修士論文

題目

中心部以外にサブサンプリング画像を用いる適応的拡大ダイヤモンド探索

指導教員

近藤 利夫 教授

平成 26 年度

三重大学大学院 工学研究科 情報工学専攻
計算機アーキテクチャ研究室

立野 篤 (413M518)

内容梗概

動画像高精細化の需要が高まる中，増大するデータ量の低減に向けた符号化の圧縮率向上の要求が強まっている．このような状況の中，H.264/AVC の 2 倍にまで圧縮性能を高めた H.265/HEVC が注目されている．しかし，処理の複雑性も大幅に増加したため，符号化速度の大幅な低下を来している．この符号化速度低下を抑えるには処理量の大半を占める動き検出処理の高速化が不可欠である．H.265 の参照ソフトウェアに組み込まれている TZSearch は高精度・高効率を両立する探索アルゴリズムのひとつながら，高速化の程度はまだまだ不十分である．

動画像に対するブロックマッチングでは，その評価値の SAD が大きい点から最小点へ向かって単調に変化する傾向がある．また，予測ベクトルで指定される位置付近に SAD 最小箇所が存在することが多い．これに対し TZSearch では，SAD 最小箇所であることの確実性を担保するために探索範囲に応じる一定範囲までの拡大型ダイヤモンド探索は省けないことから，遠隔点の探索は無駄打ちになる確率が高い．他にも，拡大型ダイヤモンド探索では探索点が疎らで画像の再利用率が低いため，キャッシュメモリのヒット率が低下しやすい．ヒット率低下はロード命令の所要サイクル数の増加を招き，演算量の低減率に見合った高速化を妨げる問題もある．

本論文ではこれらの問題の解決を目指し，TZSearch に抜本的な改良を加えた動き検出法を提案した．具体的には，拡大型ダイヤモンド探索においてその探索パタン内の SAD 最小箇所と探索中心からの距離に応じて次の探索のパタンサイズを設定する手法，中心付近を除きサブサンプリング画像を用いることで演算量低減を図る手法，小規模の探索のみで早期に探索を打ち切る手法の 3 つを組み合わせた高速動き検出アルゴリズムである．H.265 の参照ソフトウェアに実装して性能の評価を行い，圧縮効率や画質の低下を最小限に抑えながら，TZSearch に比べて AD 演算回数が平均 85.58%低減され，メモリアクセス効率が平均 23.47%改善されることを明らかにした．

三重大学大学院　工学研究科

Abstract

Improvement of video compression performance is important more than ever before because the data amount has been greatly increased by becoming high definition videos practical. Therefore, H.265/HEVC being twice as efficient as H.264/AVC is predicted to be a mainstream. However, the computational complexity has been greatly increased and one has not got the sufficient encoding speed. The improvement of motion estimation is an essential to reduce the encoding speed degradation. TZSearch that is used in H.265 reference software is one of the highly accurate and efficient motion estimation. However, the speed of the motion estimation has not been improved sufficiently yet.

In the block matching for the videos, SAD of the evaluation value tends to change monotonically toward from large point to the minimum point. The minimum SAD point usually exists in the neighborhood of the prediction motion vector position. In TZSearch, the search in distant points is often vain. This is because, the expanding diamond search cannot reduce an extent area that executes searching in order to ensure the certainty of being the minimum SAD point. And, it is difficult to reuse of the reference image in TZSearch because the search points are sparse. Therefore, the hit rate of a cache memory is easy to deteriorate. The hit rate deterioration increases the number of cycles of load operations and interferes with speed up.

In this paper, the author proposed a new motion estimation that is significant improvement over TZSearch. It has three effective techniques to achieve high accuracy and efficiency. First technique is adaptive size setting for the expanding diamond search pattern according to the distance from the current search center to the next search center. Second technique is usage of the sub-sampled image for block matching at the distant search points. Third technique is early termination of the search. The author implemented the proposed algorithm on the reference software. The algorithm reduces 85.58% of AD operations and improve memory access efficiency by 23.47% compared to TZSearch under the condition that minimizing the deterioration of compression ratio and video quality.

三重大学大学院　工学研究科

目 次

図 目 次

表 目 次

1　まえがき

近年，地上デジタル放送への完全移行や，4K テレビの普及など，動画像高精細化の需要はますます高まっている．しかし，動画像の高精細化に伴い，そのデータ量は大幅に増大しており，圧縮率のさらなる向上が望まれている．このため，現在主流の動画像圧縮符号化標準である MPEG-2 や H.264/AVC に対し，画質はそのままで，圧縮率を MPEG-2 の 4 倍，H.264/AVC の 2 倍にまで高めた H.265/HEVC が 2013 年に標準化され，今後の主流となると予想される．

しかし，この最新の動画像圧縮符号化標準 H.265/HEVC は，圧縮率は向上しているものの，処理の複雑性は大幅に増加している．これまで，符号化処理全体の 70-80%を占め，高速処理の障害となってきた動き検出の複雑性も大きく増加している．可変ブロックサイズが H.264/AVC の 4x4〜16x16 の 7 種類から，8x4/4x8〜64x64 の 24 種類と大幅に拡張されたからである．また，動画像の高精細化自体も処理時間を増加させる大きな要因となっている．現在主流のフルハイビジョン 1920x1080 から，4K 解像度 3840x2160 になると縦横が共に 2 倍で，処理量は 4 倍となる．また，今後予定されているスーパーハイビジョン 7680x4320 になると，さらに縦横が 2 倍になり，処理量 16 倍にもなる．

このような状況の中，符号化処理を高速に行うため，さまざまな動き検出アルゴリズムが提案されてきた．なかでも，H.264/AVC の参照ソフトウェアに搭載された EPZS は高精度・高効率を両立する，最も優れたソフトウェア処理向きアルゴリズムのひとつであり，H.265/HEVC の参照ソフトウェアにも改良版で追跡型探索を拡大型ダイヤモンドパタンで行うようにした TZSearch が搭載されている [1]．その性能の高さから，最近も一層の演算量低減を狙う改良の試みがいくつか報告されている [2]．しかし，TZSearch はもちろん，いずれの試みも局所最適解に陥ることを避けるため，探索点を疎らにとる拡大型ダイヤモンド探索を繰り返し用いており，演算量の低減率が大きくない上に，参照画像の再利用率が低く，転送ネックをきたしやすい弱点もある．

本論文では，このような観点から TZSearch に抜本的な改良を加えた動き検出法を提案する．具体的には，拡大型ダイヤモンドパタンによる追跡探索（拡大型ダイヤモンド探索）において，その探索パタン内の SAD 最小箇所と探索中心からの距離に応じて次の探索のパタンサイズ（探索範囲）を設定する手法，中心付近を除きサブサンプリング画像を用いることで演算量低減を図る手法，小規模の探索のみで早期に探索を打ち切

る手法の 3 つを組み合わせたアルゴリズムを提案し，H.265/HEVC の参照ソフトウェアに実装して，その性能を評価し，TZSerach に比べて演算量などがどれだけ低減されるかを明らかにする．

以下，2 章で動き補償について，3 章で既存のアルゴリズムとその問題点，4 章で提案手法，5 章で評価，6 章でその考察を述べる．

2　動画像圧縮と動き補償の概要

2.1　動画像圧縮の概要

ブロードバンド時代を迎え，光ファイバ通信やLTE通信が広く普及したことで以前よりも帯域が広くなっている．また，Blu-rayといった大規模記憶媒体の登場などにより，動画像圧縮の重要性は低くなってきているように思われるかもしれない．しかし，4Kテレビの登場やスーパーハイビジョンが予定されているように動画像の高精細化が進み，情報量が非常に増加していることや，圧縮しないままでは帯域を大幅に専有してしまうことなどから，以前にも増して動画像圧縮の重要性は高くなっている．

動画像圧縮は動き補償とDCTの2つの技術をベースとして，高い圧縮性能が実現されている．動き補償とは，被写体の動き量を検出して，その結果で効果的な予測画面を作る技術である．動き補償が画面間の圧縮技術であるのに対して，DCT(Discrete Consine Transform: 離散コサイン変換) は画面内の圧縮技術である．DCTにより画像を周波数成分に変換することで圧縮しやすくする．例えば，人間の眼の特性は，低周波成分に敏感で高周波成分には鈍感であるので，DCTで画像を周波数成分に変換し，高周波成分を取り除いてから逆DCTで画素に戻すことで圧縮できる．このように動画像は動き補償とDCTの2つの技術をベースに圧縮されている．

2.2　動き補償

DCTが1枚のフレーム（画像）で行われる圧縮であるのに対して，動き補償は複数のフレームを用いて行われる圧縮である．動画像では現フレームの次に来るフレームは現フレームと同じだと予測し，両者の差分のみを送ることで圧縮が可能である．しかし，物体が動いた場合，その部分の情報量は増加してしまう．動き補償は，そのような物体が動いた場合にも情報量を増やさずに圧縮する技術である．

例えば，自動車や人などの物体が動く場合，動いた方向と量（動きベクトル）を知ることができれば，その情報を付加して予測フレームを作ることができ，その予測フレームとの差分を送ることで，物体が動いた場合にも高い圧縮率を実現している．

2.3 動き検出

動き補償を行うためには物体の動きベクトルを知る必要があるが，それを調べるのが動き検出である．動画像圧縮では先に述べた動き補償やDCTなどさまざまな処理が基本的にブロック単位で行われる．この動き検出においてもブロックを単位として処理が行われる．まず，符号化対象ブロックを基準に過去または未来の復元フレーム内で探索範囲を定める．そしてその探索範囲内でブロックマッチングを行っていきコストSAD（Sum of Absolute Difference）が最も小さい箇所を検出する．ブロックマッチングとは，ブロックの対応する画素どうしを比較し，その差分を足し合わせていくことでSADを算出する処理である．そのSADが最も小さい箇所が，符号化対象ブロックの動いた先と考えられるので，符号化対象ブロックとSAD最小箇所との距離と方向が動きベクトルとなる．

2.4 H.265/HEVC

動画像圧縮では，これまでにMPEG-2やH.264/AVCといった符号化標準が広く普及してきた．そのような符号化標準の最新のものがH.265/HEVCである．H.265/HEVCは，同等の画質でMPEG-2の4倍，H.264/AVCの2倍の圧縮率を有しており，非常に高い圧縮性能を持った符号化標準である．この高い圧縮性能を実現できた要因のひとつは可変ブロックサイズの種類が大幅に増加したことである．H.264/AVCでは4x4〜16x16の7種類であったのに対してh.265/HEVCでは8x4/4x8〜64x64の24種類に拡張された．

ブロック分割の方法は，まず画像が64x64のブロックに分割され，各ブロックは最も符号化効率が良くなるように，さらに細かく分割される．分割の形状を図2.1に示す．各ブロックが図2.1のいずれかの形状で木構造的に分割されていく．また，分割された64x64ブロックの例を図2.2に示す．H.265/HEVCでは図2.1下段のような非対称のブロック分割が新たに導入されたことも特徴のひとつである．

可変ブロックサイズにより圧縮性能は大きく向上したが，処理の複雑性は，種類がH.264/AVCなどに比べて大幅に拡張されたため，非常に増加している．例えば動き検出においても，可変ブロックサイズの種類が増えたことで，同じ位置のブロックでも多くの種類のブロックについて探索しなければならなくなるため，処理の複雑性が大幅に増加してしまう．可変ブロックサイズ以外にも動画像の高精細化などにより，演算量は

一層増えていくと予想される．このため，動画像符号化の更なる高速化が以前にも増して求められる．

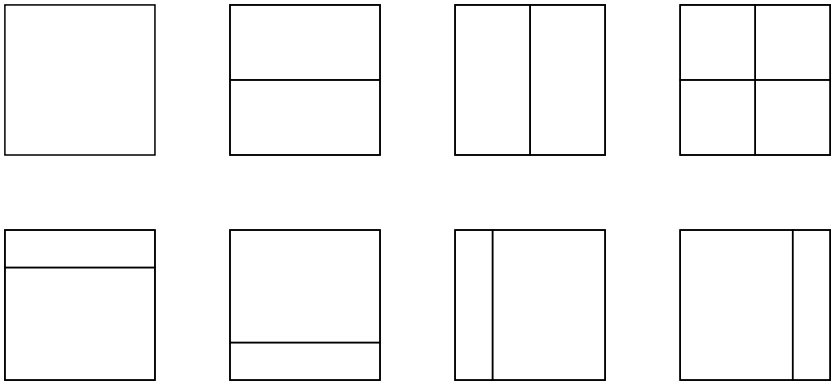

図 2.1: ブロックの分割形状

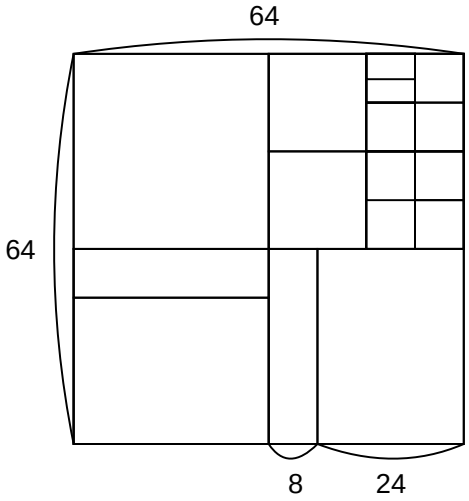

図 2.2: ブロック分割の例

3　既存の動き検出アルゴリズムとその問題点

3.1　代表的な動き検出アルゴリズム

動き検出アルゴリズムは，これまでに様々なものが提案されてきている．それらは大別すると，ハードウェア処理向きとソフトウェア処理向きに分けられる．

ハードウェア処理向きの動き検出アルゴリズムの代表的なものとしては，全探索や粗密階層探索 [3] が挙げられる．これらのアルゴリズムは，探索点数は多いものの，アルゴリズムが単純でハードウェア化が比較的容易である．

