卒業論文

題目

汎用プロセッサ向き斜交軸変換高速化機構に関する研究

指導教員

近藤 利夫 教授

2017年

三重大学 工学部 情報工学科
コンピュータアーキテクチャ研究室

近藤　亮(413820)

内容梗概

近年 ITS(高度道路交通システム) の発展の中で，車載カメラで周囲の道路状況を認識する，安全運転支援用のステレオ画像処理システムの研究が盛んに行われている．この技術を利用する際の前処理の 1 つとしてアフィン変換が用いられている．しかし，アフィン変換は不連続なメモリアクセスが必要となるため並列アクセスが機能し難く，容易に高速化できない問題点がある．当研究室ではこの問題の解決に向けて，アフィン変換の代替方法として並列処理が可能な斜交軸変換を提案しており，ステレオマッチングによる距離検出に対する精度要求をほぼ満たせることを示してきた．また，汎用プロセッサの SIMD 命令は斜交軸変換の高速化に有効ながら改善の余地が残されていた．

そこで本研究では，斜交軸変換の更なる高速化を可能とする新たな SIMD 命令の仕様を提案し，それをサポートするデータパスまで設計した．このデータパスを用いる場合と既存の x86 プロセッサを用いる場合について，それぞれの幾何学変換のアセンブリ言語プログラムを示すことで，ステップ数が半分以下に低減されることを明らかにした．

Abstract

Recently, in the development of ITS(intelligent transport system),an active research of stereo image processing system for vehicle safe driving support that can recognize road condition by on-board camera has been carried out. Affine transformation is used as one of the preprocessing for using this technology. However, there is a problem that the high-speed processing cannot be realized because Affine transformation needs discontinuous memory accesses and this parallel data access function is unsupported in current computer. To solve this probrem, this laboratory have proposed the oblique axis transformation that can perform a parallel access function to Affine transformation and show that it can almost satisfy accuracy requirement for measuring headway distances by stereo matching. Moreover SIMD instructions of general-purpose processor needs to be improved although it is effective for speeding up of the oblique axis transformation.

The research propose new SIMD instructions that can further speeding up the oblique axis transmission, and designed a data path to support these new SIMD instructions. Compared with the current x86 processor, we showed that the number of steps is reduced to less than half by showing the assembly language program of each geometric transformation.

目 次

図 目 次

表 目 次

1　まえがき

1.1　背景

近年 ITS(高度道路交通システム) の開発，発展が進んでいる．その中で，画像認識技術を適用する試みが盛んに行われており，この技術を利用して距離を検出し自動でブレーキをかけるシステムや，障害物を抽出する技術が実用化されている．このシステムには，ステレオ画像処理の技術が採用されており，さらに高度な運転システムの実現にはステレオ画像処理の高速化が必要不可欠である．しかし，キャプチャ画像の変換で使用されるアフィン変換では，不連続なメモリアクセスが必要になり並列アクセスに不向きであるため，高速化が困難という問題がある．

そこで，当研究室ではアフィン変換に代わる変換方法として並列処理が可能な斜交軸変換を提案している．汎用プロセッサの SIMD（Single Introduction Multiple Data) 命令は斜交軸変換の高速化に有効ながら, 改善の余地が残されている．

1.2 研究の目的

本研究では，整数精度での斜交軸変換の高効率実行を可能とする新たなSIMD命令の仕様を明らかにし, それをサポートするデータパスまで設計し，先行研究で使用されていた既存のSIMD命令のみの場合と斜交軸変換のステップ数を比較し, 新SIMD命令の有効性を示す．また, これらの手法が車載カメラの画像処理方法として適切なものであることを示す.

2　ステレオ画像処理による距離算出

2.1　ステレオ画像処理

ステレオ画像処理とは同一の対象物を異なる２つの視点から観測し, 両撮影面上にある対象物の位置のずれによって, その対象物の位置を測る方法である. 距離計測の前提として, 前方障害物を検知する必要があり, それは道路平面上に存在する高さを持つ物体で，画像変換を行った際に生ずるずれを認識することにより行われる. 距離計測はその画像を元に行う. また今回利用するのは平面投影ステレオ法と呼ばれる画像変換である. これは左画像中に存在するすべての点が道路面と同じ高さを持つと仮定し, 左画像を右画像へ逆投影する. この逆投影画像と実際の右画像との間の差を求め, その差分値が大きい領域を障害物であると判断する手法である.

2.2 ステレオ画像による距離計算

ステレオ画像処理による前方障害物の距離検出の手順は

画像入力 → 画像変換 → 視差算出 → 距離算出

であり, この中で最も処理時間を必要とするのは差分絶対値和（SAD;Sum of Absolute Differences）を利用した視差検出である. 一般的にこの処理の演算量を削減するために前処理としてステレオ画像の補正処理を行う. ステレオ画像は通常, 探索ラインが走査線と一致するのが理想であるが, 実際に二つのカメラをそのように配置するのは事実上不可能である. この画像処理は入力画像の探索ラインを同一水平線上に乗るよう補正するもので, ステレオ画像の平行化と呼ばれ幾何学変換を施すことによって実現される.

