuensi tinggi. Hal ini misal dapat dilihat dari perbandingan nilai CR antara citra fireworks dengan big_tree.

Ketika melihat pada tabel 4.5 hingga tabel 4.9, terdapat pola umum yaitu bila jumlah dekomposisi mengalami pengingkatan maka terjadi pengingkatan nilai PSNR hingga ambang batas tertentu. Setelah itu, nilai PSNR mengalami penurunan seiring dengan bertambahnya jumlah dekomposisi. Sementara itu, nilai CR mengalami pengingkatan seiring dengan meningkatnya nilai CR.

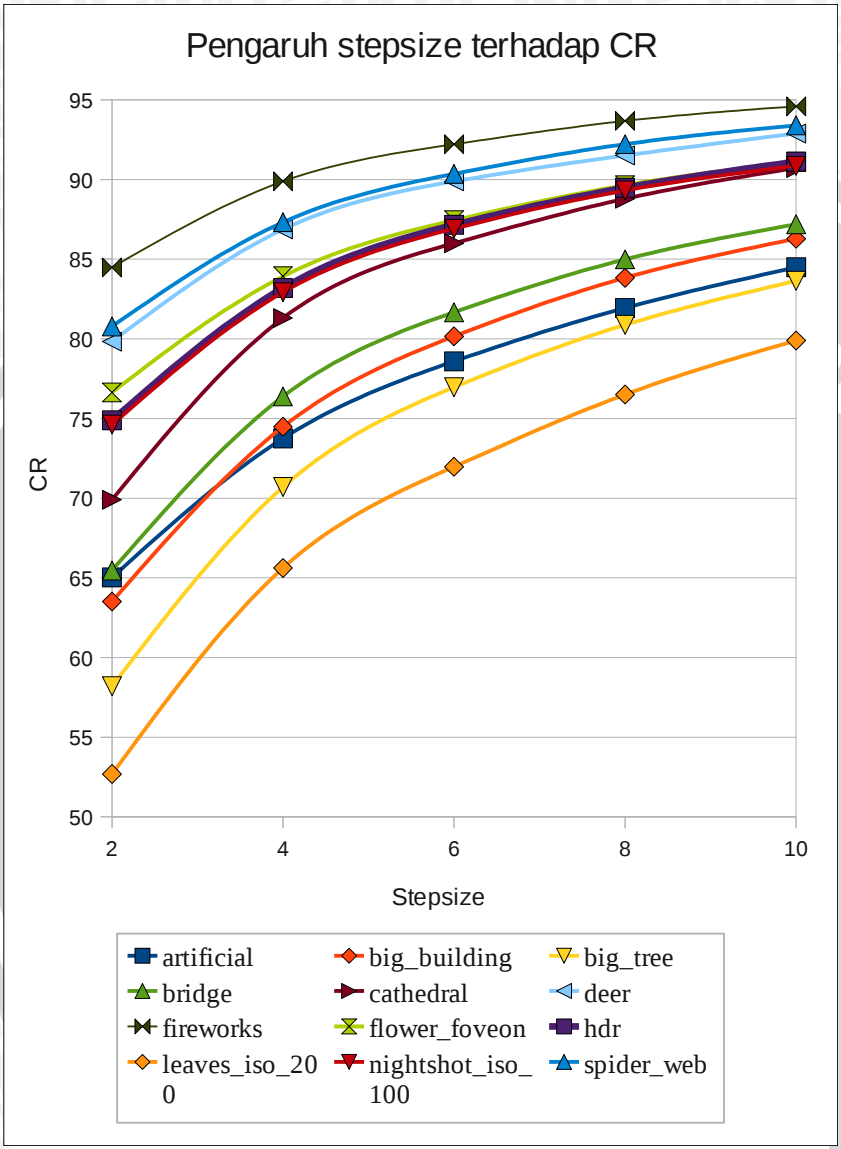
Pola lainnya yang dapat diperoleh dari tabel 4.5 hingga 4.9 adalah pengaruh nilai stepsize terhadap CR dan PSNR. Pada gambar 4.1, dapat dilihat pengaruh peningkatan jumlah stepsize terhadap nilai CR. Sementara itu pada tabel 4.8 dapat dilihat pengaruh nilai stepsize terhadap nilai PSNR.

