汎用グラフィクスハードウェアを用いた 2次元/3次元剛体位置合わせの高速化

五味田 遵[†,☆] 合田 圭吾[††] 川崎 康博[††]
伊野 文彦[††] 萩原 兼一[††]

画像の位置合わせとは，撮影方法・時刻の異なる画像の組に対し，画像中の被写体の位置を対応付ける技術である．この技術は手術支援の分野において重要である．しかし，計算量が多いため CPU を用いた実装では位置合わせに 10 数分を要し，時間制約の強い手術支援の用途に対してはさらなる高速化が必要である．一方，PC 用グラフィクスカードが搭載する GPU（Graphics Processing Unit）は近年著しく性能が向上している．そこで本研究は，GPU を位置合わせに用いることでその実行時間の短縮を目指す．本稿では，2 次元/3 次元剛体位置合わせを構成する主要な 3 つの処理（3 次元画像の投影像生成，近傍フィルタ，集約演算）を GPU 上に実装し，その性能を測定することで GPU を用いた位置合わせの有用性を検証する．評価実験の結果，GPU を用いる手法により 10 秒程度の実行時間で位置合わせを行えた．PC クラスタを用いた実装との処理時間の比較においても，優位性を確認した．また，位置合わせ精度に関して，位置合わせ前の正解に対するずれが 10 mm 程度であるとき，ほぼすべての場合で正解に近い解を得た．

Accelerating 2-D/3-D Rigid Registration Using Commodity Graphics Hardware

Jun Gomita,[†,☆] Keigo Goda,[††] Yasuhiro Kawasaki,[††]
Fumihiko Ino[††] and Kenichi Hagihara[††]

Image registration is a technique for finding point correspondences between two different images taken at different times and/or in different modalities. This technique plays an important role in computer-aided surgery. However, CPU implementations take more than 10 minutes to complete a registration task due to a large amount of computation. Therefore, some acceleration techniques are required to use this technique for surgical assistances, where response time is strictly limited in a short time. One acceleration technique is to use graphics processing units (GPUs) equipped on PC graphics cards, which are rapidly increasing performance. The objective of our work is to reduce registration time by using GPUs. This paper presents a fast 2-D/3-D rigid registration method capable of accelerating three key procedures on the GPU: volume projection, neighbor filtering, and reduction operation. We investigate the usability of our method in terms of registration time and accuracy. The experimental results show that our GPU-based method successfully completes a registration task in about 10 seconds, demonstrating shorter registration time than a cluster-based method. Furthermore, we obtain successful alignments for almost all of registration tasks if the initial registration error is less than 10 mm.

1. はじめに

2 次元/3 次元剛体位置合わせ[1),2)] とは，異なる条件下で同一の物体を撮影した 2 次元画像 I_F および 3 次元画像 V に対し，I_F の座標系に V の座標系を対応付ける，すなわち I_F が V の投影であると見なし，その投影面に対する V の位置姿勢を求める技術である．この技術は，画像処理により外科手術を支援する際に基礎的な役割を果たすため，重要である．たとえば，大腿骨の骨折を整復する手術では，術中に撮

† 大阪大学基礎工学部情報科学科
Department of Informatics and Mathematical Science, School of Engineering Science, Osaka University
†† 大阪大学大学院情報科学研究科
Graduate School of Information Science and Technology, Osaka University
☆ 現在，東京大学大学院情報理工学系研究科
Presently with Graduate School of Information Science and Technology, The University of Tokyo

影した 2 次元 X 線像に術前の 3 次元 CT 像を位置合わせすることで骨片の位置姿勢を取得し，骨片を正確な位置に固定することを支援する．しかし，位置合わせのための計算量が多いため，短い応答時間を必要とする手術支援においてはその高速化が必要である．

このような時間制約の強い用途に対し，PC クラスタを用いる並列計算により位置合わせを高速化する手法[3)]がある．この手法では，CPU で逐次実行する場合に約 17 分を要する位置合わせを 128 個の CPU を用いる並列計算により 34 秒に短縮している．しかし，この手法を医療現場で用いる場合，PC クラスタの導入や維持管理にかかるコストなどが課題となる．

一方，3 次元グラフィクスの高速処理を目的とする GPU（Graphics Processing Unit）[4)] は近年著しい性能向上をとげており，単精度ではあるものの，CPU に対して 6 倍以上の浮動小数点演算性能（FLOPS）を達成している[5)]．さらに，プログラム可能な GPU が普及するにつれ，従来の描画用途だけでなく，汎用計算への応用が注目されている．応用例としては，LU 分解[6)] や高速フーリエ変換[7)]，本研究にも関連するが，画像間の類似度計算[8)]（後述）などがある．

GPU は市販の PC にも搭載されているため，導入が容易である．GPU を汎用計算に用いることで，従来 PC クラスタを用いて高速処理していた大規模な計算を，1 台の PC によって，同程度の実行時間で行える可能性がある．

そこで本研究では，CPU に比べて高速であり，また PC クラスタに比べて導入の容易な GPU を用いて，2 次元/3 次元剛体位置合わせの高速化を目指す．提案手法は，文献 3) の位置合わせ手法を基に，下記にあげる主要な 3 つの処理を GPU 上に実装する．

(1) ボリュームレンダリングによるデジタル再構成 X 線画像（DRR: Digitally Reconstructed Radiograph）生成
(2) 近傍フィルタによる微分画像生成
(3) 正規化相互相関係数（NCC: Normalized Cross Correlation）評価のための集約演算

この際，効率の良い位置合わせを実現するために，GPU のアーキテクチャ特性を考慮し，ベクトル演算などの描画処理のための高速化機構を活用する．

提案手法の新規性は，上記 (1)〜(3) のすべてを GPU 上で実行する点にあり，(2) および (3) のみを実行する文献 8) とは異なる．また，(3) に対しては，計算精度の良い集約手法を基にしていて，CPU による実装と同等の位置合わせ精度を実現する．

以降では，まず 2 章で GPU のアーキテクチャ，3 章で位置合わせ問題を解説する．次に，4 章および 5 章で提案手法およびその評価実験の結果を示す．続いて，6 章で関連研究および本研究の位置づけを述べる．最後に，7 章で本稿をまとめる．

2. 汎用プロセッサとしての GPU

GPU は本来，描画の高速化を目的とするプロセッサであり，一般にパイプライン構造を持つ（**図 1**）．このパイプラインは，プログラム可能な 2 つの演算器で主に構成されており，それらは VP（Vertex Processor）および FP（Fragment Processor）と呼ばれる．これらはラスタライザと呼ばれるユニットを挟んで連結される．

