卒 業 研 究 報 告

題　　目

ＭＰＥＧ２デコーダにおける動き補償回路の設計

指 導 教 員

橘 昌良

報 告 者

学籍番号　1051009
氏名　　垣本真志

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

１章　はじめに

MPEG の内容を理解すると共に、動画像の復号化における動き補償部分について C 言語でプログラムした結果を確認し、検討をする。

２章では MPEG の概要と、歴史などについて説明する。

３章では MPEG の内部構造について詳しく説明する。

４章ではこの研究内容の動き補償回路についてのさらに詳しい説明と、プログラムの内容などについて説明する。

５章では４章のプログラムを用いて得た結果、そして考察を記す。

６章ではまとめ，謝辞、参考文献を記す。

2章　ＭＰＥＧの概要

２・１　ＭＰＥＧとは何か

ＭＰＥＧとは“Moving Picture Experts Group”の頭文字をとったもので、目的は動画像とオーディオの符号化方式の標準化である。簡単に言うと動画像のデータを圧縮してデータ量を減らし、その圧縮されたデータを逆の手順で元の動画像に戻し再生する技術である。

コンピュータをＬＡＮやインターネット等の回線に接続し、情報を互いにやりとりする事が頻繁に行われるようになり、メールや文章、音声（音楽)、静止画像、動画像を送ったり受信するにともない、音声や静止画像は初めにサンプリングした音質や画像の鮮明さも関係するが膨大な情報量になった。それをインターネットでそのまま送ったりすると転送時間や、回線の混雑、メモリ容量の拡大など様々な支障がでてくる。そして、音声と静止画像の連続が組み合わさった動画像となるとますます情報量は膨大になり、それらを回避するためにデータ量の削減が要求される。

それには画像、動画像を送る場合にデータ量を圧縮して小さくしてやる事により、少ないデータ量で扱えるようにし、通信速度の向上や回線、メモリ容量の有効利用が見込める。この技術が最近世間でもよく聞くようになった JPEG，MPEG と言われる符号化方式である。

動画像は情報量が多いといわれるが実際どれくらい多いのか。たとえば電話回線でＡ４ファックス 1 枚を精細モード（白黒）で送るとする。画素数は仮に１７００×２３００とすると、

１７００×２３００＝3,９１０,０００ ビット

になる。
ＴＶの画像をみてみると、ファックスほど細かい文字はないので仮に２５０×４００画素とし、色は白黒だけの場合、

２５０×４００＝１００,０００ ビット（画素）

となる。
これは１と０を 10 万個使って、1 画面を作ることになる。

次に濃淡（明るさ）の違いを表すとすれば 1 画素が 8 ビットになるので情報量も 8 倍となり

２５０×４００×８＝８００,０００ ビット

さらにカラー画像の場合は赤、緑、青（ＲＧＢ）の 3 成分が必要になり、さらに 3 倍の

２５０×４００×８×３＝2,４００,０００ ビット

になる。
その上、動画になると 1 秒間に３０枚の画像を伝送しないといけないので 1 秒あたりの情報量は、

２５０×４００×８×３×３０＝７２,０００,０００ ビット

となりファックスの精細モードの約 20 倍、電話回線での音声情報（1 秒間）の約１０００倍もの情報量になる。

MPEG と良く似た名前の圧縮技術で JPEG といわれるものがある。JPEG とは国際標準化機関 ISO と CCITT の 2 つの団体による合弁グループ「Joint Photographic Experts Group」の頭文字をとったもので、写真などの静止画像に対する符号化の規格である。

JPEG 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た非可逆変換方式であり、JPEG が圧縮の過程でイメージを美しく、かつサイズを小さくするために原画と異なる色とその配列を作り出している。

非可逆変換とは、「元の画像と復元後の画像とでは、まったく同一の品質、画質での復元はできない。」と言う事である。しかしながら、可逆変換方式ではできないような大幅な圧縮を実現させている。

MPEG の圧縮符号化は、高圧縮率を実現するためにさまざまな基本技術が組み合わされているが、簡単には MPEG には、JPEG に動き補償が加わったものとして解釈できる。

２・２ ＭＰＥＧの歴史

当初の計画としては、MPEG１から MPEG４までの 4 種類が予定されていたが途中の見直しにより MPEG３が消滅した。それらを当初の予定と比較して見てみる。

当初の予定

MPEG１ … 家庭で使用する程度の画像品質で１．５Mb/s 程度の符号化
MPEG２ … 放送局で要する程度の高品質で１６Mb/s 程度の符号化
MPEG３ … 高品質テレビを５０～８０ Mb/s で符号化
MPEG４ … 無線通信用で６４Kb/s 以下での符号化

実際には、以下のようになる。

ＭＰＥＧ１

動画フォーマットとして１９９３年に標準化が完了している。主な使用目的はＶｉｄｅｏ ＣＤ等。

正式名称は“ Coding of moving pictures and associated audio for digital storage media at up to bout 1.5 Mbps”

ＭＰＥＧ２（兼ＭＰＥＧ３）

現行ＴＶ映像を放送用品質で通信、配信、蓄積などを利用することを目的としてスタートし、符号化速度も１０Ｍｂｐｓ以下を想定していた。しかし、そのあとの議論によりさらに広い範囲への応用も含めた汎用符号化とすることに変更された。これにより当初はＭＰＥＧ３にＨＤＴＶレベルの符号化を検討していたがＭＰＥＧ２に取り込まれる形になりＭＰＥＧ３は歴史上には“ 欠番 ”となる。

正式名称は

“Generic coding of moving pictures and associated audio”

ＭＰＥＧ４

携帯用ＴＶ電話等を主な用途として６４Ｋｂｐｓ未満の極低レートがターゲットであったが、その後インターネットによる画像通信が急速に立ち上がり、それに伴いＭＰＥＧ４の位置付けも自然画像やＣＧ、アニメーション画像とそれに付随する音声、音楽の総合的符号化へと変わっていった。

１９９６年にはマルチメディアコンテントの記述インターフェースを規定する目的で MPEG７が活動を開始。

２・３ MPEG１と MPEG２の相違点

まず画像の表示方式であるがプログレッシブ画像とインタレース画像がある。

インタレース画像はプログレッシブ画像における奇数番目と偶数番目の走査線を交互に表示する。プログレッシブに比べ半分の情報量になるのが利点である。

プログレッシブ画像というのは順次走査画像とよばれ、画面上の走査線を一括してスキャンし、毎秒 60 フレームを奇数フィールド、偶数フィールドともに 1 枚絵として描いていく方式である。走査線を奇数フィールド、偶数フィールドに分けて表示するインターレースに比べ、ちらつきがなく密度の濃い映像が得られるという利点がある。

MPEG１はプログレッシブ画像のみを対象とする符号化処理であるが MPEG２はプログレッシブ画像に加えてインタレース画像をも対象とする符号化処理であり、インタレース画像の符号化効率を向上させるためのさまざまな技術が盛り込まれている。

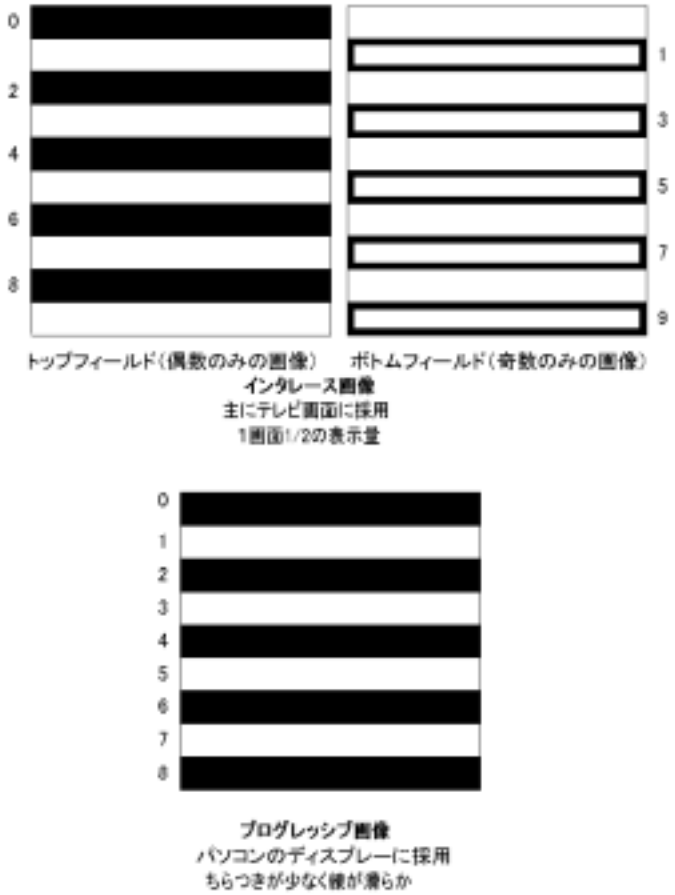


図 2-1　インタレース画像とプログレッシブ画像

２・４　ＭＰＥＧアルゴリズムの基本構成

ＭＰＥＧで規定される部分は、符号器が出力するビットストリームの構造と、そのビットストリームの復合化方法であり、符号化方法は含まれていない。つまり自由度を広くしてビットレート等をユーザーの選択に任せるという事である。

ビットレートとは、データ転送で、１秒間に転送するビット数の事で、一般にビットレートが高いほど質の高い映像が転送される。

ＭＰＥＧとは別の音声圧縮技術の一つで、ＡＤＰＣＭ（Adaptive Differential Pulse Code Modulation）方式というのがある。音声をデジタルデータに変換する方式の音が連続的に変化すること利用して、直前に数値化したデータとの差を記録することによってデータ量を減らすことができる。

音声符号化３２ｋｂｐｓＡＤＰＣＭ（Adaptive Differential Pulse Code Modulation）アルゴリズムでは、符号化と復号化の方法が細かく厳密に定められており、これにそのまま動画像を含めようと考えるとドキュメントの分量が膨大になってしまい、事実上そのような規定方法は不可能になる。