また，ソフトウェア処理向きの動き検出アルゴリズムとして，初期の Three Step Search[4]，New Three Step Search[5]，スパイラルサーチ [6] をはじめ，近年提案された PMVFAST[7]，UMHexagonS[8]，EPZS[9] などが挙げられる．これらのアルゴリズムは，ハードウェア化が比較的困難であるものの，高効率の探索アルゴリズムを組み合わせることで探索点数が大幅に低減されている．

3.2　ハードウェア処理向き検出法

3.2.1　全探索

全探索は最も単純な動き検出法で，探索範囲全てをくまなく探索する．全ての点を探索するため，探索精度は最も高くなる．また，探索点が疎らにならないため並列化が容易であり，データの再利用率も非常に高く，転送のオーバヘッドが小さい．しかし，探索範囲全ての点を探索するため，処理量が非常に膨大になるため速度を要求される場面で，全探索単体で使われることは少なく，3.2.2 の粗密階層探索法などと組み合わせて使われる．

3.2.2　粗密階層探索法

最初に画素を間引いて情報量を削った粗い画像（サブサンプリング画像）で探索し，おおよその位置を絞ってから密な画像で狭い範囲のみを探索する．密な画像で探索した場合に比べて，サブサンプリング画像で探索した方がより少ない演算量で同等の探索を行うことが可能なので，演

算量が低減できる．しかし，情報量を削ったサブサンプリング画像で探索しているために，探索精度が低下しやすいというデメリットがある．

3.3 ソフトウェア処理向き検出法

3.3.1 Three Step Search(TSS)

Three Step Search(TSS) の探索例を図 3.3 に示す．TSS のアルゴリズムは，まず予測ベクトルなどから探索中心を定め，探索中心と大きめの正方形パタンで周囲 8 点を探索し，SAD 最小箇所に探索中心を移動する．次は少し大きめの正方形パタンで周囲 8 点を探索し，SAD 最小箇所に探索中心を移動する．最後に周囲 8 点を探索し，SAD 最小箇所を MV(Motion Vector: 動きベクトル) とする．

TSS は 3 段階の探索で動きベクトルを検出する探索法である．この探索法では最初に広い範囲を探索してから探索範囲を絞っていき，精度を上げていっている．そのため，探索点数が非常に少なく高速である．しかし，H.265/HEVC などの探索範囲 ±64 のように非常に広い探索範囲の場合，TSS では狭い範囲しか探索できず，探索精度が低下しやすい．

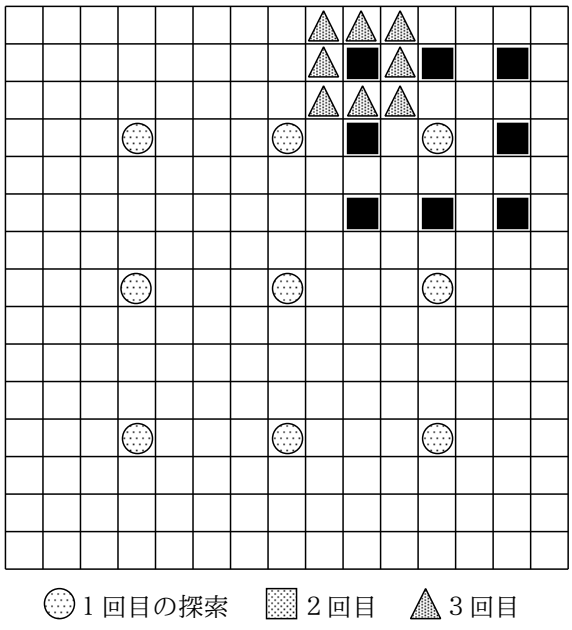

図 3.3: Three Step Search

3.3.2 PMVFAST

動き検出では探索開始位置の選択が非常に重要で，それによって探索精度や速度が大きく変わってくる．PMVFAST は探索開始位置をなるべく多くの候補から選択することで，より良い探索開始位置を選択することに主眼を置いている．

PMVFAST では探索開始位置の決定に，予測ベクトル，動きゼロの位置，左，上，右上ブロックの持つ MV，直前のフレームの対応するブロックを予測する際に使用した MV を使用しており，非常に多くの候補から選択していることが特徴である．

PMVFAST のアルゴリズムは探索開始位置を決定したら，選択された探索開始位置によってラージダイヤモンドサーチもしくはスモールダイヤモンドサーチを行うという単純なものである．しかし，その効果は大きく，後の動き検出アルゴリズムに大きく影響を与えた手法のひとつである．

3.4 TZSearch の問題点

高速動き検出アルゴリズムの多くは予測ベクトルを用いることでおおよその位置を決め，そこから SAD が小さい方へと向かっていく手法がとられている．予測ベクトルの精度は高く，その近傍に SAD 最小箇所が存在していることが多いので，少ない探索点数で SAD 最小箇所を見つけることができる．しかし，探索点数の少なさから，SAD 最小箇所が予測ベクトル位置から遠く離れていた場合などにローカルミニマムに陥る可能性がある．

最新の動画像圧縮符号化標準 H.265/HEVC の参照ソフトウェアに搭載された TZSearch は高効率・高精度両立の点で評価の高いアルゴリズムのひとつである．TZSearch では図 3.4 と図 3.5 に示す 2 つの探索パタンが主に使用される．図 3.4 は拡大型ダイヤモンドパタンで，距離が 2 の累乗の点を拡大しながら探索していくものである．図 3.5 はラスタパタンで，探索範囲内の点を一定間隔で探索していくものである．以下に TZSearch のアルゴリズムを示す．

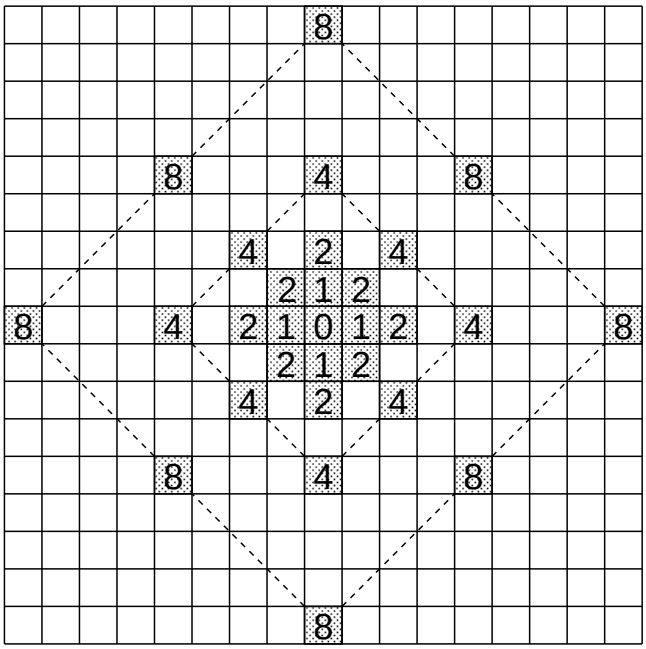

図 3.4: 拡大型ダイヤモンドパタン

Step 1. 初期探索範囲を設定する．また，予測ベクトルで指定される予測位置と動きゼロの位置を評価し，SAD が小さい方の位置を探索中心とする．

Step 2. 拡大型ダイヤモンド探索を行う．このとき，探索中心に隣接する 8 点で SAD 最小箇所が見つかれば，その点に隣接する探索中心から距離 3 の未探索点 2 点も探索する．もし SAD 最小箇所が探索中心と一致するならばその点を MV とし，探索を終了する．一致しなければ Step 3. へ移行する．

Step 3. Step 2. で見つかった暫定 SAD 最小箇所の探索中心からの距離が設定された iRaster（デフォルトでは 5）より大きければラスタパタンによる探索を行う．

Step 4. 探索中心を暫定 SAD 最小箇所に移し，Step 2. に戻る．

動画像では SAD の大きい箇所から SAD 最小箇所へ向かって単調に変化する傾向がある [11]．また，予測ベクトルで指定される位置付近に SAD

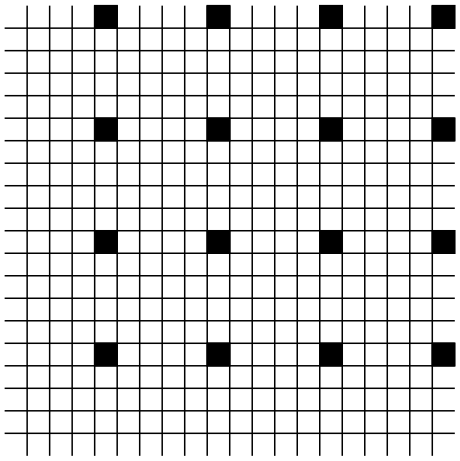

図 3.5: ラスタパタン

最小箇所が存在することが多いものの確実ではない．最小点であることの確実性を担保するために，探索範囲に応じる一定範囲までの拡大型ダイヤモンド探索は省けないものの，遠隔点の探索は無駄打ちになる確率が高い．

他にも，拡大型ダイヤモンド探索では探索点が疎らで画像の再利用率が低く，キャッシュメモリのヒット率が低下しやすい．ヒット率低下はロード命令の所要サイクル数の増加を招き，演算量の低減率に見合った高速化が得られない問題点もある．

4　提案手法

4.1　適応的な拡大範囲の変更

従来の拡大型ダイヤモンド探索では，SAD の大きい箇所から SAD 最小箇所へ向かって単調に減少していく傾向を活かせていないのに加え，探索点が疎らになるという問題点がある．これらの問題点の解消に向けて，拡大型ダイヤモンド探索の拡大範囲を，直前の拡大ダイヤモンド探索の SAD 最小箇所（暫定 SAD 最小箇所）が見つかった距離によって適応的に変更する探索法（適応的拡大ダイヤモンド探索: Adaptive Expanding Diamond Search[AEDS]）を採る．

具体的には，暫定 SAD 最小箇所の拡大中心からの距離の半分を次の探索範囲をとして設定する．ただし，探索精度向上のため，探索範囲の下限は ±2 とする．図 4.6 に最初の暫定 SAD 最小箇所の拡大中心からの距離が 16 であるときの例を示す．この場合，次の探索範囲は暫定 SAD 最小箇所から ±4 に設定する．また，距離が 4 以下の 2 であっても，次の探索範囲は ±2 となる．

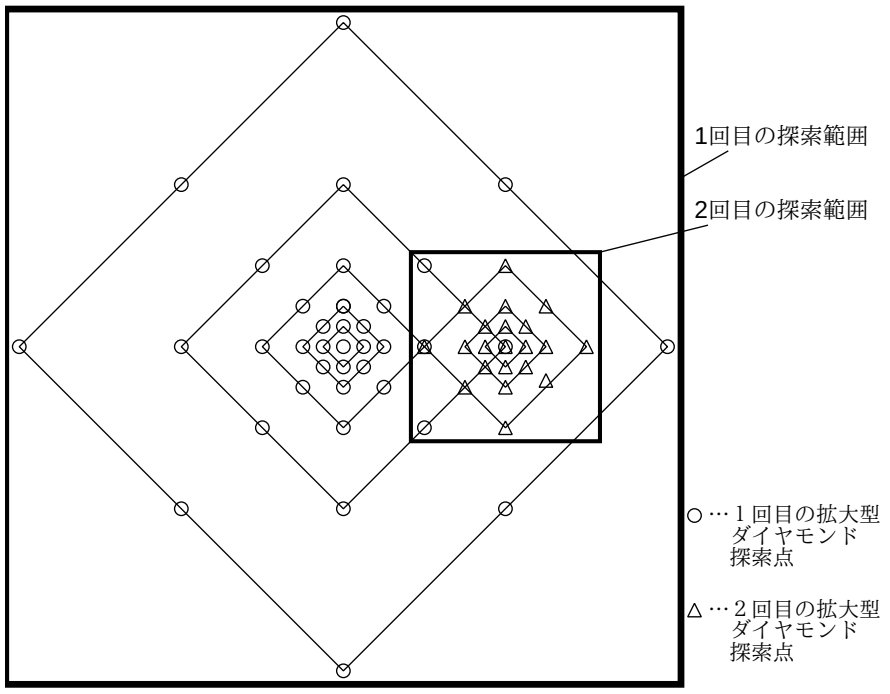

図 4.6: 適応的拡大型ダイヤモンド探索

4.2　サブサンプリングの適用

これまでに提案されてきた多くの動き検出アルゴリズム（例えばPMV-FASTやEPZSなど）にその特性が利用されてきたように，予測ベクトルの予測精度は非常に高く，予測ベクトルに一致するか，その近傍でSAD最小箇所が見つかることが多い[10], [11]．そのため遠い箇所の探索は，予測が外れた場合にしか意味を持たない．また，拡大型探索では遠く離れるほど探索点と探索点の間隔が粗くなるため，高い探索の精度は得られていない．それでも全体として高い検出精度が得られていることから，遠い箇所の探索精度は，予測ベクトル付近でSAD最小箇所が見つからない場合に，それよりましな点を検出できる程度の精度が確保されればよいと推定される．