VP および FP は，それぞれ MIMD 型[9)] および SIMD 型[9)] の演算器であり，データを並列処理できる．演算対象として IEEE754 準拠[10)] の 32 ビット浮動小数点数を扱え，ベクトル演算により 4 個のスカラ値を 1 度に処理できる．この 4 要素ベクトルは，座標 (X, Y, Z, W) あるいは色 (R, G, B, A) を表現するためのものであり，これらをベクトル演算することで効率の良い描画処理を可能とする．また，これらの演算器が参照できるビデオメモリはメインメモリとは独立していて，より高速である．しかし，その容量はメインメモリよりも少なく，現在の GPU では多くても 512 MB である．

GPU を描画処理以外の用途に用いる場合，処理対象のデータをテクスチャ（描画体に貼り付ける模様画像）として保持し，これを参照しながらビデオメモリへの描画を繰り返すという方法をとる．この際，VP と比較して FP の方が一般に高性能である[11)] ため，FP のみを用いる実装が多い．なお，VP はパイプライン

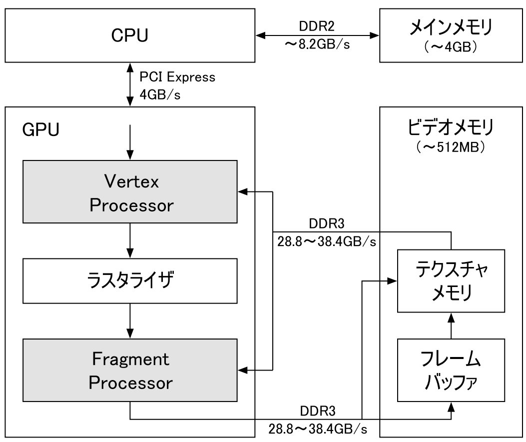


図 1 GPU アーキテクチャ
Fig. 1 GPU architecture.

の上流に位置するため，処理結果をビデオメモリに直接出力できない．一方，下流の FP は処理結果をビデオメモリへ直接出力できる．

FP は SIMD 型の演算器であるため，FP 上で動作するアルゴリズムはテクスチャ上の各画素を互いに独立に処理できる（データ並列性を持つ）必要がある．さもなければ，アーキテクチャの特性を活かせないため，十分な性能を得られない．

描画処理では GPU の処理結果をビデオメモリ上のフレームバッファ（画面）に出力すればよいが，非描画処理では処理結果の取り出しが必要である．そのため，CPU から GPU へのデータ転送に加え，GPU から CPU へのデータ転送（リードバック）が必要となる．これらのデータ転送は AGP もしくは PCI Express といったバスを経由するため，頻繁な転送は性能を低下させる．さらに，現在の GPU は転送の間パイプライン処理を停止させることも，性能を低下させる要因の 1 つである．

GPU を用いる汎用計算において，その計算精度は注意すべき問題の 1 つである．この問題は，GPU が本来，汎用計算ではなく描画処理に用いることを想定した設計に基づくことに起因する．たとえば，GPU は倍精度の浮動小数点数を扱えない．また，単精度においても計算誤差が IEEE 規格に厳格でない[12]．

3. 2 次元/3 次元位置合わせの概要

2 次元/3 次元位置合わせ[1] の手法には，特徴点の抽出に基づくもの[13),14] やボクセル値の比較に基づくもの[1),15),16] などがある．前者は高速であるが，画像から特徴点を手作業で適切に抽出する必要があり，位置合わせ精度はその抽出の正確さに依存する．一方，後者は計算量が多いものの，精度は安定している．加えて，後者は位置合わせ問題を 3 次元データの描画問題に帰着する解法であり，GPU の高速化対象と合う．ゆえに，本研究では文献 3) や文献 8) 同様，後者の手法を用いる．

この手法は，位置合わせ問題を最適化問題に帰着する．具体的には，3 次元画像 V の位置姿勢 P に関するコスト関数 C を最急降下法[17] で最適化することにより，位置合わせを実現する（**図 2**）．ここで，コスト関数 C としては V の投影である DRR と 2 次元画像 I_F との類似度で与える．この類似度を最大化する P を決定する．なお，類似度は大きいほど比較画像が類似することを示し，DRR と I_F が一致するとき最大となる．P は剛体変換パラメータ $(T_x, T_y, T_z, R_x, R_y, R_z)$ で与え，T_x，T_y および T_z は V の平行移動，R_x，R_y および R_z は V の回転移動を表す．図中のステップサイズ D は，評価 1 回あたりにおける P の各パラメータの変化量を表し，P を階層的に最適化するために用いる．D が最下層に至るまで P の移動（パラメータの更新）および評価を繰り返し，最下層で得た結果を解とする．

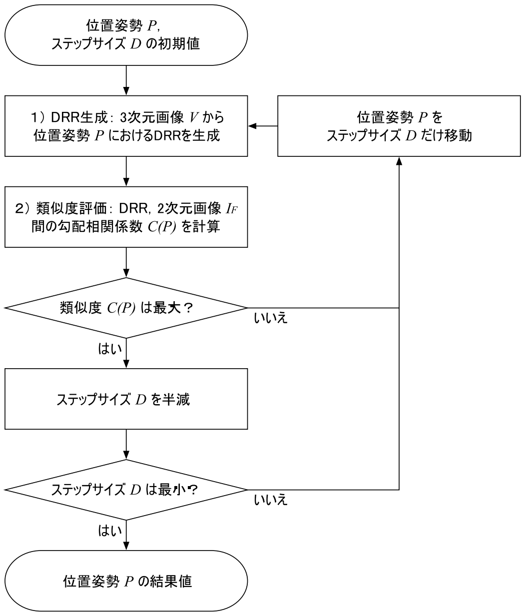


図 2 2 次元/3 次元位置合わせのフロー図
Fig. 2 Flowgraph of 2-D/3-D registration.

類似度の指標としては，勾配相関係数（GC: Gradient Correlation）を用いる．GC は比較画像の画素値補正を必要とせず，また，補正を必要としない方法の中で最も高い頑健性を持つことが知られている[16]．2 枚の画像 A，B 間の GC の値 $G(A, B)$ は次式で与える．

$$G(A,B) = \frac{1}{2}\left[N\left(\frac{\partial A}{\partial x}, \frac{\partial B}{\partial x}\right) + N\left(\frac{\partial A}{\partial y}, \frac{\partial B}{\partial y}\right)\right] \tag{1}$$

なお，関数 N は NCC である．また，$\partial A/\partial x$，$\partial B/\partial x$，$\partial A/\partial y$ および $\partial B/\partial y$ は画像 A，B それぞれの水平方向および垂直方向に対する微分画像である．

2 枚の画像 A，B 間の NCC の値 $N(A, B)$ は次式で与える．

$$N(A,B) = \frac{S_{AB} - S_A S_B/n}{\sqrt{S_{A^2} - (S_A)^2/n}\,\sqrt{S_{B^2} - (S_B)^2/n}} \tag{2}$$