Pada grafik 4.1, masing-masing citra diambil rata-rata nilai CR untuk tiap nilai *stepsize*. Berdasarkan gambar tersebut, terjadi peningkatan nilai CR ketika nilai stepsize mengalami peningkatan. Dapat dilihat juga bagaimana citra yang memiliki data berfrekuensi tinggi lebih banyak, semisal big_building dan big_tree, memiliki nilai CR yang lebih rendah dibandingkan dengan citra yang didominasi oleh data berfrekuensi rendah, seperti fireworks.

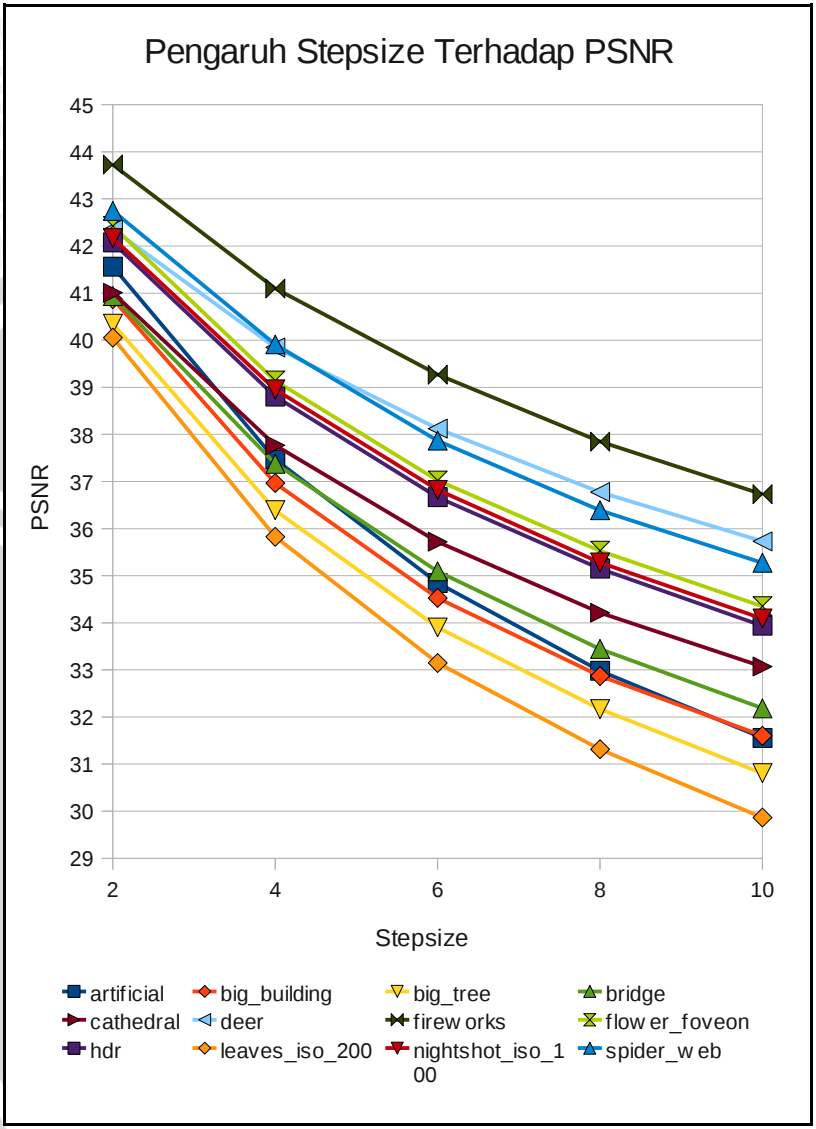
Grafik 4.2 merepresentasikan data pada tabel 4.6 untuk dijadikan contoh bagaimana pengaruh stepsize terhadap nilai PSNR. Pada gambar tersebut dapat dilihat bagaimana nilai PSNR mengalami penurunan ketika nilai stepsize ditingkatkan.

Pola lainnya adalah hubungan antara nilai CR dengan nilai PSNR. Secara umum, dapat dilihat pada tabel 4.5 hingga 4.9, ketika nilai CR mengalami peningkatan, maka nilai PSNR mengalami penurunan. Terjadi pengecualian ketika nilai stepsize yang digunakan lebih banyak. Pengecualian yang terjadi adalah nilai PSNR meningkat seiring meningkatnya nilai CR sampai batas tertentu.

