社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2008－ARC－177 (20)
2008－HPC－114 (20)
2008／3／5

CUDAによる高速なコーンビーム再構成の実装と性能評価

興津佑輔[†] 伊野文彦[††] 萩原兼一[††]

本稿では，高速なコーンビーム再構成を実現するために，GPU（Graphics Processing Unit）の汎用計算環境であるCUDA（Compute Unified Device Architecture）を用いた手法を提案する．提案手法では，GPU上のグローバルメモリおよびローカルメモリに対するアクセス回数を削減する．また，アクセス遅延を隠蔽する．評価実験として512^2画素からなる360枚の投影像を用い，512^3ボクセルのボリュームを再構成した結果，5.7秒を要した．この実行時間は，既存のOpenGL実装より2.6秒高速であり，SSEを用いるCPU実装と比較して23.7倍高速である．また，提案手法およびCPU実装の再構成結果には視認できる違いはなかった．

Implementation and Performance Evaluation of Fast Cone Beam Reconstruction using CUDA

Yusuke Okitsu,[†] Fumihiko Ino[††] and Kenichi Hagihara[††]

This paper presents a fast method for cone beam reconstruction implemented using compute unified device architecture (CUDA), which is a general purpose computation environment for the graphics processing unit (GPU). The proposed method reduces the number of accesses to global memory and local memory equipped on the GPU. It also hides the latency needed for memory access. Evaluation results show that the proposed method takes 5.7 seconds to reconstruct a 512^3-voxel volume from 360 512^2-pixel projection images. This execution time is 2.6 seconds faster than an OpenGL-based implementation. It is also 23.7-fold faster than a CPU implementation using SSE. There are no visible differences between our reconstruction results and those generated by the CPU.

1. はじめに

GPU[1]（Graphics Processing Unit）はグラフィクス処理を高速化するためのプロセッサである．近年，性能向上が著しく，単精度のみであるがCPUを超える浮動小数点演算性能を持つ．CUDA[2]（Compute Unified Device Architecture）は，GPUの汎用計算環境でありGPUを並列計算機として扱える．C言語を拡張した言語でプログラム可能であり，GPGPU（General Purpose GPU）を容易に実現できる．

一方，コーンビーム再構成はコーンビームCT（Computed Tomography）で撮影した複数枚の投影像からボリュームを生成する．コーンビームCTはCT撮影法の一種であり，X線源と平面状の検出器を回転させながら撮影する．そのため，従来のCTと比べて術中の撮影が容易である．しかし，術中に用いる場合，実時間の再構成が不可欠である．

実時間の再構成を実現している既存研究として文献3)が存在する．この文献では，グラフィクスライブラリであるOpenGL[4]を用い，フィルタリング処理および逆投影処理からなるFDK（Feldkamp）法[5]を実装している．FDK法がメモリ集中型のアルゴリズムであるのに対し，この実装はGPU・ビデオメモリ間の転送帯域幅に関して理論値の70%を引き出している．一方，CUDAを用いた既存研究として文献6)が存在する．しかし，逆投影処理のみを実装するものであり，再構成全体の性能や実装の詳細は不明である．

そこで本稿では，CUDAを用いたコーンビーム再構成の高速化手法を提案する．また，CUDAを用いることでGPUの性能をどこまで引き出せるか検証する．提案手法は，コーンビーム再構成におけるアクセス遅延を隠蔽し，メモリアクセス回数を削減する．

以降では，まず2節でCUDAのアーキテクチャを説明する．次に，3節で実装に用いるFDK法につい

† 大阪大学基礎工学部情報科学科
Department of Information and Computer Sciences, School of Engineering Science, Osaka University
†† 大阪大学大学院情報科学研究科コンピュータサイエンス専攻
Department of Computer Science, Graduate School of Information Science and Technology, Osaka University