なお，n は画像の画素数である．また，S_A，S_{A^2}，S_B および S_{B^2} は画像 A，B それぞれの画素値の和および 2 乗和であり，S_{AB} は画像 A，B の画素値の積の和である．

水平方向および垂直方向の各微分画像の生成にはガウシアン 1 次微分フィルタを用いる．このフィルタは画像の輪郭を抽出するとともに，ノイズを軽減する特性を持ち，位置合わせの頑健性を向上する．

画像 I に対する，ガウシアン 1 次微分フィルタによる水平方向微分画像 $\partial I/\partial x$ および垂直方向微分画像 $\partial I/\partial y$ は次式で与える．

$$\frac{\partial I}{\partial x}(x,y)=\sum_{i,j}\frac{-i}{2\pi\sigma^4}e^{-\frac{i^2+j^2}{2\sigma^2}}I(x+i,y+j) \quad (3)$$

$$\frac{\partial I}{\partial y}(x,y)=\sum_{i,j}\frac{-j}{2\pi\sigma^4}e^{-\frac{i^2+j^2}{2\sigma^2}}I(x+i,y+j) \quad (4)$$

なお，フィルタ半径を R ピクセルとすると，$-R \leq i, j \leq R$ である．また，σ はガウス関数の標準偏差である．

ここで，実際の位置合わせでは入力画像のノイズを軽減させる目的で R をある程度大きくとる場合がある．一方，式 (3) および式 (4) の計算量はフィルタ半径 R の 2 乗に従うため望ましくない．そこで，一般に行われているように，2 次元フィルタを垂直方向 1 次元フィルタおよび垂直方向 1 次元フィルタの 2 つのフィルタに分解することにより計算量を削減する．

$$P_{x1}(x,y) = \sum_{j} e^{-j^2/2\sigma^2} p(x, y+j) \quad (5)$$

$$P_{x2}(x,y) = \sum_{i} \frac{-i}{2\pi\sigma^4} e^{-i^2/2\sigma^2} p(x+i, y) \quad (6)$$

$$P_{y1}(x,y) = \sum_{j} \frac{-j}{2\pi\sigma^4} e^{-j^2/2\sigma^2} p(x, y+j) \quad (7)$$

$$P_{y2}(x,y) = \sum_{i} e^{-i^2/2\sigma^2} p(x+i, y) \quad (8)$$

以上をまとめると，提案手法が基にする位置合わせ手法は，1）DRR 生成および 2）画像の類似度評価のための GC 計算からなる．後者はさらに，2a）微分画像生成および 2b）NCC 計算に分類できる．

4. GPU を用いた 2-D/3-D 位置合わせ

図 3 に，GPU を用いた位置合わせの設計概要を示す．提案手法における，高速化を目的とした GPU への実装の指針は，次の 2 点である．

- CPU による逐次計算で性能ボトルネックとなる部分を GPU へ移動し，処理時間を短縮すること
- 性能ボトルネックとなりうる CPU–GPU 間の転送量を削減すること

前者に対しては，DRR 生成および微分画像生成を GPU 上で行うことでこれらを高速化する．2 次元/3 次元位置合わせを CPU 上で逐次実行する場合，その実行時間の 9 割強を DRR 生成および，類似度評価における微分画像生成が占める．したがって，GPU を用いてこれら 2 つの処理を高速化できれば，位置合わせ実行時間の短縮が見込める．

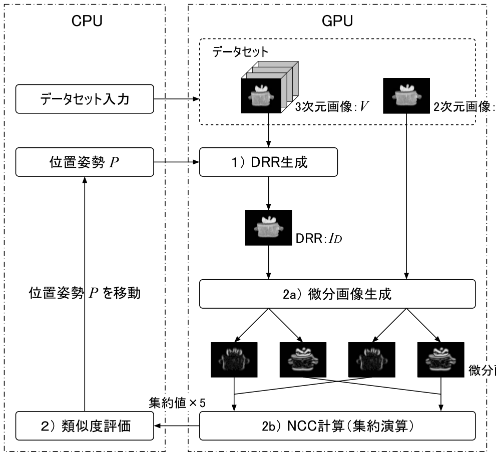


図 3 GPU を用いた 2 次元/3 次元位置合わせ
Fig. 3 2-D/3-D registration on GPU.

後者に対しては，GPU で NCC 計算の前処理を行うことで GPU からの転送量を削減する．NCC 計算はデータ並列性を持たないため，GPU での効率的な実現は容易でない．しかし，微分画像生成を GPU で行う場合，CPU–GPU 間のデータ転送が性能ボトルネックとなるため，NCC 計算も GPU 上に実装しなければ性能は低下する．NCC 計算を CPU で行う場合，類似度評価のたびに GPU から CPU へ微分画像を転送する必要がある．この際の転送量は画像サイズ相当である．一方，GPU で NCC 計算のための集約演算を行えば，転送量は 5 つの集約値 S_A，S_{A^2}，S_B，S_{B^2} および S_{AB} に削減でき（4.3 節で後述），これらは RGBA チャネルからなる 2 ピクセル（2·16 B）で保持できる．

以降では，それぞれの処理および実装について述べる．

4.1 DRR の生成

提案手法は，3 次元テクスチャを用いる TBVR (Texture-Based Volume Rendering)[18] により DRR を生成することで，処理時間を短縮する．

GPU を用いてボリュームレンダリングを行う手法には，Ray-Casting および TBVR の 2 つがある．これら 2 つを比較すると，一般的に，Ray-Casting は生成する投影像の精度に優れるが処理時間が長い．一方，TBVR は精度でやや劣るが処理時間が短い．

TBVR は，テクスチャマッピングやアルファブレンディングといった，GPU 本来の用途である描画処理のための機構を利用した方法であり，効率的である．この TBVR にはさらに，2 次元テクスチャを用いる手法と 3 次元テクスチャを用いる手法がある．

前者は，視線に対する 2 次元テクスチャの角度に依存して奥行き方向の標本間隔が変わるため，生成する投影像の精度が劣る．また，3 次元空間の各軸方向ごとにテクスチャを保持する必要があるため，ビデオメモリの消費量が多い．したがって，提案手法では 3 次元テクスチャによる手法[18] を用いる．

4.2 微分画像の生成

提案手法は，2 枚の画像に対する処理をベクトル化することで全体のスループットを向上する．

ガウシアン 1 次微分フィルタは近傍参照型のフィルタであり，データ並列性を持つ．すなわち，異なる画素の値を独立に計算することが可能である．このため，FP 上での実装が容易であり，かつ FP による並列計算により高い性能を達成できる．