2.3　算術式の導出

ステレオ画像処理の簡単な原理は三角測量である．対象物がカメラ間の中心線上にあるとする．対象物と２台のカメラを線で結ぶと三角形が出来，これを利用して計測をする．距離算出式の導出を以下に示す．

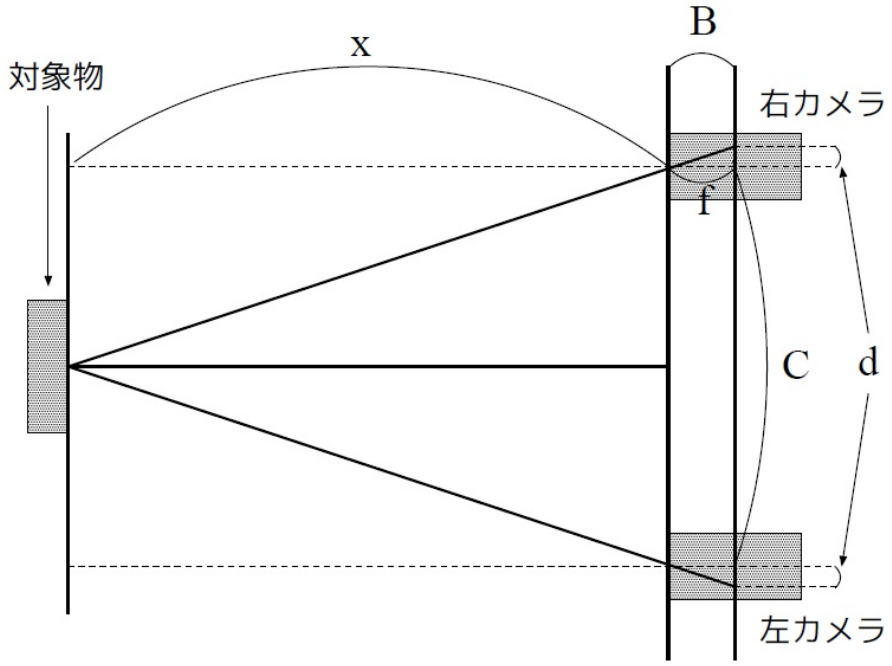


図 2.1: 三角測量

x：レンズから対象物までの距離　f：焦点距離　d：視差
C：カメラ間の距離　B：レンズから撮影面までの距離

$$\text{図 2.1 より } C : x = d : B, \quad \text{これより} \quad d = \frac{CB}{x} \tag{1}$$

$$\text{焦点の公式より} \frac{1}{x} + \frac{1}{B} = \frac{1}{f}, \quad \text{これより } B = \frac{fx}{x - f} \tag{2}$$

$$\text{式 (1)(11) より } d = \frac{fC}{x - f}, \quad \text{これより } x - f = \frac{fC}{d} \tag{3}$$

2.4　アフィン変換とその問題点

アフィン変換は，幾何学変換の一つであり，ユークリッド幾何学的な線形変換と平行移動の組み合わせによる図形や形状の移動，変換方式である．幾何学的性質が保たれるため高精度な距離算出が可能である．特に，平面物体とそれを任意の位置から撮影した画像との対応に適している．あらかじめ変換元と変換後の対応点 4 組があれば，次の行列式よりその座標値から一意の変換パラメータを算出することができる．

$$\begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \\ Y_1 \\ Y_2 \\ Y_3 \\ Y_4 \end{bmatrix} = \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -X_1x_1 & -X_1y_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -X_2x_2 & -X_2y_2 \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -X_3x_3 & -X_3y_3 \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -X_4x_4 & -X_4y_4 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -Y_1x_1 & -Y_1y_1 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -Y_2x_2 & -Y_2y_2 \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -Y_3x_3 & -Y_3y_3 \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -Y_4x_4 & -Y_4y_4 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \\ a_8 \end{bmatrix}$$

※ (X_i,Y_i) は変換元の座標,(x_i,y_i) は変換後の座標

この行列式によって求められた $a_1,\ldots,a_8$ のパラメータを以下の式 (4)(5) に代入することによりアドレス計算を行う．

$$u = \frac{ia_1 + ja_2 + a_3}{ia_7 + ja_8 + 1} \tag{4}$$

$$v = \frac{ia_4 + ja_5 + a_6}{ia_7 + ja_8 + 1} \tag{5}$$

※ (i,j) は変換前のアドレス,(u,v) は変換後のアドレス

車載ステレオ画像処理の場合, カメラのキャリブレーションから変換パラ

メータをあらかじめ求めておくことができるが, これらの式の示す用に, 毎回のアドレス計算による不連続なメモリアクセスが必要となるため処理上のボトルネックとなっている. しかし, 画像に対しランダム走査を行ってしまうので,SIMD 等の並列アクセスへの適用は難しく高速化が困難であるとされている.