て述べる．4節で提案手法について説明し，5節で評価実験の結果を示す．最後に6節で本稿をまとめる．

2. CUDA の特徴

CUDA では，GPU を複数のマルチプロセッサ（MP）からなる並列計算機として扱う．さらに MP は複数のストリームプロセッサ（SP）を内蔵する．各 MP は高速だが小容量の共有メモリを持ち，同一 MP 内の SP 間で共有できる．また，GPU 全体で使用できるデバイスメモリとして，グローバルメモリ（GM），テクスチャメモリおよび定数メモリがある．テクスチャメモリおよび定数メモリはキャッシュ機能を持ち，GM よりもアクセス遅延が短い．CPU はあらかじめこれらにデータを転送できる．一方，GPU は GM のみ書き込める．また，GM の特長として，連続領域に対する参照（Coalesced 参照[2]）が高速である点が挙げられる．また，テクスチャメモリは，ハードウェアによる画素値の線形補間が可能である．

次に，CUDA でのプログラムの動作について説明する．CUDA では，1つのジョブを複数のブロックに分割して実行する．1回のジョブ実行をカーネル実行と呼ぶ．1つのブロックは1つの MP に割り当てられ，MP には最大8個のブロックを割り当てることができる．1つの MP に複数個のブロックが割り当てられた場合，アクセス遅延を他のブロックを実行することで隠蔽できる．しかし，MP に割り当て可能なブロック数は，1ブロックの共有メモリ使用量およびレジスタ使用量で決まり，常にアクセス遅延を隠蔽できるとは限らない．また，ブロックは複数のスレッドで構成される．CUDA では，このスレッドのプログラムを作成する．スレッドは SP 上で動作し，プログラム中の変数はレジスタに格納される．レジスタ数が足りない場合，ローカルメモリ（LM）に格納される．LM は自動的にデバイスメモリ上に確保される領域である．GM と異なり，Coalesced 参照を実現できない．そのため，Coalesced 参照を実現した GM と比較すると低速である．

3. Feldkamp 法

FDK 法[5] はフィルタリング処理と逆投影処理で構成され，入力として K 枚の $U \times V$ 画素の投影像 P_0〜P_{K-1} を与え，出力として N^3 ボクセルのボリューム F を得る．**図1** に n 枚目の投影像 P_n とボリューム F の座標系の関係を示す．図1において，uv 座標系にある平面が投影像を表し，xyz 座標系にある立方体がボリュームである．d' は投影像の中心と X 線源まで

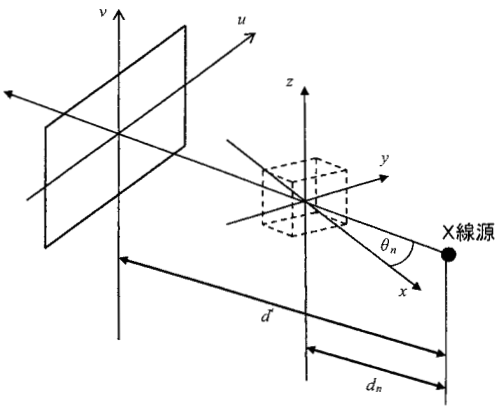


図1 FDK 法における座標系

での距離を表す．d_n はボリュームの中心と X 線源までの距離，θ_n は投影像の中心と X 線源を結んだ直線に対し，xyz 座標系の x 軸がなす角度である．

まず，フィルタリング処理は，投影像の座標 (u, v) の画素値 $P_n(u, v)$ と重み $W_1(u, v, n)$ を積算したものとフィルタ $F(u)$ の u 方向の畳み込み処理である．この処理は次の式 (1) で与えられる．

$$Q_n(u, v) = \sum_{s=-S}^{S} F(s) W(s, v) P_n(s, v) \qquad (1)$$

重みとしては式 (2) を，フィルタとしては式 (3) で与えられる Shepp-Logan フィルタ[7] を用いる．

$$W_1(u, v) = \frac{d'}{\sqrt{d'^2 + u^2 + v^2}} \qquad (2)$$

$$F(u) = \frac{2}{\pi^2 (1 - 4u^2)} \qquad (3)$$

次に逆投影処理は，入力として K 枚の補正済み投影像 Q_0〜Q_{K-1} を与え，出力としてボリューム F を得る．ボリュームの座標 (x, y, z) のボクセル値は次の式 (4) で与えられる．

$$f(x, y, z) = \frac{1}{2\pi K} \sum_{n=0}^{K-1} W_2(x, y, n) \, Q_n(u(x, y, n), v(x, y, z, n)) \qquad (4)$$