ここで，水平方向微分画像，垂直方向微分画像それぞれに対する垂直方向 1 次元フィルタ（式 (5) および (7)）および水平方向 1 次元フィルタ（式 (6) および (8)）の参照する入力画像の画素に着目すると，これらは一致している．したがって，これらの計算はベクトル化可能であり，次のように記述できる．

$$\boldsymbol{P}_{xy1}=\sum_j\left[\begin{pmatrix} e^{-j^2/2\sigma^2} \\ \dfrac{-j}{2\pi\sigma^4}e^{-j^2/2\sigma^2}\end{pmatrix}*\boldsymbol{p}(x,y+j)\right] \quad (9)$$

$$\boldsymbol{P}_{xy2}=\sum_i\left[\begin{pmatrix} \dfrac{-i}{2\pi\sigma^4}e^{-i^2/2\sigma^2} \\ e^{-i^2/2\sigma^2}\end{pmatrix}*\boldsymbol{p}(x+i,y)\right] \quad (10)$$

なお，$*$ はベクトル要素ごとの積を表し，$\boldsymbol{P}_{xy1}$，$\boldsymbol{P}_{xy2}$ および $\boldsymbol{p}$ はベクトル値である．

まとめると，ベクトル化を終えた式 (9) および (10) をフラグメントプログラムとして実装すればよい．なお，微分画像生成において扱う 2 次元画像 I_F および DRR はいずれも濃淡画像であり，各画素はスカラ値で表現できる．一方，FP は 4 要素ベクトル値を扱うため，このようなベクトル化による計算の効率化が可能となる．

また，2 次元フィルタを 1 次元フィルタに分解することにより（3 章），隣接する画素どうしで重複して行われていた入力画像の参照を含む計算を省ける．したがって，この分解は計算量を削減するだけでなく，テクスチャからの画素値の取得回数，すなわちビデオメモリに対するアクセス回数も削減できる．

4.3 NCC の計算

提案手法は，ベクトル演算を用いて複数の集約演算を一括して行うことで演算回数を削減する．また，固定の 2 次元画像 I_F に対する処理を 1 回に省略することで計算量を削減する．

NCC 計算は，画素値の総和や 2 乗和などを算出する集約演算に帰着できる．集約演算は FP を用いる手法[19]（並列総和計算）がすでに知られている．並列総和計算は，集約演算を部分和計算の集合体として構成し，個々の部分和計算どうしを並列に実行する．この手法はデータ並列性を備えており，かつ各段階の計算を SIMD 演算で処理できる．このため，FP 上に実装できる．

NCC の値を求めるためには，先述のように 5 つの集約値 S_A，S_{A^2}，S_B，S_{B^2} および S_{AB} を求める必要がある．FP はベクトル演算が可能であるので，入力画像の画素値（4 要素ベクトル）の各要素に異なる値を入れて集約演算を行うと，部分和画像の画素値の各要素は要素ごとの部分和となる．このことから，5 つの集約値を求めるために必要なスカラ値に対する 5 回の計算は，ベクトル値に対する 2 回の計算で行うことができる．このように演算回数を削減することは，リードバック回数の削減と処理時間の短縮を同時に達成する．

位置合わせにおいては，2 次元画像 I_F および DRR I_D それぞれの微分画像間における NCC の値を求めることになる．ここで，一方の I_F は処理全体を通じて固定であり，その画素値の和および 2 乗和は一定である．したがって，I_F に対しては初めの 1 回のみ結果を得れば，以降は I_D に対してのみ集約演算を行えば NCC の値を算出できる．

4.4 実 装

提案手法の実装には，C++ 言語，OpenGL ライブラリ[20] および Cg（C for graphics）ツールキット[21] を用いた．プログラムの規模は約 6,000 行である．基本的には，DRR 生成，微分画像生成および並列総和計算の各処理に応じて頂点プログラムおよびフラグメントプログラムをバインドし，テクスチャへの描画を繰り返す．紙面の都合上，並列総和計算を例に実装を示す．

図 4 に，並列総和計算に対する実装の概要を示す．この例では，3×3 ピクセルからなる領域 $(0,0)$〜$(2,2)$ を 1 ピクセル $(0,0)$ に集約している．頂点プログラムでは，$(2,2)$ を除く 8 つのピクセル☆の相対座標を

☆ 実験で用いた GPU のプロファイル CG_PROFILE_VP40 では，VP から FP へ同時に転送できるレジスタの値が 8 つに制限されているため，9 つすべてを転送できない．これらを VP 上で処理することは計算量削減による高速化につながる．詳細は文献 22) を参照されたい．

```
void runReduction(Region &region, // 描画領域
      TextureObject *rtexture, // 初期化済テクスチャオブジェクト
      RenderTexturePBuffer *pbuffer, // 初期化済ピクセルバッファ
      BufferSpecifier &rtextureBuffer, // 微分画像を保持するバッファ
      BufferSpecifier &drawBuffer) // 出力用バッファ
{
      cgGLEnableProfile(vertexProfile and fragmentProfile);
      cgGLBindProgram(vertexProgram and fragmentProgram); // 図 (b) および (c) 参照
      glDrawBuffer(drawBuffer); // 出力用バッファを指定
      rtexture->bindTexture(); // テクスチャを有効化
      pbuffer->bindTexImage(rtextureBuffer); // 微分画像を入力テクスチャとしてバインド
      glClear(GL_COLOR_BUFFER_BIT); // 出力用バッファをクリア
      glRecti(region); // 指定領域を描画
      glFlush(); // 発行済 OpenGL コマンドを強制実行
      pbuffer->releaseTexImage(rtextureBuffer); // 入力テクスチャを解放
      cgGLDisableProfile(vertexProfile and fragmentProfile);
}
```

(a) CPU program

```
// VP から FP へ渡す座標を保持するための構造体
struct ReductionCoords {
      float4 position : POSITION; float2 coord0 : TEXCOORD0; float2 coord1 : TEXCOORD1;
      float2 coord2 : TEXCOORD2; float2 coord3 : TEXCOORD3; float2 coord4 : TEXCOORD4;
      float2 coord5 : TEXCOORD5; float2 coord6 : TEXCOORD6; float2 coord7 : TEXCOORD7;
};
// 3×3 領域集約演算において，集約すべき座標を計算
ReductionCoords reductionVertex9(float4 position : POSITION, // 頂点座標（範囲 [0.33,0.66]）
      uniform float4x4 modelViewProjMatrix : state.matrix.mvp) // 位置姿勢（モデルビュー変換）を表す行列
{
      ReductionCoords output;
      output.position = mul(modelViewProjMatrix, position); // 頂点座標の計算
      output.coord0 = position.xy * 3 − 1.0f; // [0.33,0.66] の入力に対し，参照範囲がテクスチャ全体 [0,1] に なるよう変換
      output.coord1 = output.coord0 + float2(1.0, 0.0); output.coord2 = output.coord0 + float2(0.0, 1.0);
      output.coord3 = output.coord0 + float2(1.0, 1.0); output.coord4 = output.coord0 + float2(0.0, 2.0);
      output.coord5 = output.coord0 + float2(2.0, 0.0); output.coord6 = output.coord0 + float2(1.0, 2.0);
      output.coord7 = output.coord0 + float2(2.0, 1.0); // レジスタ不足により座標 (2, 2) は FP 側で計算（本文参照）
      return output;
}
```