3　斜交軸変換と高速化

3.1　斜交軸変換

当研究室では、すでに、SIMD 並列処理に適した幾何学変換処理として斜交軸変換が研究されている．斜交軸変換とは, x 軸, y 軸を求めたパラメータを元に画像を傾ける変換である. 変換後の画像の傾きは変換元と変換後の 2 組によって求められる. 以下変換元の座標を (x_i,y_i), 変換後の座標 (X_i,Y_i) とする.X 座標の傾き A, y 座標の傾き B とすると A,B は以下の 2 式 (6)(7) によって求められる.

$$A = \frac{(x_2 - x_1) - (X_2 - X_1)}{Y_1 - Y_2} \tag{6}$$

$$B = \frac{(y_2 - y_1) - (Y_2 - Y_1)}{X_1 - X_2} \tag{7}$$

変換した入力画像と基準画像の対応点のずれを補正する必要があるため, 移動量の計算をする. 左右のずれを δx, 上下のずれを δy とする. 2 つのパラメータは次の 2 つの式 (8)(9) によって算出される.

$$\delta x = A(height - y_i) - (x_i - X_i) \tag{8}$$

$$\delta y = (Y_i - y_i) - BX_i \tag{9}$$

なお、変換元の座標 (xi,yi) と, 変換後の座標 (Xi,Yi) の決定は、自動化が実現しておらず、本研究の対象ではないため、目視で行った。

3.1.1 x 軸方向への傾き

ここでは式 (6) の説明をする.A とは, 左画像の任意の 2 点を結ぶ直線m と右画像の対応する 2 点を結ぶ直線 n の同じ高さでの x 座標の移動量を表す. 実際は図 (3.5) のように綺麗に 2 直線が並ぶことはないが, 2 直線がどの位置に存在していても「x 軸方向への増加量」とその差は計算が可能であることを利用している.

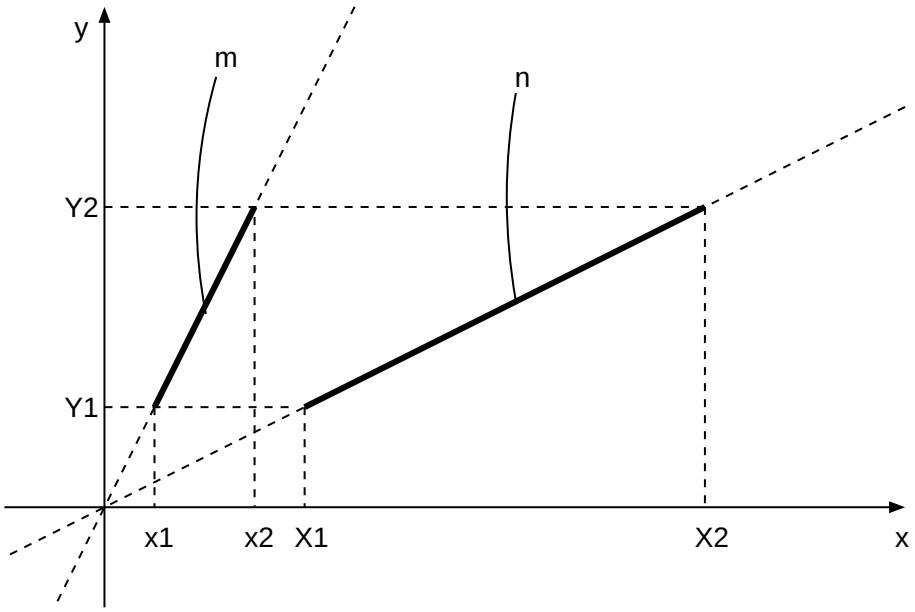


図 3.2: 水平方向への傾き

直線m上にある 2 点を $(x_1,Y_1)(x_2,Y_2)$, 直線 n 上にある 2 点を $(X_1,Y_1)(X_2,Y_2)$ をとる. すると 2 直線は

$$m: \quad x = \frac{(x_2 - x_1)}{(Y_2 - Y_1)}y \qquad n: \quad x = \frac{(X_2 - X_1)}{(Y_2 - Y_1)}y \tag{10}$$

と表せる. これより n 上の点 $(X_i,\, y_j)$ と m 上の点 $(x_i,\, y_j)$ の関係は

$$n - m : \quad X_i - x_i = \frac{(X_2 - X_1) - (x_2 - x_1)}{(Y_2 - Y_1)} y_j \tag{11}$$

式 (6) を代入すると

$$X_i = A y_j + x_i \tag{12}$$

式（12) は, 直線 n の x の増加量は直線 m の x の増加量に A を加えることで表せることを示す.

3.1.2　y 軸方向への傾き

式 (7) も同様にして求めることができる.B とは Y 座標の移動量を表す. 式（(6) の時はｙの増加量を固定したが, 今回はｘの増加量を固定することによって求める. 直線m上にある 2 点を $(X_1,y_1)(X_2,y_2)$, 直線ｎ上にある 2 点を $(X_1,Y_1)(X_2,Y_2)$ をとる. すると 2 直線は

$$m: \quad x = \frac{(y_2 - y_1)}{(X_2 - X_1)}y \qquad n: \quad x = \frac{(Y_2 - Y_1)}{(X_2 - X_1)}y \tag{13}$$

と表せる. これより n 上の点 $(X_i,\ Y_j)$ と m 上の点 $(X_i,\ Y_j)$ の関係は

$$n - m: \quad Y_j - y_j = \frac{(Y_2 - Y_1) - (y_2 - y_1)}{(X_2 - X_1)}x_i \tag{14}$$

式 (7) を代入すると

$$Y_i = Bx_j + y_i \tag{15}$$

式（(20) は, 直線ｎの y の増加量は直線mの y の増加量に B を加えることで表せることを示す.

3.1.3　y 軸方向への誤差補正

式 (9) の説明をする.δy とは左右の画像の対応点の y 座標の差である.