式中の値 $Q_n(u, v)$ は補正済み投影像の座標 (u, v) の画素値，$W_2(x, y, n)$ は重みを表している．これらの値は，次の式 (5)〜(7) で与えられる．

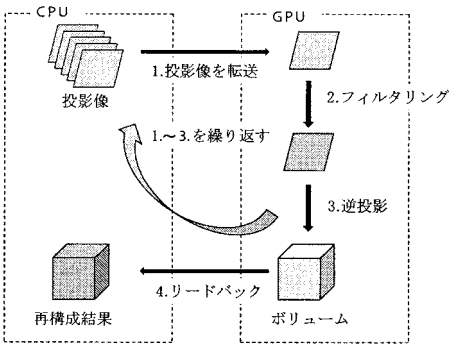


図 2 基本実装の概要

$$u(x,y,n) = \frac{d'(-xsin\theta_n + ycos\theta_n)}{d_n - xcos\theta_n - ysin\theta_n} \quad (5)$$

$$v(x,y,z,n) = \frac{d'z}{d_n - xcos\theta_n - ysin\theta_n} \quad (6)$$

$$W_2(x,y,n) = \left(\frac{d_n}{d_n - xcos\theta_n - ysin\theta_n}\right)^2 \quad (7)$$

4. 提案手法

本章では，提案手法の基となる基本実装を示したのち，逆投影処理の高速化手法を示す．

4.1 基本実装

投影像およびボリュームのデータサイズが大きいため，ビデオメモリ容量の制限により，これらすべてを同時に格納できない．そこで，**図 2** のように，ボリュームを GM に，投影像をメインメモリに格納し，1 枚の投影像を逐一 GM に転送する．その後，フィルタリング処理を実行し補正済み投影像をテクスチャメモリに転送する．最後に逆投影処理を実行する．基本実装では，この一連の処理をすべての投影像について繰り返す．**図 3** に基本実装の擬似コードを示す．なお，実際の実装ではループアンローリングを施し高速化を図っている．

フィルタリング処理では，式 (1) に示したように，u 方向の畳み込み処理をする．そのため，u 方向のデータのみが必要となり，v 方向については独立である．この独立性により，処理を投影像の 1 行ごとに分割できる．基本実装では，1 行の処理を 1 ブロックとする．また，畳み込み処理をする場合，同じデータが何度も必要となる．これは，共有メモリに投影像 1 行分のデータを配列として格納し共有することで，GM へのアクセス回数を削減できる．図 3 の 13～16 行目に示すように各スレッドがロードする画素のインデックス

```
入力：投影像 P_0, P_1,... ,P_{K-1}．フィルタサイズ S および
      投影パラメータ d', d_0, d_1..., d_{K-1},
      θ_0, θ_1,..., θ_{K-1}
出力：ボリューム F
1: Reconstruction()
2: begin
3: for i = 0 to K − 1 do begin
4:    投影像 P_i をグローバルメモリに転送;
5:    Q_i ← Filtering(P_i,K);
6:    補正済み投影像 Q_i をテクスチャとしてバインド;
7:    F ← Backprojection(Q_i,d_i,θ_i);
8:    end;
9: ボリューム F をリードバック;
10: end;
11: function Filtering(P_n,S)
12: begin
13: BID ← ブロック ID;
14: TID ← スレッド ID;
15: u ← INDEX_U(TID);
16: v ← INDEX_V(BID);
17: 共有メモリに配列 P[U] を確保
18: P[u] ← W_1(u,v,n)P_n(u,v);
19: for i = −K to K do begin
20:    Q ← Q + P[u + i]F(i);
21:    end;
22: end;
23: function Backprojection(Q_n,n)
24: BID ← ブロック ID;
25: TID ← スレッド ID;
26: x ← X(BID,TID);
27: y ← Y(BID,TID);
28: u ← u(x,y,n);
29: v ← v(x,y,0,n);
30: v' ← v'(x,y,n);
31: for z = 0 to N − 1 do begin
32:    f_xyz ← W_2(x,y,n)Q_i(u,v);
33:    v ← v + v';
34:    end;
35: end;
```

