卒 業 研 究 報 告

題　　目

JPEG におけるエントロピー符号化回路の設計

指 導 教 員

橘 昌良

報 告 者

学籍番号: 1030205

氏名:　所谷　憲典

平成　15 年　1 月　27 日

高知工科大学 電子･光システム工学科

目次

第１章 はじめに

１・１ 本研究の背景

今現在では､多くの情報がデジタル化され情報のやり取りがネットワーク上で数多く見られる。そのため、記憶媒体の節約や通信にかかるコストや時間を短縮するために､データを小さくするためのデータ圧縮が盛んに研究されている。それは、静止画像であれば JPEG、動画像であれば MPEG である。

現在使用しているデータ圧縮は、図１.１のように入力した原画像データをＤＣＴ→量子化→エントロピー符号化の順番で圧縮し､そしてその圧縮されたデータをエントロピー→量子化→ＤＣＴの順番で再生画像へと変換するものである。これは、符号器がデータ圧縮符号による符号化をすることによって、データ量を削減しそのデータを送りやすいものへと変換する｡そして圧縮されたデータを復号器が復号化して、元のものに近いものに戻すことを示す。

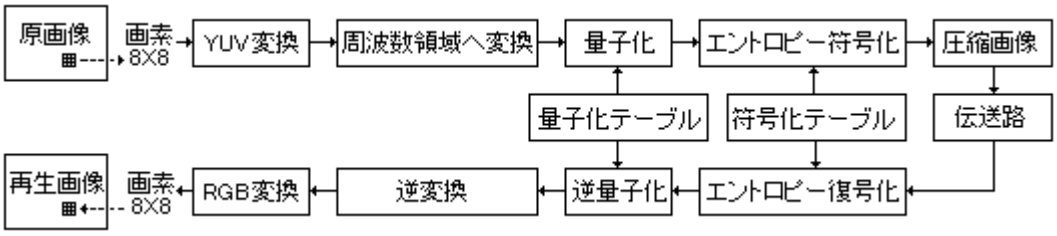

図１．１　符号化・複合化の構造

画像や動画像とは、輝度成分や色差成分などでできているため、データ量が膨大になる。そのため、そのデータ量を圧縮するために考えられたのがデータ圧縮技術である。そのデータ圧縮に必要な符号化技術とは、大きく分けて時空間冗長度除去・量子化・エントロピー符号化などがあり、そしてその中にも次のような圧縮方法が存在している。

・時空間冗長度除去
　　―インタレース
　　―動き補償
　　―色信号変換
　　―周波数変換
　　・ 離散コサイン変換
　　・ Karhunen Loeve 変換
　　・ 帯域分割
　　・ ウェーブレット変換
・量子化
　　―非線形量子化
　　―適応ステップサイズ
　　―ベクトル量子化
・エントロピー符号化
　　―ハフマン符号化
　　―算術符号化
　　―Lempel-Ziv アルゴリズム

これらの符号化技術は、画像品質を損なうことなくかなりの圧縮が可能になるため、膨大なデータ量である画像は、少ないデータ量となり、使いやすいものへと変化する。

そこで、本論文の研究対象としたのがエントロピー符号化である。エントロピー符号化とは簡単に言えば、符号量を削減するための方法で出現度の多いものに短い符号を割り当てることによりデータ量を少なくすることである。そこで私は、そのエントロピー符号化であるハフマン符号化に興味を持ち、このハフマン符号化をＣ言語によって設計を行う。

１・２ 本研究の位置付け

本論文では、まず画像情報の圧縮技術にどのようなものがあるか説明し、その使い道について述べる。そして、この本論文の主となるエントロピー符号化の性質やハフマン符号化を説明し、最後にエントロピー符号化であるハフマン符号の設計を行う。

１・３ 本論文の構成

第 2 章では、画像情報とその圧縮技術、それに JPEG についても説明を行う。第 3 章では、エントロピー符号化について述べる。第 4 章では、エントロピ

ー符号化の実現に向けての、設計の手順を述べる。第 5 章では、実際に作ったエントロピー符号化におけるハフマン符号化のＣプログラムを動かし、その結果を示して考察を行う。

第 2 章 画像情報とその圧縮技術

２・１ 画像データの性質

人間の視野に訴える画像情報は、文字・静止画像・動画像に分類される。ということは画像を効率よく圧縮するためには、自分の都合に合わせて処理する場合に、こうした分類に基づいてそれぞれの性質を理解した上で圧縮することが非常に有効である。

画像の特徴として、データ量が多いということがある。特に従来の画像信号とは、非常に広い信号帯域を持っているため、非常に膨大なデータ量を要した。そのため、少しだけ圧縮したとしてもデジタル画像はかなり大きなデータ量となるため、データの伝送などに長い時間がかかったり、伝送しないにしても容量を大きく使うため使い勝手の悪いものであったりする。現在では光ファイバや符号化技術の進歩のおかげで、デジタル画像の処理・伝送が比較的容易に行うことが出来るようになり、私達の生活の中で、なくてはならないものとなった。

画像には、もう一つ大きな特徴がある。それは、画像データの容量がかなりの圧縮が可能であるということである。それは、画像データ全体の中にかなり明確な規則性をもっているため、それをうまく把握できればデータ容量全体をかなり小さくできるということである。もちろん画像の内容や使い方で、どの程度まで画像品質を保つかを明確にしなければならない。例えるなら、非常に細かく書かれている回路設計図を圧縮するとなると、画像品質を落とした場合に細かいところまで見ることができなくなるため、大きな圧縮は向かないということである。しかし次に、通常の写真や身の回りの印刷物を圧縮するとなると話は先ほどとは異なる。なぜなら、ある程度画像品質を落としたとしても、細かい部分に目をやる必要のないこれらのものは、見て何か分かれば良いので、設計図のような細かさは必要なくなるため、設計図以上の圧縮が可能となる。しかし、品質を落としすぎて大部分の人に気づかれては意味がないため、圧縮をする場合は、そのものの色、内容、用途などを考え、それらによって圧縮率を変化させるのが最もよい。

２・２ 画像データと音声データの圧縮比率

画像データと音声データの比較を表２．１に示す。

種類	容量	サイズ	利用
画像(カラー)	12Mbyte	2048×2048	テレビ等
モノクロ	4Mbyte	2048×2048	
音 8kHz	48kbyte	3秒間	電話
44.1～48kHz	264.6～288kbyte	3秒間	CD，プロ用PC録音システム

表２．１ 画像と音声の容量

次に、画像と音声の圧縮効果の比較をみると、表２．２のようになる。この意味は、画像なら 10 分の１に圧縮されたとしても、人間の目では圧縮による劣化を認知することできないことを示し、音声では 2 分の 1 の圧縮なら、認知できないことを示す。これだけでも画像と音声では、5 倍もの圧縮率が異なり画像の圧縮がはるかに行いやすいことがわかる。もちろん数値は対象とした画像・音声データの種類と圧縮アルゴリズムの種類により多少異なるであろうが、画像は音声よりも圧縮しやすいことが分かる。

種類	認知できない圧縮率	認知できる圧縮率
画像	1／10	1／40
音声	1／2	1／4

表２．２ 圧縮による認知率

２・３ 画像データ圧縮のための基本技術

画像データファイルの容量を小さくするための手段は多数ある。これらは、視覚的な冗長度と統計的な冗長度を削ることや、アナログ信号の帯域の圧縮や

可逆符号化の実現方法である。そしてその基本方法は下のようなものがある、これらを、簡単に説明する。

インタレース

動画像の圧縮方法で、最も簡単な方法の一つは、動画像から時間軸方向に標本化して得られる静止画像の数を減らしてしまうという方法で、つまり一秒間のコマ数を通常６０～８０のところをそれ以下におさえて、減らしたコマの分だけデータ量を減らす方法である。