このようにMPEGはこうした問題を回避するためビットストリームの構造と復号化のみのアルゴリズム規定法を用いている。

３章　ＭＰＥＧの内部構造

３・１　　符号器、復号器の構造

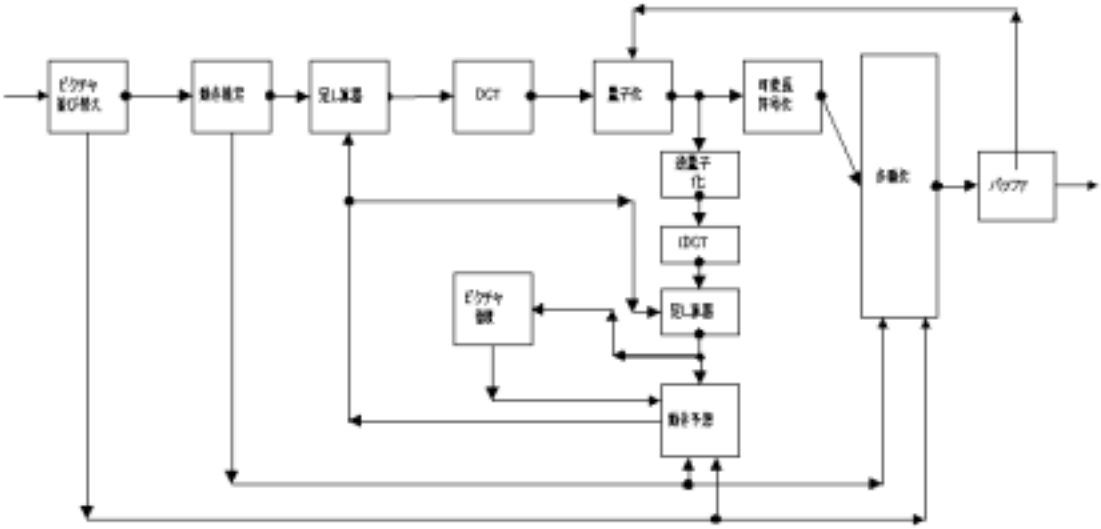


図3-1　MPEG符号器の構成

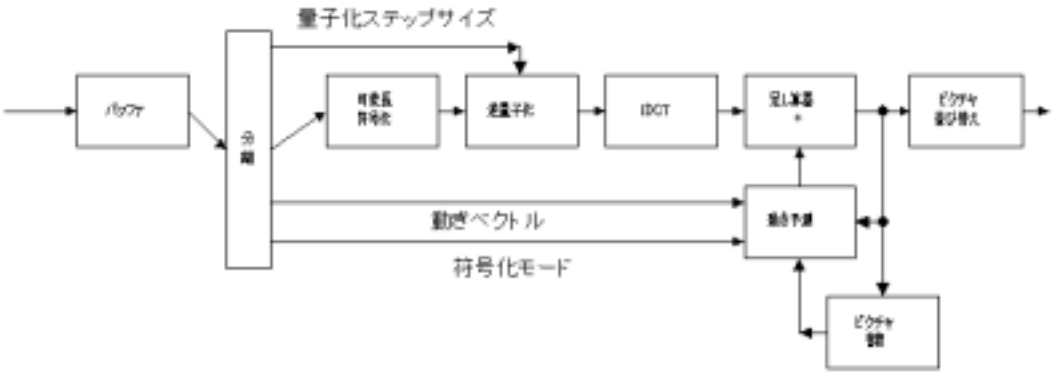


図3-2　MPEG復号器の構成

図 3-1 図 3-2 は符号器、復号器を表している。これらの図を見るとＭＰＥＧはＪＰＥＧのＤＣＴ方式と同様に復号画像の画素値が入力画像の画素値に一致しない有ひずみ符号化である事がわかる。また無歪み符号化は選択できない。

３・２　ビットストリームの構造

ビットストリームとは２次元の画像の情報を１次元の情報に並び替え、情報を実際に蓄積したり伝送したりするために使用される。

このビットストリームはシーケンスヘッダ、シーケンス拡張、GOP ヘッダ、ピクチャヘッダ，ピクチャデータ、拡張とユーザデータの階層があり、これを実現するためにはビットストリーム内の個々のフラグやパラメータがどの階層に属しているかを具体的に示すための目印が必要である。この目印の役目を負うのが開始符号（start code）と呼ばれる８ビットの特別の符号がある。

３・３　プロファイルとレベル

MPEG２が汎用符号化規定とされるのはさまざまなアプリケーション（プログラム）を想定して機能や品質を選択できるアルゴリズム構成となっているからである。すなわち、ＭＰＥＧ２の規定には符号器の規定が存在せず、符号器が出力するビットストリームの内容の規定、およびそのビットストリームからさまざまなパラメータやフラグを読み出して、「希望の品質を持つ画像」を復号化する復号化手順の規定で MPEG2 は構成されている。

この「希望の品質を持つ画像」の言葉の中に広範囲の応用に対応可能な能力を持つという意味が内在されている。ただし、考え得るすべての応用を網羅した復号器を定義してそれらの間の互換性を保証することは現実的でない。

このような複数の機能・品質のサブセットを定義するために MPEG２ではプロファイル（Profile）とレベル(Level)の考え方を導入している。

プロファイルとはビットストリームを構成するパラメータやフラグのサブセットを表し、これにより符号化、復号化アルゴリズムの基本構成が定義される。各プロファイルは異なるパラメータのサブセットを持ち、付加的なパラメータはビットストリーム内に存在する拡張と呼ばれるパラメータのサブセットで指示させるようになっている。

一方、レベルとはビットストリーム内のパラメータに加えられる制約条件を表し、これによりプロファイルで定まる一定のアルゴリズム構成において達成される品質が定義される。パラメータの制約条件とは、例えば画像のサイズやビットレートなどに加えられる制限の事である。

３・４　動き補償

ＭＰＥＧにおける圧縮に大きな効果を発揮する重要な部分が動き補償と呼ばれるものである。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（連続したフレーム）において、それらの画素の集合の動きに注目して、この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。

例えばカメラを固定して撮影するとする。画面の左端にボールがあるという設定で、そこをボールが画面の左端から画面の右端にころがって移動したという動画像を撮ったとする。

ボールが左端にある時から、右に移動するまでの間のフレーム数を仮に１０枚とし、この時に画面上で動いたのはボールのみで背景はまったく動いていない。ボールの動く範囲以外の景色は変化しないので背景の部分は１枚目から１０枚目まで共通として扱う事ができ、変化していくのは１枚目のボールの位置から２枚目のボールの位置、2 枚目～3 枚目、3 枚目～4 枚目の変化した部分といったようにフレーム全体ではなく、動いた所の「差分」を扱う事によりデータ量を圧縮していくわけである。

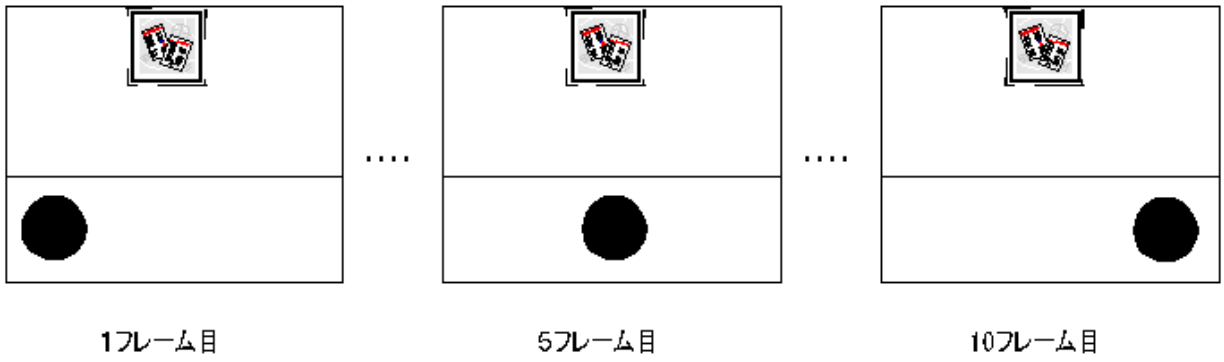


図 3-3　ボールの転がっていく動画

３・５　ピクチャタイプ

前述の動き補償の所で述べた圧縮の方法は同じ背景の場合には非常に有効である。しかし、実際扱われる動画は画面の切り替わりやカメラの移動などにより時間によって背景、対象物共に絶えず変化していく。瞬時に画像が切り替わった時に対応するためにピクチャタイプというものがある。

ピクチャタイプは３つのタイプがあり、イントラピクチャ（Intra-coded picture)、予測符号化ピクチャ(Predictive-coded picture)、双方向予測符号化ピクチャ(Bidirectionally predictive-coded picture)がある。簡単にＩピクチャ、Ｐピクチャ、Ｂピクチャとも呼ぶ。