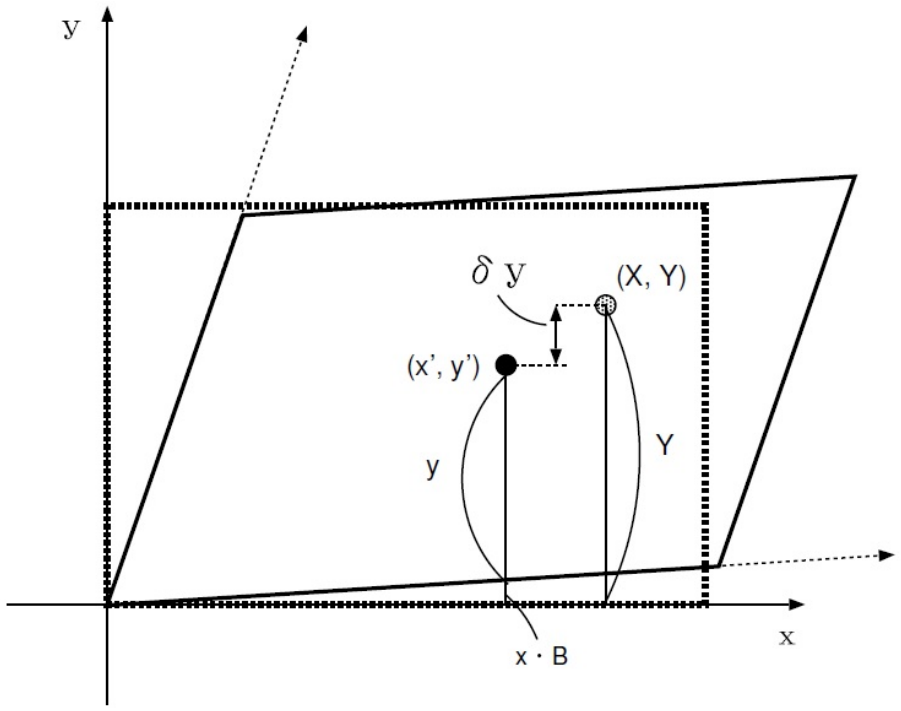


図 3.3: 垂直方向のずれ算出

図 (3.3) に示すように, 左画像の任意の点を (x,y), 画像に B の式を適応した後の左画像の点を (x', y'), 右画像の点を (X,Y) とすると

$$y' = y + Bx \tag{16}$$

$$Y = \delta y + y' \tag{17}$$

となり, 式 (16)(17) より δy が算出される.

3.1.4 x 軸方向への誤差補正

式 (8) の説明をする.δx とは左右の画像の対応点の x 座標の差である.

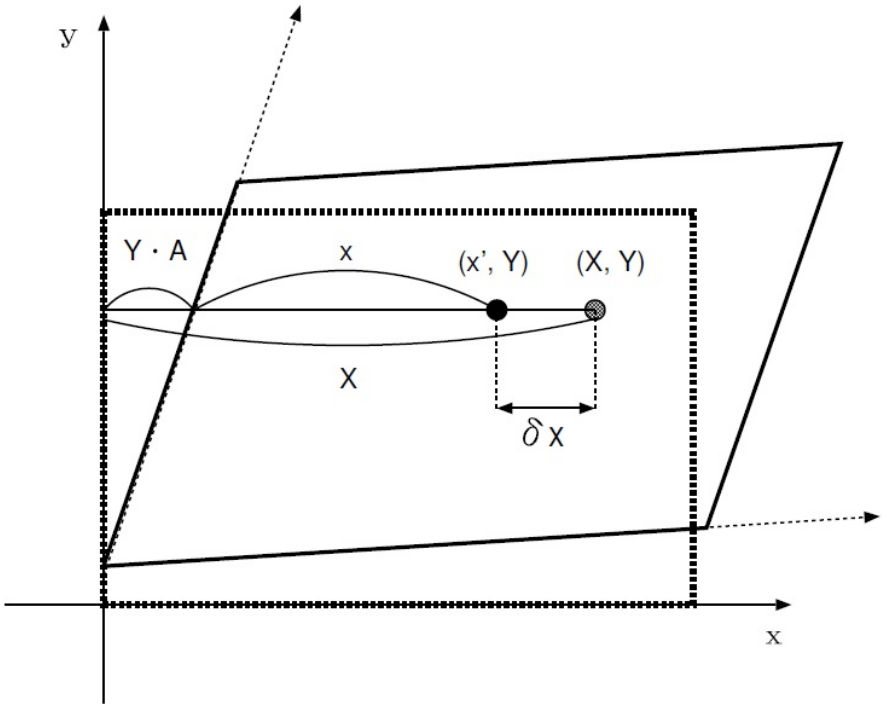


図 3.4: 水平方向のずれ算出

前章で Y 成分の位置を合わせた. 次は X 成分の位置を合わせる処理をする. 図 (3.4) に示すように. 左画像の任意の点を (x,y), 画像に B の式を適応した後の左画像の点を (x', Y), 右画像の点を (X,Y) とすると

$$x' = x + yA \tag{18}$$

$$X = \delta x + x' \tag{19}$$

となり, 式 (18)(19) より δx が算出される.