しかしこの方法には大きな欠点がある。それはコマ数を減らしていくと動きに滑らかさが無くなり動画像として見ることが不可能となってしまうことだ。そのため、動画像として見るためには、最低一秒間に静止画像 50 枚程度必要である。

そこで考えられたのがインタレースである。インタレースとは、一枚の画像を偶数番目の走査線から構成される画像と、奇数番目の走査線から構成される画像を二つに分けてしまい、こうして得た倍の画像で動画像を表示する方法である。この方法は図２．２を見てもらえるとおりで、二分の一の画像の量で一つのものとして見ることができる。

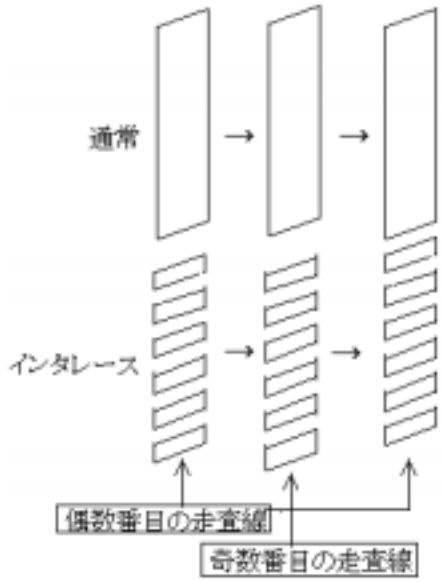

図２．２ インタレース

色信号変換

カラー画像は、赤（Ｒ：Red）と緑（Ｇ：Green）そして青（Ｂ：Blue）の三つの信号から構成されている。こうしたカラー画像の三つのＲＧＢ画像信号には強い相関関係があるため、これらの三つのＲＧＢ画像を別な 3 組の信号に変

換することにより、ダイナミックレンジの総和を縮めることが可能になる。こうした変換の中で、今現在もっとも使われているものとして、一つの輝度信号と二つの色信号にするものや、色信号の高い周波数成分を切り捨てることでデータ量を少なくすることなどである。これらの方法は、人間の目の仕組みを逆手にとった方法であり、人間の目は緑という色ははっきりと見ることができるのに対して、青や赤等の色の空間周波数を認識することが、非常に難しいという特徴を持っているため、その空間周波数帯域をとってしまうことでデータ量の削減を行っている。通常のテレビ放送での輝度信号と二つの色信号の帯域は、４：２：２もしくは４：１：１が使われており、ようするに二つの色信号あわせて輝度信号の同じくらいかその半分の周波数帯域が使われている。

色信号変換の欠点として、印刷・写真・美術・染色などのプロフェッショナル用途に用いる場合には、色が表示できなかったり、もしくは二つの異なる領域の色の違いが判別できなかったりしてしまうところがあるため、使う場所や用途によって使い分けることが大切である。

周波数変換

画像には、劣化が目立ちやすいものと目立ちにくいものがある。目立ちにくいものとしては、画像にある空間周波数の高周波成分の劣化などがある。つまり高周波成分は、人間の目では認識できないため多少の劣化があったとしても大きく目立つことが少ない。したがって、画像にある高周波成分を削るということは、画像信号の帯域を狭められるとともに、画像データ量の削減となる。

こうした高い周波数成分を削るという処理は、原理的にはアナログでもデジタルでも行えるが、要求される処理の精度やデジタルシステムによって、はじめて実用的に使うことができるようになった。

どのようなデジタルシステムにより空間周波数の高い周波数成分を削る処理をするかは、大きく分けて二つあるそれは、

- 画像の周波数スペクトルを計算し、その計算して導かれた結果の所定の部分を取り出す方法 (DCT)
- フィルタバンクを用いる方法（サブバンド法）

＊フィルタバンクとは、特性の異なる複数のフィルタの集合体

この二つがあり、二つの違いは処理を周波数領域で行うか、時間領域で行うかである。

そしてこの二つの方法の性質として、DCT の方は画像にブロックひずみと呼ばれる格子状のひずみが明確な形で現れる。そしてサブバンド法のほうは DCT のようなひずみは現れないが、画面全体がぼやけた様な劣化が起きるため用途によって使い分ける必要がある。

エントロピー符号化

エントロピー符号化とは、０と１のビット列に対して、そのビットの統計な性質を利用して、より小さなビット列に変換する符号のことである。そして可逆符号化とは、符号化されたビット列をもとの形(ビット列)に完全に復元することをいう。

そしてこれらの符号化は、今現在多数存在する。なぜなら、統計的に性質をどの様に考えるかに関して統一性がないからである。要するにこの統計的な性質とは、「信号源にどのような考え方のモデル構造を作るか」というのが議論である。もちろん色々なエントロピー符号化には、圧縮率の違いがあるものもあるが、可逆符号化であるため１/2～1/3 が限界となり、特殊なデータを使わなければそれ以上の圧縮を見込めない為、どれが一番いいということが難しい状態である。

動き補償

動き補償は動画像に適用される。動き補償とは、時間軸方向に動画像信号を標本化して得られた、ある時間間隔ごとの連続した複数の静止画像間において、画像画素の集合の動きに注目して、動きを検出することにより高能率な符号化を実現しようとするものである。要するに、フレーム間である画像の動きが分かれば、次に使うフレームは前回のフレームの再利用でまかなうことができるということである。もっと簡単に言えば、ある背景があってその背景の中で、人が一個所で動き回っているとする。すると、背景は変わらずに人だけ動くことになるから、背景のデータは常に再利用でき、人の動きだけのデータで動画像が見えるということである。このように動き補償は画像データ量をかなり削減することができる。ただし、動き補償は確かに高い符号化効率をもたらすが、そのためにかなりの演算量が必要であることや、この動きをどの範囲でどのように検出するかという問題が生じるため、今も研究は続けられている。

ベクトル量子化

ベクトル量子化とは、複数のサンプル信号値に対して、あるビットパターンを符号化として割り当てる方法である。よってサンプル信号値の分布状態に対し、適切なグループ分けができれば、情報論理の上限の符号化効率を実現できる。しかしここで問題が出てくる。それは、このグループ分けをどのようにすればよいかということである。この問題は、サンプル信号値の分布状態をすべて把

握することができるのなら答えは簡単だが、それをすべて把握するのは今現在では、事実上不可能なため、ある程度推定で補っている。そのため符号化効率は、不確かなデータ量(予測されたデータ)が必要となるため、この情報理論の上限まで圧縮することはできない。

ほとんどの場合サンプル信号値は、デジタル信号に変換せれているので、複数のサンプル信号値は、ある程度のビット数のデジタル信号である。ということは、このベクトル量子化はエントロピー符号化と同じようなもので、デジタル信号を別のより小さなビット数のデジタル信号へと変換するものである。しかし、その量子化とエントロピー符号化の違いは、エントロピー符号化では可逆が保証されているのにたいして、量子化では可逆が保証されていない。このことは重要な問題であり、可逆が保証されるための方法が今もなお現在研究されている。

ウェーブレット変換

ウェーブレット変換とは、周波数領域の分解能と時間領域の分解能が異なる複数のフィルタバンクを用いて、信号の性質を分析する手段である。よって高周波数成分と低周波数では、時間解像度を変えることができるので、信号の多重解像度解析が可能である。多重解像度解析は、急激な信号変化に対しては周波数成分の詳細構造の分析をある程度犠牲にして、その場所を特定する必要がある。逆に緩慢な信号変化に対しては、その場所よりも周波数変化の詳細を、より正確に分析する必要がある。DCT のような変換では、変換次数が固定であるため、このような多重解像度を持つ信号分析はできないが、多重解像度で分析できることにより、信号の変な変化に対して分析能力を高くできるため、画像の解析・認識・符号化への応用の他にも、地震波形･音響波形、量子力学など、非常に広範囲へに応用が期待されている。

２・４ ＪＰＥＧについて

JPEG (Joint Picture Expert Group)