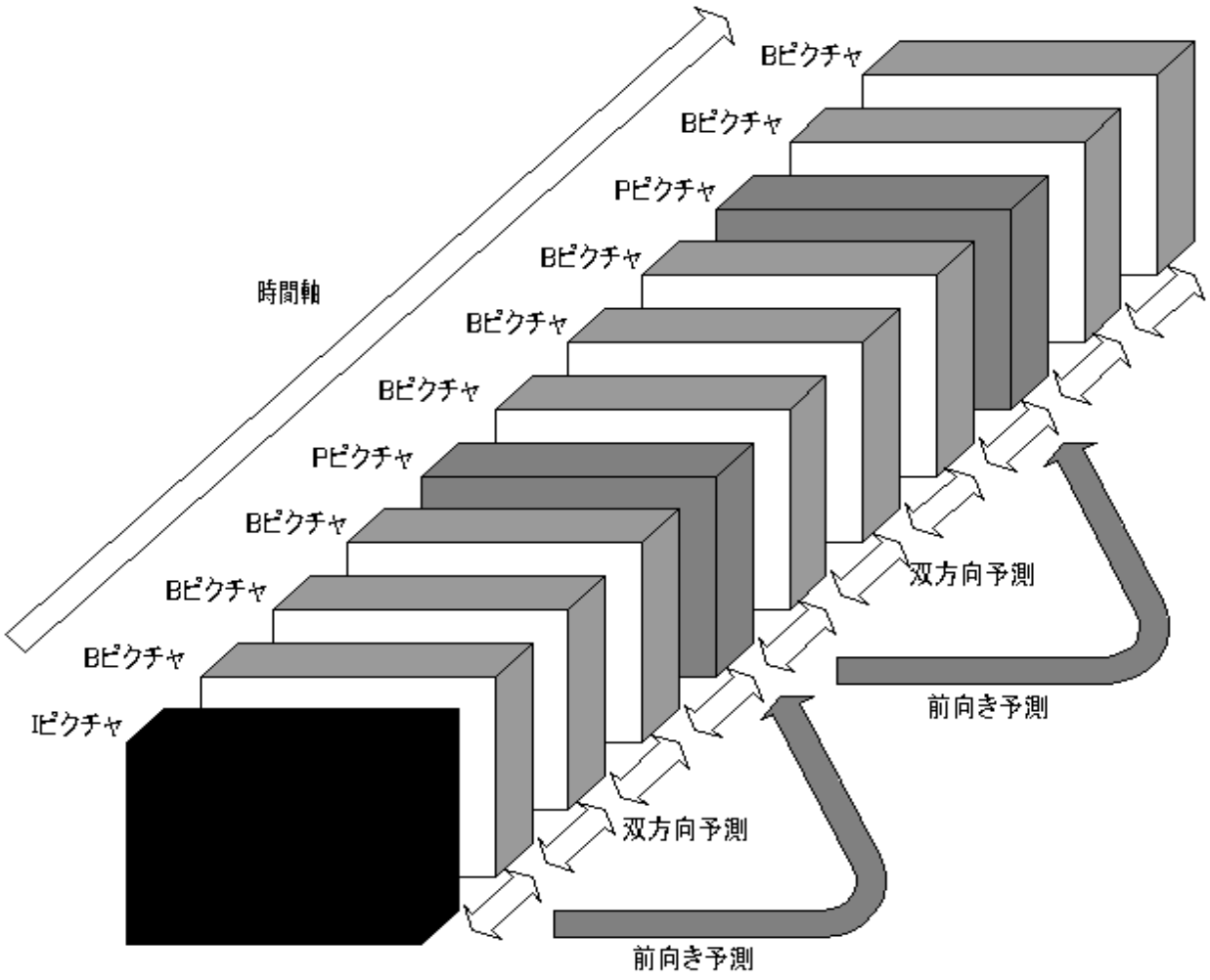


図 3-4　３つのピクチャタイプの構成例

I ピクチャ

I ピクチャは他のピクチャの情報を使用せず、JPEG のようにそれ自身のピクチャ情報のみで符号化される。3 つのタイプの中で一番圧縮率が低いが独立して復号化が可能であるためランダムアクセス時のアクセス点として利用される。

P ピクチャ

過去の I ピクチャまたは P ピクチャを参照ピクチャとして、時間軸上で前向き動き予測符号化される。I ピクチャより圧縮率は高いがその復号化には過去の I ピクチャの要素が必要である。

B ピクチャ

過去と将来の I ピクチャまたは P ピクチャを参照ピクチャとして、時間軸上で前向きおよび後ろ向き動き予測符号化される。3 つのピクチャの中で一番圧縮率が高いが、B ピクチャの復号化には将来の P ピクチャの復号化が先に完了していなくてはならないため、復号化画像を表示する時点で遅延が生じる。

３・６　ビデオデータの階層構造

MPEG では動画像のデータを符号化、復号化を複数の階層に分けておこなっている。
これは、動画像データの統計的な性質が時間的、および空間的に変化することに伴い、符号化効率を向上させるためにキメの細かいパラメータ制御を行うため、ビットエラー等に伴って正しい符号化データ情報が失われた場合に復号化された画像への影響を限定した範囲に抑えるため、ビットストリームの途中からでも画像の復号化が行えるというハンドリング性のため、などである。
構造は最上位に位置するシーケンス階層から最下位のブロック階層まで、合計６階層から構成されている。

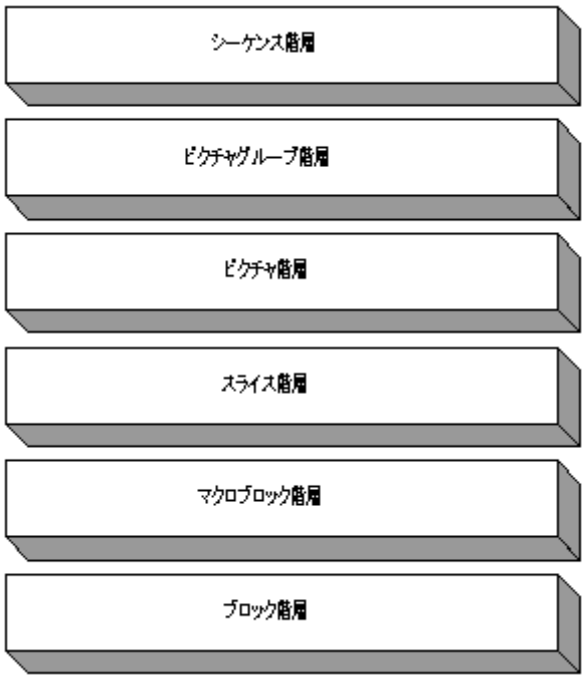


図 3-5 ビデオデータの階層構造

4 章　動き補償回路の設計

４・１　動き補償回路の全体構成

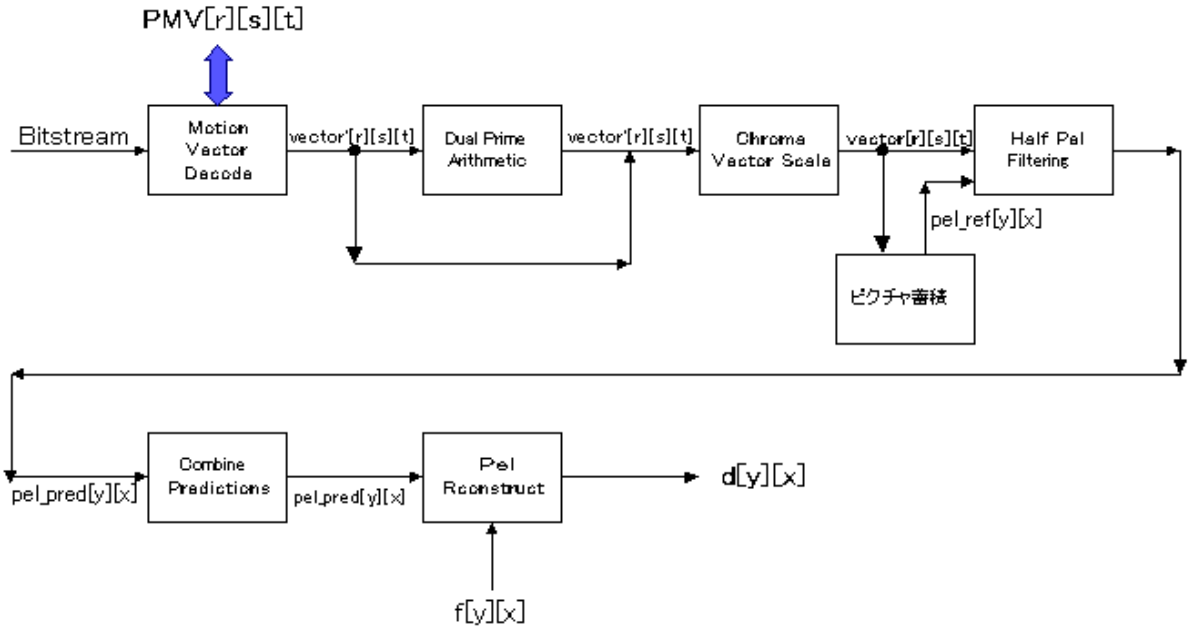


図4-1　動き補償回路の全体構成

図 4-1 はビットストリームから動きベクトルに関する可変長符号を読み込み、動き補償値 pel_pled[y][x]を動き補償前の画素値 f[y][x]に加える事により、空間領域での画素データ d[y][x]を得るまでの処理のブロック構成を示している。

入力されたビットストリームから、最初に演算ブロック（Motion Vector Decode） によって動きベクトル vector'[r][s][t]が復号化される。次に必要に応じて演算ブロック（Dual Prime Arithmetic）により、フィールド予測の効率を向上させる Dual-prime 予測が行われる。

さらに演算ブロック Chroma Vector Scale によって、色差成分用の動きベクトルのスケールが行われ、実際に動き補償に使用される動きベクトル vector[r][s][t]が得られる。演算ブロック HalfPelFiltering は、参照ピクチャの画素値 pel_ref[y][x]に基づいて半画素精度の動き予測値 pel_pred[y][x]を作成する。

演算ブロック CombinePrediction は、双方向予測が行われる場合のように、２つの予測値

の平均化を行う。最後に演算ブロック PelReconstruct において、すでに述べた逆量子化や逆ＤＣＴが完了した空間領域のデータ f[y][x]に動き予測値 Pel_Pred[y][x]を加えれば、動き補償された画素値 d[y][x]が得られる。

HalfPelFiltering は、輝度成分と色差成分に対する動きベクトルを入力すると、次にこれらの動きベクトルと参照ピクチャを用いて実際の動き予測値を作成する動き予測でも重要な部分である。ＭＰＥＧ２ではすべての動きベクトルは半画素精度で求められるため、画素の中間点の値を求める計算が必要になる。

今回の実験では、HalfPelFiltering の部分を設計することにする。

入力データは、16×16 画素単位の ChromaVectorScale より与えられる vector[r][s][t]（動く向き）と、ピクチャ蓄積からの pel_ref[y][x]（動く前の元の画像）の 2 つとなり、出力は半画素単位の動き予測値である pel_pred[y][x]を出力する。