図 3 基本実装の擬似コード

を計算する．そして，ブロック内の各スレッドが分担して投影像 1 行分のデータを共有メモリの配列にロードする（18 行目）．その後，19～21 行目に示す畳み込み処理を各スレッドが担当する画素について実行する．

次に，逆投影処理について説明する．図 3 の 24～29 行目に示すように，ブロック ID およびスレッド ID から各スレッドが担当するボリュームの座標 (x,y) を計算する．その後，30～34 行目に示すように同じ (x,y) を持つすべてのボクセル値を計算する．(x,y) の値が同じである場合，座標値 u と重み W_2 は変化しない．そのため，座標値 u と重み W_2 は一度計算するだけでよい．また，z 座標に依存する座標値 v についても，最初に式 $v'(x,y,n) = d'/(d_n - x\cos\theta_n - y\sin\theta_n)$ を計算し，その結果を再利用することで計算量を削減できる（30 行目と 33 行目）[3]．ここで計算した座標

(u, v) は整数にはならない．そのため，線形補間を施し画素値を計算する必要がある．そこで，線形補間機能を持つテクスチャメモリに投影像 Q_i を格納する．

4.2 高速化手法

提案手法の逆投影処理における高速化手法は次の5つからなる．

(1) Coalesced 参照の実現：MP 内部の SP が同時にアクセスするボクセルを連続するメモリ領域に格納する．これにより，Coalesced 参照となる条件を満たす．

(2) GM へのアクセス回数の削減：1 回のカーネル実行で処理する投影像の数を 1 枚から 4 枚に増やし，各投影像についての逆投影結果を LM 上の変数に加算する．4 枚の投影像を処理した後に GM に加算するため，GM へのアクセスは 4 回中 1 回で済む．そのため，GM へのアクセス回数を 1/4 に削減できる．

(3) 先行計算によるアクセス遅延の隠蔽：1 回のカーネル実行で処理する投影像数を 12 枚に増やす．具体的には，4 枚単位の投影像の処理を 3 回実行する．これにより，GM へのアクセス遅延およびテクスチャ参照時のアクセス遅延を先行計算で隠蔽できる．

(4) LM へのアクセス回数の削減：4 枚の投影像について，逆投影計算を 1 つの式にまとめる．この変更により，LM へのアクセス回数を 4 回から 1 回に削減できる．

(5) LM へのアクセスの除去：LM へのアクセスを GM へのアクセスに変更する．また，変更した GM へのアクセスで Coalesced 参照を実現する．

上記の高速化手法のうち，(1)，(3) および (5) はアクセス遅延の隠蔽，(2) および (4) がメモリアクセス回数の削減に分類できる．

5. 評価実験

提案手法の性能および再構成結果の画質を評価するために，実機を用いて実験した結果を示す．

5.1 実験方法

実験に用いた PC は，CPU として Core 2 Duo E6850 (3GHz) を持ち，GPU として nVIDIA GeForce 8800 GTX を備える．また，主記憶容量は 4GB であり，ビデオメモリ容量は 768MB である．OS は Windows XP であり，ドライバのバージョンは 169.21 である．また，CUDA 1.1 を用いた．

表 1 に，実験に使用した実装の一覧を示す．実装 A〜F は高速化手法 (1) 〜 (5) を組み合わせたものである．実装 A が基本的な実装，実装 F がすべての提案手法を実装したものである．また，実験に使用する投影像として**表 2** にまとめるデータ 1 および 2 を用いる．データ 1 は Shepp-Logan Phantom (SLP) と呼ばれる仮想的なデータである．データ 2 は動脈瘤を含む人体頭部を撮影した臨床データである．