JPEG は、静止画圧縮に対して開発された規格で動画像圧縮より圧縮が容易なだけではなく、圧縮効率と復元効率が優れており、現在最も広く使われている静止画圧縮方式である。(カラー静止画像の世界標準となっている)

JPEG の圧縮方法

カラーイメージは明度と色の二つで成り立っている。JPEG の圧縮方式は、「人間の目は明るさの変化には鋭敏だが、色の変化には鈍感である」という特性を利用して、画像の明度のバリエーション情報を保存し、色の情報を利用して圧縮する方法である。復元された画像が同じような明度のバリエーションを持つ限り、人間の目には元の画像と変わりなく見える。

ほかの圧縮方法

画像の保存方式として JPEG は静止画、特に写真の保存に優れている。JPEG のほかにも GIF や TIFF などがあり、GIF は単純な画像（アニメのような画像）について、JPEG よりも高い圧縮をかけることができるが、色数の少なさのため、写真のような複雑な画像の保存にはむいていない。TIFF は、サポートしていないブラウザもあり、普及度や互換性にやや難点があるが、今後の改良が期待されている。

第３章 エントロピー符号化

３・１ エントロピー符号器信号の作成

エントロピー符号化(無ひずみ圧縮)は量子化された信号に対して行う。ただし DC 成分とそれ以外の 63 個の AC 成分は、エントロピー符号化が両者の統計的な性質が大きく異なるため、別々に考えなくてはならない。DC 成分 Sq00 に対しては、スキャンの順番において、直前のブロックの DC 成分との差分がエントロピー符号化される。ようするに、i 番目のブロックである DC 成分の量子化値を、Sq00(i)としたときに、実際のエントロピー符号化される信号は、

$$Diff = Sq00(i) - Sq00(i-1) \quad \cdots\cdots\cdots\cdots \text{式(1)}$$

で計算される差分値である。これは、DC 成分(画像の絵柄の平均値)が隣接ブロック間で大きく変化しないため、差分値の分散を小さくできる。一方 AC 成分については、ジグザグスキャンの順番に従って、2 次元信号から 1 次元信号に変換してからエントロピー符号化部に引き渡される。これは、AC 成分の大きさが高周波数成分程小さく、さらに量子化ステップサイズも高周波数成分ほど大きいため、高周波成分では大部分の AC 成分が 0 となるという性質を利用して、０値の AC 成分ができるだけ長く連続するように 1 次元信号を作成している。

３・２ ハフマン符号化のアルゴリズム

ハフマン符号化とは、もっとも出現率の多い符号に、少ないビット数を当てることにより、全体のビット数を削減する方法である。一つの例としてこの例題を見てみる。

アルファベット“A”,“B”,“C”,“D”,“E”の五つの文字からなる以下の文章があるとする。

CEACDABABCEABADACADABABADEACBADABCADAEE

この文字列は 39 文字であり、表 3.1 のような出現頻度になっていて、“A”の出現頻度が高いことになる。ここで、この文章をデジタル・データに変換するところだが、もっとも単純な方法は、それぞれのアルファベットに対して 1 対 1 に対応する固定長のビット列を割り当てることである。(表 3. 1 に固定長ビット列を対応させた例を示す)要するに五つのアルファベットを３ビットの符号により現すことである。すると全体で使われている文字数は 39 文字なので、39×3 ＝117 ビットよって 117 ビットのデータを用いてこの文章をデジタルで表現す

ることが可能である。しかし、すべてのアルファベットに対して 3 ビットの符号を割り当てる際、デジタル・データへと変換したが、“ A ”の出現頻度が高いことがわかっているのであれば、出現頻度の高いアルファベットに対してビット数の短い符号を割り当てれば、全体として少ないビット数で現すことができる。このときの符号の割り当て方は、ハフマン符号を使うことにより実現できる。(ハフマン符号化した表を表３．２に示す)このハフマン符号を用いれば,通常“ BAD ”という 3 文字の単語は“ 100000110 ”と９ビットのビット列なるのに対して、“ 1000110 ”という 7 ビットのビット列に変換することができる。

逆に、“ 01100111 ”となるビット列が与えられた場合は、最初のビットから解析して、“ ADAE ”という元の文字列を生成することができる。“ 01100111 ”というビット列の解析の仕方は、先頭のビットより順に見ていき、表にマッチするハフマン符号を発見することにより復号化する。符号長が固定ではなく可変なので、前から順に解析しないと次の符号との境界を発見することができない。最初からやっていくと、まず符号“ 0 ”すなわちアルファベット“ A ”で符号長１の符号を発見する。この時点で“ A ”というデコード結果が得られ、符号長１ となるので、すぐ後に次の符号との境界があることが判明する。次にこの境界より解析を行う。3 ビットを解析した時点で, 符号“ 110 ”すなわち、アルファベット“ D ”で符号長 3 の符号を発見する。この時点で“ D ”というデコード結果が得られ、符号長 3 なので、次の位置に境界があることが判明する。このようにハフマン符号化とは、より少ないビット数でデータを現すことができる。

ハフマン符号での注意だが「どの符号もほかの符号の先頭の一部と同じにならない」という条件が成立していなくてはならない。なぜなら,“ 1000110 ” と“ 100 ”が存在する場合に、一番始めから解析していくハフマン符号としては、“ 1000110 ”の信号が“ 100…の部分で反応してしまい“ 100 ”と“ 0110 ”の二つの信号と勘違いしてしまうから、このことを注意して考え、重なることのないようにビットを振り分けなければ、ハフマン符号化は成り立たない。

文字列

CEACDABABCEABADACADABABADEACBADABCADAEE

記号	出現頻度	対応ビット列	符号のビット長	トータル・ビット数
A	15	000	3	15×3=45
B	7	100	3	7×3=21
C	6	101	3	6×3=18
D	6	110	3	6×3=18
E	5	111	3	5×3=15
				合計ビット117

表3、1固定長ビット列

記号	出現頻度	対応ビット列	符号のビット長	トータル・ビット数
A	15	0	1	15×1=15
B	7	100	3	7×3=21
C	6	101	3	6×3=18
D	6	110	3	6×3=18
E	5	111	3	5×3=15
				合計ビット87

表３．２ ハフマン符号化

ハフマン符号化への変換

DCT 方式のハフマン符号化は、64 個の DCT 係数に対して DC 成分と AC 成分で別々に行う。DC 成分の符号化は、隣接する直前のブロックの DC 成分を求めてそれの差分をハフマン符号化する。

次に AC 成分に対しては、量子化により大部分の係数値が０となることから、非零の AC 成分とその AC 成分の前に位置する零値の AC 成分の個数を組み合わせて、零ランレングスを考慮したハフマン符号化を行う。

ジグザグスキャンによる並び替え

ジグザグスキャンとは、2 次元の DCT 変換係数をハフマン符号化するために、まず始めに 2 次元配列から１次元配列へと変換するために使われる。

８×８の DCT 変換係数からなる 2 次元信号を Squv(u,v=0,1,・・・,7)とし、これをさらに図３．１に示した順序でシグザグスキャンして１次元信号に変換した信号を ZZ（ｋ）（ｋ＝0,1,・・・,63）とする。すなわち、

$$ZZ(0)=Sq00, ZZ(1)=Sq01, ZZ(2)=Sq10, \cdot\cdot ZZ(63)=Sq77 \quad 式(2)$$

となり、ZZ(0)がDC成分、ZZ(0)～ZZ(63)がAC成分である。DC成分は、式(１)で示されたように、直前のブロックにおけるDC成分に量子化値の差分信号Diffをハフマン符号化する。またAC成分はZZ(1),ZZ(2),・・・ZZ(63)の連続する零の個数と非零のAC成分を組み合わせてハフマン符号化する。詳しいＤＣ差分値の符号化とＡＣ成分の符号化の説明は次の項目で説明する。