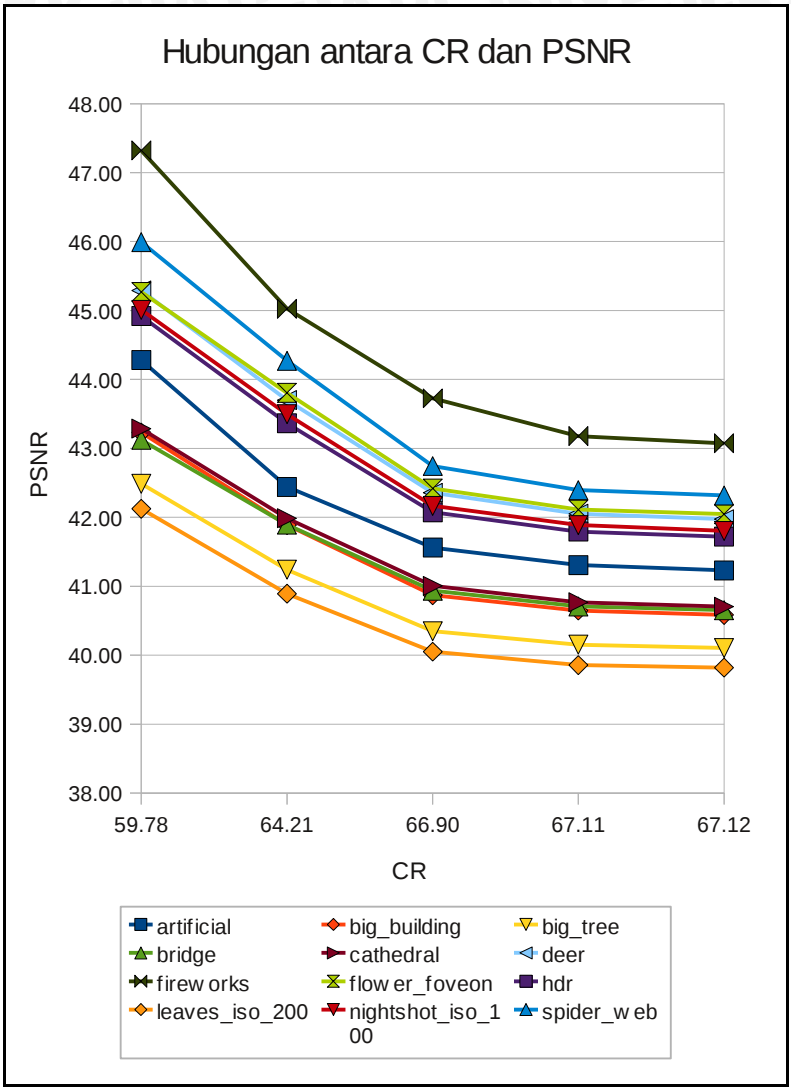
Grafik 4.3 menunjukkan bagaimana hubungan nilai CR dan PSNR dimana data pada tabel 4.5 dijadikan sebagai contoh. Gambar tersebut menunjukkan bahwa nilai PSNR menurun seiring dengan meningkatnya nilai CR.



Grafik 4.1 Grafik pengaruh nilai stepsize terhadap CR



Grafik 4.2 Pengaruh stepsize terhadap PSNR



Grafik 4.3 Hubungan antara CR dan PSNR

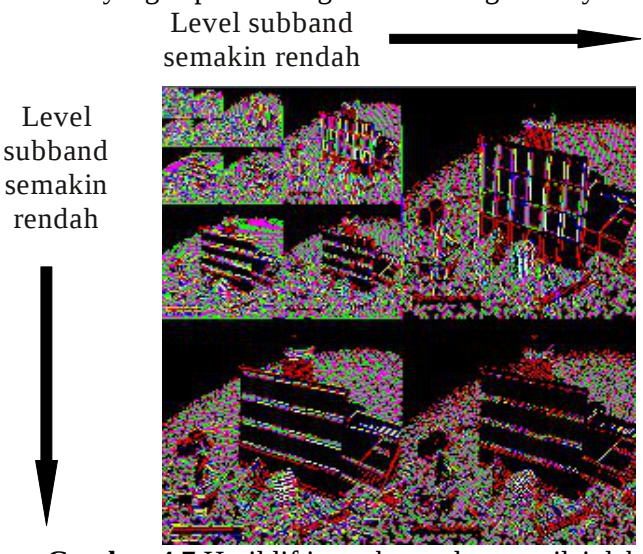
4.5 Analisa Hasil

4.5.1 Analisa Tingkat Kompresi

Terdapat dua variabel yang mempengaruhi nilai tingkat kompresi. Variabel tersebut adalah jumlah dekomposisi dalam

proses lifting scheme dan ukuran stepsize dalam kuantisasi. *Lifting scheme* melakukan proses transformasi citra ke dalam *subband-subband* yang datanya memiliki tingkat frekuensi yang berbeda. Pembagian level dari subband hasil transformasi lifting scheme, dapat dilihat pada gambar 4.7. Citra pada bagian subband tertinggi (pojok kiri atas) mengandung koefisien transformasi yang penting dimana koefisien ini menjadi representasi dari citra asli. Data pada subband tertinggi memiliki nilai frekuensi yang rendah.