3.2 並列処理による高速化

前述の通り, 射影変換では性質上, 画素単位でメモリを読み出しする必要があった. 本稿の斜交軸変換では, ラスタ走査可能な 2 つの斜交軸変換を組み合わせることで画像変換を行う.

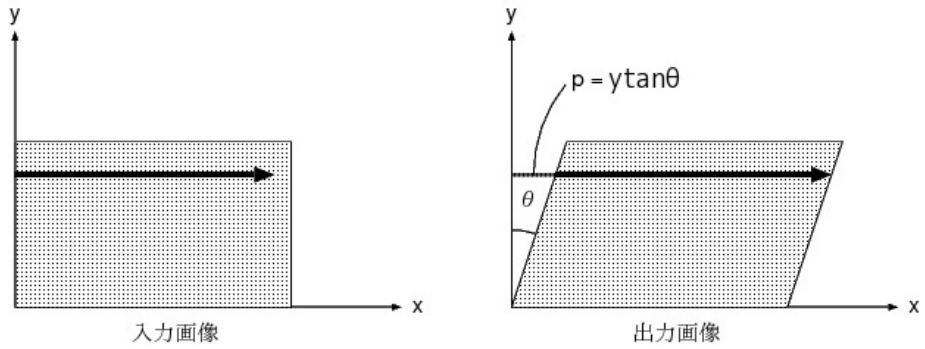


図 3.5: x 軸に平行な直線の変換

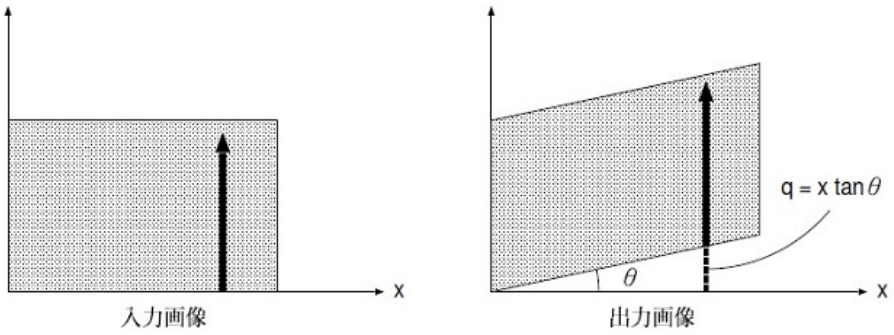


図 3.6: y 軸に平行な直線の変換

図 3.5,3.6 の様に, x 軸と平行な直線を水平方向に p 画素移動する変換を行う. 同様に y 軸と平行な直線を垂直方向に q 画素移動する変換を行う.

したがって, 入力画像も変換画像も水平方向への変換については複数の画素を一括してメモリアクセスすることが可能となる. 垂直方向に関してもメモリアクセスが容易になる. このようにデータを一括処理することが可能なことから, 並列処理を活かすことで, 画像変換において高速化が可能である.

3.3 先行研究及びハードウェアへの適応

先行研究では，ソフトウェア上での斜交軸変換が提案されていた．アフィン変換に比べ処理時間を高速化することは実現されていた．しかし先行研究のソフトウェアでは，128 × 128 画素の画像のみしか対応しておらず，解像度の高い画像に対しての変換ができていない問題があった．そこで，当研究室考案の SIMD データパスに斜交軸変換が処理できるような回路を組み込み，さらなる高速化としてステップ数の削減を目指すと同時にソフトウェアでは対応しきれなかった高解像度の画像に対しての処理ができることを示す.

3.4 x86 プロセッサ SIMD 命令の問題点

x86 プロセッサにもバイト単位でシフト演算するような命令は存在する. しかし, シフトにより空いたバイトはゼロクリアされるため斜交軸変換の処理で必要なシフトによりはみ出る画素データのの埋め込みにうまく対応できない問題点がある．また,2 つのレジスタを連結させてシフトを行う命令でもレーン境界を越えてシフトされることがないため, 斜交軸変換の高速化には適していない．したがって，斜交軸変換の高速化のためには，シフトによりはみ出る画素データに対応した新たな命令が必要となる.

4　提案手法

4.1　方針

斜交軸変換をソフトウェアで高速化する場合，誤差精度の問題が発生してしまうため高速化に限界がある．そこで，斜交軸変換をハードウェアで高速化することを目指す．当研究室考案の既存のデータパスは，64ビット命令を上位と下位の 2 命令に分割し，スカラ演算部とベクトル演算部を同時に処理できるような複数命令発行方式である．データパスは7段パイプライン構成になっており，動画像符号化の高速化のために使用されていたため，データパス内に SAD 演算の機能が組み込まれている．また, 動画像符号化計算に有効活用するために，4×8 の二次元データに対するロードストアを可能にするキャッシュが提案されている．本研究では，当研究室考案のデータパスにすでに設計されている，スカラレジスタとベクトルレジスタを使い分けることにより効率よく処理を進めることを提案する．256 ビット幅であるベクトルレジスタの利点を生かし，解像度の高い画像での処理にも対応する．

4.2 新 SIMD 命令の提案

高速化実現のために，画素単位でレジスタデータを処理できる機能を追加した．変換処理がベクトルレジスタの方で行われるように命令を設定し，新規追加した演算器へ処理信号が送られるように設計した. 従来手法では，与えられたパラメータに基づいた値を逐次的にシフトしていたため，ステップ数が増大していた．そこで，当研究室考案のデータパスにおいてキャッシュへのメモリアクセスが 8 画素単位であることを有効活用し，レジスタ内の画素データをキャッシュへ書き戻す位置を適切な位置までずらす工夫を行った．これらの工夫を施すことにより，シフト量は最大 7 画素になる. また，従来手法ではシフトによってはみ出る画素データに関して考慮されていなかったが，新規命令案においてはみ出る画素データを再び演算器に書き戻すことによりデータが欠落することを防ぎ，なおかつステップ数が増大しないように設計した.