(b) Vertex program

```
// RGB チャネルに対する 3×3 領域集約和演算
float3 reductionSum9RGB(ReductionCoords input,
      uniform samplerRECT sampRect : TEXUNIT0) : COLOR
{
      float2 coord8 = input.coord0 + float2(2.0, 2.0); // VP で処理しきれない座標 (2, 2) の計算
      float3 output = texRECT(sampRect, input.coord0).rgb;
      output += texRECT(sampRect, input.coord1).rgb; output += texRECT(sampRect, input.coord2).rgb;
      output += texRECT(sampRect, input.coord3).rgb; output += texRECT(sampRect, input.coord4).rgb;
      output += texRECT(sampRect, input.coord5).rgb; output += texRECT(sampRect, input.coord6).rgb;
      output += texRECT(sampRect, input.coord7).rgb;
      output += texRECT(sampRect, coord8).rgb; // 座標 (2, 2) の参照
      return output;
}
```

(c) Fragment program

図 4 並列総和計算のための CPU および GPU プログラム
Fig. 4 CPU and GPU programs for parallel reduction.

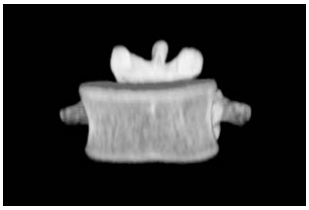
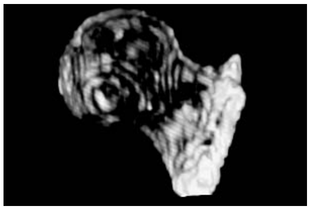
計算し，その結果を出力する．フラグメントプログラムでは，その 8 つの座標を受け取り，それらのピクセル値の総和を計算する．CPU 側のプログラムでは，これらのプログラムや入力用のテクスチャをバインドしたうえで，指定する領域を描画する．なお，集約を終えたのち，集約値をメインメモリへ転送する際には glReadPixels() を用いてピクセル値を参照する．

5. 評価実験

GPU を用いた 2 次元/3 次元剛体位置合わせの有用性を検証するため，その実行時間および残差[24)]（解として得る位置姿勢と，正解の位置姿勢とのずれ）を，**表 1** に示す GPU および CPU を用いて測定した．

データには椎骨および大腿骨（**表 2**）を用いた．ここで，表中における ROI（Region of Interest）とは，

表 2　データセット
Table 2　Dataset specification.

	椎骨	大腿骨
		
3 次元画像（ボクセル）	512 × 512 × 204	256 × 256 × 367
・ファイルサイズ*	102 MB	46 MB
・ROI（ボクセル）	300 × 300 × 48	54 × 38 × 55
2 次元画像（ピクセル）	300 × 200（×2）	
・ファイルサイズ*	36 KB	56 KB

*：ファイルは PNG 形式

表 1　実験環境
Table 1　Experimental environment.

	PC クラスタ版	GPU 版
CPU	Pentium 3 1.0 GHz × 2	Pentium 4 2.8 GHz
RAM	2 GB	
GPU	なし	Quadro FX 3400
コアクロック		350 MHz
VRAM		256 MB
メモリクロック		900 MHz
メモリ帯域		28.8 GB/s
フィルレート		4.2 Gpixels/s
PC 間接続	Myrinet[23] 2000	なし

位置合わせの対象となる領域を表す．この領域のみを切り出すことで計算量を削減し，また位置合わせ精度を向上させることができる[15]．なお，今回の実験では正解の位置姿勢を特定するため，あらかじめ特定の位置姿勢における DRR を生成し，これを 2 次元画像 I_F として用いた．また，ステップサイズの初期値は 2（ボクセルあるいは度），終了条件である最小値は 0.1 とする．DRR 生成において用いた投影面のサイズは 350×250 ピクセルである．また，$\sigma = 3$ および $R = 9$ とした．

5.1　実行時間に関する考察

GPU を用いることにより，2 次元/3 次元剛体位置合わせを 10 秒程度で実行できている（**表 3**）．このことから，CPU を用いて逐次計算した場合に十数分を要する位置合わせは，GPU を用いることで，手術支援の要求する時間制約を満たすと考える．

また，その実行時間の内訳（**表 4**）を見ると，CPU による逐次計算でその大部分を占めていた DRR 生成および微分画像生成の処理時間を削減できている．なお，表 4 中の逐次版の列は，GPU 版と同スペックの CPU 1 個による逐次計算の結果である．PC クラスタ版の列は，表 1（PC クラスタ版）記載の PC 32 台（64 CPU）からなる PC クラスタによる並列計算の結果である．ここで比較に用いた CPU 版位置合わせは，ボリュームレンダリングに Ray-Casting を用いているなど，提案手法とアルゴリズムが異なる．そのため，厳密な比較はできないが，CPU 版に比べ GPU 版の処理時間における優位性は確認できる．

次に，GPU で集約演算を行うことの効果を調べる．そのために，提案手法および GPU から CPU へ微分画像を転送して NCC 値を計算する手法（手法 A）の処理時間を比較する．CPU で微分画像から NCC の値を求める処理時間は 9 ミリ秒ないし 3 ミリ秒であり（表 4），この時間に GPU で生成した微分画像を CPU へ転送する時間を加算することで，手法 A の処理時間を算出できる．測定の結果，GPU から CPU へ 4 枚の微分画像を転送する時間は 6 ミリ秒程度である．すなわち，手法 A の処理時間は 15 ミリ秒ないし 9 ミリ秒であり，提案手法では 5.3 ミリ秒であるから，後者が有効である．GPU で集約演算を行うことで，CPU–GPU 間の転送量をおよそ 1.3 MB（$= 350 \cdot 250 \cdot 16$）から 32 B（$= 2 \cdot 16$）に削減でき，転送時間を 0.2 ミリ秒に削減できる．実際に手法 A を実装した結果，椎骨および大腿骨データセットの位置合わせに対し，各々 14 秒および 9 秒を要した．

5.2　精度に関する考察

先に述べたように，GPU はアーキテクチャに起因して計算精度に問題がある．加えて本研究では，ボリュームレンダリングの際，Ray-Casting と比較して精度の劣る TBVR を用いており，これも精度を落と

表 3 提案手法による位置合わせ結果
Table 3 Registration results obtained by the proposed method.