0	1	5	6	14	15	27	28
2	4	7	13	16	26	29	42
3	8	12	17	25	30	41	43
9	11	18	24	31	40	44	53
10	19	23	32	39	45	52	54
20	22	33	38	46	51	55	60
21	34	37	47	50	56	59	61
35	36	48	49	57	58	62	63

図３．１ ジグザグスキャンの順番

３・３ ハフマン符号化の手順

3･3･1 DC差分値の符号化

DC差分値Diffの符号を行うには表３．３に示してあるDiffの値に、対応するカテゴリSSSSとハフマン符号を求める。ここでのSSSSとは、DC差分値をグループ分けするための変数である。Diffが整数の場合にはDiffを２進表現した場合に上位ビットから見て、一番初めの"１"が立つLSBの番号であり、Diffが負の場合には Diff－１を２進表現した場合に上位ビットから見て一番初めの"0"が立つLSBの番号である。カテゴリSSSSに対応するハフマン符号のみでは、グループは特定できても具体的なグループ内のDiffの値が表現できないため、ハフマン符号の後に付加ビットを加えてDiffの値を特定する。SSSSはこの付加ビットのビット長に対応しており、Diffが正の場合にはDiffのLSBからSSSSビ

ットを、Diff が負の場合には Diff－１の LSB から SSSS ビットが付加ビットとなる。これにより、復号器では付加ビットの先頭ビットが１の場合には Diff が正、0 の場合には Diff が負であると判断できる。以上述べたように DC 差分値のハフマン符号化は、DC 差分値の大きさのカテゴリを最初にハフマン符号で指定し、その後に続く付加ビットで、カテゴリ内の DC 差分値の具体的な値を指定する形を、

（カテゴリのハフマン符号）＋(付加ビット)・・・・式(3)

の順番に結合された可変長符号を用いて行われる。

Diff	SSSS	ハフマン符号	付加ビット
−2047,...,−1024,1024,...,2047	11	111111110	00000000000 01111111111,10000000000 11111111111
−1023,...,−512,512,...1023	10	11111110	0000000000 0111111111,1000000000 1111111111
−511,...,−256,256,...511	9	1111110	000000000 011111111,100000000 111111111
−255,...,−128,128,...255	8	111110	00000000 01111111,10000000 11111111
−127,...,−64,64,...127	7	11110	0000000 0111111,1000000 1111111
−63,...,−32,32,...63	6	1110	000000 011111,100000 111111
−31,...,−16,16,...31	5	110	00000 01111,10000 11111
−15,...,−8,8,...15	4	101	0000 0111,1000 1111
−7,...,−4,4,...7	3	100	000 011,100 111
−3,−2,2,3	2	011	00,01,10,11
−1,1	1	010	0,1
0	0	00	なし

表３．３　DC 差分値(輝度成分)符号化用ハフマン符号の例とビット

3･3･2 AC 成分の符号化

AC 成分の符号化とは、非零の AC 成分とその非零係数の前にある連続する零の AC 成分の個数を組み合わせて可変長符号を構成し、さらに DC 差分値の符号化と同様に付加ビットを加える。具体的な符号化手順は、最初に AC 成分を ZZ(1)から順番に見ていき、非零の AC 成分があれば、それ以前の零値だった AC 成分の個数を 4 ビットの RRRR(よって、0≦RRRR≦15 の範囲を取る)とし、さらに現在の非零の AC 成分に対応するカテゴリを 4 ビットの SSSS（SSSS の求め方は DC 差分値符号化の場合と一緒）として表し、これらの組み合わせを用いて表３．４の中で対応するハフマン符号を求める。このように零値のランレングスを加味した可変長符号化を行うことにより、連続する零値の個数を一括して一つの符号で表現できるため、高い符号化効率が得られる。

以上述べた AC 成分の処理には二つの特別な場合がある。そのうちの一つは零値の AC 成分が 16 個以上連続する場合である。すなわち、零ランレングスを表

現するための RRRR が 4 ビットであることにより 16 以上の零ランレングスは表現できない。そのため、連続する零値が 15 以下となるまで 16 個の零のグループに対して一つの RRRR/SSSS＝15/0 の組み合わせに対応する符号を出力する。もう一つの特別な処理は最後の AC 成分 ZZ(63)が 0 の場合であり、このときはブロックの最後に残った零値の AC 成分に対して一括して RRRR/SSSS＝0/0 の組み合わせに対応する符号を出力した後、直ちに当該ブロックの符号化処理を終了する。なお、ブロック内の最後の係数 ZZ(63)が非零の場合には EOB は用いられない。

ZRL または EOB 以外のハフマン符号の後には、具体的な AC 成分の大きさを特定するための付加ビットを加える。ようするに、非零 AC 成分が正の場合には AC 成分の値の LSB から SSSS ビットを、非零 AC 成分が負の場合には AC−1 の値の LSB から SSSS ビットを抽出し、これを付加ビットとする。DC 差分値の場合と同様に、復号器では付加ビットが 1 の場合に AC 成分は正、0 の場合には AC 成分は負であると判断できる。

SSSS / RRRR	0	1	2	･･･	10
0	1010(EOB)	00	01	･･･	1111111110000011
1	なし	1100	11011	･･･	1111111110001000
2	なし	11100	11111001	･･･	1111111110001110
･･･	なし	･･･	･･･	･･･	･･･
15	11111111001(ZRL)	1111111111110101	1111111111110110	･･･	1111111111111110

表３．４ Ac 成分符号化用ハフマン符号

３・４ ハフマン復号化の手順

復号化処理は、符号化処理を逆にたどることにより、復号化が可能になる。

3･4･1 DC 差分値の復号化

DC 差分値の復号化は、次の 3 つからなる
１、圧縮データから対応するハフマン符号を見つけて復号化し、符号化要素のカテゴリ SSSS を得る。
２、引き続いてビットストリームから SSSS ビット分の付加ビットを読み込み、DC 差分値 Diff をする。
３、Diff の先頭ビットが 0 の場合は、DC 差分値が負であるため、Diff＋1 を求めた後に SSSS＋1 ビット目の LSB 以上に"1"を詰めて負数に変換する。
最終的な DC 成分 ZZ(0)の再生には、上記手順によって得られた DC 差分値 Diff に、直前のブロックにおける DC 成分の再生値 Pred を加えることによって得られる。すなわち、

$$ZZ(0)＝Diff＋Pred \quad \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \text{式(4)}$$

となり、ハフマン符号化の復号化が完成する。

3･4･2 AC 成分の復号化

AC 成分の復号化は DC 差分値の復号化よりやや複雑で、以下の 6 つからなる。
１、AC 成分の配列 ZZ(i)(i≧1）を全て 0 にリセットする。
２、ビットストリームから対応するハフマン符号化を見つけて復号化し、零ランレングスと非零 AC 成分のカテゴリの組み合わせとなっている符号化要素シンボル RS を得る。
３、RS から AC 成分カテゴリ SSSS と零ランレングス RRRR を得る。
４、SSSS が 0 かつ RRRR が 15 であれば、AC 成分のアドレスを 16 個分更新して処理(2)に戻る。これは、16 個分の AC 成分を 0 とすることに等しい。
５､SSSS が 0 かつ RRRR が 15 であれば(すなわち EOB が検出されたらということ)、当該ブロックの復号化処理を終了する。これは、残りの AC 成分をすべて 0 とすることに等しい。
６、SSSS が 0 でなければ付加ビットを読み出し、付加ビットの先頭ビットが 0 の場合は AC 成分が負であるため、AC＋1 を求めた後に SSSS+1 ビット目の LSB 以上に"1"を詰めて負数に変換する。この時点で、DCT 係数のアドレスが 63 に達していれば復号化処理を終了し、63 に満たない場合は処理(2)に戻る。
この 6 つの手順を踏むことで、ＡＣ成分の復号化が可能となる。