Representasi data berdasarkan *subband-subband* yang berbeda ini memungkinkan adanya pembuangan atau pengurangan data pada daerah bersubband rendah. Pengurangan data pada daerah bersubband rendah ini memiliki pengaruh yang kecil terhadap kualitas citra yang dipersepsi oleh sistem penglihatan manusia. Hal ini dikarenakan sistem penglihatan manusia cenderung lebih sensitif terhadap perubahan yang terjadi pada data berfrekuensi rendah (Grgic, 2001). Sehingga semakin tinggi nilai dekomposisi, semakin tinggi pula jumlah subband yang berfrekuensi tinggi yang berarti semakin banyak data yang dapat dibuang atau dikurangi nilainya.



Gambar 4.7 Hasil lifting scheme dengan nilai dekomposisi = 8

Pembuangan atau pengurangan data pada masing-masing subband dilakukan dalam proses kuantisasi. Kuantisasi yang diterapkan adalah kuantisasi skalar yang menerapkan *deadzone*. Berapa besar *deadzone* ini ditentukan oleh nilai *stepsize*. Dengan demikian, semakin besar nilai stepsize, berarti semakin besar ukuran penyeragaman pada data, dan ini pun berujung pada semakin banyak data yang bernilai 0 dan rendah, terutama pada subband rendah.

Untuk melihat bagaimana jumlah 0 bertambah pada proses kompresi, dilakukan perhitungan jumlah 0 pada citra artificial.ppm. Sebagai contoh. Perhitungan ini menggunakan stepsize bernilai 2. Jumlah 0 pada citra setelah proses lfiting scheme ditampilkan pada tabel 4.10. Sementara itu, jumlah 0 setelah kuantisasi ditampilkan pada tabel 4.11.

Tabel 4.10 Jumlah 0 yang dihasilkan proses lifting scheme terhadap citra artificial.ppm

Jumlah Dekomposisi	Jumlah 0
1	89365
2	104754
4	107410
6	107432
8	107432

Tabel 4.11 Jumlah 0 yang dihasilkan tahapan kuantisasi dengan nilai stepsize = 2 pada citra artificial.ppm

Jumlah Dekomposisi	Jumlah 0
1	102412
2	120306
4	123487
6	123515
8	123515

Jumlah data bernilai 0 pada citra asli sebelum mengalami proses kompresi adalah 48994. Dapat dilihat berdasarkan tabel 4.10, bahwa proses lifting scheme meningkatkan jumlah data bernilai 0 pada citra. Semakin tinggi jumlah dekomposisi, maka data bernilai 0 pun bertambah.

Berdasarkan tabel 4.11, jumlah 0 semakin bertambah banyak dengan diterapkannya kuantisasi skalara berbasis *deadzone*. Penambahan ini juga menunjukkan bahwa semakin besar ukuran stepsize, maka semakin berkurang jumlah pixel unik yang ada pada citra. Keseragaman ini memungkinkan bertambahnya tingkat kompresi pada citra.

Proses transformasi lifting scheme dan kuantisasi menghasilkan data bersebelahan yang lebih seragam. Keseragaman ini akan dikompresi lebih dengan menggunakan metode *Run Length Encoding* (RLE) dan *Huffman Encoding*. Metode RLE memampatkan jumlah kode yang diperlukan untuk merepresentasikan data. Sementara itu Huffman Encoding memampatkan representasi kode sehingga kode yang lebih sering muncul memeroleh panjang kode yang lebih pendek, sedangkan kode yang lebih jarang memperoleh panjang kode yang lebih panjang.

Berdasarkan pemaparan di atas, tingkat kompresi mengalami peningkatan ketika jumlah dekomposisi bertambah. Penerapan stepsize dengan ukuran yang lebih besar juga berujung pada tingkat kompresi yang lebih tinggi.