表 1 各実装の詳細

実装内容	実装					
	A	B	C	D	E	F
Coalesced 参照の実現	×	○	○	○	○	○
GM へのアクセス回数の削減	×	×	○	○	○	○
先行計算によるアクセス遅延の隠蔽	×	×	×	○	○	○
LM へのアクセス回数の削減	×	×	×	×	○	○
LM へのアクセスの除去	×	×	×	×	×	○

表 2 投影像の仕様

パラメータ	データ 1 SLP	データ 2 人体頭部
U：投影像サイズ横（ピクセル）	512	1024
V：投影像サイズ縦（ピクセル）	512	640
K：投影像数（枚）	360	96
N：ボリュームサイズ（ボクセル）	512	512

表 3 既存手法との比較

	実行時間 (s)	投影像/秒
CPU 実装[9]	135.4	2.8
AG-GPU[8]	8.9	40.5
OpenGL 実装[3]	8.7	41.4
AG-GPU w/ EFK[8]	6.8	52.9
提案手法	5.7	63.2

5.2 性能評価

表 3 に，提案手法および既存手法[3),8)] のデータ 1 の実行時間を示す．なお，OpenGL 実装[3)] は，文献中では投影像 308 枚のデータで実験しているが，本稿ではデータ 1 を用いて計測した結果を示す．表 3 を見ると，提案手法が最速であり，文献 8) 中で最速の実装である 6.8 秒より 1.1 秒高速である．この実装は再構成する領域を限定しており，領域を限定できない投影データの場合は実行時間が増加する．再構成する領域を限定せず最も高速なものは OpenGL 実装の 8.3 秒であるが，提案手法はこれより 2.6 秒高速である．

表 4 に，実装 A〜F および OpenGL 実装における実行時間の内訳を示す．提案手法は高速化手法を実装することにより逆投影処理を高速化している．Coalesced 参照[2)] の実装で 9 倍，GM へのアクセス回数の削減で 4 倍，先行計算によるアクセス遅延の隠蔽で 1.7 倍高速化している．また，LM へのアクセス回数

表 4　各実装の実行時間 (s)

内訳	実装						
	A	B	C	D	E	F	OpenGL
初期化	0.1						0.5
投影データ転送	0.3	0.2	0.3	0.2	0.2	0.2	0.4
畳み込み処理	0.7						2.1
逆投影処理	458.6	51.0	12.8	7.6	6.0	4.4	3.2
リードバック	0.3						0.3
その他	—						1.7
合計	460.0	52.3	14.2	8.9	7.4	5.7	8.3

表 5　浮動小数点演算性能および実効帯域幅の比較

	実効演算性能 (GFLOPS)	実効帯域幅 (GB/s)
実装 F	31.2	51.5
OpenGL 実装	99.3	70.8

の削減で 1.3 倍，LM へのアクセスを GM へのアクセスに変更することで 1.4 倍高速化している．まとめると，アクセス遅延の隠蔽で約 21 倍，メモリアクセス回数の削減で約 5 倍高速化しており，全体では約 105 倍高速化している．その中でもアクセス遅延を隠蔽する Coalesced 参照が効果的である．この原因は，他の高速化手法と比べて Coalesced 参照の実現で削減できるクロックサイクル数が多いためだと考えられる．

次に，実装 F と OpenGL 実装を比較する．表 4 を見ると，全体の実行時間は実装 F が高速である．しかし，実行時間の大半を占める逆投影処理に着目すると OpenGL 実装の方が高速である．この原因について，実効演算性能と GPU・ビデオメモリ間の実効帯域幅の観点から考察する．**表 5** に実装 F の逆投影処理における実効演算性能と実効帯域幅を示す．まず，実効演算性能であるが，実装 F は OpenGL 実装と比較して FLOPS 値が小さい．この理由は，計算結果を再利用できる回数が実装 F の方が多いためである．実装 F では，式 (6) の値を 512 回再利用している．一方，OpenGL 実装では 16 回にとどまる．そのため，提案手法における逆投影処理の演算数は OpenGL 実装の半分以下となる．しかし，FDK 法はメモリ集中型のアルゴリズムであり，実行時間の大半を転送時間が占める．そのため，演算数を半分以下にしても実行時間への影響は小さく，FLOPS 値が小さくなる．