動きベクトルは[r][s][t]の 3 次元配列を用いて表され、いずれも０または１の値をもつ。
画素は１６×１６のマクロブロックで扱う。よって[y][x]には０～１５の値が入る。
実験に用いる画像データは、動きベクトルが変化した場合にわかりやすい白黒の縞模様（図４－２参照）とした。０は白色を表し、２５５は黒を表す。

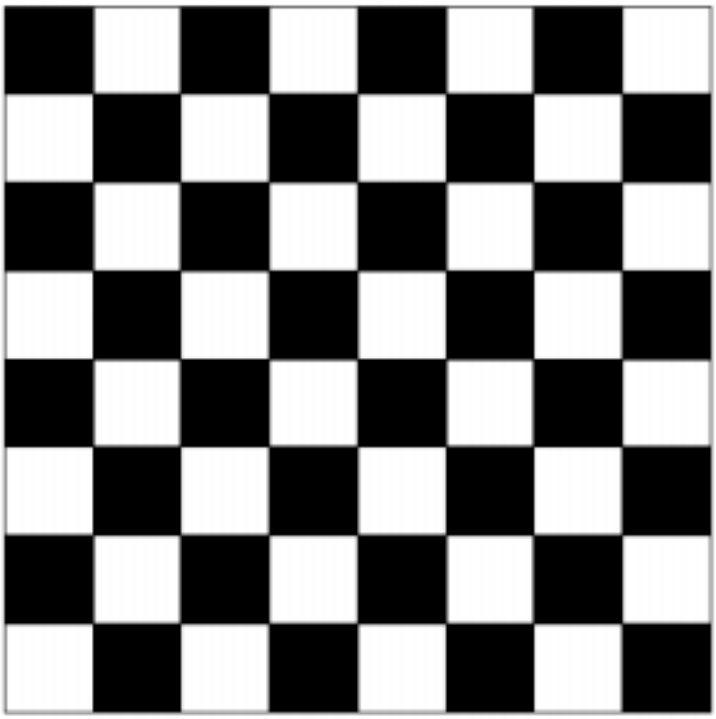


図4-2 入力画像のデータ

{255,255, 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 },
{255,255, 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 },
{ 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 ,255,255},
{ 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 ,255,255},
{255,255, 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 },
{255,255, 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 },
{ 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 ,255,255},
{ 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 ,255,255},
{255,255, 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 },
{255,255, 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 },
{ 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 ,255,255},
{ 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 ,255,255},
{255,255, 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 },
{255,255, 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 },
{ 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 ,255,255},
{ 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 ,255,255}

図 4-3 入力画像の数値データ

vector[r][s][t]のｒ、ｓ、ｔは０，１のどちらかの値が入り、[r]が[０]の場合ｘ軸での０から７までの前半部分を表し、[1]の場合は 8 から１５までの後半部分を表す。（図４－４参照）
[ｓ]は０，１のどちらかの値が入り、[０]の場合 y 軸での０から７までの前半部分を表し、[1]の場合は 8 から１５までの後半部分を表す。
[t]はベクトルの垂直成分か水平成分を表し、0 の時水平方向で、１の時垂直方向を表す。

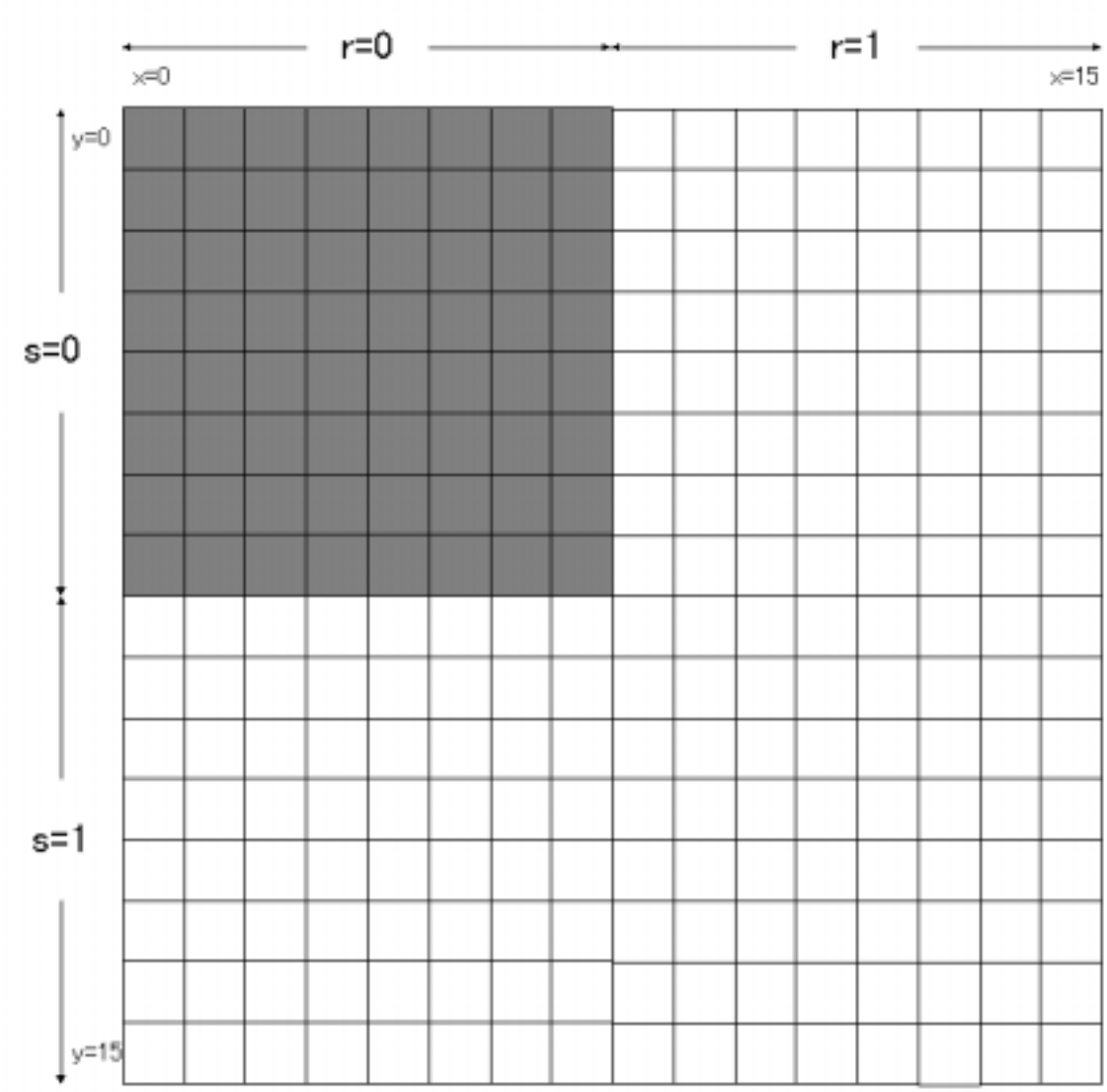


図４-４

ベクトルの組み合わせは

vector[0][0][0]=X

vector[0][0][1]=Y

の２通りで、灰色の 8×8 部分（図 4-4）になる。

X、Y には各ディスプレイ、または放送形式によって画素の違いがあるが、

	ヨコ（X）		タテ（Y）	
HDTV	１９２０	×	１１５２	（画素）
現行テレビ	７２０	×	５７６	（画素）

のまでの上限値の２倍の値まで入れることができる。（半画素精度のため）

この 16×16 のマクロブロックがいくつも敷き詰められて１枚の画像になる。

４．２　プログラム

ここでは今回使うプログラミングの重要な所についての説明をおこなう。

プログラム１

```
for (t=0;t<2;t++)
 {
        int_vec[t] = vector[r][s][t] / 2 ;
        if (vector[r][s][t]-(2*int_vec[t])!=0)      half_flag[t]=1 ;
        else                                        half_flag[t]=0 ;
 }
```

上記は動きベクトル値を 2 で割り、それを計算した値が０でない場合（ベクトル要素が奇数の時）に半画素の動きがあるということを判別するプログラムである。０の場合はベクトルが偶数なので全画素となる。

プログラム２

```
if ((half_flag[0]==0)&&(half_flag[1]==0))
    for (y=8;y<16;y++)
        {
          for (x=0;x<8;x++)
              {
                pel_pred[y][x]=pel_ref[y+int_vec[1]][x+int_vec[0]] ;
               }
        }
if ((half_flag[0]==0)&&(half_flag[1]==1))
    for (y=8;y<16;y++)
        {
          for (x=0;x<8;x++)
             {
                  pel_pred[y][x]=(pel_ref[y+int_vec[1]][x+int_vec[0]]+
                                  pel_ref[y+int_vec[1]+1][x+int_vec[0]])/2 ;
              }
        }
```

```
if ((half_flag[0]==1)&&(half_flag[1]==0))
    for (y=8;y<16;y++)
        {
          for (x=0;x<8;x++)
            {
              pel_pred[y][x]=(pel_ref[y+int_vec[1]][x+int_vec[0]]+
                              pel_ref[y+int_vec[1]][x+int_vec[0]+1])/2 ;
            }
        }
if ((half_flag[0]==1)&&(half_flag[1]==1))
    for (y=8;y<16;y++)
      {
       for (x=0;x<8;x++)
          {
            pel_pred[y][x]=(pel_ref[y+int_vec[1]][x+int_vec[0]]+
                            pel_ref[y+int_vec[1]][x+int_vec[0]+1]+
                            pel_ref[y+int_vec[1]+1][x+int_vec[0]]+
                            pel_ref[y+int_vec[1]+1][x+int_vec[0]+1])/4       ;
          }
      }
```

上のプログラムは半画素フラグが０と１の各組み合わせにより４つの場合に分けられて計算する部分である。この記述した部分は、全体（16×16）の左上８×８の領域のプログラムなので、ｘとｙの値を変えたプログラムをこの下にあと 3 回繰り返すようになる。

残りのプログラムについては、計算して出てきたデータを１６×１６に並べてデータファイルに表示させるプログラムでプログラミングは終わる。

４・３　シミュレーション

プログラミングが終わったらコンパイルし、エラーがなければビルド、実行をする。
まず、HalfPelFiltering では動きベクトルとピクチャ蓄積のデータを入れてやる必要がある。
ピクチャ蓄積のデータはプログラム内にて定義してあるので、動きベクトルの値を適当に変化させて、どのように入力した白黒のデータが動いたかをみてみる。