これらの観点から，探索中心の近傍点では通常の復元画像を用い，それ以外ではサブサンプリングした復元画像を用いるブロックマッチングにより粗い探索を行うことで，探索精度を落とさずに演算量低減を図る．SAD最小箇所が存在すると予想される探索点には最初から通常の復元画像を用いているので，探索範囲全てにサブサンプリングした復元画像を用いる粗密階層探索に比べて探索精度が落ち難い．

今回，実装を容易にするため，サブサンプリング画像の生成は図4.7に示すように，補間無しで2x2画素または4x4画素の左上の画素のみを使用して生成する．また，通常の復元画像を用いてSADを計算する場合と，サブサンプリングした復元画像を用いてSADを計算する場合とでは，画素数の違いからSADが異なってしまう．そこで，サブサンプリングした復元画像を用いた場合のSADに，画素数の比率の逆数を乗ずることで補正をかける．

この手法では，どの距離からサブサンプリング画像を使用するか，どの精度のサブサンプリング画像を用いるのかによって複数の組み合わせが存在する．以下では，様々な条件で実験を行った結果から，サブサンプリング画像を用いる最適な条件について考える．実験は表4.1に示す4種類のシーケンスで動きの多い箇所33フレームをエンコードしておこなった．

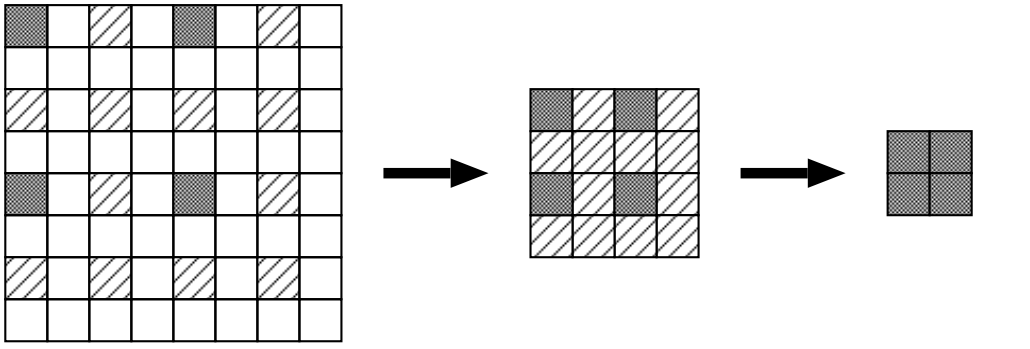

図 4.7: サブサンプリング画像の生成

表 4.1: 実験に使用したテストシーケンス

画像名	解像度	フレーム数	フレーム番号	フレームレート
BasketballDrill	832x480	33	250-282	50
BQMall	832x480	33	40-72	60
Kimono	1920x1080	33	60-92	24
ParkScene	1920x1080	33	180-212	24

図 4.8，図 4.9，図 4.10 は 2 画素精度サブサンプリング画像を使用し始める距離を変えて実験した結果であり，4.1 の AEDS からの増加率や低減率を表している．AD(Absolute Difference) 演算回数とは画素の差分をとり，その絶対値を求める回数である．図 4.8 から，サブサンプリング画像を用いることで AD 演算回数を低減できていることがわかる．サブサンプリング画像を使用し始める距離が中心に近いほど，低減効果が大きく表れている．また，図 4.9 と図 4.10 から，距離 2 や距離 4 からサブサンプリング画像を使用し始めた場合，PSNR やビットレートの悪化が顕著に見られるのに対し，距離 8 以上では悪化は小さい．

以上の結果から，2 画素精度サブサンプリング画像を使用するのは高速化と符号化効率の両方を考慮して，距離 8 から使用するのが最適であるといえる．

2 画素精度の他にも 4 画素精度についても同様に考える．図 4.11，図

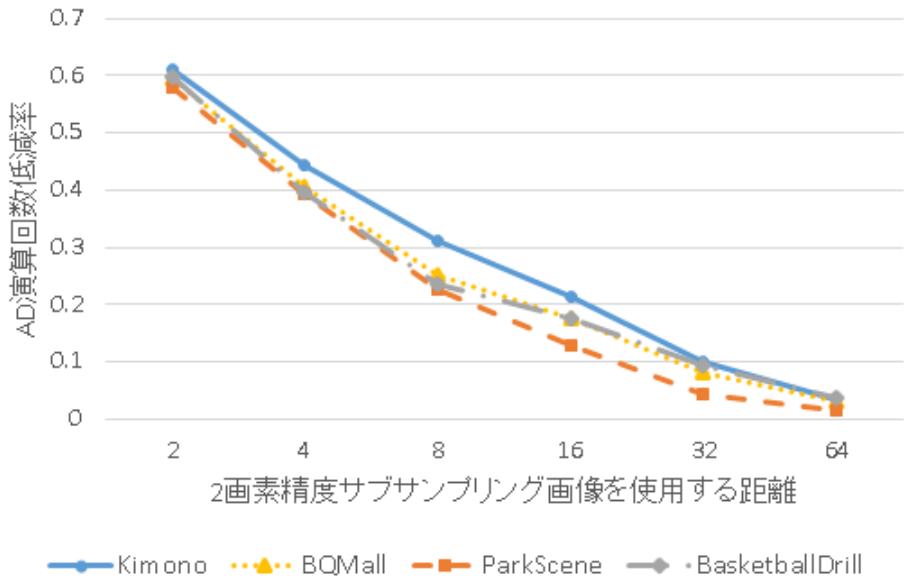

図 4.8: 2 画素精度画像使用時の AD 演算回数低減率

4.12，図 4.13 は 4 画素精度サブサンプリング画像を使用し始める距離を変えて実験した結果であり，距離 8 から 2 画素精度画像を用いる場合からの増加率や低減率を表している．図 4.11 から，4 画素精度サブサンプリング画像を用いることで AD 演算回数が低減できることがわかるが，その低減率は，AEDS からの 2 画素精度サブサンプリング画像を用いたときの効果に比べて非常に小さい．また，図 4.12 と図 4.13 から符号化効率が改善されることもない．これらの結果から，処理の複雑性を増加させる 4 画素精度サブサンプリング画像の使用は避けた方が良いといえる．

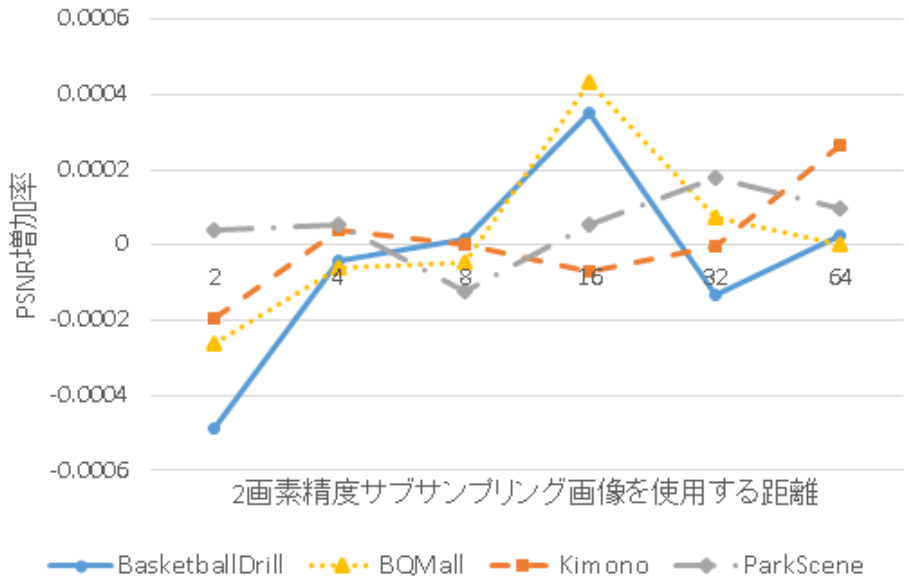

図 4.9: 2 画素精度画像使用時の PSNR 増加率

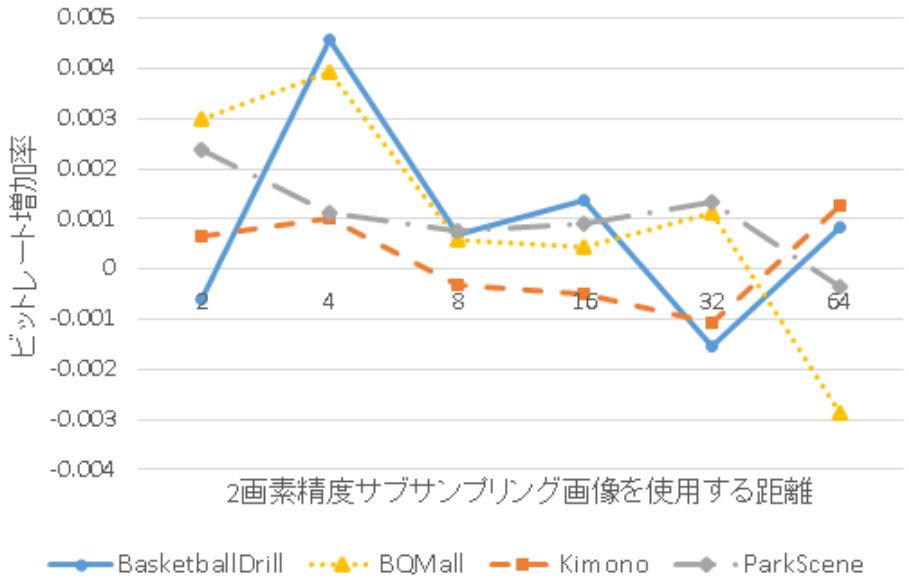

図 4.10: 2 画素精度画像使用時のビットレート増加率

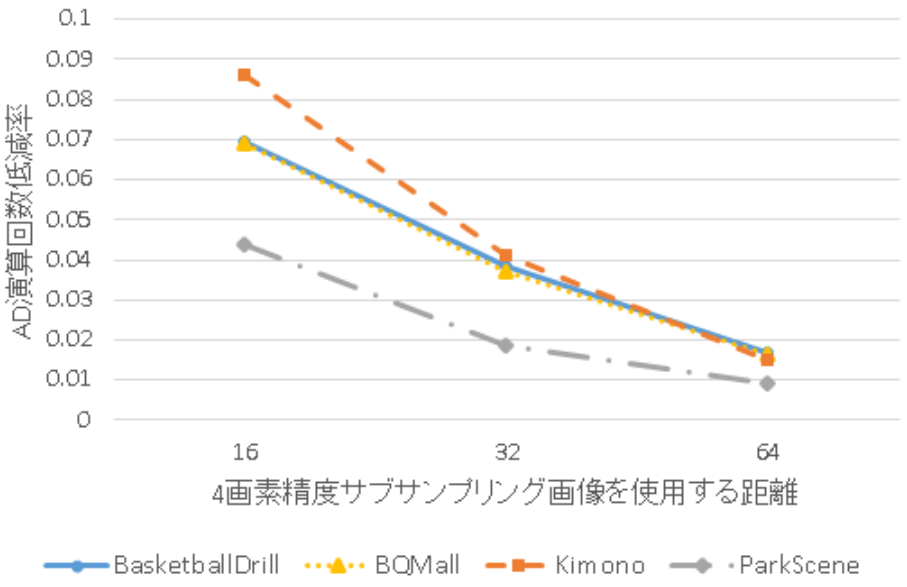

図 4.11: 4 画素精度画像使用時の AD 演算回数低減率

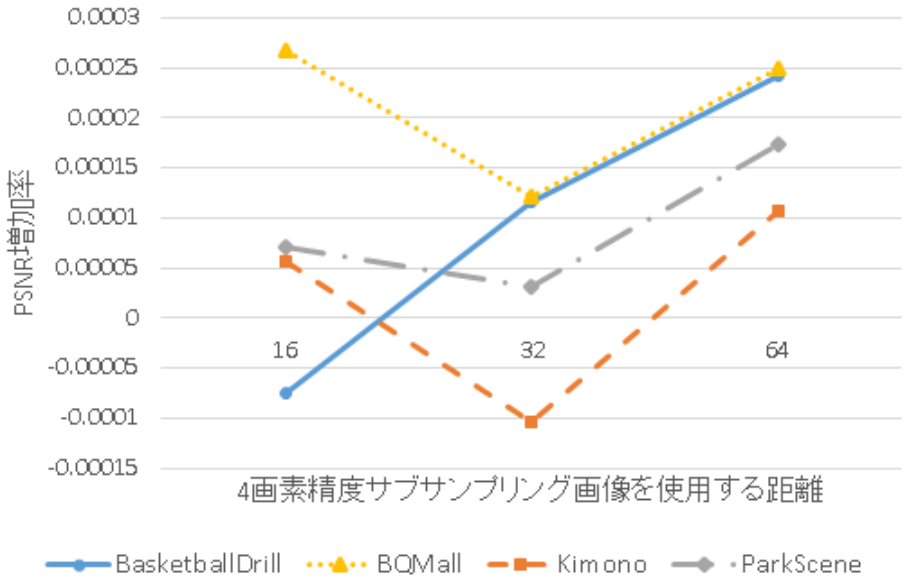

図 4.12: 4 画素精度画像使用時の PSNR 増加率

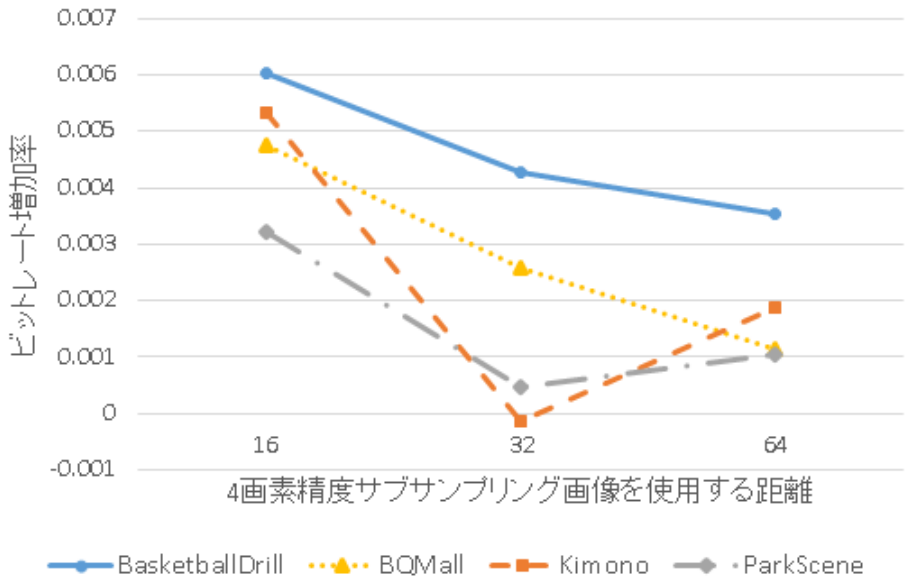

図 4.13: 4 画素精度画像使用時のビットレート増加率

4.3 早期打ち切り

これまでに様々なアルゴリズムに早期打ち切り手法が組み込まれてきたように，その効果は非常に大きい [2]．本動き検出でもさらに演算量を低減するために早期打ち切り手法の組み込みを図る．4.2 で述べたように予測ベクトルの予測精度は非常に高く，その近傍で SAD 最小箇所が見つかることが多い．そこで，最初に小規模の追跡型探索を行い，その結果により AEDS をおこなうかどうかを決定する．今回は，追跡型探索によく用いられる図 4.14 のダイヤモンドパタンで実験を行った．