次に実効帯域幅について考察する．実装 F は実効帯域幅が 51.5GB/s と理論値の 60%程度であるのに対し，OpenGL 実装は 70.8GB/s と理論値の 71%程度である．この理由として，実装 F ではレジスタ使用量の制限のため MP 上で同時に 1 ブロックしか実行できない．そのため，2 節で述べたようにアクセス遅延を隠蔽できない．一方，単純に比較できないが，OpenGL 実装はレジスタ使用量が 8 個である．そのため，処理するフラグメント[4] を切り替えることでアクセス遅延を隠蔽していると考えられる．

以上の結果より，実装 F の性能ボトルネックについてまとめる．まず，実効演算性能が理論値 345.6GFLOPS の 10%程度であるため，演算部分は逆投影処理の性能ボトルネックではないと考えられる．一方，実効帯域幅は理論値 86.4GB/s の 60%程度である．FDK 法がメモリ集中型アルゴリズムであることを考えると，更に実効帯域幅を出せると考えられる．しかし，MP 内のレジスタ数の制限によりアクセス遅延の隠蔽が不十分である．これが，実装 F が OpenGL 実装と比較して低速である原因であると考えられる．

次に，提案手法と OpenGL 実装の逆投影処理を除く処理を比較する．表 4 を見ると，フィルタリング処理は，実装 F では 0.7 秒であるのに対し，OpenGL 実装では 2.1 秒と 3 倍の時間を要している．この原因として，OpenGL 実装では共有メモリをプログラムで明示的に参照できない．一方，CUDA では共有メモリをプログラムで明示的に参照できるため，GM へのアクセス回数を少なく制御できる．

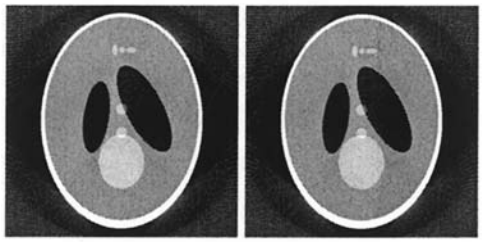
最後にその他について比較する．OpenGL 実装では，その他としてフラグメントプログラムの初期化やベクトル化のためのデータ変換処理を実行し，1.7 秒を要している．一方，実装 F ではこれらの処理はほぼ無視できる．理由として，実装 F はベクトル化を施していない．また，フラグメントプログラムの初期化に対応する処理の時間はほぼ無視できる．これは，CUDA ではパラメータを定数メモリへ高速に転送できるのに対し，OpenGL ではパラメータの転送に時間を要するためである．

以上の結果から，実装 F が OpenGL 実装より高速な理由をまとめると，(1) 共有メモリをプログラムで明示的に参照でき，(2) フラグメントプログラムの初期化が必要ない，という CUDA の特徴を生かして逆投影処理を除く処理時間を高速化したためである．

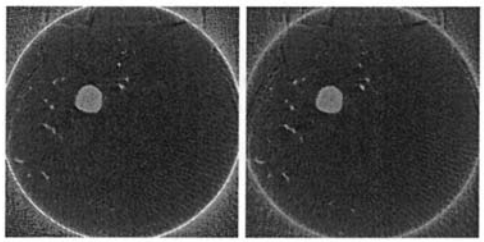
5.3　提案手法の画質評価

提案手法と CPU 実装で再構成したボリュームの画質を比較する．**表 6** に，データ 1 および 2 を再構成したボリュームにおけるボクセル値の誤差比率と PSNR (Peak Signal to Noise Ratio) を示す．PSNR は 40dB を超えると，画質の差を視認できなくなる評価指標である．また，**図 4** に再構成したボリュームの $z = 255$ における断面図を示す．