今回の実験ではｒ＝０、ｓ＝０でｔのみ１と０の値を変化させた２通りのデータだけでよい。それ以外のベクトルの組み合わせでも２シフトまでなら対応できるよう、付録のソースでは 16×16 以外のシフトしてくるデータも含めた 18×18 の入力画像とした。

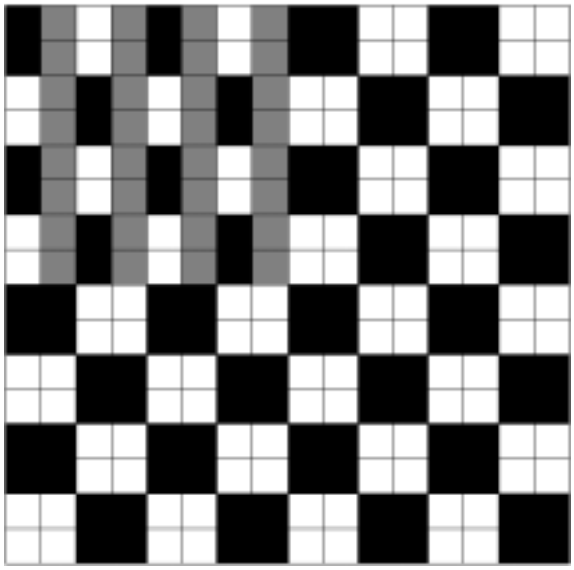
５章 結果と考察

vector[0][0][0]=1

vector[0][0][1]=0

255,127, 0,127,255,127, 0,127,255,255, 0, 0,255,255, 0, 0
255,127, 0,127,255,127, 0,127,255,255, 0, 0,255,255, 0, 0
0,127,255,127, 0,127,255,127, 0, 0,255,255, 0, 0,255,255
0,127,255,127, 0,127,255,127, 0, 0,255,255, 0, 0,255,255
255,127, 0,127,255,127, 0,127,255,255, 0, 0,255,255, 0, 0
255,127, 0,127,255,127, 0,127,255,255, 0, 0,255,255, 0, 0
0,127,255,127, 0,127,255,127, 0, 0,255,255, 0, 0,255,255
0,127,255,127, 0,127,255,127, 0, 0,255,255, 0, 0,255,255
255,255, 0, 0,255,255, 0, 0,255,255, 0, 0,255,255, 0, 0
255,255, 0, 0,255,255, 0, 0,255,255, 0, 0,255,255, 0, 0
0, 0,255,255, 0, 0,255,255, 0, 0,255,255, 0, 0,255,255
0, 0,255,255, 0, 0,255,255, 0, 0,255,255, 0, 0,255,255
255,255, 0, 0,255,255, 0, 0,255,255, 0, 0,255,255, 0, 0
255,255, 0, 0,255,255, 0, 0,255,255, 0, 0,255,255, 0, 0
0, 0,255,255, 0, 0,255,255, 0, 0,255,255, 0, 0,255,255
0, 0,255,255, 0, 0,255,255, 0, 0,255,255, 0, 0,255,255

出力データ

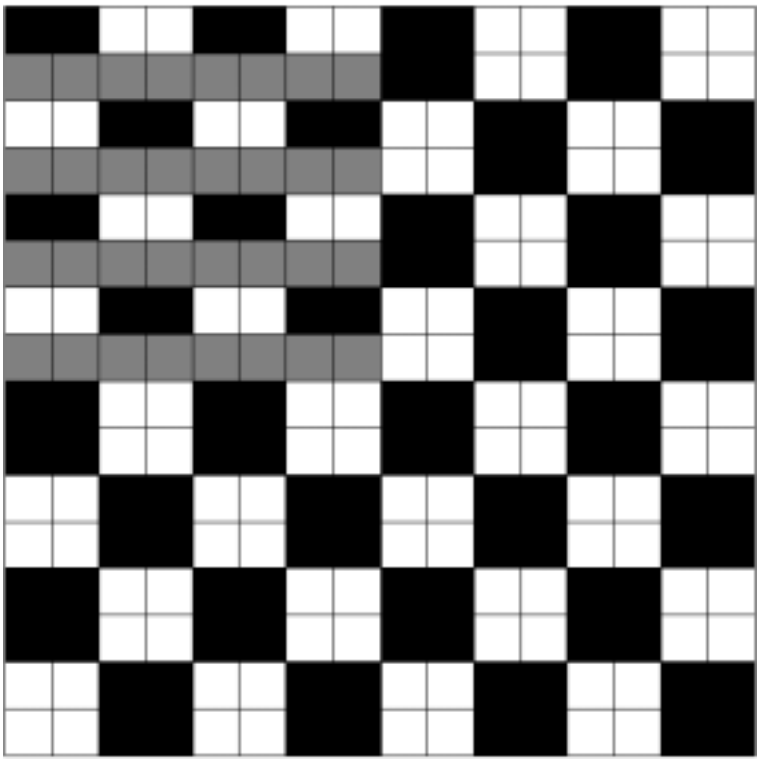


出力データより画像に変換

vector[0][0][0]=0

vector[0][0][1]=1

255,255, 0, 0,255,255, 0, 0,255,255, 0, 0,255,255, 0, 0
127,127,127,127,127,127,127,127,255,255, 0, 0,255,255, 0, 0
0, 0,255,255, 0, 0,255,255, 0, 0,255,255, 0, 0,255,255
127,127,127,127,127,127,127,127, 0, 0,255,255, 0, 0,255,255
255,255, 0, 0,255,255, 0, 0,255,255, 0, 0,255,255, 0, 0
127,127,127,127,127,127,127,127,255,255, 0, 0,255,255, 0, 0
0, 0,255,255, 0, 0,255,255, 0, 0,255,255, 0, 0,255,255
127,127,127,127,127,127,127,127, 0, 0,255,255, 0, 0,255,255
255,255, 0, 0,255,255, 0, 0,255,255, 0, 0,255,255, 0, 0
255,255, 0, 0,255,255, 0, 0,255,255, 0, 0,255,255, 0, 0
0, 0,255,255, 0, 0,255,255, 0, 0,255,255, 0, 0,255,255
0, 0,255,255, 0, 0,255,255, 0, 0,255,255, 0, 0,255,255
255,255, 0, 0,255,255, 0, 0,255,255, 0, 0,255,255, 0, 0
255,255, 0, 0,255,255, 0, 0,255,255, 0, 0,255,255, 0, 0
0, 0,255,255, 0, 0,255,255, 0, 0,255,255, 0, 0,255,255
0, 0,255,255, 0, 0,255,255, 0, 0,255,255, 0, 0,255,255



vector[0][0][0]=2

vector[0][0][1]=0

255, 0, 0,255,255, 0, 0,255,255,255, 0, 0,255,255, 0, 0

255, 0, 0,255,255, 0, 0,255,255,255, 0, 0,255,255, 0, 0

0,255,255, 0, 0,255,255, 0, 0, 0,255,255, 0, 0,255,255

0,255,255, 0, 0,255,255, 0, 0, 0,255,255, 0, 0,255,255

255, 0, 0,255,255, 0, 0,255,255,255, 0, 0,255,255, 0, 0

255, 0, 0,255,255, 0, 0,255,255,255, 0, 0,255,255, 0, 0

0,255,255, 0, 0,255,255, 0, 0, 0,255,255, 0, 0,255,255

0,255,255, 0, 0,255,255, 0, 0, 0,255,255, 0, 0,255,255

255,255, 0, 0,255,255, 0, 0,255,255, 0, 0,255,255, 0, 0

255,255, 0, 0,255,255, 0, 0,255,255, 0, 0,255,255, 0, 0

0, 0,255,255, 0, 0,255,255, 0, 0,255,255, 0, 0,255,255

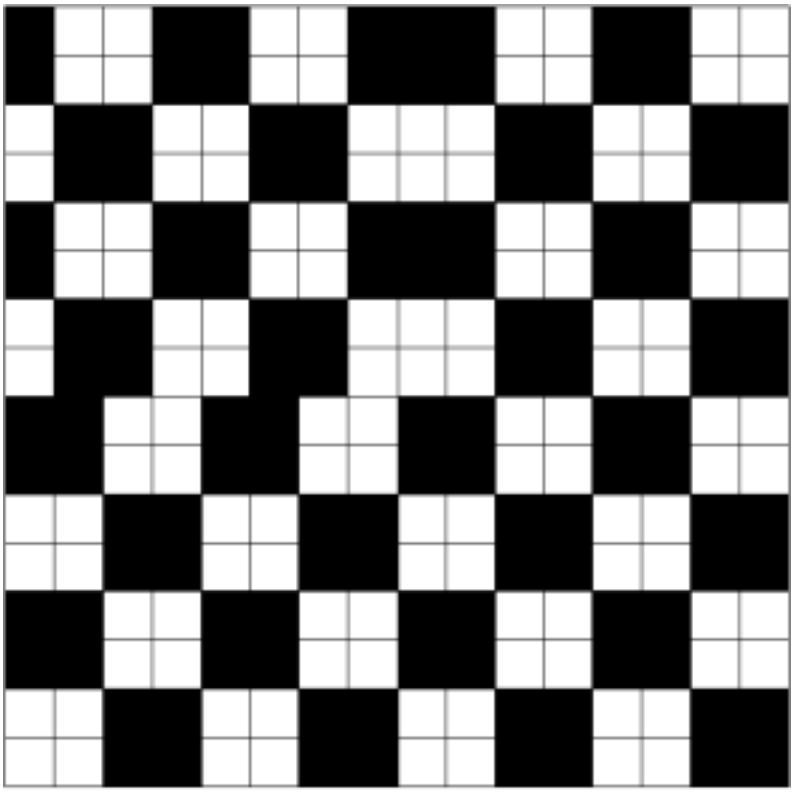
0, 0,255,255, 0, 0,255,255, 0, 0,255,255, 0, 0,255,255