4.5.2 Analisa Nilai Peak Signal-to-Noise Ratio (PSNR)

Nilai PSNR menunjukkan kualitas citra hasil kompresi. Semakin rendah nilai PSNR, berarti jumlah informasi yang hilang semakin bertambah. Sifat dari kompresi *lossy* adalah adanya informasi yang hilang pada citra hasil kompresi.

Pembuangan atau pengurangan informasi terjadi pada tahapan kuantisasi. Yang menjadi perhatian adalah proses kuantisasi skalar menerapkan kuantisasi pada tiap level subband dengan ukuran stepsize yang sama. Pengurangan atau pembuangan informasi terjadi tidak hanya pada data berfrekuensi tinggi, pada data berfrekuensi rendah juga. Data pada level subband yang paling tinggi, data merepresentasikan

nilai global yang terdapat pada citra asli. Ketika bagian tersebut paling rendah dikenakan kuantisasi, maka terjadi perubahan pada nilai global citra yang pada akhirnya menurunkan kualitas citra itu sendiri.

Pembuangan atau pengurangan informasi ditentukan oleh ukuran *stepsize* yang diterapkan. Semakin besar ukuran stepsize maka semakin besar pula jangkauan nilai yang diseragamkan. Penyeragaman ini berakibat pada hilangnya informasi yang ada citra, sehingga hal tersebut mempengaruhi nilai PSNR.

Pada penelitian ini, terdapat fenomena yang menarik yaitu ketika jumlah dekomposisi dintingkatkan dari 1 menjadi 2, nilai PSNR meningkat. Sementara itu, bila jumlah dekomposisi ditingkatkan lagi, nilai PSNR mengalami penurunan. Untuk menemukan penyebab terhadap fenomena ini dilakukan uji coba terhadap empat data uji yang dimana empat data tersebut dipilih secara acak dari data uji yang digunakan pada skripsi ini. Pada masing-masing data uji, dilakukan proses kompresi tanpa menerapkan tahapan kuantisasi maupun tahapan *entropy coding*. Hasil dapat dilihat pada tabel 4.12.

Tabel 4.12 Uji PSNR pada citra pasca *lifting scheme*

Jumlah dekomposisi	artificial	big_building	bridge	fireworks
1	46.02	45.91	45.94	45.91
2	46.02	45.91	45.94	45.91
4	46.02	45.91	45.94	45.91
6	46.02	45.91	45.94	45.91
8	46.02	45.91	45.94	45.91

Hasil pada tabel 4.12 menunjukkan bahwa nilai PSNR sama untuk masing-masing citra meskipun jumlah dekomposisi ditingkatkan. Hal ini berarti anomali pada data disebabkan oleh penerapan kuantisasi. Untuk menguji hipotesa ini, maka dilakukanlah pengujian lanjutan dari model di atas.

Model kuantisasi yang digunakan tetap menggunakan

kuantisasi skalar berbasi *deadzone*. Perbedaanya terletak pada model pengujian dimana *subband* tertinggi dari hasil *lifting scheme* tidak dikenakan kuantisasi. Hasil pengujian ini dapat dilihat pada tabel 4.13. Masing-masing data uji dikenakan kuantisasi dengan nilai stepsize sebesar 6.

Tabel 4.13 Uji PSNR pada citra dengan tidak menerapkan kuantisasi pada subband tertinggi. Nilai stepsize = 6

Jumlah dekomposisi	artificial	big_building	bridge	fireworks
1	36.508	36.186	36.658	40.285
2	35.133	34.873	35.375	39.181
4	34.640	34.351	34.895	38.715
6	34.602	34.310	34.857	38.679
8	34.600	34.306	34.853	38.673

Tabel 4.13 menunjukkan bahwa tidak terjadi anomali pada hasil PSNR. Hal ini berbeda dengan tabel 4.7 sebelumnya, dimana terdapat pengingkatan nilai PSNR ketika jumlah dekomposisi dinaikkan dari 1 menjadi 2. Hasil uji coba di atas menunjukkan bahwa anomali disebabkan oleh penerapan kuantisasi pada bagian subband tertinggi. Penemuan ini dapat menjadi bahan untuk penelitian selanjutnya.

Berdasarkan pemaparan di atas, nilai PSNR lebih dipengaruhi oleh ukuran stepsize. Sementara itu, jumlah dekomposisi memiliki pengaruh yang kecil terhadap nilai PSNR. Kontribusi jumlah dekomposisi terhadap nilai PSNR terletak pada bagaimana transformasi lifting scheme merepresentasikan data ke dalam bentuk yang lebih mudah untuk dipampatkan lebih banyak pada tahapan kuantisasi.