	椎骨データセット				大腿骨データセット			
初期残差（mm）	評価回数	実行時間（秒）	結果残差（mm）	成功回数	評価回数	実行時間（秒）	結果残差（mm）	成功回数
2〜 4	242	9.0	0.10	10	194	4.8	0.52	10
4〜 6	249	9.3	0.13	10	209	5.2	0.45	10
6〜 8	292	10.9	0.09	10	257	6.3	0.51	10
8〜10	281	10.5	0.09	10	271	6.7	0.94	9
10〜12	341	12.6	1.46	8	292	7.2	0.68	9
12〜14	372	13.8	1.11	7	297	7.3	0.73	9
14〜16	359	13.2	5.86	4	334	8.2	1.86	9
16〜18	345	12.7	7.09	4	347	8.6	5.37	5
18〜20	365	13.4	11.13	2	351	8.7	4.82	5
20〜22	376	13.8	10.71	2	382	9.4	6.58	5

表 4 評価 1 回あたりの処理時間（ミリ秒）
Table 4 Processing time per step.

	椎骨データセット			大腿骨データセット		
	CPU 版			CPU 版		
内訳	逐次版	PC クラスタ版	GPU 版	逐次版	PC クラスタ版	GPU 版
DRR 生成	2940	142	26.4	810	31	14.1
微分画像生成	142	7	5.2	197	7	5.3
NCC 計算	9	46	5.3	3	14	5.3
・集約演算	—	—	4.9	—	—	4.9
・転送	—	—	0.2	—	—	0.2
小計（ミリ秒）	3091	195	36.9	1010	52	24.7
評価回数	300					
実行時間	15 分	58 秒	11 秒	5 分	15 秒	7 秒

表 5 CPU 実装[3] による位置合わせ結果
Table 5 Registration results obtained by the CPU-based method [3].

	椎骨データセット				大腿骨データセット			
初期残差（mm）	評価回数	実行時間（秒）	結果残差（mm）	成功回数	評価回数	実行時間（秒）	結果残差（mm）	成功回数
2〜 4	231	48.3	0.27	10	124	8.0	1.13	8
4〜 6	240	49.4	0.25	10	145	9.1	1.63	6
6〜 8	238	49.5	0.24	10	151	9.4	3.27	2
8〜10	245	51.0	1.70	9	145	9.0	12.71	1
10〜12	346	69.1	3.12	8	158	9.7	9.34	0
12〜14	282	57.6	8.92	4	152	9.4	11.19	1
14〜16	311	62.8	7.25	5	156	9.6	15.18	1
16〜18	331	67.2	13.73	2	154	9.5	16.46	0
18〜20	418	82.1	13.17	3	184	11.1	22.63	0
20〜22	403	78.3	18.48	1	158	9.8	22.23	0

す要因となる．

表 3 に，本研究の実装における位置合わせ結果の残差[24] と成功回数を示す．また，**表 5** に CPU を用いた位置合わせ[3] の結果を示す．なお，PC クラスタ版と逐次版の結果は実行時間を除いて同一である．

ここでは精度の指標として，位置姿勢のボクセル単位でのずれを見る．結果残差が 0.5 ボクセル（椎骨：0.66 mm，大腿骨：1.56 mm に相当）以下であるものを成功とする．なお，各初期残差に対して試行は 10 回である．結果から，初期残差が 10 mm 程度であるとき，ほぼすべての試行で位置合わせに成功できている．

なお，位置合わせを実際の手術支援に用いる際には入力の 2 次元画像として X 線像を与える．したがって，2 次元画像として DRR を使用する今回の実験結果が，ただちに実用における精度を保証するものではない．しかし，CPU を用いる位置合わせと同程度の

表 6　Hybrid 法[8] による位置合わせ結果
Table 6　Registration results obtained by the hybrid method [8].

	椎骨データセット				大腿骨データセット			
初期残差（mm）	評価回数	実行時間（秒）	結果残差（mm）	成功回数	評価回数	実行時間（秒）	結果残差（mm）	成功回数
2～ 4	263	8.6	2.09	2	226	4.7	1.22	10
4～ 6	308	10.0	1.85	1	230	4.8	1.17	10
6～ 8	332	10.8	3.37	1	291	6.1	6.12	5
8～10	310	10.1	6.20	0	296	6.2	9.86	1
10～12	385	12.5	10.74	0	301	6.3	11.98	0
12～14	430	13.9	8.24	0	284	6.0	11.27	0
14～16	321	10.3	19.47	0	314	6.6	15.96	0
16～18	384	12.3	19.39	0	346	7.3	14.50	0
18～20	366	11.6	16.98	0	302	6.4	19.31	0
20～22	340	11.1	20.18	0	282	5.9	21.04	0

表 7　関連研究の比較
Table 7　Comparison of related work.

<table>
<tr><td>比較項目</td><td colspan="2">提案手法</td><td colspan="2">文献 8)</td><td colspan="2">文献 3)</td></tr>
<tr><td>計算機環境</td><td colspan="4">GPU 装備の PC</td><td colspan="2">PC クラスタ（p CPUs）</td></tr>
<tr><td>位置合わせ手法</td><td colspan="6">文献 1)，15)，16)</td></tr>
<tr><td rowspan="2">DRR 生成</td><td colspan="2" rowspan="2">GPU
3-D texture-based VR</td><td colspan="2">CPU</td><td colspan="2">p CPUs</td></tr>
<tr><td colspan="4">Ray-Casting VR</td></tr>
<tr><td rowspan="2">微分画像生成</td><td colspan="4">GPU</td><td colspan="2">p CPUs</td></tr>
<tr><td colspan="2">ベクトル演算</td><td colspan="4">スカラ演算</td></tr>
<tr><td>NCC 計算</td><td>GPU
並列ベクトル総和</td><td>CPU
仕上げの計算 *</td><td>GPU
mipmap 機構</td><td>CPU
SSE 演算と仕上げの計算</td><td>p CPUs
並列スカラ総和</td><td>CPU
仕上げの計算</td></tr>
<tr><td>処理時間</td><td colspan="2">○</td><td colspan="2">×</td><td colspan="2">○</td></tr>
<tr><td>精度</td><td colspan="2">○</td><td colspan="2">×</td><td colspan="2">○</td></tr>
<tr><td>コスト</td><td colspan="2">○</td><td colspan="2">○</td><td colspan="2">×</td></tr>
</table>

*：5 つの集約値 S_A，S_{A^2}，S_B，S_{B^2} および S_{AB} を基に，式 (2) を計算すること

精度を有することは確認できた．

表 6 に，既存手法[8]（文献に倣い，Hybrid 法と呼ぶ）による結果を示す．この結果は，GPU における mipmap 機構を用いて微分画像を 1/4 に平均化したのち，その平均画像を CPU に転送し，残りを CPU の SSE 演算[25] を用いて集約したものである．なお，本来の Hybrid 法は DRR 生成に CPU を用いるが，今回は GPU を用いた．表 3 と比較すると，Hybrid 法は提案手法よりも位置合わせの精度が低い．この原因は後述する（6 章）．

また，Hybrid 法では一部の実行時間が提案手法よりも短い．この理由は，位置合わせの失敗が原因で，提案手法よりも最適化に要する評価回数が少ないためである．評価 1 回あたりの処理時間は，提案手法とほぼ同じである．ただし，本来の Hybrid 法は DRR 生成に長い時間を要することに注意されたい．