255,255, 0, 0,255,255, 0, 0,255,255, 0, 0,255,255, 0, 0

255,255, 0, 0,255,255, 0, 0,255,255, 0, 0,255,255, 0, 0

0, 0,255,255, 0, 0,255,255, 0, 0,255,255, 0, 0,255,255

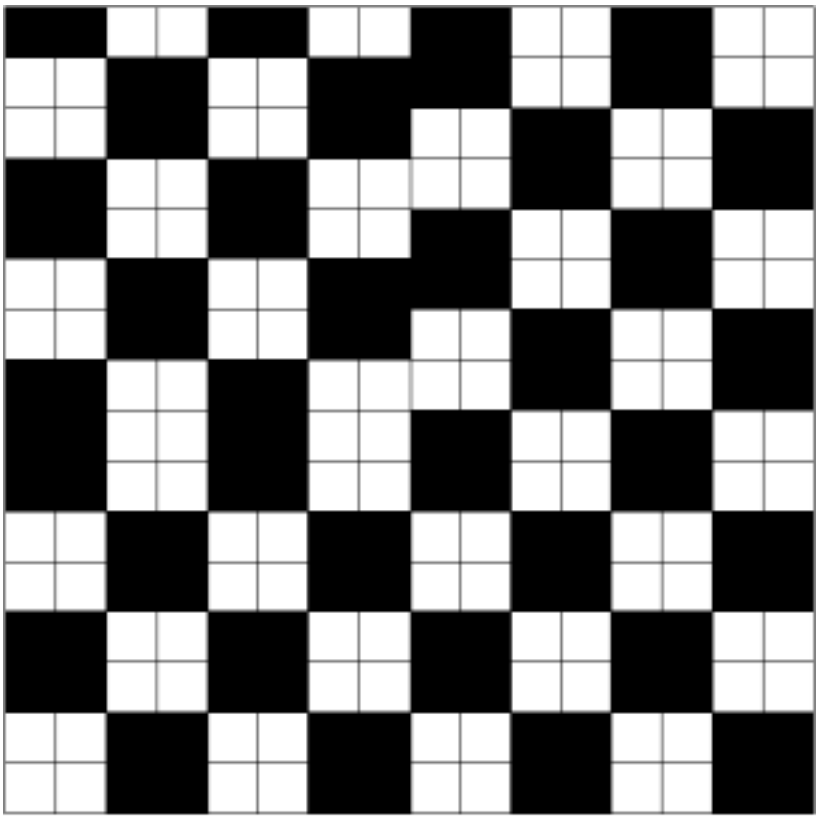
0, 0,255,255, 0, 0,255,255, 0, 0,255,255, 0, 0,255,255



vector[0][0][0]=0

vector[0][0][1]=2

255,255, 0, 0,255,255, 0, 0,255,255, 0, 0,255,255, 0, 0
0, 0,255,255, 0, 0,255,255,255,255, 0, 0,255,255, 0, 0
0, 0,255,255, 0, 0,255,255, 0, 0,255,255, 0, 0,255,255
255,255, 0, 0,255,255, 0, 0, 0, 0,255,255, 0, 0,255,255
255,255, 0, 0,255,255, 0, 0,255,255, 0, 0,255,255, 0, 0
0, 0,255,255, 0, 0,255,255,255,255, 0, 0,255,255, 0, 0
0, 0,255,255, 0, 0,255,255, 0, 0,255,255, 0, 0,255,255
255,255, 0, 0,255,255, 0, 0, 0, 0,255,255, 0, 0,255,255
255,255, 0, 0,255,255, 0, 0,255,255, 0, 0,255,255, 0, 0
255,255, 0, 0,255,255, 0, 0,255,255, 0, 0,255,255, 0, 0
0, 0,255,255, 0, 0,255,255, 0, 0,255,255, 0, 0,255,255
0, 0,255,255, 0, 0,255,255, 0, 0,255,255, 0, 0,255,255
255,255, 0, 0,255,255, 0, 0,255,255, 0, 0,255,255, 0, 0
255,255, 0, 0,255,255, 0, 0,255,255, 0, 0,255,255, 0, 0
0, 0,255,255, 0, 0,255,255, 0, 0,255,255, 0, 0,255,255
0, 0,255,255, 0, 0,255,255, 0, 0,255,255, 0, 0,255,255



vector[0][0][0]=1

vector[0][0][1]=1

255,127, 0,127,255,127, 0,127,255,255, 0, 0,255,255, 0, 0

127,127,127,127,127,127,127,127,255,255, 0, 0,255,255, 0, 0

0,127,255,127, 0,127,255,127, 0, 0,255,255, 0, 0,255,255

127,127,127,127,127,127,127,127, 0, 0,255,255, 0, 0,255,255

255,127, 0,127,255,127, 0,127,255,255, 0, 0,255,255, 0, 0

127,127,127,127,127,127,127,127,255,255, 0, 0,255,255, 0, 0

0,127,255,127, 0,127,255,127, 0, 0,255,255, 0, 0,255,255

127,127,127,127,127,127,127,127, 0, 0,255,255, 0, 0,255,255

255,255, 0, 0,255,255, 0, 0,255,255, 0, 0,255,255, 0, 0

255,255, 0, 0,255,255, 0, 0,255,255, 0, 0,255,255, 0, 0

0, 0,255,255, 0, 0,255,255, 0, 0,255,255, 0, 0,255,255

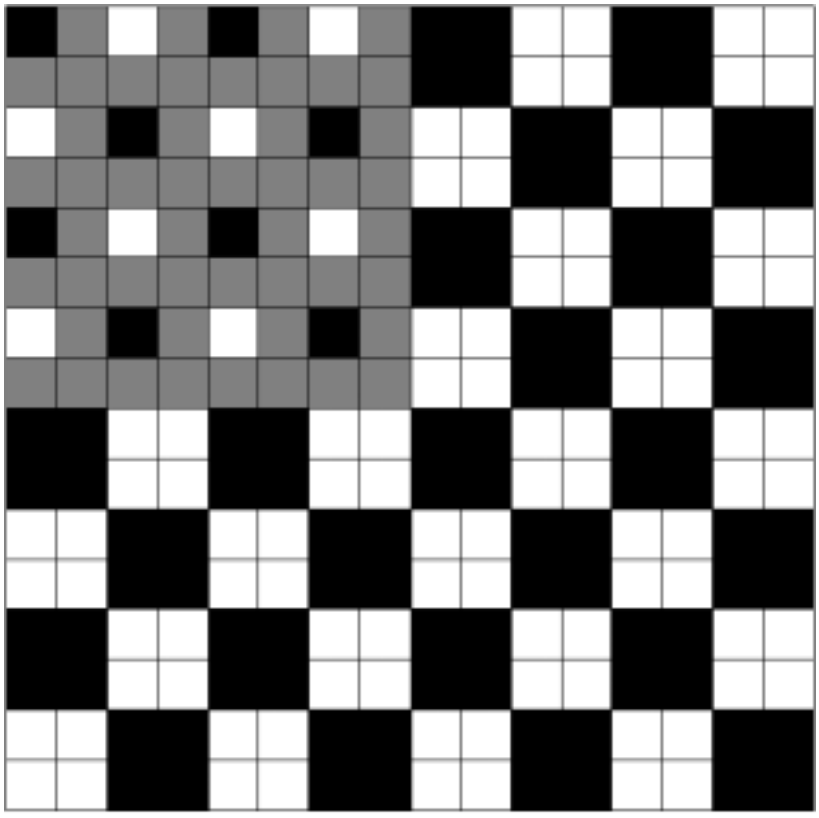
0, 0,255,255, 0, 0,255,255, 0, 0,255,255, 0, 0,255,255

255,255, 0, 0,255,255, 0, 0,255,255, 0, 0,255,255, 0, 0

255,255, 0, 0,255,255, 0, 0,255,255, 0, 0,255,255, 0, 0

0, 0,255,255, 0, 0,255,255, 0, 0,255,255, 0, 0,255,255

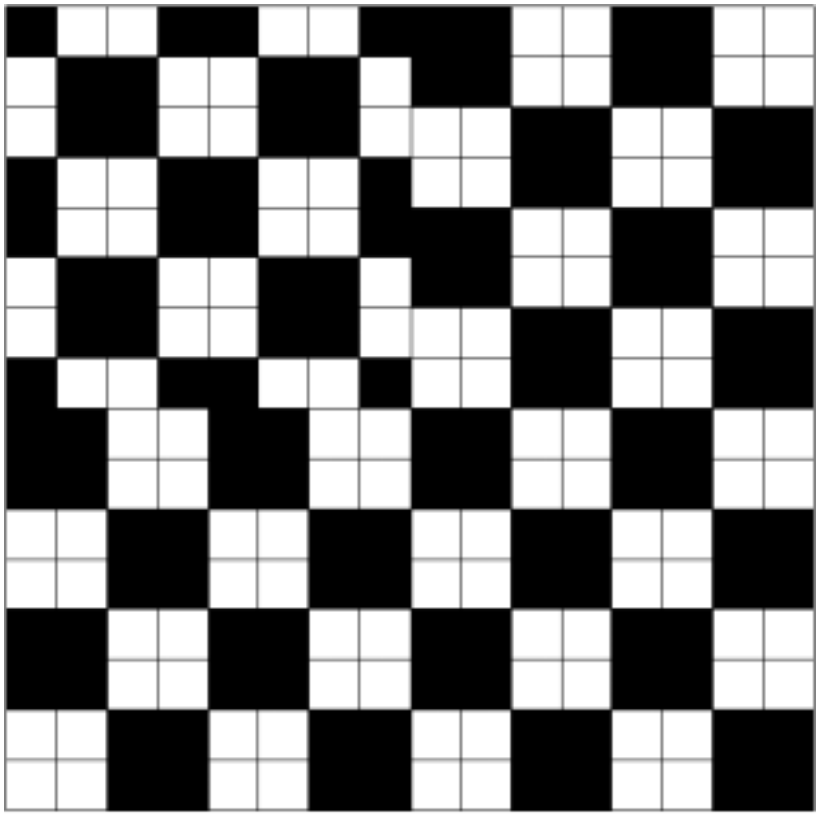
0, 0,255,255, 0, 0,255,255, 0, 0,255,255, 0, 0,255,255