4.6 Hasil Optimal

Hasil optimal proses kompresi didasarkan pada perolehan tingkat kompresi tertinggi dengan pencapaian PSNR tertinggi. Hasil optimal ini ditampilkan pada tabel 4.14.

Tabel 4.14 Hasil Optimal Kompresi Citra

CITRA	STEPSIZE	LIFTING_LEVEL	CR	PSNR
artificial	2	1	59.78	44.28
big_building	2	1	53.90	43.23
big_tree	2	1	48.36	42.49
bridge	2	1	55.13	43.12
cathedral	2	1	59.52	43.29
deer	2	1	69.72	45.29
fireworks	2	1	79.88	47.32
flower_foveon	2	1	65.24	45.27
hdr	2	1	64.35	44.92
leaves_iso_200	2	1	43.94	42.12
nightshot_iso_100	2	1	64.89	45.01
spider_web	2	1	70.67	45.99
artificial	4	1	69.11	38.91
big_building	4	1	66.35	37.93
big_tree	4	1	62.36	37.37
bridge	4	1	68.85	38.22
cathedral	4	1	72.56	38.62
deer	4	2	86.16	41.00
fireworks	4	1	86.74	42.84
flower_foveon	4	2	83.12	40.19
hdr	4	2	82.22	39.81
leaves_iso_200	4	1	57.63	36.79
nightshot_iso_100	4	2	81.86	39.91
spider_web	4	2	86.68	41.19
artificial	6	2	77.79	35.43
big_building	6	2	79.35	35.02
big_tree	6	2	75.92	34.33
bridge	6	2	80.83	35.55
cathedral	6	2	85.07	36.21
deer	6	2	89.05	38.82
fireworks	6	2	91.56	40.18
flower_foveon	6	2	86.63	37.70
hdr	6	2	86.16	37.35
leaves_iso_200	6	1	64.18	33.58
nightshot_iso_100	6	2	85.93	37.44
spider_web	6	2	89.68	38.65
artificial	8	2	81.18	33.48
big_building	8	2	82.98	33.20
big_tree	8	2	79.79	32.45
bridge	8	2	84.06	33.77
cathedral	8	2	87.82	34.47
deer	8	2	90.79	37.01
fireworks	8	2	93.12	38.54
flower_foveon	8	2	88.83	35.91
hdr	8	2	88.50	35.54
leaves_iso_200	8	2	75.62	31.59
nightshot_iso_100	8	2	88.34	35.60
spider_web	8	2	91.60	36.76

artificial	10	2	83.77	31.99
big_building	10	2	85.40	31.78
big_tree	10	2	82.51	30.96
bridge	10	2	86.24	32.32
cathedral	10	2	89.77	33.10
deer	10	2	92.01	35.59
fireworks	10	2	94.01	37.19
flower_foveon	10	2	90.26	34.40
hdr	10	2	90.17	34.07
leaves_iso_200	10	2	78.95	30.04
nightshot_iso_100	10	2	89.99	34.05
spider_web	10	2	92.80	35.26

Untuk hasil optimal terbaik, dipilih berdasarkan pada citra dengan nilai CR paling besar. Berdasarkan tabel 4.14 dapat dilihat bahwa citra dengan hasil kompresi optimal terbaik adalah fireworks dimana tingkat kompresi bernilai 94.01 % dan nilai PSNR adalah 37.19.

BAB V
PENUTUP

5.1 Kesimpulan

Dari hasil analisa ujicoba yang menggunakan parameter uji jumlah dekomposisi bernilai 1,2,4,6,8 dan nilai stepsize 2,4,6,8,10 diperoleh kesimpulan sebagai berikut:

1. Rancangan sistem kompresi citra *lossy* yang diterapkan terdiri atas konversi warna dari RGB menjadi YCbCr, transformasi *lifting scheme* Daubechies 4, kuantisasi skalar, Run Length Encoding dan Huffman. Untuk proses dekompresi, langkahnya adalah kebalikan dari langkah kompresi.
2. Hasil pengukuran terhadap tingkat kaulitas citra hasil kompresi terbaik diperoleh oleh citra fireworks, dengan tingkat kompresi sebesar 94.01% dan nilai PSNR sebesar 37.19.
3. Tingkat kompresi dipengaruhi oleh jumlah dekomposisi dan nilai stepsize yang digunakan. Semakin bertambahnya jumlah dekomposisi dan dikombinasikan dengan nilai stepsize yang bertambah, maka tingkat kompresi semakin tinggi.
4. Hasil PSNR dipengaruhi oleh jumlah dekomposisi dan nilai stepsize yang digunakan. Secara umum, semakin bertambahnya jumlah dekomposisi dan nilai stepsize yang digunakan, semakin berkurang nilai PSNR. Namun pada beberapa hasil ujicoba, nilai PSNR dapat ditingkatkan dengan meningkatkan jumlah dekomposisi.
5. Ketika tingkat kompresi mengalami peningkatan, maka pola umumnya adalah nilai PSNR mengalami penurunan.