第４章 JPEG アルゴリズムの実現

アルゴリズムの基本ブロックを図４．１のように分類して説明する。

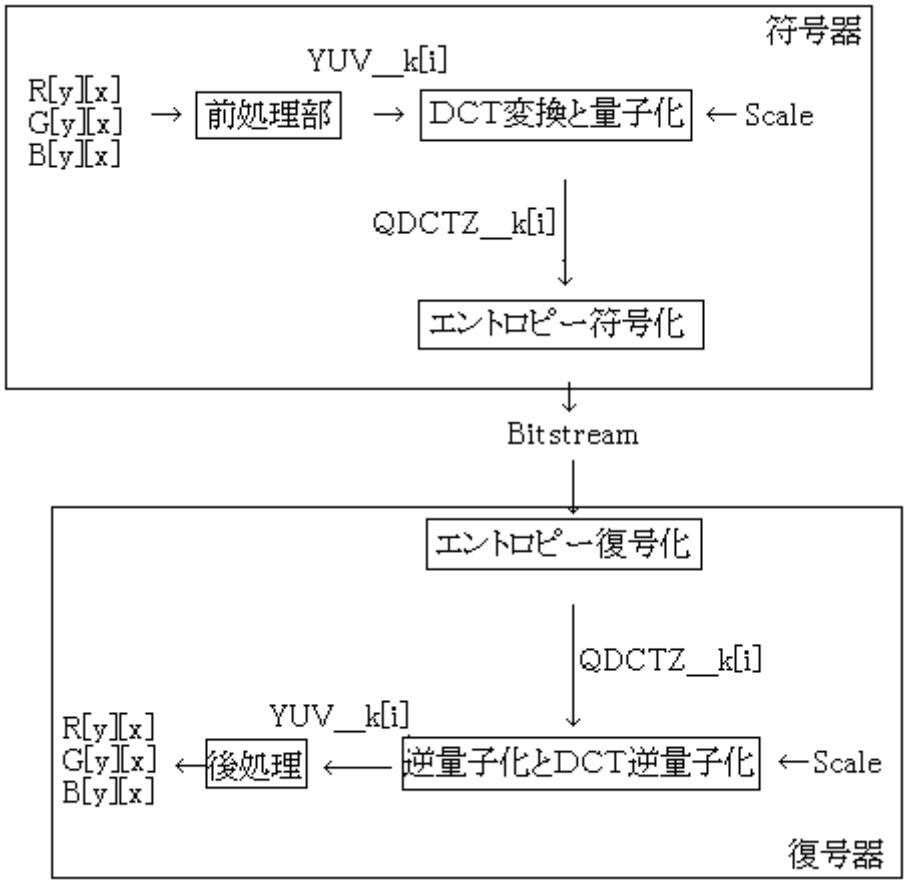

図 4. 1 符号化と復号化の構成

この図４．１は RGB の色信号を入出力信号として、DCT 変換/逆変換とエントロピー符号化/復号化は輝度と色差成分 YUV に変換して行う。符号化と復号化の間の情報はビットストリームを介して行われ、その媒体はデータ蓄積装置であったり伝送路であったりする。外部から入力されるパラメータは、量子化ひずみの大きさを直接制御する量子化スケールファクタ Scale である。通常は Scale=1 が用いられるが、さらに高品質な符号化を行う場合のは Scale＜１とすれば良い。

また色差間引きの方法には、4：4：4, 4：2：2, 4：1：1 がある。各色差間引きにおける MCU の構成は図４．2 に示す。なお本論文では、色差成分は空間周波数の帯域が輝度成分と比較して狭いことや視覚的に劣化が検知されなくいため、

縦横 1/2 の間引きを行う 4：1：1 の場合について行う。

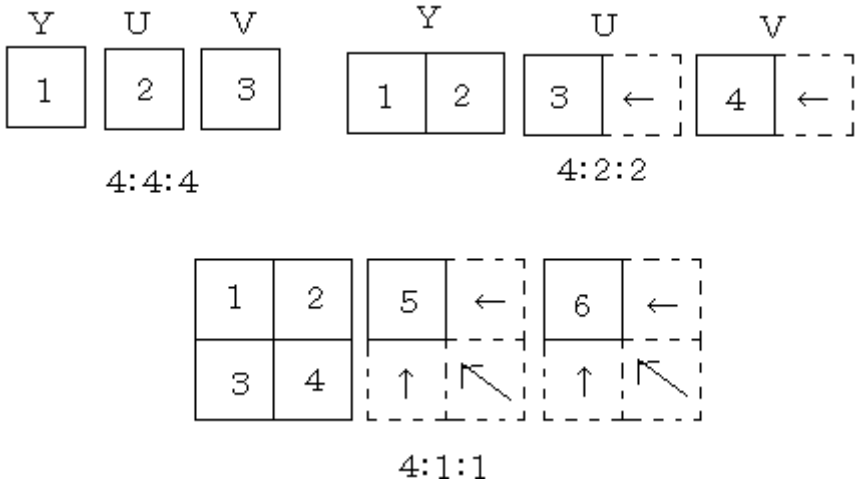

図４．2 色差間引き方法

４．１ ハフマン符号化・復号化の実現

ハフマン符号化

ハフマン符号化の入出力信号は、変換符号化部で作成された DCT 変換係数の量子化値が格納されている QDCTZ__k[i](k=1,・・・6)である。また出力信号は圧縮符号化されたビットストリームである。(図４．３にその形を示す)ハフマン符号化用のテーブルは、輝度成分と色差成分とで別々に定義することが可能であり、表４．１に示すように輝度成分と色差成分のそれぞれに対して DC 差分用と AC 成分用がある。

演算ブロック HuffmanEncode は、ハフマン符号化処理を実現している。最初に DC 差分値を求め、この差分値をハフマン符号化する。次に AC 成分のハフマン符号化を行うが、これは以下の三つの場合に分けてハフマン符号化を行う。

・EOB が検出された場合

・零ランレングスが 16 個以上となった場合

・その他の非零 AC 成分と零ランレングスの組み合わせの場合

である。

テーブル	意味
DcL_C,DcL_L	輝度成分のDC差分用ハフマン符号　(16ビット下詰め)と符号長
AcC_C,AcC_L	輝度成分のAC成分用ハフマン符号　(16ビット下詰め)と符号長
DcC_C,DcC_L	色差成分のDC差分用ハフマン符号　(16ビット下詰め)と符号長
AcL_C,AcL_L	色差成分のAC成分用ハフマン符号　(16ビット下詰め)と符号長

表４．１ ハフマン符号化テーブル

演算ブロック HuffmaneEncode 内の EncodeDc と EncodeAc は、それぞれ DC 差分値と AC 成分を実際にハフマン符号化テーブルを参照することによって符号化する。

・EncodeDc では最初に DC 差分値の大きさカテゴリｓ(表３．３参照)を決定し、ｓに対するハフマン符号をビットストリームに書き込む､さらにDC差分値のLSBからｓビット分を引き続いてビットストリームに書き込む。

・EncodeAc での処理も基本的には EncodeDc と同様であるが、大きさカテゴリｓに対応するハフマン符号を決定する時に、零ランレングス Run との組み合せによってテーブル参照（表３．４）する点が異なっている。

さらに EncodeDc と EncodeAc 内で使用する演算ブロック PutVlc は、上記手順に従って得られた符号を、ハフマン符号器の出力であるビットストリームに書き込むための処理を行う。

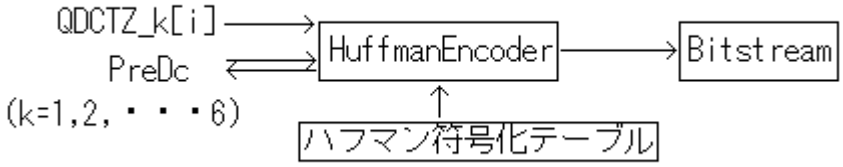

図４．３ ハフマン符号化

ハフマン復号化

ハフマン復号化の入力信号は、圧縮符号化されたビットストリームであり、出力信号はハフマン復号化された DCT 変換係数の量子化値 QDCTZ__k[i](k=1,・・・,6)である。(図４．４に図で示す) ハフマン復号化のテーブルは、輝度成分と色差成分とで別々に定義することが可能であり、表４．２に示すように輝度成分と色差成分のそれぞれに対して DC 差分用と AC 成分用がある。