vector[0][0][0]=2

vector[0][0][1]=2

255, 0, 0,255,255, 0, 0,255,255,255, 0, 0,255,255, 0, 0
0,255,255, 0, 0,255,255, 0,255,255, 0, 0,255,255, 0, 0
0,255,255, 0, 0,255,255, 0, 0, 0,255,255, 0, 0,255,255
255, 0, 0,255,255, 0, 0,255, 0, 0,255,255, 0, 0,255,255
255, 0, 0,255,255, 0, 0,255,255,255, 0, 0,255,255, 0, 0
0,255,255, 0, 0,255,255, 0,255,255, 0, 0,255,255, 0, 0
0,255,255, 0, 0,255,255, 0, 0, 0,255,255, 0, 0,255,255
255, 0, 0,255,255, 0, 0,255, 0, 0,255,255, 0, 0,255,255
255,255, 0, 0,255,255, 0, 0,255,255, 0, 0,255,255, 0, 0
255,255, 0, 0,255,255, 0, 0,255,255, 0, 0,255,255, 0, 0
0, 0,255,255, 0, 0,255,255, 0, 0,255,255, 0, 0,255,255
0, 0,255,255, 0, 0,255,255, 0, 0,255,255, 0, 0,255,255
255,255, 0, 0,255,255, 0, 0,255,255, 0, 0,255,255, 0, 0
255,255, 0, 0,255,255, 0, 0,255,255, 0, 0,255,255, 0, 0
0, 0,255,255, 0, 0,255,255, 0, 0,255,255, 0, 0,255,255
0, 0,255,255, 0, 0,255,255, 0, 0,255,255, 0, 0,255,255



実験の結果のデータを見てみると、入力画像がベクトル値を入力すると、その分だけ画像がその方向に半画素レベルでシフトしていくのがわかる。

これにより、このデータは正しく成功していると言える。

６章 まとめ

今回の動き補償回路の設計は目標であったＶＨＤＬで設計するのが最終目標であった。そこまで到達できなかったのは残念であるが、最初は動画像の事もＣ言語の事もほとんど無知であった私がこうして説明し、発表も出来るようになったのは自分自身が少なからずレベルアップした証拠だと思う。プログラミングをしたのもよい経験になったと思う。

謝辞

卒業研究をするにあたり、親切丁寧に御指導していただいた橘助教授、そしてあたたかく見守っていただいた原教授、矢野教授に熱くお礼申し上げます。

参考文献

[1]小野・鈴木　わかりやすいＪＰＥＧ・ＭＰＥＧ２の技術　オーム社　（2001）
[2]越智・黒田　ＪＰＥＧ&ＭＰＥＧ図解でわかる画像圧縮技術　日本実業出版　（1999）
[3]杉江・堀　Ｃプログラミングの学習　培風館　（2000）
[4]塚越　はじめての C＋＋　－演習と解説－　技術評論社　（2000）