5.2 Saran

Hal-hal yang dapat dilakukan untuk mengembangkan penelitian ini adalah:

1. Melakukan penelitian terhadap pengaruh bentuk dekomposisi yang digunakan pada tahap *lfiting scheme* terhadap tingkat kompresi dan kualitas citra yang dihasilkan.
2. Melakukan perbandingan antara jenis lifting scheme terhadap

tingkat kompresi dan kualitas citra yang dihasilkan. Perbandingan ini juga memfokuskan pada bagaimana pengaruh jumlah dekomposisi yang digunakan terhadap alat ukur kualitas kompresi.

3. Kuantisasi menjadi bagian yang penting dalam kompresi *lossy*. Penelitian dapat dikembangkan dengan menggunakan kuantisasi berbasis vektor.
4. Menggunakan nilai-nilai kuantisasi yang berbeda untuk subband yang berbeda.
5. Menggunakan metode entropy coding yang melakukan encoding terhadap lebih dari satu simbol sekaligus. Contoh metode tersebut adalah arithmetic coding.

DAFTAR PUSTAKA

Al-Abudi, Bushra K. 2005. *Color Image Compression Using Wavelet Transformation.* University Of Baghdad.

Albayrak, Songul. 2001. *Color Quantization By Modified K-Means Algorithm.* Yildiz Technical University.

Celebi, M. Emre. 2009. *Effective Initialization Of K-Means For Color Quantization*. Louisiana State University

Chang, Shaorong. 2006. *A Modified SPIHT Algorithm for Image Coding With a Joint MSE and Classification Distortion Measure.* IEEE Transactions on Image Processing, Vol. 15, No. 3,

Daubechies, Ingrid. 1997. *Factoring Wavelet Transforms Into Lifting Steps*. Princeton University

Dipperstein, Michael. 2008. *Arithmetic Code Discussion and Implementation*. http://michael.dipperstein.com/arithmetic/index.html. 23 Maret 2010.

Fowler, Martin. 2004. *UML Distilled.* USA: Addison-Wesley.

Gershikov, Evgeny. 2007. C*orrelation vs. Decorrelation Of Color Components In Image Compression.* Israel Institure Of Technology.

Gonzalez, Rafael C. 1993. *Digital Image Processing.* USA: Prentice Hall

Grgic, Sonja. 2001. *Performance Analysis of Image Compression Using Wavelet.* IEEE Transactions On Industrial Electronics, Vol. 48 No. 3

Hamilton, Eric. 1992. *JPEG File Interchange Format V 1.02*. USA: C-Cube Microsystems

Howard, Paul G. 1992. *Practical Implementations of Arithmetic*

Coding. Brown University

Lafore, Robert. 1998. *Object Oriented Programming in C++*. USA: Macmillan Computer Publishing

Polikar, Robert. 2001. *The Engineers Ultimate Guide To Wavelet Analysis.* http://users.rowan.edu/~polikar/WAVELETS/WTtutorial.html. 15 Desember 2009.

Popescu-Bodorin, Nicolae. 2008. *Fast K-Means Image Quantization Algorithm and Its Application to Iris Segmentation.* Universitatea din Pitesti

Said, Amir. 2004. *Introduction to Arithmetic Coding - Theory and Practice.* Palo Alto: HP Laboratories

Salomon, David. 2004. *Data Compression: The Complete Reference.* USA: Springer

Shi, Yun Q. 2008. *Image and Video Compression for Multimedia Engineering.* USA: CRC Press

Spires, Wade. 2005. *Lossless Image Compression Via The Lifting Scheme.* University Of Central Florida

Strom, Jacob. 1996. *Dead-zone Quantization in Wavelet Image Compression.* Tidak tersedia.