4.3 追加モジュール

新 SIMD 命令を実現するため，新規モジュールの追加を行った．当研究室考案の SIMD 併用型データパスでは複数命令発行方式をとっており，上段のスカラ演算部分と下段のベクトル演算部分を別々で実行することが可能となっている．今回設計した SHIFT_ALU というモジュールは，ベクトル演算部分に設計した．また，既存のモジュールである SHIFT_ALU というモジュールと同じパイプライン上に設計した．SHIFT_ALU に入力されるレジスタデータは 1 つであり，その他の入力値は 4 ビット幅のオペコード値と 6 ビット幅の funct 値である．出力されるでレジスタデータは 2 つあり，1 つはキャッシュに書き戻すためのデータであり，もう 1 つはシフトによりはみ出た画素データを格納したものであり再び SHIFT_ALU に入力される．

4.4 ブロック図

提案したデータパスの構成図を図4.7に示す．新規追加したSHIFT_ALUは点線の部分である．命令が発行されると，キャッシュから読みだされた画素データがベクトルレジスタからSHIFT_ALUに格納される．SHIFT_ALU内ではパラメータ量に応じて演算が進められる．出力データは2つあり，1つはキャッシュに書き戻す画素データでもう一つはシフトによりはみ出た画素データでありSHIFT_ALUの中に書き戻される．

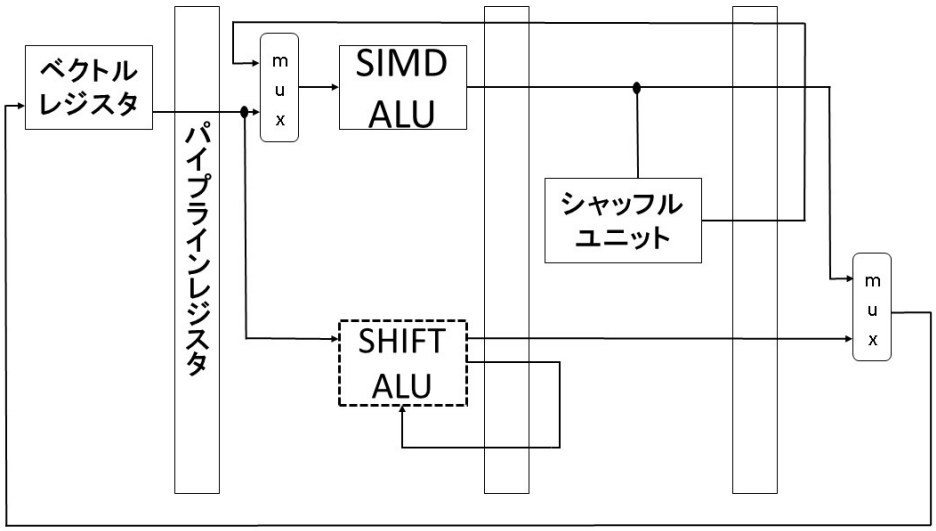


図 4.7: 提案データパス構成

5　性能評価

5.1　ステップ数評価

提案手法の命令セットで斜交軸変換を実行したときのステップ数と，東芝製画像認識プロセッサ「Visconti」に内蔵されている専用ハードウェアでアフィン変換を実行したときのステップ数を比較し評価を行う．また，既存の x86 プロセッサ SIMD 拡張命令セットである SSE でのみで斜交軸変換を実行したときのステップ数とも比較を行う. 評価に使用する画像サイズは 640 × 480 画素であり，画像を傾けるためのパラメータは予め求めてあるものとし，画像変換に関わるステップ数のみを評価対象とした.

5.2　考察

1 行あたりの画像変換処理にかかるステップ数の比較グラフを図 5.8 に示す．一番左が既存の x86 拡張命令セットである SSE のみで実行したもので，1 行あたり 390 ステップかかっている．中央は東芝製画像認識プロセッサ「Visconti」で実行したものでありステップ数は 240 ステップである．一番右が提案手法を用いて実行したものであり，ステップ数は 110 ステップであった．この結果から，提案手法は SSE と比較してステップ数を約 72%低減できた．また「Visconti」と比較してもステップ数を約 54%低

減することができた．理由としては，既存の命令では定義されていなかった，シフトによってはみ出るデータに対して対応できた点が大きいと考えられる．よって，高速化の観点から斜交軸変換はアフィン変換よりも優れていることがいえる．