テーブル	意味
DcL_Val,DcL_Min,DcL_Max,DcL_Huf	輝度成分のDC差分用 VALPTR,MINCODE,MAXCODE,HUFFVAL
AcL_Val,AcL_Min,AcL_Max,AcL_Huf	輝度成分のAC成分用 VALPTR,MINCODE,MAXCODE,HUFFVAL
DcC_Val,DcC_Min,DcC_Max,DcC_Huf	色差成分のDC差分用 VALPTR,MINCODE,MAXCODE,HUFFVAL
AcC_Val,AcC_Min,AcC_Max,AcC_Huf	色差成分のAC成分用 VALPTR,MINCODE,MAXCODE,HUFFVAL

表４．２ ハフマン符号化テーブル

演算ブロック HuffmanDcoder は、ハフマン復号化処理を実現している。最初に DC 差分値を復号化し、これに前ブロックの DC 復号値を加えて現ブロックの DC 成分を再生する。次に AC 成分のハフマン復号化を行い、EOB が検出された場合と、その他の非零の AC 成分と零ランレングスの組み合わせの場合とに場合分けして、AC 成分を復号化している。

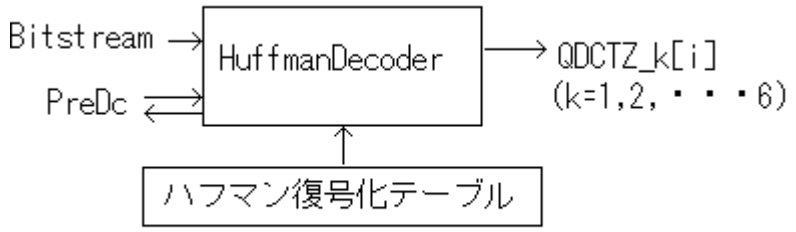

図４．４ ハフマン復号化

HuffmanDecoder 内の演算ブロック DecodeDc と DecodeAc は、それぞれ DC 差分値と AC 成分を実際にハフマン復号化テーブルを参照することによって復号化する。DecodeAc での処理も基本的には DecodeDc と同様であるが、ハフマン復号値から零ランレングス Run と大きさカテゴリｓを分離する点が異なっている。さらに DecodeDc と DecodeAc 内で使用する演算ブロック GetVlc は、ビットストリームからハフマン符号を指定されたビット数分だけ読み出す処理を行う。
これらハフマン符号化や復号化の C 言語での設計は、第五章で実現している。

第５章 Ｃ言語プログラム

５・１ Ｃ言語―プログラム

設計したのはハフマン符号化の輝度成分によるＤｃ差分値の符号化と復号化です。この符号化の構成は、第 3 章と第 4 章に説明した工程を踏むことにより実現する。

ＤＣ成分の符号化プログラム

```
#include <stdio.h>            /*注意だがＳとハフマンの値は表 3.3 を参照*/
#include <stdlib.h>
void main(void)
{
int QDCTZ[64]={16,0,0,0,0,0,0,0,180,0,0,0,0,0,0,0,
               0,0,0,0,0,0,0,0,210,0,0,0,0,0,0,0,    /*量子化値の代入*/
               0,0,0,0,0,0,0,0,312,0,0,0,0,0,0,0,
               0,0,0,0,0,0,0,0,936,0,0,0,0,0,0,0};
int PreDc=0;
int s;
int huffman;
int fukabit;
int bitstream_out;
int bits;
int Dc;
int k;
       Dc=QDCTZ[0]-PreDc;                          /*Dc 差分値の計算*/
       PreDc=PreDc+Dc;

       if(Dc==0)  /*これより下は Dc 差分値によってハフマンの値を決める*/
      {
         s=0;
         huffman=0;
         printf("bitstream_out(%d,2)\n",huffman);
         }
```

```
if(Dc==-1||Dc==1)
    {
     s=1;
     huffman=2;
     printf("bitstream_out(%d,3)\n",huffman);
       if(Dc==-1){fukabit=~Dc;
                  printf("bitstream_out(%d,%d)\n",fukabit,s);}
                  if(Dc==1){fukabit=Dc;
                  printf("bitstream_out(%d,%d)\n",fukabit,s);
                  }
    }

    if((Dc<=-2&&Dc>=-3)||(Dc>=2&&Dc<=3))
    {
     s=2;
     huffman=3;
     printf("bitstream_out(%d,3)\n",huffman);
     if(Dc<=-2&&Dc>=-3){fukabit=~Dc;
                        printf("bitstream_out(%d,%d)\n",fukabit,s);}
                        if(Dc>=2&&Dc<=3){fukabit=Dc;
                        printf("bitstream_out(%d,%d)\n",fukabit,s);
                         }
    }

    if((Dc<=-4&&Dc>=-7)||(Dc>=4&&Dc<=7))
    {
     s=3;
     huffman=4;
     printf("bitstream_out(%d,3)\n",huffman);
     if(Dc<=-4&&Dc>=-7){fukabit=~Dc;
                        printf("bitstream_out(%d,%d)\n",fukabit,s);
                          }
     if(Dc>=4&&Dc<=7){fukabit=Dc;
                        printf("bitstream_out(%d,%d)\n",fukabit,s);
                         }
    }
```

```
if((Dc<=-8&&Dc>=-15)||(Dc>=8&&Dc<=15))
{
 s=4;
 huffman=5;
 printf("bitstream_out(%d,3)\n",huffman);
 if(Dc<=-8&&Dc>=-15){fukabit=~Dc;
                  printf("bitstream_out(%d,%d)\n",fukabit),s;
                    }
 if(Dc>=8&&Dc<=15){fukabit=Dc;
                  printf("bitstream_out(%d,%d)\n",fukabit,s);
                    }
  }

 if((Dc<=-16&&Dc>=-31)||(Dc>=16&&Dc<=31))
 {
 s=5;
 huffman=6;
 printf("bitstream_out(%d,3)\n",huffman);
 if(Dc<=-16&&Dc>=-31){fukabit=~Dc;
                  printf("bitstream_out(%d,%d)\n",fukabit,s);
                      }
 if(Dc>=16&&Dc<=31){fukabit=Dc;
                  printf("bitstream_out(%d,%d)\n",fukabit,s);
                      }
}

 if((Dc<=-32&&Dc>=-63)||(Dc>=32&&Dc<=63))
 {
 s=6;
 huffman=14;
 printf("bitstream_out(%d,4)\n",huffman);
 if(Dc<=-32&&Dc>=-63){fukabit=~Dc;
                  printf("bitstream_out(%d,%d)\n",fukabit,s);
                      }
 if(Dc>=32&&Dc<=63){fukabit=Dc;
```

```
                printf("bitstream_out(%d,%d)\n",fukabit,s);
                    }
 }

if((Dc<=-64&&Dc>=-127)||(Dc>=64&&Dc<=127))
{
s=7;
huffman=30;
printf("bitstream_out(%d,5)\n",huffman);
if(Dc<=-64&&Dc>=-127){fukabit=~Dc;
                printf("bitstream_out(%d,%d)\n",fukabit,s);
                      }
if(Dc>=64&&Dc<=127){fukabit=Dc;
                 printf("bitstream_out(%d,%d)\n",fukabit,s);
                  }
 }

if((Dc<=-128&&Dc>=-255)||(Dc>=128&&Dc<=255))
{
s=8;
huffman=62;
printf("bitstream_out(%d,6)\n",huffman);
if(Dc<=-128&&Dc>=-255){fukabit=~Dc;
                printf("bitstream_out(%d,%d)\n",fukabit,s);
                      }
if(Dc>=128&&Dc<=255){fukabit=Dc;
                 printf("bitstream_out(%d,%d)\n",fukabit,s);
                   }
 }

if((Dc<=-256&&Dc>=-511)||(Dc>=256&&Dc<=511))
{
s=9;
huffman=126;
printf("bitstream_out(%d,7)\n",huffman);
if(Dc<=-256&&Dc>=-511){fukabit=~Dc;
```

```
                    printf("bitstream_out(%d,%d)\n",fukabit,s);
                        }
        if(Dc>=256&&Dc<=511){fukabit=Dc;
                     printf("bitstream_out(%d,%d)\n",fukabit,s);
                        }
         }

        if((Dc<=-512&&Dc>=-1023)||(Dc>=512&&Dc<=1023))
        {
        s=10;
        huffman=254;
        printf("bitstream_out(%d,8)\n",huffman);
        if(Dc<=-512&&Dc>=-1023){fukabit=~Dc;
                    printf("bitstream_out(%d,%d)\n",fukabit,s);
                        }
        if(Dc>=512&&Dc<=1023){fukabit=Dc;
                    printf("bitstream_out(%d,%d)\n",fukabit,s);
                        }
         }

        if((Dc<=-1024&&Dc>=-2047)||(Dc>=1024&&Dc<=2047))
        {
        s=11;
        huffman=510;
        printf("bitstream_out(%d,9)\n",huffman);
        if(Dc<=-1024&&Dc>=-2047){fukabit=~Dc;
                    printf("bitstream_out(%d,%d)\n",fukabit,s);
                        }
        if(Dc>=1024&&Dc<=2047){fukabit=Dc;
                     printf("bitstream_out(%d,%d)\n",fukabit,s);
                        }
         }
}
```