図 4.15 は，4.2 に早期打ち切り手法を実装したときの演算量低減効果である．縦軸は早期打ち切りを実装していない状態からの低減率を表しているおり，横軸は小規模の追跡型探索を行う回数の上限を表している．図 4.15 から，早期打ち切り手法を用いることで演算量が更に低減できることがわかる．グラフの右に行くほど打ち切られる確率が高く，AEDS を省略できるので，この上限を大きくするほど演算量を低減できる．しかし，今回の実験結果によると，上限を 4 より大きくしても，効果が頭打ちになる．

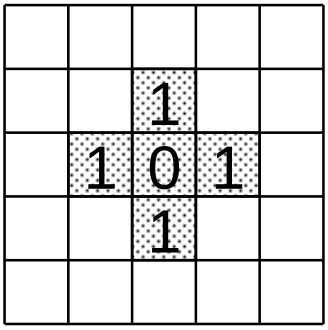

図 4.14: ダイヤモンドパタン

図 4.16 と図 4.17 はそれぞれ早期打ち切りを実装していないものからの PSNR とビットレートの増減率である．とくに図 4.17 からは，上限を大きくするほどビットレートが増加する傾向にあることがわかる．これは拡大型ダイヤモンド探索が行われる回数が減ったことで，探索精度が低下したと考えられる．今回の実験結果では上限 3 または 4 で，PSNR とビットレートの変動がどの画像においても小さい．

以上の結果から，早期打ち切りの小規模追跡型探索を行う回数の上限は 4 が最適といえる．

4.4 提案手法の詳細

提案手法は 4.1，4.2，4.3 の 3 つの手法を組み合わせたもので，以下の手順で動きベクトルを検出する．

Step 1. 初期探索範囲を設定する．通常はデフォルトの ±64 を用いる．また，予測ベクトルで指定される予測位置と動きゼロの位置を評価し，SAD が小さい方の位置を探索中心とする．

Step 2. ダイヤモンドパタンによる追跡型探索を最大で 4 回まで行い，SAD 最小箇所が探索中心と一致するならば，その点を MV とし，探索を終了する．一致しなければ Step 3. へ移行する．

Step 3. 拡大型ダイヤモンド探索を行う．拡大していく際，探索中心からの距離が 8 以上であれば，2 画素精度サブサンプリング画像を用いてブロックマッチングを行う．このとき，探索中心に隣接する 8

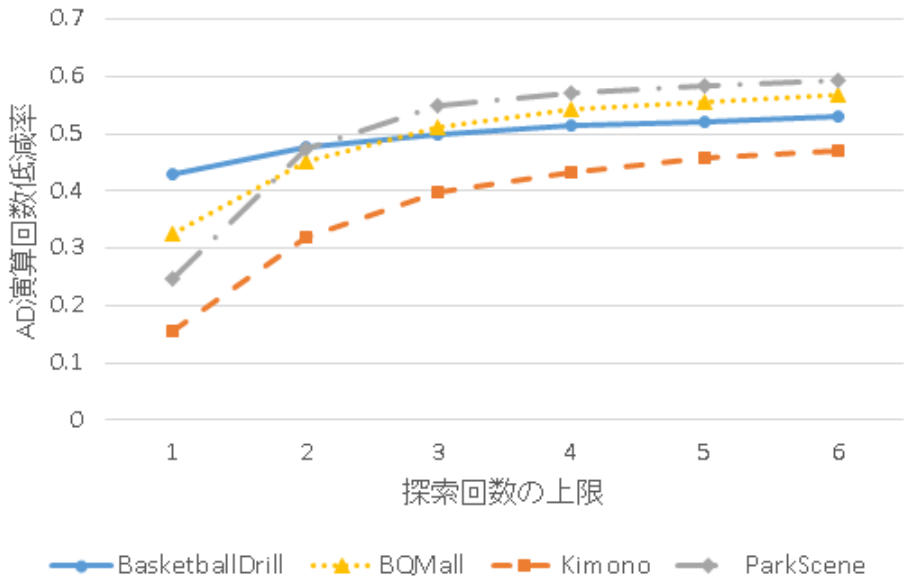

図 4.15: 早期打ち切りによる演算量低減効果

点で SAD 最小箇所が見つかれば，その点に隣接する探索中心から距離 3 の未探索点 2 点も探索する．もし SAD 最小箇所が探索中心と一致するならば，その点を MV とし，探索を終了する．一致しなければ Step 4. へ移行する．

Step 4. 暫定 SAD 最小箇所と探索中心との距離に応じて，次の探索範囲を新しく設定する．どのように設定するかは 4.1 で示した規則に従う．

Step 5. 探索中心を暫定 SAD 最小箇所に移し，Step 3. に戻る．

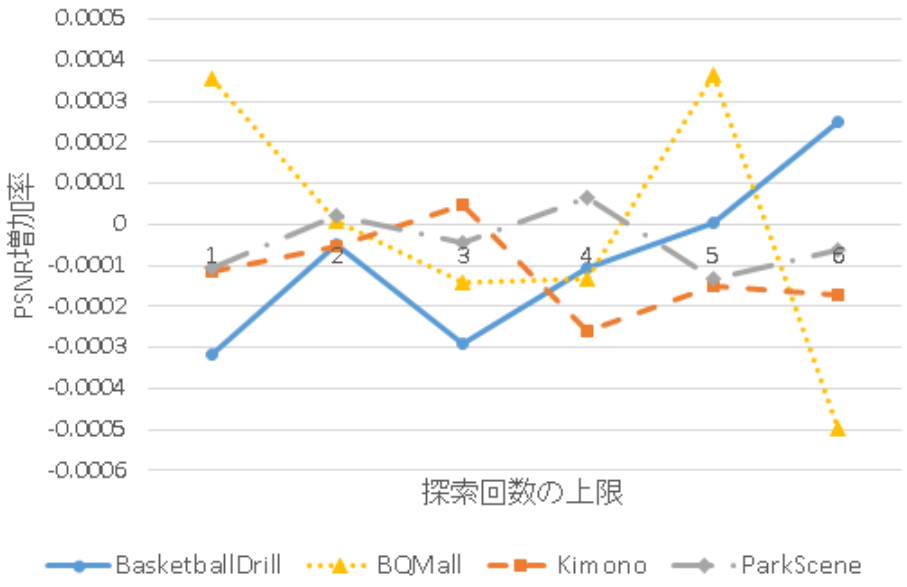

図 4.16: 早期打ち切りの PSNR 増加率

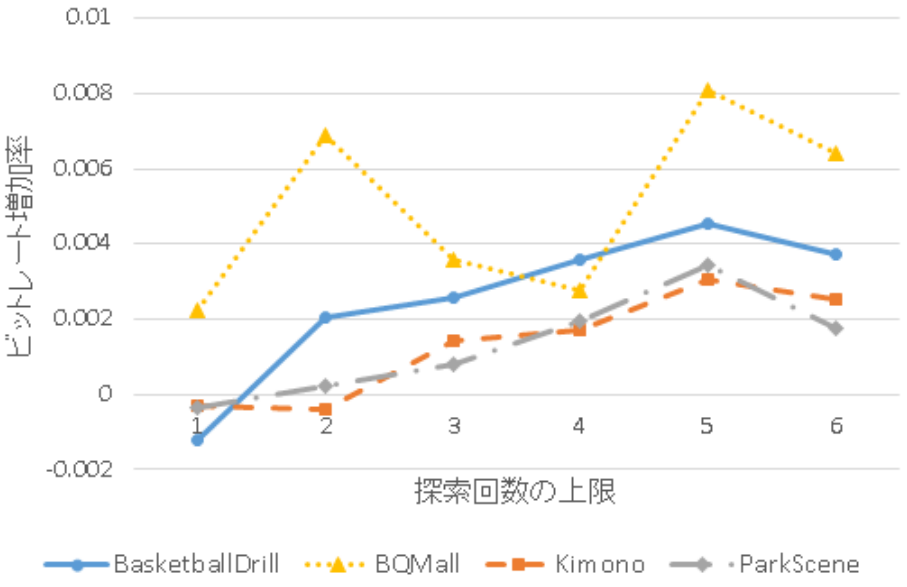

図 4.17: 早期打ち切りのビットレート増加率

5　評価

H.265/HEVC の参照ソフトウェア HM-10.0 に提案手法を実装し，その効果を確認した．評価に用いたテストシーケンスは共通実験条件におけるテストシーケンスのうち，表 5.2 に示す 4 種類である．また，エンコード条件は表 5.3 に示すとおりである．

表 5.2: テストシーケンス

画像名	解像度	フレーム数	フレームレート
BasketballDrill	832x480	500	50
BQMall	832x480	600	60
Kimono	1920x1080	240	24
ParkScene	1920x1080	240	24

表 5.3: エンコード条件

符号化構造	ランダムアクセス GOP Size = 8 (IBBBBBBBBBBBBB…)
探索範囲	±64
CU サイズ / 深さ	64 / 4
QP	22, 27, 32, 37

評価はビットレート，YPSNR，AD 演算回数，1 ロードあたりのサイクル数を採り，BD-rate や低減率を算出した．YPSNR とは，Y(輝度) における PSNR であり，この数値が高いほど画質が良い．BD-rate とは，同一の PSNR を仮定したときのビットレートの増加率を表しており，この数値が高いほど圧縮効率が悪化したということである．AD 演算回数は 4.3 で説明したものと同じである．1 ロードあたりのサイクル数とは，画像のロードにかかるサイクル数をロードの数で割ったものであり，キャッシュメモリに格納される画像の再利用率が高いほど小さくなる．このサイクル数により，キャッシュメモリからレジスタへの転送速度がボトルネックとなる場合の最短の処理サイクル数が制約される．この計測は，HM-10.0 上に表 5.4 の構成のキャッシュメモリのシミュレータを組み込むことでお

こなった．この構成は一般的に使われている ARM プロセッサに搭載されているキャッシュメモリ構成をもとに設定した．

表 5.4: 想定キャッシュメモリ構成

メモリ	容量	データ格納構造	レイテンシ
1 次キャッシュ	32KB	4way セットアソシアティブ	1cycle
2 次キャッシュ	256KB	ダイレクトマップ	10cycle
主記憶	—	———————	200cycle

以上の条件でおこなった評価の結果を表 5.5 に示す．また，図 5.18-5.21 にそれぞれのシーケンスの RD 曲線を示す．RD 曲線（レート-歪曲線）とは，シーケンス全体にわたるビットレートを横軸，PSNR を縦軸にプロットした曲線になっており，RD 曲線が左上に位置するほど PSNR が高く，必要となるビットレートが少ないことを意味している．図中の A の曲線が TZSearch を使用してエンコードしたときのもので，B が提案手法を使用したときのものである．