付録　Cのソース

```
#include <stdio.h>
#include <stdlib.h>

main()
{
    FILE *str_ptr;
    str_ptr = fopen("infile.txt","w");

    int vector[2][2][2] ;
    int     int_vec[2];
    int     pel_ref[18][18]    =
    {
        {255,255, 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 ,255,255},
        {255,255, 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 ,255,255},
        { 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 },
        { 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 },
        {255,255, 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 ,255,255},
        {255,255, 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 ,255,255},
        { 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 },
        { 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 },
        {255,255, 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 ,255,255},
        {255,255, 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 ,255,255},
        { 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 },
        { 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 },
        {255,255, 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 ,255,255},
        {255,255, 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 ,255,255},
        { 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 },
        { 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 },
        {255,255, 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 ,255,255},
        {255,255, 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 ,255,255, 0 , 0 ,255,255},
    };
```

```
int     half_flag[2];
int     pel_pred[16][16];
int     x;
int     y;
int     r;
int     s;
int     t;

vector[0][0][0]=1;
vector[0][0][1]=0;
vector[0][1][0]=0;
vector[0][1][1]=0;
vector[1][0][0]=0;
vector[1][0][1]=0;
vector[1][1][0]=0;
vector[1][1][1]=0;



/* 1 */
  r=0;
  s=0;

for (t=0;t<2;t++)
    {


        int_vec[t] = vector[r][s][t] / 2 ;                    /*動きベクトル値を
２で割る*/
                if  (vector[r][s][t]-(2*int_vec[t])!=0)     half_flag[t]=1 ;   /*半画素フラ
グを１にセット*/
                else                                        half_flag[t]=0 ;   /*半画
素フラグを０にセット*/


       }

   if ((half_flag[0]==0)&&(half_flag[1]==0))
```

```
            for (y=0;y<8;y++)
             {
                        for (x=0;x<8;x++)
                         {
pel_pred[y][x]=pel_ref[y+int_vec[1]][x+int_vec[0]] ;
             }
          }

if ((half_flag[0]==0)&&(half_flag[1]==1))
for (y=0;y<8;y++)
 {
                        for (x=0;x<8;x++)
                         {
      pel_pred[y][x]=(pel_ref[y+int_vec[1]][x+int_vec[0]]+
                                  pel_ref[y+int_vec[1]+1][x+int_vec[0]])/2 ;
                          }
   }

if ((half_flag[0]==1)&&(half_flag[1]==0))
for (y=0;y<8;y++)
 {
                        for (x=0;x<8;x++)
                         {
      pel_pred[y][x]=(pel_ref[y+int_vec[1]][x+int_vec[0]]+
                                  pel_ref[y+int_vec[1]][x+int_vec[0]+1])/2 ;
              }
   }

if ((half_flag[0]==1)&&(half_flag[1]==1))
for (y=0;y<8;y++)
 {
                        for (x=0;x<8;x++)
                         {
      pel_pred[y][x]=(pel_ref[y+int_vec[1]][x+int_vec[0]]+
                                  pel_ref[y+int_vec[1]][x+int_vec[0]+1]+
```

```
        pel_ref[y+int_vec[1]+1][x+int_vec[0]]+

        pel_ref[y+int_vec[1]+1][x+int_vec[0]+1])/4  ;
                                }
          }

         /* 2 */
           r=1;
       s=0;

   for (t=0;t<2;t++)
    {

                int_vec[t] = vector[r][s][t] / 2 ;                                  /*動きベクトル
値を２で割る*/
                    if  (vector[r][s][t]-(2*int_vec[t])!=0)      half_flag[t]=1 ;   /*半画素フラ
グを１にセット*/
                    else                                                  half_flag[t]=0 ;   /*半画
素フラグを０にセット*/


     }

    if ((half_flag[0]==0)&&(half_flag[1]==0))
                for (y=0;y<8;y++)
                 {
                          for (x=8;x<16;x++)
                           {
                pel_pred[y][x]=pel_ref[y+int_vec[1]][x+int_vec[0]] ;
                        }
              }

        if ((half_flag[0]==0)&&(half_flag[1]==1))
        for (y=0;y<8;y++)
         {
                          for (x=8;x<16;x++)
                           {
```

```
                pel_pred[y][x]=(pel_ref[y+int_vec[1]][x+int_vec[0]]+
                                        pel_ref[y+int_vec[1]+1][x+int_vec[0]])/2 ;
                                }
            }

          if ((half_flag[0]==1)&&(half_flag[1]==0))
          for (y=0;y<8;y++)
           {
                                for (x=8;x<16;x++)
                                 {
                        pel_pred[y][x]=(pel_ref[y+int_vec[1]][x+int_vec[0]]+
                                                pel_ref[y+int_vec[1]][x+int_vec[0]+1])/2 ;
                                }
            }

          if ((half_flag[0]==1)&&(half_flag[1]==1))
          for (y=0;y<8;y++)
           {
                                for (x=8;x<16;x++)
                                 {
                        pel_pred[y][x]=(pel_ref[y+int_vec[1]][x+int_vec[0]]+
                                                pel_ref[y+int_vec[1]][x+int_vec[0]+1]+

pel_ref[y+int_vec[1]+1][x+int_vec[0]]+

pel_ref[y+int_vec[1]+1][x+int_vec[0]+1])/4  ;
                                }
            }

            /* 3 */
            r=0;
        s=1;

    for (t=0;t<2;t++)
      {
```

```
        int_vec[t] = vector[r][s][t] / 2 ;                    /*動きベクトル値を
２で割る*/
                if  (vector[r][s][t]-(2*int_vec[t])!=0)     half_flag[t]=1 ;  /*半画素フラ
グを１にセット*/
                else                                          half_flag[t]=0 ;  /*半画
素フラグを０にセット*/

    }

  if ((half_flag[0]==0)&&(half_flag[1]==0))
              for (y=8;y<16;y++)
               {
                        for (x=0;x<8;x++)
                         {
            pel_pred[y][x]=pel_ref[y+int_vec[1]][x+int_vec[0]] ;
                        }
              }

     if ((half_flag[0]==0)&&(half_flag[1]==1))
     for (y=8;y<16;y++)
      {
                        for (x=0;x<8;x++)
                         {
               pel_pred[y][x]=(pel_ref[y+int_vec[1]][x+int_vec[0]]+
                                    pel_ref[y+int_vec[1]+1][x+int_vec[0]])/2 ;
                          }
       }

     if ((half_flag[0]==1)&&(half_flag[1]==0))
     for (y=8;y<16;y++)
      {
                        for (x=0;x<8;x++)
                         {
             pel_pred[y][x]=(pel_ref[y+int_vec[1]][x+int_vec[0]]+
                                    pel_ref[y+int_vec[1]][x+int_vec[0]+1])/2 ;
                        }
```

```
            }

          if ((half_flag[0]==1)&&(half_flag[1]==1))
          for (y=8;y<16;y++)
           {
                          for (x=0;x<8;x++)
                           {
                    pel_pred[y][x]=(pel_ref[y+int_vec[1]][x+int_vec[0]]+
                                          pel_ref[y+int_vec[1]][x+int_vec[0]+1]+

pel_ref[y+int_vec[1]+1][x+int_vec[0]]+

pel_ref[y+int_vec[1]+1][x+int_vec[0]+1])/4  ;
                                  }
            }

           /* 4 */
             r=1;
       s=1;

   for (t=0;t<2;t++)
     {

          int_vec[t] = vector[r][s][t] / 2 ;                    /*動きベクトル値を
２で割る*/
                  if  (vector[r][s][t]-(2*int_vec[t])!=0)      half_flag[t]=1 ;   /*半画素フラ
グを１にセット*/
                  else                                          half_flag[t]=0 ;   /*半画
素フラグを０にセット*/

       }

    if ((half_flag[0]==0)&&(half_flag[1]==0))
                  for (y=8;y<16;y++)
                  {
                          for (x=8;x<16;x++)
```

```
                    {
        pel_pred[y][x]=pel_ref[y+int_vec[1]][x+int_vec[0]] ;
                  }
          }

        if ((half_flag[0]==0)&&(half_flag[1]==1))
        for (y=8;y<16;y++)
         {
                    for (x=8;x<16;x++)
                     {
                pel_pred[y][x]=(pel_ref[y+int_vec[1]][x+int_vec[0]]+
                                    pel_ref[y+int_vec[1]+1][x+int_vec[0]])/2 ;
                      }
          }

        if ((half_flag[0]==1)&&(half_flag[1]==0))
        for (y=8;y<16;y++)
         {
                    for (x=8;x<16;x++)
                     {
               pel_pred[y][x]=(pel_ref[y+int_vec[1]][x+int_vec[0]]+
                                   pel_ref[y+int_vec[1]][x+int_vec[0]+1])/2 ;
                  }
          }

        if ((half_flag[0]==1)&&(half_flag[1]==1))
        for (y=8;y<16;y++)
          {
                    for (x=8;x<16;x++)
                     {
                pel_pred[y][x]=(pel_ref[y+int_vec[1]][x+int_vec[0]]+
                                    pel_ref[y+int_vec[1]][x+int_vec[0]+1]+

pel_ref[y+int_vec[1]+1][x+int_vec[0]]+

pel_ref[y+int_vec[1]+1][x+int_vec[0]+1])/4  ;
```

```
                    }
                }

fprintf(str_ptr,"%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d
,%3d¥n",
                    pel_pred[0][0],pel_pred[0][1],pel_pred[0][2],pel_pred[0][3],
            pel_pred[0][4],pel_pred[0][5],pel_pred[0][6],pel_pred[0][7],
            pel_pred[0][8],pel_pred[0][9],pel_pred[0][10],pel_pred[0][11],
            pel_pred[0][12],pel_pred[0][13],pel_pred[0][14],pel_pred[0][15]);

fprintf(str_ptr,"%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d
,%3d¥n",
                    pel_pred[1][0],pel_pred[1][1],pel_pred[1][2],pel_pred[1][3],
            pel_pred[1][4],pel_pred[1][5],pel_pred[1][6],pel_pred[1][7],
            pel_pred[1][8],pel_pred[1][9],pel_pred[1][10],pel_pred[1][11],
            pel_pred[1][12],pel_pred[1][13],pel_pred[1][14],pel_pred[1][15]);

fprintf(str_ptr,"%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d
,%3d¥n",
                    pel_pred[2][0],pel_pred[2][1],pel_pred[2][2],pel_pred[2][3],
            pel_pred[2][4],pel_pred[2][5],pel_pred[2][6],pel_pred[2][7],
            pel_pred[2][8],pel_pred[2][9],pel_pred[2][10],pel_pred[2][11],
            pel_pred[2][12],pel_pred[2][13],pel_pred[2][14],pel_pred[2][15]);

fprintf(str_ptr,"%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d
,%3d¥n",
                    pel_pred[3][0],pel_pred[3][1],pel_pred[3][2],pel_pred[3][3],
            pel_pred[3][4],pel_pred[3][5],pel_pred[3][6],pel_pred[3][7],
            pel_pred[3][8],pel_pred[3][9],pel_pred[3][10],pel_pred[3][11],
            pel_pred[3][12],pel_pred[3][13],pel_pred[3][14],pel_pred[3][15]);

fprintf(str_ptr,"%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d
,%3d¥n",
                    pel_pred[4][0],pel_pred[4][1],pel_pred[4][2],pel_pred[4][3],
            pel_pred[4][4],pel_pred[4][5],pel_pred[4][6],pel_pred[4][7],
            pel_pred[4][8],pel_pred[4][9],pel_pred[4][10],pel_pred[4][11],
```

```
        pel_pred[4][12],pel_pred[4][13],pel_pred[4][14],pel_pred[4][15]);

fprintf(str_ptr,"%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d
,%3d¥n",
                pel_pred[5][0],pel_pred[5][1],pel_pred[5][2],pel_pred[5][3],
        pel_pred[5][4],pel_pred[5][5],pel_pred[5][6],pel_pred[5][7],
        pel_pred[5][8],pel_pred[5][9],pel_pred[5][10],pel_pred[5][11],
        pel_pred[5][12],pel_pred[5][13],pel_pred[5][14],pel_pred[5][15]);

fprintf(str_ptr,"%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d
,%3d¥n",
                pel_pred[6][0],pel_pred[6][1],pel_pred[6][2],pel_pred[6][3],
        pel_pred[6][4],pel_pred[6][5],pel_pred[6][6],pel_pred[6][7],
        pel_pred[6][8],pel_pred[6][9],pel_pred[6][10],pel_pred[6][11],
        pel_pred[6][12],pel_pred[6][13],pel_pred[6][14],pel_pred[6][15]);

fprintf(str_ptr,"%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d
,%3d¥n",
                pel_pred[7][0],pel_pred[7][1],pel_pred[7][2],pel_pred[7][3],
        pel_pred[7][4],pel_pred[7][5],pel_pred[7][6],pel_pred[7][7],
        pel_pred[7][8],pel_pred[7][9],pel_pred[7][10],pel_pred[7][11],
        pel_pred[7][12],pel_pred[7][13],pel_pred[7][14],pel_pred[7][15]);


fprintf(str_ptr,"%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d
,%3d¥n",
                pel_pred[8][0],pel_pred[8][1],pel_pred[8][2],pel_pred[8][3],
        pel_pred[8][4],pel_pred[8][5],pel_pred[8][6],pel_pred[8][7],
        pel_pred[8][8],pel_pred[8][9],pel_pred[8][10],pel_pred[8][11],
        pel_pred[8][12],pel_pred[8][13],pel_pred[8][14],pel_pred[8][15]);

fprintf(str_ptr,"%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d
,%3d¥n",
                pel_pred[9][0],pel_pred[9][1],pel_pred[9][2],pel_pred[9][3],
        pel_pred[9][4],pel_pred[9][5],pel_pred[9][6],pel_pred[9][7],
        pel_pred[9][8],pel_pred[9][9],pel_pred[9][10],pel_pred[9][11],
```

```
        pel_pred[9][12],pel_pred[9][13],pel_pred[9][14],pel_pred[9][15]);

fprintf(str_ptr,"%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d
,%3d¥n",
                pel_pred[10][0],pel_pred[10][1],pel_pred[10][2],pel_pred[10][3],
        pel_pred[10][4],pel_pred[10][5],pel_pred[10][6],pel_pred[10][7],
        pel_pred[10][8],pel_pred[10][9],pel_pred[10][10],pel_pred[10][11],
        pel_pred[10][12],pel_pred[10][13],pel_pred[10][14],pel_pred[10][15]);

fprintf(str_ptr,"%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d
,%3d¥n",
                pel_pred[11][0],pel_pred[11][1],pel_pred[11][2],pel_pred[11][3],
        pel_pred[11][4],pel_pred[11][5],pel_pred[11][6],pel_pred[11][7],
        pel_pred[11][8],pel_pred[11][9],pel_pred[11][10],pel_pred[11][11],
        pel_pred[11][12],pel_pred[11][13],pel_pred[11][14],pel_pred[11][15]);

fprintf(str_ptr,"%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d
,%3d¥n",
                pel_pred[12][0],pel_pred[12][1],pel_pred[12][2],pel_pred[12][3],
        pel_pred[12][4],pel_pred[12][5],pel_pred[12][6],pel_pred[12][7],
        pel_pred[12][8],pel_pred[12][9],pel_pred[12][10],pel_pred[12][11],
        pel_pred[12][12],pel_pred[12][13],pel_pred[12][14],pel_pred[12][15]);

fprintf(str_ptr,"%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d
,%3d¥n",
                pel_pred[13][0],pel_pred[13][1],pel_pred[13][2],pel_pred[13][3],
        pel_pred[13][4],pel_pred[13][5],pel_pred[13][6],pel_pred[13][7],
        pel_pred[13][8],pel_pred[13][9],pel_pred[13][10],pel_pred[13][11],
        pel_pred[13][12],pel_pred[13][13],pel_pred[13][14],pel_pred[13][15]);

fprintf(str_ptr,"%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d
,%3d¥n",
                pel_pred[14][0],pel_pred[14][1],pel_pred[14][2],pel_pred[14][3],
        pel_pred[14][4],pel_pred[14][5],pel_pred[14][6],pel_pred[14][7],
        pel_pred[14][8],pel_pred[14][9],pel_pred[14][10],pel_pred[14][11],
        pel_pred[14][12],pel_pred[14][13],pel_pred[14][14],pel_pred[14][15]);
```

```
fprintf(str_ptr,"%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d,%3d
,%3d¥n",
                        pel_pred[15][0],pel_pred[15][1],pel_pred[15][2],pel_pred[15][3],
            pel_pred[15][4],pel_pred[15][5],pel_pred[15][6],pel_pred[15][7],
            pel_pred[15][8],pel_pred[15][9],pel_pred[15][10],pel_pred[15][11],
            pel_pred[15][12],pel_pred[15][13],pel_pred[15][14],pel_pred[15][15]);

 fclose(str_ptr);
 return(1);
}
```