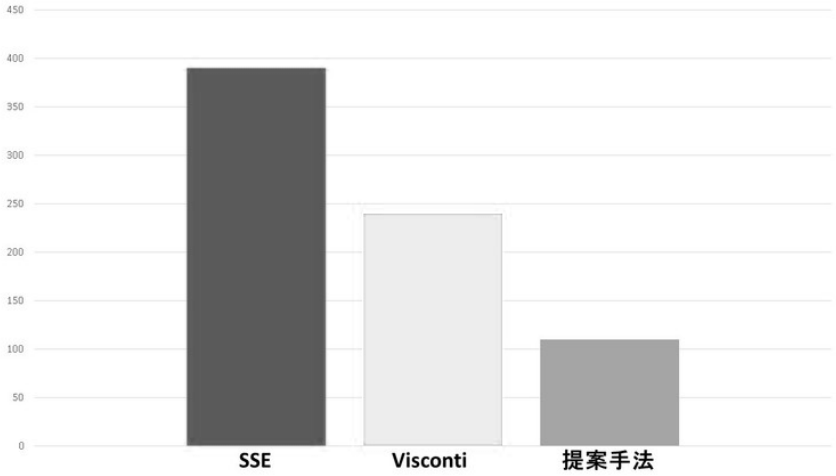


図 5.8: 1 行あたりのシフト処理にかかるステップ数

6 関連研究

6.1 差分絶対値和

差分絶対値和（SAD;Sum of Absolute Differences) とは同じ大きさの二枚の画像を比較するとき, 対応する各画素について二枚の画像間の差分を取り, その絶対値を合計したもの. この数値が小さいほど, 二つの画像は類似していることを示す.

二枚の画像間の m×n 画素探索ウィンドウの SAD は以下の式 (20) によって求められる.

$$SAD = \sum_{i=-m/2}^{m/2} \sum_{j=-n/2}^{n/2} (|A(x_i, y_j) - B(x_i, y_j)|) \tag{20}$$

6.2 SIMD 演算

SIMD(Single Instruction Mlutiple Data) とは一つの命令で複数のデータに対して処理を行う命令セットである. 汎用命令では加算等の命令を実行する際, 6.1 に示すようにソースおよびデスティネーションの２つのオペランドを取り, 命令を実行しその結果をデスティネーションオペランドに格納する. ２つのオペランドを必要とするが, 解は１つしか得られない. それに対し,SIMD 命令ではソース及びデスティネーションに複数のデータを与え, それを一つの命令でまとめて実行する.6.2 の場合一つの命令で

汎用レジスタ（Destination）
D
+
汎用レジスタ（Source）
S
↓
汎用レジスタ（Destination）
D + S

表 6.1: 汎用命令

８つの演算を実行し, デスティネーション格納している. 独立した複数のデータから求めた演算の解であり, 一つの命令で８つの解が求められたことになる.SIMD は単位時間あたりのデータ処理量を増加させるというアプローチで並列化を実現している.

SIMD レジスタ（Destination）							
D7	D6	D5	D4	D3	D2	D1	D0
+							
SIMD レジスタ（Source）							
S7	S6	S5	S4	S3	S2	S1	S0
↓							
SIMD レジスタ（Destination）							
D7+S7	D6+S6	D5+S5	D4+S4	D3+S3	D2+S2	D1+S1	D0+S0

表 6.2: SIMD 命令

7　あとがき

7.1　まとめ

本研究では，アフィン変換の代替方法である斜交軸変換の高速化に対応するハードウェア構成を検討し，はみ出た画素データを後続の命令で取り組むような新規SHIFT命令を提案した．結果として，従来のSIMD命令のみの使用に比べステップ数が約72 %低減されることを示した．また，市販の車載向け画像認識プロセッサのアフィン変換実行時と比較しても，ステップ数が約54 %低減されることを示した．

7.2　今後の課題

今後の課題として，本研究では面積評価を行っていない．そこで今後は，SIMDデータパスのサイクルタイムや実装面積の増加につながらないか，レイアウト設計まで行って評価していく必要がある．また，斜交軸変換の距離算出精度に関しては，アフィン変換に比べて劣ってしまっている．先行研究の結果から，アフィン変換の距離誤差率が平均4.3 %であるのに対し斜交軸変換は平均6.2 %である．自動車の更なる安全向上のために距離算出精度の誤差低減の観点からも研究を進めていく必要がある．さらにこれまでの研究では，変化量を求めるための対応点を道路上の白

線としていた．しかし，道路上に白線が常に存在しているとは限らない．そのため，距離算出を行う際には常に存在している障害物を対応点として検出することで安定した結果が得られるのではないかと考えられる．

謝辞

本研究を行うにあたり，御指導，御助言頂きました近藤利夫教授，並びに多くの助言を頂きました佐々木敬泰助教，深澤祐樹研究員に深く感謝いたします．また，様々な局面にてお世話になりました研究室の皆様にも心より感謝致します．

参考文献

[1] 水野貴誠，SIMD プロセッサに適した車載キャプチャ画像変換法の研究，三重大学卒業論文，2010 年．

[2] 河合恵奈，斜交軸変換によるステレオ画像処理の高速化の研究，三重大学卒業論文，2011 年．

[3] 深田幸宏，SIMD プロセッサを用いた斜交軸変換の高速化の研究，三重大学卒業論文，2012 年．

[4] 渡邊敬太，高効率動き検出に対応する SIMD 併用型汎用データパス構成法の研究，三重大学大学院修士論文，2015 年.