Ｄｃ差分値の復号化プログラム

```
#include<stdio.h>          /*注意だがSとハフマンの値は表3.3を参照*/
#include<stdlib.h>
void main(void)
{
int i;
int code;
int max[17]={0xffff,0xffff,0x0000,0x0006,0x000e,0x001e,  /*ハフマン値
             0x003e,0x007e,0x00fe,0x01fe,0xffff,0xffff,      の最大値*/
             0xffff,0xffff,0xffff,0xffff,0xffff};
int min[17]={0x0000,0x0000,0x0000,0x0002,0x000e,0x001e,     /*ハフマン値
              0x003e,0x007e,0x00fe,0x01fe,0x0000,0x0000,     の最小値*/
              0x0000,0x0000,0x0000,0x0000,0x0000};
int val[17]={0,0,0,1,6,7,8,9,10,11,0,0,0,0,0,0,0};     /*大きさの順番*/
int huf[12]={0x00,0x01,0x02,0x03,0x04,0x05,0x06,
             0x07,0x08,0x09,0x0a,0x0b};
int Dc;
int DC_out;
int next;
int PreDc=0;
int bits=3;
int huffman=6;
int QDCTZ;
int j;
int s;
int sifuto=4;
i=1;
code=0x0000;
next=(huffman>>sifuto);
code=next;
while(code>max[i-1])        /*ハフマンの値からDc差分値の領域を探す*/
     {
     i=i+1;
     next=(huffman>>sifuto--);
     code=next;
```

```
    }
       if(code<max[i])
          {
          j=val[i]+code-min[i];
          s=huf[j];                    /*s の値を決定する*/
       printf("%d\n",s);

          if(s!=0)
            {
             if((Dc&(1<<(s-1)))==0) /*以下ｓの値によって QDCTZ を決定*/
                 {
                  Dc=Dc-((1<<s)-1);
                     QDCTZ=Dc+PreDc;
                     PreDc=PreDc+Dc;
                     printf("DC_out(%d)\n",QDCTZ);
                   }
               if((Dc&(1<<(s-1)))!=0)
                  {
                  QDCTZ=Dc+PreDc;
                     PreDc=PreDc+Dc;
                  printf("DC_out(%d)\n",QDCTZ);
                     }

        else
                    {
                    Dc=0;
                    printf("DC_out(%d)\n",Dc);
                       }
               } }

}
```

５・２ 実行結果

Ｄｃ差分値プログラム結果
入力
Dc 成分＝16
前 Dc 成分＝０
量子化値＝P23 を参照

出力
bitstream=(6,3)　　bitstream=(ハフマン符号、ビット数)

bitstream=(16,5)　　bitstream=(付加ビット、ビット数)

となった。

Ｄｃ差分値の復号化では、
入力
bitstream=(6,3)　　bitstream=(ハフマン符号、ビット数)

bitstream=(16,5)　　bitstream=(付加ビット、ビット数)

PreDc＝０

出力
QDCTZ＝１６　　QDCTZ＝Dc 成分

となり、Dc 成分と復号化した値が同じなため、Dc 成分の符号化・復号化が完成した。
ここでは、輝度成分の符号化・復号化を行ったが、色成分の符号化・復号化のプログラムは値が違うだけで、プログラムの書き方はまったく一緒なので、輝度成分のみを設計した。あと符号化・復号化の入力と出力の別のパターンを図 5.1 に示しているので、これらの結果から Dc 差分値による C 言語プログラムが完成したと考えている。

符号化			復号化	
入力	出力		入力	出力
(2, 5)	011	-3	入力値は符号化によって出た出力を入れる。	2
(16, 5)	101	11		16
(156, 23)	11110	133		156
(1356, 250)	111111110	1106		1356
(現Dc成分, 前Dc成分)	ハフマン符号	付加ビット		現Dc成分

図５．１ 入力と出力

まとめ

Ｃプログラムを実行してみると、符号器に使用する値と復号器によって出された値が一致したため、Ｄｃ成分のハフマン符号化・復号化が完成した。このＣプログラム作る際、Ｃ言語の知識が乏しいため小さなミスなどにより、実行結果が上手く動かず時間を大きくロスしてしまい、VHDL による設計を行うことができなかった。今後の課題として VHDL によるＤｃ成分の回路設計、また画像圧縮についての更なる知識を得ることを試みたい。

謝辞

本論文の作成にあたり、終始適切な助言を賜り、また丁寧に指導して下さった橘 昌良先生に感謝します。それに、適宜指導をいただいた原央教授、矢野政顯教授にも厚く御礼申し上げます。

同窓生の皆さん、研究室のメンバー（垣本真志・中田勝）には常に刺激的な議論を頂き、精神的にも支えられました。ほんとうにありがとうございました。これからもお世話になると思いますが、よろしくお願いいたします。

参考文献

[1] 小野定康/鈴木純司：「ＪＰＥＧ/ＭＰＥＧ２の技術」，オーム社(2001)
[2] 原島博：「画像情報圧縮」，オーム社(1991)
[3]「Design Wave Magazine」，和田友久，2001 年 11 月号
[4] 杉江日出澄/山本格：「C プログラミングの学習」，培風館(2000)
[5] 佐々木一郎：「C 言語入門」，朝倉書店(1992)