6. 関 連 研 究

位置合わせの高速化に対するアプローチは，アルゴリズムの計算量を削減する手法および高性能ハードウェアを用いる手法に分類できる．

前者に属するものとして，位置合わせにおいて計算量の大半を占める DRR 生成に対し，少ない計算量で行う手法が提案されている．たとえば，初期位置姿勢の近傍の位置姿勢における 3 次元画像の DRR をあらかじめ計算しておき，位置合わせの際にはそれらを用いて DRR を近似的に生成する手法[26),27] がある．さらにこれを発展させ，可能であれば先に生成した DRR の画素値を再利用することで DRR 生成を高速化する手法[28] がある．また，DRR 生成の回数を削減することで高速化する手法[29] もある．この手法では，3 次元画像の位置姿勢を移動する際，投影面の法線方向を軸とする回転移動については移動前後の投影像が合同になることに着目し，DRR 生成を省略する．一方，これらのアプローチの中で画像間の類似度計算に着目した研究は，我々の知る限り存在しない．

後者に属するものの比較を，**表 7** にまとめる．まず，評価実験においても比較に用いたが，PC クラスタを用いる手法[3] がある．この手法の問題点については先に述べたとおりである．さらには，GPU を用い

る手法があり，DRR 生成あるいは類似度計算に対するものが存在する．我々の提案手法もこれに含まれる．

類似度計算に対するものとして，提案手法と同様，CPU–GPU 間の転送量を削減することで高速化する Hybrid 法[8] がある．Hybrid 法は，GPU において mipmap 機構[30] を利用して集約演算を行い，サイズを縮約した平均画像を CPU へ転送し，残りを CPU において SSE[25] を用いて集約する．この方法の欠点は，集約の 1 ステップごとに平均をとるために丸め誤差が生じ，その誤差が反復の過程で拡大することである．また，集約する画像をテクスチャとして保持する必要があるため，演算対象である画素値を 8 ビット整数（0〜255）で扱わなければならない．これもまた，精度を落とす要因である．この誤差を拡大しうる GPU での集約回数を削減すべく，Hybrid 法では，GPU で集約を終えずに平均画像を CPU へ転送する．しかし，結果として，CPU–GPU 間の転送量を削減しきれていない．

一方，提案手法は GPU での集約演算に並列総和計算を用いており，この方法は画素値を 32 ビット浮動小数点数として扱える．加えて，平均化による丸め誤差もない．これらのことから，GPU における集約演算の計算精度に関して，Hybrid 法に優れる．CPU 上での計算は高精度であるが，GPU の計算精度に対して，転送された時点において含まれる誤差を解消するほどの差はないと考える．また，CPU–GPU 間の転送量に関しては，提案手法が GPU からリードバックするのは 5 つの数値であり，平均画像を転送する Hybrid 法に比べて削減量が多い．

7. おわりに

本稿では，2 次元/3 次元剛体位置合わせに対し，GPU を用いて高速化する手法を提案した．提案手法は，実行時間を短縮するために，CPU を用いて逐次計算する場合にその実行時間の大半を占める DRR 生成および微分画像生成を GPU 上で高速処理する．さらに，NCC 計算のための集約演算を GPU で処理することにより，GPU から CPU へのデータ転送量を削減する．集約演算には並列総和計算を用いることにより，mipmap 機構を用いる手法よりも高い位置合わせ精度を実現する．

GPU を用いて提案手法の実行時間を測定した結果，10 秒程度で位置合わせを行えた．この実行時間は手術支援用途に対して十分に短い．また，位置合わせ精度についても，CPU を用いる場合と同程度であることを確認した．これらの結果から，GPU を用いる 2 次元/3 次元位置合わせの有用性は高いと考える．

今後の課題としては，GPU の特性を活かすより効率的な手法の検討や，実用に向けたインタフェースの整備などを行いたい．

謝辞　本研究の一部は，科学研究費補助金特定領域研究（17032007）および基盤研究（B）（2）（18300009）の補助による．また，臨床画像を提供していただいた大阪大学大学院医学系研究科田村進一教授，佐藤嘉伸助教授，東京大学インテリジェント・モデリング・ラボラトリー中島義和助教授に深謝いたします．有益なご意見をいただいた査読者の方々に深く感謝いたします．

参 考 文 献

1) Lemieux, L., Jagoe, R., Fish, D.R., Kitchen, N.D. and Thomas, D.G.T.: A patient-to-computed-tomography image registration method based on digitally reconstructed radiographs, *Medical Physics*, Vol.21, No.11, pp.1749–1760 (1994).
2) Hajnal, J.V., Hill, D.L. and Hawkes, D.J. (Eds.): *Medical Image Registration*, CRC Press, Boca Raton, FL (2001).
3) 川崎康博，伊野文彦，田代孝仁，中島義和，佐藤嘉伸，菅野伸彦，田村進一，萩原兼一：術中二次元/三次元剛体位置合せのための並列化手法，電子情報通信学会論文誌，Vol.J88-D-I, No.10, pp.1534–1546 (2005).
4) Pharr, M. and Fernando, R. (Eds.): *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation*, Addison-Wesley, Reading, MA (2005).
5) Montrym, J. and Moreton, H.: The GeForce 6800, *IEEE Micro*, Vol.25, No.2, pp.41–51 (2005).
6) Galoppo, N., Govindaraju, N.K., Henson, M. and Manocha, D.: LU-GPU: Efficient Algorithms for Solving Dense Linear Systems on Graphics Hardware, *Proc. Int'l Conf. High Performance Computing, Networking, Storage and Analysis* (*SC'05*) (2005). 12 pages (CD-ROM).
7) Moreland, K. and Angel, E.: The FFT on a GPU, *Proc. SIGGRAPH/EUROGRAPHICS Workshop Graphics Hardware* (*GH'03*), pp.112–119 (2003).
8) Chisu, R.: Techniques for Accelerating Intensity-based Rigid Image Registration, Ph.D. Thesis, Technische Universität München, München, Germany (2005).
9) Grama, A., Gupta, A., Karypis, G. and Kumar, V.: *Introduction to Parallel Computing*, 2nd edition, Addison-Wesley, Reading, MA

(2003).

10) Stevenson, D.: A Proposed Standard for Binary Floating-Point Arithmetic, *IEEE Computer*, Vol.14, No.3, pp.51–62 (1981).

11) Owens, J.D., Luebke, D., Govindaraju, N., Harris, M., Krüger, J., Lefohn, A.E. and Purcell, T.J.: A Survey of General-Purpose Computation on Graphics Hardware, *EUROGRAPHICS 2005, State of the Art Report*, pp.21–51 (2005).