[5] 中井，田辺，古川，小坂谷，宮森，谷口，宮本，前田，画像認識プロセッサ Visconti と，その安心・安全への適用事例，情報処理学会論文誌：コンピュータビジョンとイメージメディア，Vol. 48，No. SIG 1(CVIM 17)，pp.1–11，2007 年 2 月

[6] 小室，鏡，石川，片山，超並列画像プロセッサのためのビットレベルコンパイラ，情報処理学会論文誌，コンピューティングシステム，Vol.48，No.SIG 13(ACS19)，pp.106–116，2007 年 8 月

[7] 松井正貴，システムオンチップの普及と MeP，東芝レビュー，Vol.58，No.5，pp.2–8，2003 年

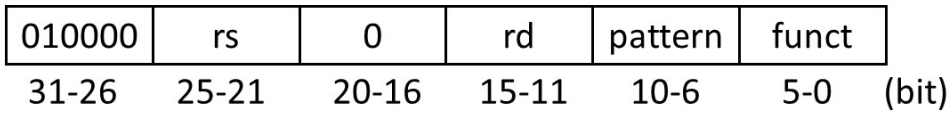
[8] 近藤勝久，メディアプロセッサを用いたスマートカー向け画像認識 LSI，東芝レビュー，Vol.56，No.8，pp.58–61，2001 年

[9] 宮本，谷口，宮森，車載画像処理システム LSI による衝突防止システム，東芝レビュー，Vol.58，No.12，pp.54–57，2003 年

A　NEW_SHIFT 命令

NEW_SHIFT reg1,reg2,imm

NEW_SHIFT



斜交軸変換実現のために新規追加した NEW_SHIFT 命令について記載する．パラメータ量に応じて画素データをシフトする命令である．funct フィールドで求められたシフト量を指定し，rs フィールドで指定した画素データをシフトする．そして，塩山結果を rd フィールドで指定したレジスタに格納することが可能である．また，pattern フィールドで右シフトか左シフトか指定できるように設計した．

B　追加モジュールの Verilog

新規追加したモジュールである SHIFT_ALU 内に記述した Verilog コードを記載する.

```
module SHIFT_ALU( src1,funct,op,knd,res2,res1);
   input   [255:0]  src1;
   input   [5:0]    funct;
   input   [3:0]    op;
   input   [255:0]   knd;
   output  [255:0]   res2;
   output  [255:0]  res1;
   wire    [255:0]  tmp;
   wire    [255:0]  tmp2;
assign tmp = f_alu(src1, funct,op);
assign tmp2 = new_alu(src1,funct,op);
assign res1 = result_alu(tmp2,knd,op);
assign res2 = ex_alu(tmp,op);
```

```
function [7:0] abs_o;
   input [7:0] current;
   input [7:0] reference;
   logic [7:0] a,b;
   logic  s;

   assign a = 8'hff ^ current;
   assign {s,b} = a + 1'h1 + reference + 9'h100;
   assign abs_o = (s) ? (8'hff ^ b) + 1 : b;
endfunction

function [255:0] result_alu;
  input [3:0] op;
  input [255:0] tmp2;
  input [255:0]  knd;

  case(op)
    4'b1010:
```

```
    result_alu = tmp2 + knd;


endcase

endfunction


function [255:0] ex_alu;

  input [3:0] op;

  input [255:0] tmp;


  case(op)

    4'b1010:

        if(funct == 6'b001000)begin

      ex_alu = tmp << 248;

      end

    else if(funct == 6'b010000)begin

      ex_alu = tmp << 240;

    end
```

```
    else if(funct == 6'b011000)begin
      ex_alu = tmp << 232;
    end
    else if(6'b100000)begin
      ex_alu = tmp << 224;
    end
    else if(6'b101000)begin
      ex_alu = tmp << 216;
    end
    else if(6'b110000)begin
      ex_alu = tmp << 208;
    end
    else if(6'b111000)begin
      ex_alu = tmp << 200;
    end
endcase
endfunction
```

```
function [255:0] new_alu;

input [255:0] src1;

input [5:0] funct;

input [3:0] op;


case(op)

  4'b1010:

  if(funct == 6'b001000)begin

  new_alu = src1[7:0];

  end

else if(funct == 6'b010000)begin

  new_alu = src1[15:0];

end

else if(funct == 6'b011000)begin

   new_alu = src1[23:0];

end

else if(6'b100000)begin

  new_alu = src1[31:0];
```

```
 end
 else if(6'b101000)begin
   new_alu = src1[39:0];
 end
 else if(6'b110000)begin
   new_alu = src1[47:0];
 end
 else if(6'b111000)begin
   new_alu = src1[55:0];
 end
 endcase
 endfunction


function [255:0] f_alu;
  input [255:0] src1;
  input [5:0] funct;
  input [3:0] op;
```

```
case(op)
  4'b1010:
  if(funct == 6'b001000)begin
    f_alu = src1 >> 8;
  end
  else if(funct == 6'b010000)begin
    f_alu = src1 >> 16;
  end
  else if(funct == 6'b011000)begin
    f_alu = src1 >> 24;
  end
  else if(funct == 6'b100000)begin
    f_alu = src1 >> 32;
  end
  else if(funct == 6'b101000)begin
    f_alu = src1 >> 40;
  end
  else if(funct == 6'b110000)begin
```

```
      f_alu = src1 >> 48;
    end
    else if(funct == 6'b111000)begin
      f_alu = src1 >> 56;
    end


 endcase
endfunction
```