＊付録＊

ＤＣ成分の符号化プログラム

```
#include <stdio.h>          /*注意だがSとハフマンの値は表3.3を参照*/
#include <stdlib.h>
void main(void)
{
int QDCTZ[64]={16,0,0,0,0,0,0,0,180,0,0,0,0,0,0,0,
               0,0,0,0,0,0,0,0,210,0,0,0,0,0,0,0,   /*量子化値の代入*/
               0,0,0,0,0,0,0,0,312,0,0,0,0,0,0,0,
               0,0,0,0,0,0,0,0,936,0,0,0,0,0,0,0};
int PreDc=0;
int s;
int huffman;
int fukabit;
int bitstream_out;
int bits;
int Dc;
int k;
       Dc=QDCTZ[0]-PreDc;                          /*Dc差分値の計算*/
       PreDc=PreDc+Dc;

       if(Dc==0)  /*これより下はDc差分値によってハフマンの値を決める*/
      {
         s=0;
         huffman=0;
         printf("bitstream_out(%d,2)\n",huffman);
         }

    if(Dc==-1||Dc==1)
        {
         s=1;
         huffman=2;
         printf("bitstream_out(%d,3)\n",huffman);
           if(Dc==-1){fukabit=~Dc;
```

```
                printf("bitstream_out(%d,%d)\n",fukabit,s);}
                if(Dc==1){fukabit=Dc;
                printf("bitstream_out(%d,%d)\n",fukabit,s);
                }
}

if((Dc<=-2&&Dc>=-3)||(Dc>=2&&Dc<=3))
{
 s=2;
 huffman=3;
 printf("bitstream_out(%d,3)\n",huffman);
 if(Dc<=-2&&Dc>=-3){fukabit=~Dc;
                    printf("bitstream_out(%d,%d)\n",fukabit,s);}
                    if(Dc>=2&&Dc<=3){fukabit=Dc;
                    printf("bitstream_out(%d,%d)\n",fukabit,s);
                     }
}

if((Dc<=-4&&Dc>=-7)||(Dc>=4&&Dc<=7))
{
 s=3;
 huffman=4;
 printf("bitstream_out(%d,3)\n",huffman);
 if(Dc<=-4&&Dc>=-7){fukabit=~Dc;
                    printf("bitstream_out(%d,%d)\n",fukabit,s);
                      }
 if(Dc>=4&&Dc<=7){fukabit=Dc;
                    printf("bitstream_out(%d,%d)\n",fukabit,s);
                     }
}

if((Dc<=-8&&Dc>=-15)||(Dc>=8&&Dc<=15))
{
 s=4;
 huffman=5;
 printf("bitstream_out(%d,3)\n",huffman);
```

```
 if(Dc<=-8&&Dc>=-15){fukabit=~Dc;
                 printf("bitstream_out(%d,%d)\n",fukabit),s;
                   }
 if(Dc>=8&&Dc<=15){fukabit=Dc;
                 printf("bitstream_out(%d,%d)\n",fukabit,s);
                   }
  }

 if((Dc<=-16&&Dc>=-31)||(Dc>=16&&Dc<=31))
 {
 s=5;
 huffman=6;
 printf("bitstream_out(%d,3)\n",huffman);
 if(Dc<=-16&&Dc>=-31){fukabit=~Dc;
                 printf("bitstream_out(%d,%d)\n",fukabit,s);
                     }
 if(Dc>=16&&Dc<=31){fukabit=Dc;
                 printf("bitstream_out(%d,%d)\n",fukabit,s);
                     }
}

 if((Dc<=-32&&Dc>=-63)||(Dc>=32&&Dc<=63))
 {
 s=6;
 huffman=14;
 printf("bitstream_out(%d,4)\n",huffman);
 if(Dc<=-32&&Dc>=-63){fukabit=~Dc;
                 printf("bitstream_out(%d,%d)\n",fukabit,s);
                     }
 if(Dc>=32&&Dc<=63){fukabit=Dc;
                 printf("bitstream_out(%d,%d)\n",fukabit,s);
                     }
  }

 if((Dc<=-64&&Dc>=-127)||(Dc>=64&&Dc<=127))
 {
```

```
s=7;
huffman=30;
printf("bitstream_out(%d,5)\n",huffman);
if(Dc<=-64&&Dc>=-127){fukabit=~Dc;
                printf("bitstream_out(%d,%d)\n",fukabit,s);
                      }
if(Dc>=64&&Dc<=127){fukabit=Dc;
                 printf("bitstream_out(%d,%d)\n",fukabit,s);
                  }
 }

if((Dc<=-128&&Dc>=-255)||(Dc>=128&&Dc<=255))
{
s=8;
huffman=62;
printf("bitstream_out(%d,6)\n",huffman);
if(Dc<=-128&&Dc>=-255){fukabit=~Dc;
                printf("bitstream_out(%d,%d)\n",fukabit,s);
                      }
if(Dc>=128&&Dc<=255){fukabit=Dc;
                 printf("bitstream_out(%d,%d)\n",fukabit,s);
                   }
 }

if((Dc<=-256&&Dc>=-511)||(Dc>=256&&Dc<=511))
{
s=9;
huffman=126;
printf("bitstream_out(%d,7)\n",huffman);
if(Dc<=-256&&Dc>=-511){fukabit=~Dc;
                printf("bitstream_out(%d,%d)\n",fukabit,s);
                      }
if(Dc>=256&&Dc<=511){fukabit=Dc;
                 printf("bitstream_out(%d,%d)\n",fukabit,s);
                    }
 }
```

```
            if((Dc<=-512&&Dc>=-1023)||(Dc>=512&&Dc<=1023))
            {
            s=10;
            huffman=254;
            printf("bitstream_out(%d,8)\n",huffman);
            if(Dc<=-512&&Dc>=-1023){fukabit=~Dc;
                            printf("bitstream_out(%d,%d)\n",fukabit,s);
                                  }
            if(Dc>=512&&Dc<=1023){fukabit=Dc;
                            printf("bitstream_out(%d,%d)\n",fukabit,s);
                                }
             }

            if((Dc<=-1024&&Dc>=-2047)||(Dc>=1024&&Dc<=2047))
            {
            s=11;
            huffman=510;
            printf("bitstream_out(%d,9)\n",huffman);
            if(Dc<=-1024&&Dc>=-2047){fukabit=~Dc;
                            printf("bitstream_out(%d,%d)\n",fukabit,s);
                                    }
            if(Dc>=1024&&Dc<=2047){fukabit=Dc;
                             printf("bitstream_out(%d,%d)\n",fukabit,s);
                                    }
             }
}
```

Ｄｃ差分値の復号化プログラム

```
#include<stdio.h>          /*注意だがSとハフマンの値は表3.3を参照*/
#include<stdlib.h>
void main(void)
{
int i;
int code;
int max[17]={0xffff,0xffff,0x0000,0x0006,0x000e,0x001e,  /*ハフマン値
             0x003e,0x007e,0x00fe,0x01fe,0xffff,0xffff,      の最大値*/
             0xffff,0xffff,0xffff,0xffff,0xffff};
int min[17]={0x0000,0x0000,0x0000,0x0002,0x000e,0x001e,     /*ハフマン値
              0x003e,0x007e,0x00fe,0x01fe,0x0000,0x0000,     の最小値*/
              0x0000,0x0000,0x0000,0x0000,0x0000};
int val[17]={0,0,0,1,6,7,8,9,10,11,0,0,0,0,0,0,0};     /*大きさの順番*/
int huf[12]={0x00,0x01,0x02,0x03,0x04,0x05,0x06,
             0x07,0x08,0x09,0x0a,0x0b};
int Dc;
int DC_out;
int next;
int PreDc=0;
int bits=3;
int huffman=6;
int QDCTZ;
int j;
int s;
int sifuto=4;
i=1;
code=0x0000;
next=(huffman>>sifuto);
code=next;
while(code>max[i-1])        /*ハフマンの値からDc差分値の領域を探す*/
     {
     i=i+1;
     next=(huffman>>sifuto--);
     code=next;
```

```
    }
       if(code<max[i])
          {
          j=val[i]+code-min[i];
          s=huf[j];                    /*s の値を決定する*/
       printf("%d\n",s);

          if(s!=0)
            {
             if((Dc&(1<<(s-1)))==0) /*以下ｓの値によって QDCTZ を決定*/
                 {
                  Dc=Dc-((1<<s)-1);
                     QDCTZ=Dc+PreDc;
                     PreDc=PreDc+Dc;
                     printf("DC_out(%d)\n",QDCTZ);
                   }
               if((Dc&(1<<(s-1)))!=0)
                  {
                  QDCTZ=Dc+PreDc;
                     PreDc=PreDc+Dc;
                  printf("DC_out(%d)\n",QDCTZ);
                     }

        else
                    {
                    Dc=0;
                    printf("DC_out(%d)\n",Dc);
                       }
               } }

}
```