Sweldens, Wim. 1995. *The Lifting Scheme: A New Philosophy in Biorthogonal Wavelet Constructions.*Katholieke Universiteit Leuven Belgium

Xiong, Zixiang. 2009. *Essential Guide to Image Processing.* USA: Elsevier

LAMPIRAN I
SOURCE CODE UNTUK METODE PEMBANGUNAN IN-ARRAY HUFFMAN TREE

```
1   void HuffmanCoding::buildHuffman(SymbolData
2   *data,SymbolData *dataHuffman,const int
3   &symbolsLength)
4   {
5     int posToInsert = 0;
6     int posToCheckHuffman = 0;
7     int posToCheck = 0;
8
9     dataHuffman[0].setLeft(data[posToCheck]);
10    dataHuffman[0].setFrequency(
11      data[posToCheck].getFrequency());
12    data[posToCheck].setParent(
13      dataHuffman[posToInsert]);
14    data[posToCheck].setIsLeft();
15    posToCheck++;
16
17    dataHuffman[0].setRight(data[posToCheck]);
18    dataHuffman[0].increaseDataFrequency(
19      data[posToCheck].getFrequency());
20    data[posToCheck].setParent(
21      dataHuffman[posToInsert]);
22    data[posToCheck].setIsRight();
23    posToInsert++;
24    posToCheck=2;
25
26    // bangun huffman tree
27    while(posToInsert < (symbolsLength - 1)) {
28    // masukkan children sebelah kiri
29    if (posToCheck < symbolsLength) {
30      if (data[posToCheck]<=
31          dataHuffman[posToCheckHuffman]) {
32        dataHuffman[posToInsert].setLeft(
33          data[posToCheck]);
34        dataHuffman[posToInsert].setFrequency(
35          data[posToCheck].getFrequency());
36        data[posToCheck].setParent(
37          dataHuffman[posToInsert]);
```

```
38          data[posToCheck].setIsLeft();
39          posToCheck++;
40        }else{
41          dataHuffman[posToInsert].setLeft(
42             dataHuffman[posToCheckHuffman]);
43          dataHuffman[posToInsert].setFrequency(
44             dataHuffman[posToCheckHuffman]
45             .getFrequency());
46          dataHuffman[posToCheckHuffman]
47             .setParent(
48             dataHuffman[posToInsert]);
49          dataHuffman[posToCheckHuffman]
50             .setIsLeft();
51                    posToCheckHuffman++;
52                }
53      } else {
54          dataHuffman[posToInsert]
55                .setLeft(dataHuffman[posToCh
56                eckHuffman]);
57          dataHuffman[posToInsert]
58                .setFrequency(dataHuffman
59                [posToCheckHuffm].getFrequency());
60          dataHuffman[posToCheckHuffman]
61                .setParent(dataHuffman
62                [posToInsert]);
63          dataHuffman[posToCheckHuffman]
64            .setIsLeft();
65           posToCheckHuffman++;
66      }
67      // masukkin chidren sebelah kanan
68      if (posToCheck < symbolsLength) {
69        if (data[posToCheck]<=
70                dataHuffman[posToCheckHuffman]) {
71            dataHuffman[posToInsert]
72              .setRight(data[posToCheck]);
73            dataHuffman[posToInsert]
74              .increaseDataFrequency(
75              data[posToCheck].getFrequency());
76            data[posToCheck]
77              .setParent(
78              dataHuffman[posToInsert]);
79            data[posToCheck].setIsRight();
```

```
80            posToCheck++;
81          } else {
82            dataHuffman[posToInsert]
83              .setRight(
84              dataHuffman[posToCheckHuffman]);
85            dataHuffman[posToInsert]
86              .increaseDataFrequency(
87              dataHuffman[posToCheckHuffman]
88              .getFrequency());
89            dataHuffman[posToCheckHuffman]
90              .setParent(
91              dataHuffman[posToInsert]);
92            dataHuffman[posToCheckHuffman]
93              .setIsRight();
94            posToCheckHuffman++;
95           }
96       } else {
97          dataHuffman[posToInsert]
98            .setRight(
99            dataHuffman[posToCheckHuffman]);
100         dataHuffman[posToInsert]
101           .increaseDataFrequency(
102           dataHuffman[posToCheckHuffman]
103           .getFrequency());
104         dataHuffman[posToCheckHuffman]
105           .setParent(
106           dataHuffman[posToInsert]);
107         dataHuffman[posToCheckHuffman]
108           .setIsRight();
109         posToCheckHuffman++;
110       }
111     posToInsert++;
    }
  }
```

Sourcecode 1 Membangun *in-array huffman tree*