表 5.5: 評価結果

		TZSearch				提案手法						
シーケンス	QP	ビットレート (kbps)	YPSNR (dB)	AD 演算回数 ($\times 10^{11}$ 回)	1 ロードあたりのサイクル数 (cycle)	ビットレート (kbps)	YPSNR (dB)	AD 演算回数 ($\times 10^{11}$ 回)	1 ロードあたりのサイクル数 (cycle)	BD-rate (%)	AD 演算回数低減率 (%)	1 ロードあたりのサイクル数低減率 (%)
Basketball Drill	22	3616.78	40.50	8.01	2.235	3642.95	40.50	0.91	2.018	1.1	88.61	9.71
	27	1755.93	37.40	7.33	2.267	1771.70	37.39	0.86	2.057		88.33	9.26
	32	865.62	34.46	6.53	2.166	873.33	34.45	0.78	2.117		88.00	2.26
	37	459.20	31.94	5.70	2.156	464.56	31.92	0.71	2.146		87.57	0.46
BQMall	22	3831.91	40.23	6.32	2.454	3850.28	40.23	0.88	2.009	0.8	86.11	18.13
	27	1823.89	37.74	5.68	2.428	1834.82	37.74	0.82	2.035		85.58	16.19
	32	931.35	35.02	5.06	2.330	936.97	35.01	0.76	2.074		84.91	10.99
	37	498.61	32.28	4.52	2.153	500.80	32.26	0.71	2.118		84.24	1.63
Kimono	22	4732.49	41.60	20.33	5.104	4743.20	41.60	2.62	2.929	0.5	87.11	42.61
	27	2159.37	39.73	17.80	4.743	2169.64	39.73	2.36	2.890		86.75	39.07
	32	1053.09	37.42	15.43	4.527	1056.97	37.42	2.10	2.863		86.37	36.76
	37	533.29	35.03	13.35	4.275	535.76	35.02	1.88	2.833		85.92	33.73
ParkScene	22	7406.56	40.05	9.82	3.65	7432.00	40.05	1.63	2.170	0.6	83.45	40.55
	27	3179.04	37.52	8.85	3.528	3192.36	37.51	1.53	2.129		82.77	39.65
	32	1450.09	34.91	8.01	3.529	1456.68	34.90	1.43	2.136		82.10	39.47
	37	670.63	32.39	7.25	3.323	674.10	32.39	1.35	2.160		81.41	35.00
									Average	0.75	85.58	23.47

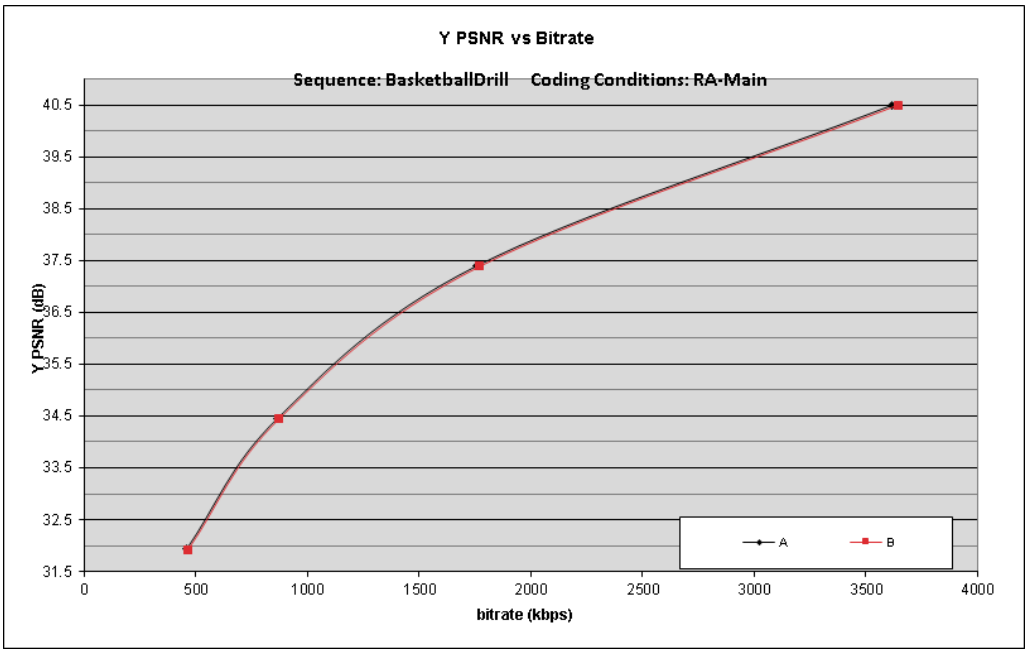

図 5.18: BasketballDrill の RD 曲線

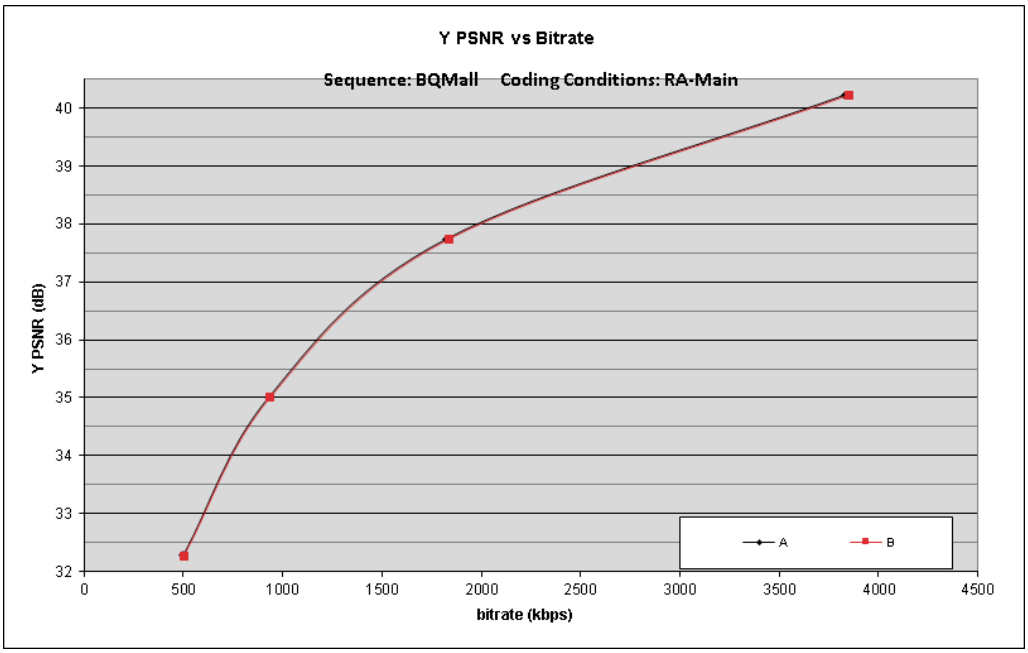

図 5.19: BQMall の RD 曲線

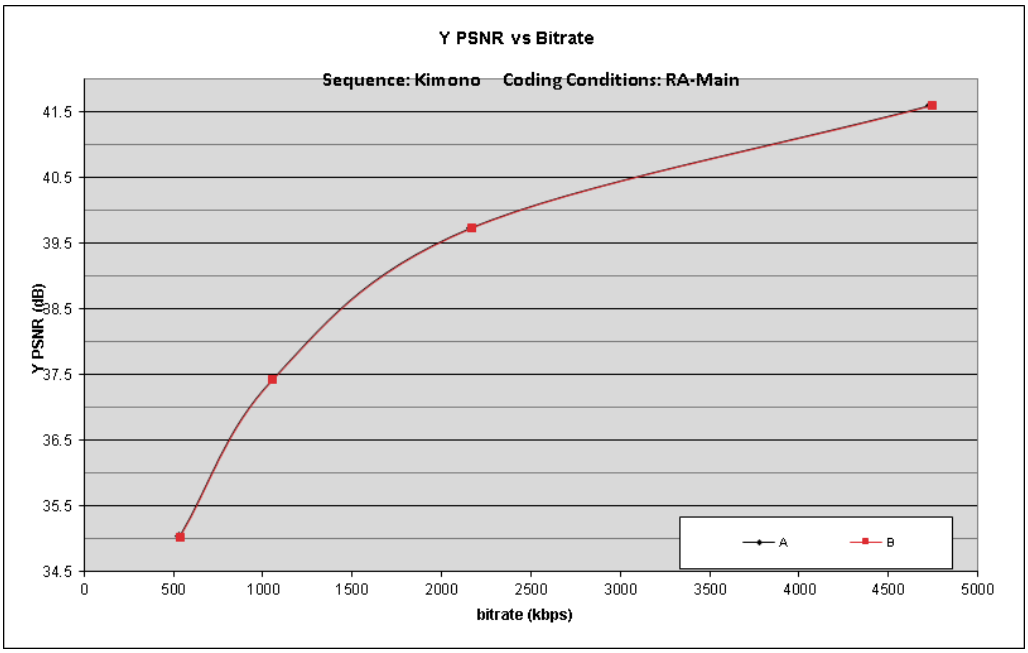

図 5.20: Kimono の RD 曲線

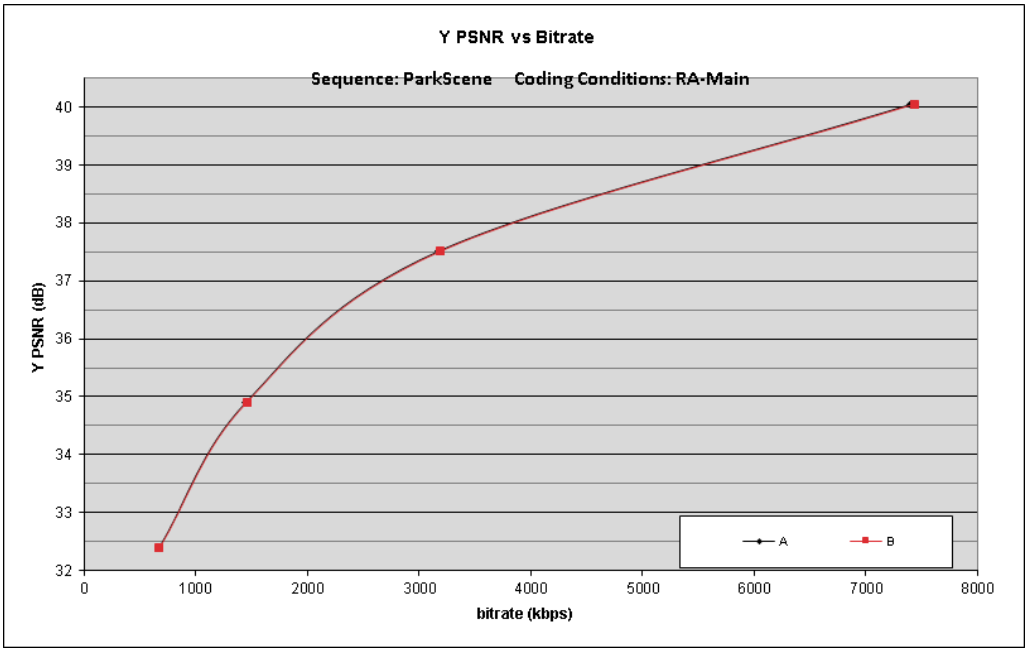

図 5.21: ParkScene の RD 曲線

6　考察

表 5.5 より，デフォルトの TZSearch と比較して提案手法は，AD 演算回数を平均 85.58%低減，1 ロードあたりのサイクル数を平均 23.47%改善できた．1 ロードあたりのサイクル数が改善できたのは不要なデータへのアクセスを避けつつデータの再利用性を向上させた結果である．

図 5.18-5.21 を見ると，RD 曲線がどのシーケンスにおいても TZSeach の曲線と提案手法の曲線がほとんど一致しており，圧縮効率の悪化は非常に小さいと言える．そして，BD-rate が最も悪化した BasketballDrill においても，その値は 1.1%と非常に小さいものとなっている．

CIME[12] は TZSeach の性能を大きく上回るアルゴリズムであり，比較的直近に発表されたものである．また，今回評価に用いた画像と同じ画像を使用していたこともあり，比較対象として選択した．表 6.6 はその比較結果である．低減率は，提案手法の方が AD 演算回数の低減率で，CIME の方は動き検出に要する時間の低減率であるが，動き検出に要する時間はおおよそ AD 演算回数に比例するので，比較には問題ないと考えられる．CIME に比べて提案手法のほうが BD-rate 及び高速化率が平均的に勝っており，提案手法は符号化率と高速化率の両面で優れていると言える．

表 6.6: CIME との比較

シーケンス	BD-rate(%)		低減率 (%)	
	CIME	提案手法	CIME	提案手法
BasketballDrill	1.6	1.1	69.86	88.13
BQMall	0.6	0.8	69.03	85.21
Kimono	1.1	0.5	73.01	86.54
ParkScene	0.6	0.6	64.83	82.43
Average	0.98	0.75	69.18	85.58

提案手法では拡大範囲が制限される上に，サブサンプリング画像が用いられるため，遠隔点における探索精度が中央付近の探索に比べて落ちやすい．今回用いたシーケンスの中でも非常に動きが速く遠隔探索の割合の大きい BasketballDrill で BD-rate が一番悪化したことは，その落ちやすさから来ていると考えられる．しかし，遠隔探索では，探索点数と個々の点の AD 演算量が低減されるため，BasketballDrill は AD 演算回数の低減率が一番大きくなった．

表 5.5 から明らかなように TZSearch は 1 ロードあたりのサイクル数が HD 画像のときに大幅に増加している．それに対して提案手法では HD 画像においても増加量は小さく，アクセス効率が大きく改善されていることがわかる．SD 画像のときよりも HD 画像のときのほうが値が大きくなる傾向があるのは，探索範囲が広がる分，キャッシュメモリのヒット率が低下してアクセス効率が悪くなっていると考えられる．

7　あとがき

本論文では，動画像符号化における動き検出を高速に行う，新たな動き検出法を提案した．提案した動き検出法は，暫定 SAD 最小箇所と探索中心からの距離に応じて拡大範囲を適応的に設定する手法，中心付近を除きサブサンプリング画像を適用する手法，探索の早期打ち切り手法の 3 つを組み合わせたものである．

そして H.265/HEVC 参照ソフトウェアに実装し，評価した結果，圧縮効率を最小限に抑えながら，参照ソフトウェアに搭載されている高速動き検出法 TZSearch に比べて AD 演算回数を平均 85.58%低減，メモリアクセス効率を平均 23.47%改善できることを示した．

今回の提案手法ではブロックサイズによらず同一の探索法を適用した．しかし，H.265/HEVC では H.264/AVC からの可変ブロックサイズの拡張によりブロックの種類が大きく増えている．このため，探索特性がサイズや形の違いにより互いにかなり異なってきていると考えられる．そこで，今後はサイズや形に応じて探索法を使い分ける手法を導入し更なる探索速度と精度の向上をはかっていく必要がある．

謝辞

本研究を遂行するにあたり，日頃から御指導，御助言を頂きました近藤利夫教授，佐々木敬泰助教，深澤祐樹研究員に感謝いたします．また，研究に協力していただいた計算機アーキテクチャ研究室の方々に感謝の意を表します.