図 4 を見ると，提案手法と CPU 実装に視認できる差はほとんどない．表 6 を見ると，PSNR も 40dB を



(a) データ 1：Shepp-Logan-Phamtom



(b) データ 2：人体頭部

図 4 投影像の再構成結果（左：CPU；右：提案手法）

表 6 画質の指標

	手法	PSNR	誤差を含むボクセル比率 (%)
データ 1	提案手法	52.6	61.9
	OpenGL 実装	80.1	14.7
データ 2	提案手法	40.2	77.0
	OpenGL 実装	75.0	12.4

超えていて，CPU 実装と比較して視認できる差はなかった．しかし，提案手法ではどちらのデータに対しても全体の 60〜80%ほどのボクセル値が CPU 実装の結果と異なる．一方，OpenGL 実装では，どちらのデータに対しても PSNR が高く，誤差を含むボクセルの比率が 20%以下である．そのため，提案手法と比較して画質が良い．画質が異なる原因については，現在原因を調査中である．

6. ま と め

本稿では，CUDA を用いてコーンビーム再構成を高速化する手法を提案した．提案手法として，GPU 上のメモリに対するアクセス回数を削減した．また，アクセス遅延を隠蔽した．

実験の結果，512^2 画素からなる投影像 360 枚から 512^3 ボクセルのボリュームを再構成するのに 5.7 秒を要した．これは，OpenGL を用いる既存研究[3),8)] よりも高速である．画質についても，PSNR が 40dB を超えている．

しかし，逆投影処理の実行時間は文献 3) のほうが高速であり，GPU の性能も提案手法は理論値の 60%程度しか引き出せていない．また，画質についても誤差を含むボクセルの割合が多いという問題がある．

今後の課題としては，プログラムのレジスタ使用量を削減したいと考えている．また，画質についても誤差を含むボクセルの割合を減らし PSNR を文献 3) と同等にしたいと考えている．

謝辞 本研究の一部は，科学研究費補助金基盤研究 (B) (2) (18300009)，若手研究 (B) (19700061) および大阪大学グローバル COE プログラム「予測医学基盤」の補助による．

参 考 文 献

1) Luebke, D. and Humphreys, G.: How GPUs Work, *Computer*, Vol. 40, No. 2, pp. 96–100 (2007).
2) nVIDIA Corporation: CUDA Programming Guide Version 1.1 (2007). `http://developer.nvidia.com/cuda/`.
3) 吉田征司，伊野文彦，西野和義，萩原兼一：GPU を用いたコーンビーム再構成の性能評価と精度検証，電子情報通信学会技術研究報告，MI2007-104, pp.219–226 (2008).
4) Shreiner, D., Woo, M., Neider, J. and Davis, T.: *OpenGL Programming Guide*, Addison-Wesley, Reading, MA, fifth edition (2005).
5) Feldkamp, L.A., Davis, L.C. and Kress, J.W.: Practical cone-beam algorithm, *J. Optical Society of America*, Vol.1, No.6, pp.612–619 (1984).
6) 李 美花，楊 海圏，小泉和人，工藤博幸：CUDA アーキテクチャを用いた高速コーンビーム CT 画像再構成，*Medical Imaging Technology*, Vol.25, No.4, pp.243–250 (2007).
7) Shepp, L.A. and Logan, B.F.: The Fourier Reconstruction of a Head Section, *IEEE Trans. Nuclear Science*, Vol.21, No.3, pp.21–43 (1974).
8) Xu, F. and Mueller, K.: Real-time 3D computed tomographic reconstruction using commodity graphics hardware, *Physics in Medicine and Biology*, Vol. 52, No. 12, pp. 3405–3419 (2007).
9) Kachelrieß, M., Knaup, M. and Bockenbach, O.: Hyperfast parallel-beam and cone-beam backprojection using the cell general purpose hardware, *Medical Physics*, Vol.34, No.4, pp. 1474–1486 (2007).