12) Hillesland, K.E. and Lastra, A.: GPU Floating Point Paranoia, *Proc. 1st ACM Workshop General-Purpose Computing on Graphics Processors* (*GP*^{2}*'04*), p.C-8 (2004).

13) West, J., Fitzpatrick, J.M., Wang, M.Y., Dawant, B.M., Maurer, C.R., Kessler, R.M. and Maciunas, R.J.: Retrospective Intermodality Registration Techniques for Images of the Head: Surface-Based Versus Volume-Based, *IEEE Trans. Medical Imaging*, Vol.18, No.2, pp.144–150 (1999).

14) McLaughlin, R.A., Hipwell, J., Hawkes, D.J., Noble, J.A., Bryne, J.V. and Cox, T.: A Comparison of 2D-3D Intensity-Based Registration and Feature-Based Registration for Neurointerventions, *Proc. 5th Int'l Conf. Medical Image Computing and Computer-Assisted Intervention* (*MICCAI'02*), *Part II*, pp.517–524 (2002).

15) Weese, J., Penney, G.P., Desmedt, P., Buzug, T.M., Hill, D.L.G. and Hawkes, D.J.: Voxel-Based 2-D/3-D Registration of Fluoroscopy Images and CT Scans for Image-Guided Surgery, *IEEE Trans. Information Technology in Biomedicine*, Vol.1, No.4, pp.284–293 (1997).

16) Penney, G.P., Weese, J., Little, J.A., Desmedt, P., Hill, D.L.G. and Hawkes, D.J.: A Comparison of Similarity Measures for Use in 2-D–3-D Medical Image Registration, *IEEE Trans. Medical Imaging*, Vol.17, No.4, pp.586–595 (1998).

17) Press, W.H., Teukolsky, S.A., Vetterling, W.T. and Flannery, B.P.: *NUMERICAL RECIPES in C: The Art of Scientific Computing*, Cambridge University Press, Cambridge, UK (1988).

18) Cullip, T.J. and Neumann, U.: Accelerating Volume Reconstruction with 3D Texture Hardware, Technical Report TR93-027, University of North Carolina at Chapel Hill (1993).

19) Fernando, R. (Ed.): *GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics*, Addison-Wesley, Reading, MA (2004).

20) Shreiner, D., Woo, M., Neider, J. and Davis, T.: *OpenGL Programming Guide*, 4th edition, Addison-Wesley, Reading, MA (2003).

21) Mark, W.R., Glanville, R.S., Akeley, K. and Kilgard, M.J.: Cg: A system for programming graphics hardware in a C-like language, *ACM Trans. Graphics*, Vol.22, No.3, pp.896–897 (2003).

22) Ikeda, T., Ino, F. and Hagihara, K.: A Code Motion Technique for Accelerating General-Purpose Computation on the GPU, *Proc. 20th IEEE Int'l Parallel and Distributed Processing Symp.* (*IPDPS'06*) (2006). 10 pages (CD-ROM).

23) Boden, N.J., Cohen, D., Felderman, R.E., Kulawik, A.E., Seitz, C.L., Seizovic, J.N. and Su, W.-K.: Myrinet: A Gigabit-per-Second Local Area Network, *IEEE Micro*, Vol.15, No.1, pp.29–36 (1995).

24) Fitzpatrick, J.M., West, J.B. and Maurer, C.R.: Predicting Error in Rigid-Body Point-Based Registration, *IEEE Trans. Medical Imaging*, Vol.17, No.5, pp.694–702 (1998).

25) Klimovitski, A.: Using SSE and SSE2: Misconceptions and Reality, *Intel Developer Update Magazine* (2001). Available at http://www.intel.com/technology/magazine/computing/sw03011.pdf

26) LaRose, D.A.: Iterative X-Ray/CT Registration Using Accelerated Volume Rendering, Ph.D. Thesis, Carnegie Mellon University, Pittsburgh, PA (2001).

27) Russakoff, D.B., Rohlfing, T., Rueckert, D., Shahidi, R., Kim, D. and Maurer, C.R.: Fast calculation of digitally reconstructed radiographs using light fields, *Proc. SPIE Medical Imaging* (*MI'03*), Vol.5032, pp.684–695 (2003).

28) Rohlfing, T., Russakoff, D.B., Denzler, J. and Maurer, C.R.: Progressive Attenuation Fields: Fast 2D-3D Image Registration Without Precomputation, *Proc. 7th Int'l Conf. Medical Image Computing and Computer-Assisted Intervention* (*MICCAI'04*), *Part I*, pp.631–638 (2004).

29) Birkfellner, W., Wirth, J., Burgstaller, W., Baumann, B., Staedele, H., Hammer, B., Gellrich, N.C., Jacob, A.L., Regazzoni, P. and Messmer, P.: A faster method for 3D/2D medical image registration — a simulation study, *Physics in Medicine and Biology*, Vol.48, No.16, pp.2665–2679 (2003).

30) Williams, L.: Pyramidal Prametrics, *Computer Graphics*, Vol.17, No.3, pp.1–11 (1983).

(平成 17 年 12 月 8 日受付)
(平成 18 年 7 月 4 日採録)

五味田　遵

平成 18 年大阪大学基礎工学部情報科学科卒業．現在，東京大学大学院情報理工学系研究科修士課程在学中．並列処理の応用に関する研究に従事．

合田　圭吾

平成 17 年大阪大学基礎工学部情報科学科卒業．現在，同大学院情報科学研究科修士課程在学中．並列処理の応用に関する研究に従事．

川崎　康博

平成 14 年大阪大学基礎工学部情報科学科卒業．平成 16 年同大学院情報科学研究科修士課程修了．現在，同大学院博士課程在学中．平成 15 年国際会議 HiPC'03 最優秀論文賞受賞．並列処理の応用に関する研究に従事．

伊野　文彦（正会員）

平成 10 年大阪大学基礎工学部情報工学科卒業．平成 12 年同大学院基礎工学研究科修士課程修了．平成 14 年同大学院同研究科博士課程中退．同年同大学助手．博士（情報科学）．平成 15 年国際会議 HiPC'03 最優秀論文賞，平成 16 年先進的計算基盤システムシンポジウム SACSIS'04 最優秀論文賞受賞．並列計算機の応用およびソフトウェア開発環境に関する研究に従事．

萩原　兼一（正会員）

昭和 49 年大阪大学基礎工学部情報工学科卒業．昭和 54 年同大学院基礎工学研究科博士課程修了．工学博士．同大学助手，講師，助教授を経て，平成 5 年奈良先端科学技術大学院大学教授．平成 6 年より大阪大学教授．平成 4～5 年文部省在外研究員（米国メリーランド大学）．平成 15 年国際会議 HiPC'03 最優秀論文賞，平成 16 年先進的計算基盤システムシンポジウム SACSIS'04 最優秀論文賞受賞．現在，並列処理の基礎および応用に興味を持っている．