参考文献

[1] HIgh Efficiency Video Coding (HEVC). HM Reference Software [online].
Available: https://hec.hhi.fraunhofer.de

[2] Pan, Zhaoqing, et al. “Early termination for TZSearch in HEVC Motion Estimation.” Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on. IEEE, 2013.

[3] Barnea, Daniel I., and Harvey F. Silverman. “A class of algorithms for fast digital image registration.” Computers, IEEE Transactions on 100.2 (1972): 179-186.

[4] T. Koga, K. Iinuma, A. Hirano, Y. Iijima and T. Ishiguro. “Motion compensated interframe coding for video conferencing”, Proc. NTC 81, pp.C9.6.1-9.6.5. 1981.

[5] Li, Renxiang, Bing Zeng, and Ming L. Liou. “A new three-step search algorithm for block motion estimation.” Circuits and Systems for Video Technology, IEEE Transactions on 4.4 (1994): 438-442.

[6] Zahariadis, Th, and D. Kalivas. “A spiral search algorithm for fast estimation of block motion vectors.” Signal Processing VIII, theories and applications. Proceedings of the EUSIPCO 96 (1996): 3.

[7] Tourapis, Alexis M., Oscar C. Au, and Ming L. Liou. “Predictive motion vector field adaptive search technique (PMVFAST)-enhancing block based motion estimation.” Proceedings of SPIE. Vol. 4310. 2001.

[8] Chen, Zhibo, et al. “Fast integer-pel and fractional-pel motion estimation for H. 264/AVC.” Journal of Visual Communication and Image Representation 17.2 (2006): 264-290.

[9] Tourapis, Alexis M. “Enhanced predictive zonal search for single and multiple frame motion estimation.” Electronic Imaging 2002. International Society for Optics and Photonics, 2002.

[10] Xiu-li, Tang, Dai Sheng-kui, and Cai Can-hui. “An analysis of TZSearch algorithm in JMVC.” Green Circuits and Systems (ICGCS), 2010 International Conference on. IEEE, 2010.

[11] Kuo, Chung-Ming, et al. “A novel prediction-based directional asymmetric search algorithm for fast block-matching motion estimation.” Circuits and Systems for Video Technology, IEEE Transactions on 19.6 (2009): 893-899.

[12] Hu, Nan, and En-hui Yang. “Fast Motion Estimation based on Confidence Interval.” IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY. VOL.24. NO.8. 1310-1322. 2014.

A　キャッシュメモリシミュレータの概要

- 評価関数ではキャッシュメモリの tag, valid, dirty を記憶する配列を用意し，参照した画素アドレスの各要素を各配列に記憶する．
- 画素アドレスは，何番目の参照画像かということ，参照ブロックの x, y 座標，画像のサイズがら導出する．
- 画素アドレスとキャッシュ配列間で 1 画素ずつ一致比較を行い，ブロックの水平 1 ライン全てがキャッシュ配列内に存在すればヒット，1 画素でも欠けている部分があればミスとする．
- 1 次キャッシュ，2 次キャッシュ，主記憶の 3 つの階層に対してアクセス回数を記憶し，データ転送に要するサイクル数を導出する．

B　キャッシュメモリシミュレータの使い方

```
/* encmain.cpp */
グローバル変数
KonEvaluation konEva;

  // オリジナルのコード
  catch (po::ParseFailure& e)
  {
    cerr << "Error parsing option \""<< e.arg
    <<"\" with argument \""
    << e.val <<"\"." << endl;
    return 1;
  }
 --------------------
  // シミュレータ用
  if(argc == 8){ // 実行時引数の取得
    konEva.setParameter(atoi(argv[5]), atoi(argv[6]),
    atoi(argv[7]));
  }else{
    konEva.setParameter(-1, -1, -1);
  }
```

```
  konEva.memAlloc(); // 領域確保
  konEva.fileOpen(cTAppEncTop.getInputFile()); // ファイルオープン
  konEva.setImageSize(cTAppEncTop.getSourceWidth(),
  cTAppEncTop.getSourceHeight()); // 画像サイズの取得
 ---------------------
  // オリジナルのコード
  // starting time
  double dResult;
  long lBefore = clock();


/* TAppEncCfg.h */
// シミュレータ用に追加。画像の取得など
  Int getSourceWidth(){ return m_iSourceWidth; }
  Int getSourceHeight(){ return m_iSourceHeight; }
  Char* getInputFile(){ return m_pchInputFile; }

/* TEncGop.cpp */
#if L0045_NON_NESTED_SEI_RESTRICTIONS
      xResetNonNestedSEIPresentFlags();
#endif
      pcPic->getPicYuvRec()->copyToPic(pcPicYuvRecOut);

      pcPic->setReconMark   ( true );
      m_bFirst = false;
      m_iNumPicCoded++;
      m_totalCoded ++;
      /* logging: insert a newline at end of picture period */
      printf("\n");
      fflush(stdout);

      delete[] pcSubstreamsOut;
      konEva.reset(); // シミュレータ用。リセットと出力
  }
```

```
/* TEncSearch.cpp xMotionEstimation() 内に追加 */
  // 探索範囲設定
  if ( bBi )  xSetSearchRange   ( pcCU, rcMv   , iSrchRng,
  cMvSrchRngLT, cMvSrchRngRB );
  else        xSetSearchRange   ( pcCU, cMvPred, iSrchRng,
  cMvSrchRngLT, cMvSrchRngRB );

  m_pcRdCost->getMotionCost ( 1, 0 );

  m_pcRdCost->setPredictor  ( *pcMvPred );
  m_pcRdCost->setCostScale  ( 2 );

  setWpScalingDistParam( pcCU, iRefIdxPred,
  eRefPicList ); /* HM-10.0で追加 */

  // シミュレータ用。
  // どのPUの座標やどの参照画像を使うかなど。
  // 参照画像のリストはシミュレータで扱いやすいようにソートする
  Int iPUX=0, iPUY=0, iPUWidth=0, iPUHeight=0, iRefPOC=0;
  pcCU->getPartPosition(iPartIdx, iPUX, iPUY,
  iPUWidth, iPUHeight); // PUの座標とサイズを取得
  iRefPOC = pcCU->getSlice()->getRefPOC(
  eRefPicList, iRefIdxPred ); // どの画像を参照するか
  if(!konEva.bSort && !(konEva.getFbn() < 0)){
    Int iNumRefIdx0 = pcCU->getSlice()->
                      getNumRefIdx(REF_PIC_LIST_0);
    Int iNumRefIdx1 = pcCU->getSlice()->
                      getNumRefIdx(REF_PIC_LIST_1);
    Int *CombRefPOCList = new Int[iNumRefIdx0 + iNumRefIdx1];
    Int iRefNum = 0;
    for(Int i=0; i<iNumRefIdx0; i++){
      CombRefPOCList[iRefNum] =
      pcCU->getSlice()->getRefPOC(REF_PIC_LIST_0, i);
```

```
  iRefNum++;
}
for(Int i=0; i<iNumRefIdx1; i++){
  CombRefPOCList[iRefNum] =
  pcCU->getSlice()->getRefPOC(REF_PIC_LIST_1, i);
  iRefNum++;
}
// 選択ソート
// http://www7b.biglobe.ne.jp/~robe/cpphtml/html03/cpp03025.html
Int iSearch=0, iMin=0, nMin=0, temp=0;
for(Int iLoop=0; iLoop<(iNumRefIdx0+iNumRefIdx1-1); iLoop++){
  iMin = iLoop;
  nMin = CombRefPOCList[iLoop];
  for(iSearch = iLoop+1; iSearch<(iNumRefIdx0+iNumRefIdx1);
  iSearch++){
    if(CombRefPOCList[iSearch]<nMin){
      iMin = iSearch;
      nMin = CombRefPOCList[iSearch];
    }
  }
  temp = CombRefPOCList[iMin];
  CombRefPOCList[iMin] = CombRefPOCList[iLoop];
  CombRefPOCList[iLoop] = temp;
}
// 重複チェック
for(Int i=0; i<(iNumRefIdx0+iNumRefIdx1-1); i++){
  if(CombRefPOCList[i] == CombRefPOCList[i+1]){
    CombRefPOCList[i] = -1;
    iRefNum--;
  }
}
//重複削除
konEva.RefPOCList = new Int[iRefNum];
konEva.refNum = iRefNum;
Int now = 0;
```

```
    for(Int i=0; i<(iNumRefIdx0+iNumRefIdx1); i++){
      if(CombRefPOCList[i] >= 0){
        konEva.RefPOCList[now] = CombRefPOCList[i];
        now++;
      }
    }
    delete[] CombRefPOCList;
    konEva.bSort = true;
  }
/* ここで取得した物を渡す。
   TEncSearch.cpp xPatternSearchFast() や xTZSearch() などの引
数に Int iPUX, Int iPUY, Int iPUWidth,
   Int iPUHeight, Int iRefPOC を追加する */


/* TEncSearch.cpp xTZSearchHelp の改良 */
/* マッチングの計測やシミュレータの始動など */
/* xTZSearch の代わりにこちらを使う */
__inline Void TEncSearch::xTZSearchHelp_range(
TComPattern* pcPatternKey, IntTZSearchStruct& rcStruct,
const Int iSearchX, const Int iSearchY, const UChar ucPointNr,
const UInt uiDistance, const UInt uiDistance2, Int& riBestDist,
Int iPUX, Int iPUY, Int iPUWidth, Int iPUHeight, Int iRefPOC )
{
  UInt  uiSad;

  Pel*  piRefSrch;

  piRefSrch = rcStruct.piRefY + iSearchY *
              rcStruct.iYStride + iSearchX;

  //-- jclee for using the SAD function pointer
  m_pcRdCost->setDistParam( pcPatternKey, piRefSrch,
  rcStruct.iYStride,  m_cDistParam );
```

```
  // fast encoder decision: use subsampled SAD
  //when rows > 8 for integer ME  サブサンプル探索設定
  if ( m_pcEncCfg->getUseFastEnc() )
  {
    if ( m_cDistParam.iRows > 8 )
    {
      m_cDistParam.iSubShift = 1;
    }
  }

  setDistParamComp(0);  // Y component 輝度で計算するように設
定

  konEva.lliMatchingCount++; // マッチング回数

  konEva.evaluation(iPUX+iSearchX, iPUY+iSearchY,
  iPUWidth, iPUHeight, iRefPOC, m_cDistParam.iSubShift,
  m_cDistParam.iSubsampling); // キャッシュへ

  // distortion
  m_cDistParam.bitDepth = g_bitDepthY;
  uiSad = m_cDistParam.DistFunc( &m_cDistParam );

  // SAD 演算回数の計測
  unsigned long long int sadNum = 0;
  Int shift = m_cDistParam.iSubShift;
  if(m_cDistParam.iSubsampling > 0){
    shift += m_cDistParam.iSubsampling;
  }
  sadNum = iPUWidth * iPUHeight;
  sadNum >>= shift;
  konEva.ulliSadNum += sadNum;

  // motion cost
  uiSad += m_pcRdCost->getCost( iSearchX, iSearchY );
```

```
  // これまでの最小の判定
  if( uiSad < rcStruct.uiBestSad )
  {
    rcStruct.uiBestSad      = uiSad;
    rcStruct.iBestX         = iSearchX;
    rcStruct.iBestY         = iSearchY;
    rcStruct.uiBestDistance = uiDistance;
    rcStruct.uiBestRound    = 0;
    rcStruct.ucPointNr      = ucPointNr;
    riBestDist              = uiDistance2;
  }
}
```

以上の他にKonEvaluation.hとKonEvaluation.cppを追加する.
実行時引数を与えることでシミュレータが動作する.

実行例: TAppEncoder.exe -c encoder_randomaccess_main.cfg -c
Football.cfg 0 2 0

後ろから3つ目の引数で1次キャッシュ容量を設定できる.
0:32KB 1:16KB 2:8KB 3:4KB 4:1GB 5:2GB 6:4GB

2つ目の引数でキャッシュの入れ替え方式を指定する.
0:ダイレクトマップ 1:2wayセット 2:4wayセット

最後の引数で2次キャッシュ容量を設定.
2次キャッシュ容量 0:256KB

2次キャッシュはダイレクトマップとなっている.

実行が終わるとcsvファイルが生成され，そこに1次キャッシュヒット回数などが出力される.

C　HM 変更点概要

- TComRdCost::xGetSAD16() など
  - 基本は同じだがサブサンプリング画像を用いた場合も対応できるようにした．
  - どの程度サブサンプリングしているか記憶するために DistParam クラスに iSubsampling という変数を追加している．この iSubsampling の値は TEncSearch の動き検出部で決定する．また，サブサンプリングの度合いによって iSubStep の値も変わるので計算する必要がある．
  - 0:サブサンプリング無し 2:2 画素精度サブサンプリング 4:4 画素精度サブサンプリング
  - iSubsampling の値によって SAD の計算を分岐している．
  - TComRdCost::xGetSAD8() など他の xGetSAD 関数も同様に変更した．
- TEncSearch::xTZSearchHelp_range()
  - 適応的に拡大範囲を変更するために探索中心との正確な距離を保存する必要があったので追加した．
  - uiDistance2 というのが正確な距離で，SAD 最小箇所が更新されたとき riBestDist を uiDistance2 の値に更新する．
- TEncSearch::xTZ8PointDiamondSearch_range()
  - 適応的拡大範囲の変更を実現するために引数などを増やす必要があったので変更した．動作自体は元のとまったく同じである．
- TEncSearch::xTZSearch_termination()
  - 基本的な動作は本文を参照すること．
  - 適応的拡大範囲の変更を実現するために元々のサーチレンジとは別の pcMvSrchRng2 を用意し，こちらを変更することで実現する．iBestDist という正確な探索中心との距離を保存する変数を用意し，この値によって pMvSrchRng2 を変更する．
  - 拡大型探索を行う際に，iDist の値によって m_cDistParam.iSubsampling を決める．

D　提案手法ソースコード

```
UInt TComRdCost::xGetSAD16( DistParam* pcDtParam )
{
  if ( pcDtParam->bApplyWeight )
  {
    return xGetSADw( pcDtParam );
  }
  Pel* piOrg   = pcDtParam->pOrg; //符号化対象画像
  Pel* piCur   = pcDtParam->pCur; //参照画像
  Int  iRows   = pcDtParam->iRows;
  Int  iSubShift  = pcDtParam->iSubShift;
  Int  iSubStep   = ( 1 << iSubShift );
  if(pcDtParam->iSubsampling > 0){ //縦方向のサブサンプリング
のため
    iSubStep <<= (pcDtParam->iSubsampling/2);
  }
  Int  iStrideCur = pcDtParam->iStrideCur*iSubStep;
//pcDtParam->iStrideCur は画像の横幅 (マージン付加)
  Int  iStrideOrg = pcDtParam->iStrideOrg*iSubStep;
//pcDtParam->iStrideOrg はブロックの横幅

  UInt uiSum = 0;

  if(pcDtParam->iSubsampling == 0){
    for( ; iRows != 0; iRows-=iSubStep )
    {
      uiSum += abs( piOrg[0] - piCur[0] );
      uiSum += abs( piOrg[1] - piCur[1] );
      uiSum += abs( piOrg[2] - piCur[2] );
      uiSum += abs( piOrg[3] - piCur[3] );
      uiSum += abs( piOrg[4] - piCur[4] );
      uiSum += abs( piOrg[5] - piCur[5] );
      uiSum += abs( piOrg[6] - piCur[6] );
      uiSum += abs( piOrg[7] - piCur[7] );
      uiSum += abs( piOrg[8] - piCur[8] );
```

```
    uiSum += abs( piOrg[9] - piCur[9] );
    uiSum += abs( piOrg[10] - piCur[10] );
    uiSum += abs( piOrg[11] - piCur[11] );
    uiSum += abs( piOrg[12] - piCur[12] );
    uiSum += abs( piOrg[13] - piCur[13] );
    uiSum += abs( piOrg[14] - piCur[14] );
    uiSum += abs( piOrg[15] - piCur[15] );

    piOrg += iStrideOrg;
    piCur += iStrideCur;
  }
}else if(pcDtParam->iSubsampling == 2){
  for( ; iRows != 0; iRows-=iSubStep )
  {
    uiSum += abs( piOrg[0] - piCur[0] );
    uiSum += abs( piOrg[2] - piCur[2] );
    uiSum += abs( piOrg[4] - piCur[4] );
    uiSum += abs( piOrg[6] - piCur[6] );
    uiSum += abs( piOrg[8] - piCur[8] );
    uiSum += abs( piOrg[10] - piCur[10] );
    uiSum += abs( piOrg[12] - piCur[12] );
    uiSum += abs( piOrg[14] - piCur[14] );

    piOrg += iStrideOrg;
    piCur += iStrideCur;
  }
}else if(pcDtParam->iSubsampling == 4){
  for( ; iRows > 0; iRows-=iSubStep )
  {
    uiSum += abs( piOrg[0] - piCur[0] );
    uiSum += abs( piOrg[4] - piCur[4] );
    uiSum += abs( piOrg[8] - piCur[8] );
    uiSum += abs( piOrg[12] - piCur[12] );

    piOrg += iStrideOrg;
```

```
      piCur += iStrideCur;
    }
  }

  uiSum <<= (iSubShift+pcDtParam->iSubsampling);

  return uiSum >> DISTORTION_PRECISION_ADJUSTMENT(
pcDtParam->bitDepth-8);
}


/*-----------------*/
/** range用。
 */
__inline Void TEncSearch::xTZSearchHelp_range(
TComPattern* pcPatternKey, IntTZSearchStruct& rcStruct,
const Int iSearchX, const Int iSearchY, const UChar ucPointNr,
const UInt uiDistance, const UInt uiDistance2,
Int& riBestDist, Int iPUX, Int iPUY, Int iPUWidth,
Int iPUHeight, Int iRefPOC )
{
  UInt  uiSad;

  Pel*  piRefSrch;

  piRefSrch = rcStruct.piRefY + iSearchY *
              rcStruct.iYStride + iSearchX;

  //-- jclee for using the SAD function pointer
  m_pcRdCost->setDistParam( pcPatternKey, piRefSrch,
  rcStruct.iYStride,  m_cDistParam );

  // fast encoder decision: use subsampled SAD when
rows > 8 for integer ME  サブサンプル探索設定
  if ( m_pcEncCfg->getUseFastEnc() )
```

```
  {
    if ( m_cDistParam.iRows > 8 )
    {
      m_cDistParam.iSubShift = 1;
    }
  }

  setDistParamComp(0);  // Y component 輝度で計算するように設
定

  konEva.lliMatchingCount++; // マッチング回数

  konEva.evaluation(iPUX+iSearchX, iPUY+iSearchY,
  iPUWidth, iPUHeight, iRefPOC, m_cDistParam.iSubShift,
  m_cDistParam.iSubsampling); // キャッシュへ

  // distortion
  m_cDistParam.bitDepth = g_bitDepthY;
  uiSad = m_cDistParam.DistFunc( &m_cDistParam );

  // SAD 演算回数の計測
  /* サブサンプリングのことも考慮 */
  unsigned long long int sadNum = 0;
  Int shift = m_cDistParam.iSubShift;
  if(m_cDistParam.iSubsampling > 0){
    shift += m_cDistParam.iSubsampling;
  }
  sadNum = iPUWidth * iPUHeight;
  sadNum >>= shift;
  konEva.ulliSadNum += sadNum;

  // motion cost
  uiSad += m_pcRdCost->getCost( iSearchX, iSearchY );

  // これまでの最小の判定
```

```
  if( uiSad < rcStruct.uiBestSad )
  {
    rcStruct.uiBestSad      = uiSad;
    rcStruct.iBestX         = iSearchX;
    rcStruct.iBestY         = iSearchY;
    rcStruct.uiBestDistance = uiDistance;
    rcStruct.uiBestRound    = 0;
    rcStruct.ucPointNr      = ucPointNr;
    riBestDist              = uiDistance2; // 追加
  }
}

/*-----------------*/

__inline Void TEncSearch::xTZ8PointDiamondSearch_range(
TComPattern* pcPatternKey, IntTZSearchStruct& rcStruct,
TComMv* pcMvSrchRngLT, TComMv* pcMvSrchRngRB,
const Int iStartX, const Int iStartY, const Int iDist,
Int& riBestDist, Int iPUX, Int iPUY, Int iPUWidth,
Int iPUHeight, Int iRefPOC )
{

xTZSearchHelp_range( pcPatternKey, rcStruct, iStartX,
iTop, 2, iDist, iDist, riBestDist, iPUX,
 iPUY, iPUWidth, iPUHeight, iRefPOC );
/* xTZSearchHelp を xTZSearchHelp_range に変更 */

}


/*-----------------*/
/**
 * レンジ+粗い+早期打ち切り
 * \param pcCU CodingUnit のクラス
 * \param pcPatternKey neighbour access class pointer 近傍アク
```

セスクラスのポインタ?近傍の画素にアクセスするためのクラス?
```
 * \param piRefY 参照画像の輝度値
 * \param iRefStride 参照画像のストライド。横幅
 * \param pcMvSrchRngLT サーチレンジ ← ↑
 * \param pcMvSrchRngRB サーチレンジ → ↓
 * \param rcMv 検出された動きベクトル
 * \param ruiSAD 最良だったSAD
 * \param iPUX PUのx座標
 * \param IPUY PUのy座標
 * \param iPUWidth PUの横幅
 * \param IPUHeight PUの高さ
 * \param iRefPOC 参照画像番号
 * \returns Void
 */
Void TEncSearch::xTZSearch_termination( TComDataCU* pcCU,
TComPattern* pcPatternKey, Pel* piRefY, Int iRefStride,
TComMv* pcMvSrchRngLT, TComMv* pcMvSrchRngRB, TComMv& rcMv,
UInt& ruiSAD, Int iPUX, Int iPUY, Int iPUWidth,
Int iPUHeight, Int iRefPOC ){
  // サーチレンジの設定
  Int   iSrchRngHorLeft   = pcMvSrchRngLT->getHor();
  Int   iSrchRngHorRight  = pcMvSrchRngRB->getHor();
  Int   iSrchRngVerTop    = pcMvSrchRngLT->getVer();
  Int   iSrchRngVerBottom = pcMvSrchRngRB->getVer();

  // 可変範囲用
  TComMv* pcMvSrchRngLT2 = new TComMv();
  TComMv* pcMvSrchRngRB2 = new TComMv();
  Int iSrchRngHorLeft2   = iSrchRngHorLeft;
  Int iSrchRngHorRight2  = iSrchRngHorRight;
  Int iSrchRngVerTop2    = iSrchRngVerTop;
  Int iSrchRngVerBottom2 = iSrchRngVerBottom;
  Int iSrchRngHorLeft_tmp   = iSrchRngHorLeft;
  Int iSrchRngHorRight_tmp  = iSrchRngHorRight;
  Int iSrchRngVerTop_tmp    = iSrchRngVerTop;
```

```
  Int iSrchRngVerBottom_tmp = iSrchRngVerBottom;
  pcMvSrchRngLT2->set(pcMvSrchRngLT->getHor(),
  pcMvSrchRngLT->getVer()); // 初期化
  pcMvSrchRngRB2->set(pcMvSrchRngRB->getHor(),
  pcMvSrchRngRB->getVer());

  TZ_SEARCH_CONFIGURATION // TZSearch の設定

  UInt uiSearchRange = m_iSearchRange; // サーチレンジ設定?
  pcCU->clipMv( rcMv ); // 予測値を入れてる?
  rcMv >>= 2; // 一行上の clipMv 中で 2 ビット左シフトしてるからそ
れを戻す。整数画素探索なので。

  // init TZSearchStruct
  IntTZSearchStruct cStruct;
  cStruct.iYStride    = iRefStride;
  cStruct.piRefY      = piRefY;
  cStruct.uiBestSad   = MAX_UINT;
  m_cDistParam.iSubsampling = 0;

  // set rcMv (Median predictor) as start point
and as best point  メディアン予測候補を評価
  xTZSearchHelp( pcPatternKey, cStruct, rcMv.getHor(),
  rcMv.getVer(), 0, 0, iPUX, iPUY, iPUWidth,
  iPUHeight, iRefPOC );

  // test whether one of PRED_A, PRED_B, PRED_C MV is
better start point than Median predictor
  // bTestOtherPredictedMV=1 に設定されている場合、隣接 3 予測候
補を評価
  if ( bTestOtherPredictedMV )
  {
    for ( UInt index = 0; index < 3; index++ )
    {
      TComMv cMv = m_acMvPredictors[index];
```

```
      pcCU->clipMv( cMv );
      cMv >>= 2;
      xTZSearchHelp( pcPatternKey, cStruct, cMv.getHor(),
      cMv.getVer(), 0, 0, iPUX, iPUY, iPUWidth,
      iPUHeight, iRefPOC );
    }
  }

  // test whether zero Mv is better start point
than Median predictor
  // BTestZeroVector=1に設定されている場合、0ベクトル位置を評価
  if ( bTestZeroVector )
  {
    xTZSearchHelp( pcPatternKey, cStruct, 0, 0, 0, 0,
    iPUX, iPUY, iPUWidth, iPUHeight, iRefPOC );
  }

  // start search
  Int  iDist = 0;
  Int  iStartX = cStruct.iBestX;
  Int  iStartY = cStruct.iBestY;
  Int  iBestDist = 0; // 最良点時のiDist保存，参照渡しで距離を
保存するように

  // 追跡
  Bool termination = false;
  for(Int i=0; i<4; i++){
    iStartX = cStruct.iBestX;
    iStartY = cStruct.iBestY;
    cStruct.uiBestDistance = 0;
    Int iTop    = iStartY - 1;
    Int iBottom = iStartY + 1;
    Int iLeft   = iStartX - 1;
    Int iRight  = iStartX + 1;
    // ダイヤモンド
```

```
if(iTop >= iSrchRngVerTop){
  xTZSearchHelp( pcPatternKey, cStruct, iStartX, iTop,
  0, 1, iPUX, iPUY, iPUWidth, iPUHeight, iRefPOC );
}
if(iLeft >= iSrchRngHorLeft){
  xTZSearchHelp( pcPatternKey, cStruct, iLeft, iStartY,
  0, 1, iPUX, iPUY, iPUWidth, iPUHeight, iRefPOC );
}
if(iRight <= iSrchRngHorRight){
  xTZSearchHelp( pcPatternKey, cStruct, iRight, iStartY,
  0, 1, iPUX, iPUY, iPUWidth, iPUHeight, iRefPOC );
}
if(iBottom <= iSrchRngVerBottom){
  xTZSearchHelp( pcPatternKey, cStruct, iStartX, iBottom,
  0, 1, iPUX, iPUY, iPUWidth, iPUHeight, iRefPOC );
}
// スクウェア
/*if(iTop >= iSrchRngVerTop){
  if(iLeft >= iSrchRngHorLeft){
    xTZSearchHelp( pcPatternKey, cStruct, iLeft, iTop,
    0, 1, iPUX, iPUY, iPUWidth, iPUHeight, iRefPOC );
  }
  xTZSearchHelp( pcPatternKey, cStruct, iStartX, iTop,
  0, 1, iPUX, iPUY, iPUWidth, iPUHeight, iRefPOC );
  if(iRight <= iSrchRngHorRight){
    xTZSearchHelp( pcPatternKey, cStruct, iRight, iTop,
    0, 1, iPUX, iPUY, iPUWidth, iPUHeight, iRefPOC );
  }
}
if(iLeft >= iSrchRngHorLeft){
  xTZSearchHelp( pcPatternKey, cStruct, iLeft, iStartY,
  0, 1, iPUX, iPUY, iPUWidth, iPUHeight, iRefPOC );
}
if(iRight <= iSrchRngHorRight){
  xTZSearchHelp( pcPatternKey, cStruct, iRight, iStartY,
```

```
      0, 1, iPUX, iPUY, iPUWidth, iPUHeight, iRefPOC );
    }
    if(iBottom <= iSrchRngVerBottom){
      if(iLeft >= iSrchRngHorLeft){
        xTZSearchHelp( pcPatternKey, cStruct, iLeft, iBottom,
        0, 1, iPUX, iPUY, iPUWidth, iPUHeight, iRefPOC );
      }
      xTZSearchHelp( pcPatternKey, cStruct, iStartX, iBottom,
      0, 1, iPUX, iPUY, iPUWidth, iPUHeight, iRefPOC );
      if(iRight <= iSrchRngHorRight){
        xTZSearchHelp( pcPatternKey, cStruct, iRight, iBottom,
        0, 1, iPUX, iPUY, iPUWidth, iPUHeight, iRefPOC );
      }
    }*/
    if(cStruct.uiBestDistance == 0){
      termination = true;
      break;
    }
  }

  if(!termination){ // 打ち切りフラグが立っていなければ普通の探
索
    iStartX = cStruct.iBestX;
    iStartY = cStruct.iBestY;
    cStruct.uiBestDistance = 0;

    // first search
    // 順次拡大型８点ダイヤモンド、あるいは順次拡大型８点スクエア
探索
    for ( iDist = 1; iDist <= (Int)uiSearchRange; iDist*=2 )
    {
      /*if(iDist > 16){
        m_cDistParam.iSubsampling = 4; //この場合４画素制度(4x4
画素の左上)
      } else */if(iDist > 4){
```

```
        m_cDistParam.iSubsampling = 2;   // どれだけ間引くか。
何ビット右シフトするのか。これの場合２画素精度（2x2 画素の左上）
      } else {
        m_cDistParam.iSubsampling = 0;
      }
      xTZ8PointDiamondSearch_range ( pcPatternKey, cStruct,
      pcMvSrchRngLT, pcMvSrchRngRB, iStartX, iStartY, iDist,
      iBestDist, iPUX, iPUY, iPUWidth, iPUHeight, iRefPOC );
      if ( bFirstSearchStop && ( cStruct.uiBestRound >=
      uiFirstSearchRounds ) ) // stop criterion
      {
        break;
      }
    }
    m_cDistParam.iSubsampling = 0;
    // calculate only 2 missing points instead 8 points
if cStruct.uiBestDistance == 1  探索中心から距離１の場合、２点探
索を行う
    if ( cStruct.uiBestDistance == 1 )
    {
      cStruct.uiBestDistance = 0;
      xTZ2PointSearch( pcPatternKey, cStruct, pcMvSrchRngLT,
      pcMvSrchRngRB, iPUX, iPUY, iPUWidth,
      iPUHeight, iRefPOC );
    }

    // 探索範囲の変更
    iSrchRngHorLeft_tmp   = cStruct.iBestX;
    iSrchRngHorRight_tmp  = cStruct.iBestX;
    iSrchRngVerTop_tmp    = cStruct.iBestY;
    iSrchRngVerBottom_tmp = cStruct.iBestY;

    switch(iBestDist){
      case 64:
        iSrchRngHorLeft_tmp   += -32;
```

```
  iSrchRngHorRight_tmp  +=  32;
  iSrchRngVerTop_tmp    += -32;
  iSrchRngVerBottom_tmp +=  32;
  uiSearchRange = 32;
  break;
case 32:
  iSrchRngHorLeft_tmp   += -16;
  iSrchRngHorRight_tmp  +=  16;
  iSrchRngVerTop_tmp    += -16;
  iSrchRngVerBottom_tmp +=  16;
  uiSearchRange = 16;
  break;
case 16:
  iSrchRngHorLeft_tmp   += -8;
  iSrchRngHorRight_tmp  +=  8;
  iSrchRngVerTop_tmp    += -8;
  iSrchRngVerBottom_tmp +=  8;
  uiSearchRange = 8;
  break;
case 8:
  iSrchRngHorLeft_tmp   += -4;
  iSrchRngHorRight_tmp  +=  4;
  iSrchRngVerTop_tmp    += -4;
  iSrchRngVerBottom_tmp +=  4;
  uiSearchRange = 4;
  break;
case 4:
  iSrchRngHorLeft_tmp   += -2;
  iSrchRngHorRight_tmp  +=  2;
  iSrchRngVerTop_tmp    += -2;
  iSrchRngVerBottom_tmp +=  2;
  uiSearchRange = 2;
  break;
default:
  iSrchRngHorLeft_tmp   += -2;
```

```
      iSrchRngHorRight_tmp  +=  2;
      iSrchRngVerTop_tmp    += -2;
      iSrchRngVerBottom_tmp +=  2;
      uiSearchRange = 2;
      break;
    }

    // 変えた探索範囲が元の探索範囲内かチェック
    if(iSrchRngHorLeft2 > iSrchRngHorLeft_tmp){
      iSrchRngHorLeft_tmp = iSrchRngHorLeft2;
    }
    if(iSrchRngHorRight2 < iSrchRngHorRight_tmp){
      iSrchRngHorRight_tmp = iSrchRngHorRight2;
    }
    if(iSrchRngVerTop2 > iSrchRngVerTop_tmp){
      iSrchRngVerTop_tmp = iSrchRngVerTop2;
    }
    if(iSrchRngVerBottom2 < iSrchRngVerBottom_tmp){
      iSrchRngVerBottom_tmp = iSrchRngVerBottom2;
    }
    iSrchRngHorLeft2 = iSrchRngHorLeft_tmp;
    iSrchRngHorRight2 = iSrchRngHorRight_tmp;
    iSrchRngVerTop2 = iSrchRngVerTop_tmp;
    iSrchRngVerBottom2 = iSrchRngVerBottom_tmp;
    pcMvSrchRngLT2->set(iSrchRngHorLeft_tmp,
    iSrchRngVerTop_tmp);
    pcMvSrchRngRB2->set(iSrchRngHorRight_tmp,
    iSrchRngVerBottom_tmp);

    // start refinement
    if ( bStarRefinementEnable &&
    cStruct.uiBestDistance > 0 )
    {
  // 順次拡大型８点ダイヤモンドあるいは８点スクエアサーチ
      while ( cStruct.uiBestDistance > 0 )
```

```
    {
      iStartX = cStruct.iBestX;
      iStartY = cStruct.iBestY;
      cStruct.uiBestDistance = 0;
      cStruct.ucPointNr = 0;
      iBestDist = 0;
      for ( iDist = 1; iDist < (Int)uiSearchRange + 1;
            iDist*=2 )
      {
        /*if(iDist > 16){
          m_cDistParam.iSubsampling = 4; //この場合 4 画素制
度 (4x4 画素の左上)
        } else */if(iDist > 4){
          m_cDistParam.iSubsampling = 2;   // どれだけ間引く
か。何ビット右シフトするのか。これの場合 2 画素精度 (2x2 画素の左上)
        } else {
          m_cDistParam.iSubsampling = 0;
        }
        xTZ8PointDiamondSearch_range ( pcPatternKey,
        cStruct, pcMvSrchRngLT2, pcMvSrchRngRB2, iStartX,
        iStartY, iDist, iBestDist, iPUX, iPUY, iPUWidth,
        iPUHeight, iRefPOC );
        if ( bStarRefinementStop && (cStruct.uiBestRound >=
        uiStarRefinementRounds) ) // stop criterion
        {
          break;
        }
      }
      m_cDistParam.iSubsampling = 0;
      // calculate only 2 missing points instead 8 points
      if cStrukt.uiBestDistance == 1  中心から距離 1 の場合、
2 点探索
      if ( cStruct.uiBestDistance == 1 )
      {
        cStruct.uiBestDistance = 0;
```

```
  if ( cStruct.ucPointNr != 0 )
  {
    xTZ2PointSearch( pcPatternKey, cStruct,
    pcMvSrchRngLT2, pcMvSrchRngRB2, iPUX, iPUY,
    iPUWidth, iPUHeight, iRefPOC );
  }
}

// 探索範囲の変更
iSrchRngHorLeft_tmp   = cStruct.iBestX;
iSrchRngHorRight_tmp  = cStruct.iBestX;
iSrchRngVerTop_tmp    = cStruct.iBestY;
iSrchRngVerBottom_tmp = cStruct.iBestY;

switch(iBestDist){
  case 64:
    iSrchRngHorLeft_tmp   += -32;
    iSrchRngHorRight_tmp  +=  32;
    iSrchRngVerTop_tmp    += -32;
    iSrchRngVerBottom_tmp +=  32;
    uiSearchRange = 32;
    break;
  case 32:
    iSrchRngHorLeft_tmp   += -16;
    iSrchRngHorRight_tmp  +=  16;
    iSrchRngVerTop_tmp    += -16;
    iSrchRngVerBottom_tmp +=  16;
    uiSearchRange = 16;
    break;
  case 16:
    iSrchRngHorLeft_tmp   += -8;
    iSrchRngHorRight_tmp  +=  8;
    iSrchRngVerTop_tmp    += -8;
    iSrchRngVerBottom_tmp +=  8;
    uiSearchRange = 8;
```

```
    break;
  case 8:
    iSrchRngHorLeft_tmp   += -4;
    iSrchRngHorRight_tmp  +=  4;
    iSrchRngVerTop_tmp    += -4;
    iSrchRngVerBottom_tmp +=  4;
    uiSearchRange = 4;
    break;
  case 4:
    iSrchRngHorLeft_tmp   += -2;
    iSrchRngHorRight_tmp  +=  2;
    iSrchRngVerTop_tmp    += -2;
    iSrchRngVerBottom_tmp +=  2;
    uiSearchRange = 2;
    break;
  default:
    iSrchRngHorLeft_tmp   += -2;
    iSrchRngHorRight_tmp  +=  2;
    iSrchRngVerTop_tmp    += -2;
    iSrchRngVerBottom_tmp +=  2;
    uiSearchRange = 2;
    break;
}

// 変えた探索範囲が元の探索範囲内かチェック
if(iSrchRngHorLeft2 > iSrchRngHorLeft_tmp){
  iSrchRngHorLeft_tmp = iSrchRngHorLeft2;
}
if(iSrchRngHorRight2 < iSrchRngHorRight_tmp){
  iSrchRngHorRight_tmp = iSrchRngHorRight2;
}
if(iSrchRngVerTop2 > iSrchRngVerTop_tmp){
  iSrchRngVerTop_tmp = iSrchRngVerTop2;
}
if(iSrchRngVerBottom2 < iSrchRngVerBottom_tmp){
```

```
          iSrchRngVerBottom_tmp = iSrchRngVerBottom2;
        }
        iSrchRngHorLeft2 = iSrchRngHorLeft_tmp;
        iSrchRngHorRight2 = iSrchRngHorRight_tmp;
        iSrchRngVerTop2 = iSrchRngVerTop_tmp;
        iSrchRngVerBottom2 = iSrchRngVerBottom_tmp;
        pcMvSrchRngLT2->set(iSrchRngHorLeft_tmp,
        iSrchRngVerTop_tmp);
        pcMvSrchRngRB2->set(iSrchRngHorRight_tmp,
        iSrchRngVerBottom_tmp);
      }
    }
  }

  // write out best match
  rcMv.set( cStruct.iBestX, cStruct.iBestY );
  ruiSAD = cStruct.uiBestSad - m_pcRdCost->getCost(
  cStruct.iBestX, cStruct.iBestY );
  delete pcMvSrchRngLT2;
  delete pcMvSrchRngRB2;
}
```