RK-007

GPGPU に基づく聴覚ディスプレイシステムにおける音像制御処理の並列化に関する検討

An Investigation on Parallelization of Sound Image Controling Process for Virtual Auditory Display based on GPGPU

及川 祐亮 * 渡邉 貫治 † 高根 昭一 † 佐藤 宗純 † 安倍 幸治 †
Yusuke Oikawa Kanji Watanabe Shouichi Takane Sojun Sato Koji Abe

1. はじめに

音場にある音源から聴取者の耳に到達する音は，音源から直接届く音や，頭部，耳介などにおいて反射・回折した音が重ね合わされたものになる．それらの音を人工的に合成することにより，聴取者に仮想的な音源 (音像) を知覚させるシステムを，音響バーチャルリアリティの分野では聴覚ディスプレイシステムと呼ぶ [1]．代表的な音信号の合成法の一つとして，無響空間内のある位置における音源から鼓膜面までの伝達関数 (HRTF: head-related transfer function)[2] の時間領域表現である頭部インパルス応答 (HRIR: head-related impulse response) を音源信号に畳み込む方法がよく知られている．

聴覚ディスプレイシステムにおいて，移動する音源や聴取者の移動による相対的な音源の移動を再現する場合，聴取者に提示する音の出力が遅れると，音源や聴取者が動いてから遅れて音像が移動することになり，現実の環境を正確に再現できていないことになる．したがって，音源や聴取者の移動を実現する場合，それらの動きに対応した音像の制御をリアルタイムに行う必要がある．音源移動の再現を含む聴覚ディスプレイとしては，矢入ら [3]，Scarpaci ら [4]，大谷ら [5] などが，伝達関数合成法に基づいて，PC(personal computer) を用いたソフトウェア聴覚ディスプレイシステム (CPU 処理) を実装している．これらのシステムは，畳み込む HRIR をリアルタイムに切り替えることで頭部運動に伴う音源移動の再現を行っており，例えば矢入らのシステムにおいては全体の遅延が 8.3 ms であると報告されている．これらのシステムで処理している音源数は 1 であるが，一般的な音空間においては複数の音源が存在するため，聴覚ディスプレイシステムの実用性をより高めるためには複数の音像を制御できることが望ましい．しかし，一般的な PC の持つ CPU のコア数は多くても 6 程度であるため，多音源となった場合に音源数の増加にほぼ比例して処理時間が増加してしまい，小さな遅延での動作を実現できなくなることが予想される．そこで，本研究では，多音源に対する演算を高速に処理するために，GPU(graphics processing units) を用いた並列処理によって聴覚ディスプレイシステムを構築することを目標とする．本論文では，構築したシステムにおける音像制御処理の処理速度を計測し，聴覚ディスプレイシステムに GPU を用いることの有効性について検討を行う．

2. GPGPU に基づく聴覚ディスプレイシステム

2.1 聴覚ディスプレイシステムの概要

聴覚ディスプレイシステムにおいて，聴取者や音源の移動に追随して音像を制御するためには，可能な限り短い処理時間で演算を行う必要がある．そのためには，1 サンプルの入力毎に演算を行うことが理想であるが，本研究のシステムでは，メモリ転送やプログラム動作のオーバヘッドなどを考慮し，有限ポイントからなるブロック単位で演算を行い出力を得るシステムとして設計することとする．

本研究で使用している NVIDIA 社製 GPU(GeForce GTX295) では 1242MHz のクロック周波数で動作するコアを 480 個搭載し，それぞれが 1 クロックにつき 2 回の浮動小数点演算能力を持つので，最大で 2 × 1242 MHz × 480 コア ≈ 1.2 TFLOPS 程度の演算能力を持つ．GPU によって並列化される演算の単位はスレッドと呼ばれ，コア内に最大で約 2×10^{12} 個のスレッドを同時に持つことができる [6]．これを利用して，畳み込み演算をできるだけ高い効率で並列化することを考える．ただし，聴覚ディスプレイシステムには，分岐処理等の GPU による並列化に適さない処理が存在するので，適した処理のみを GPU に負担させることで，全ての処理を GPU に行わせるよりも効率的なシステムを構築する．

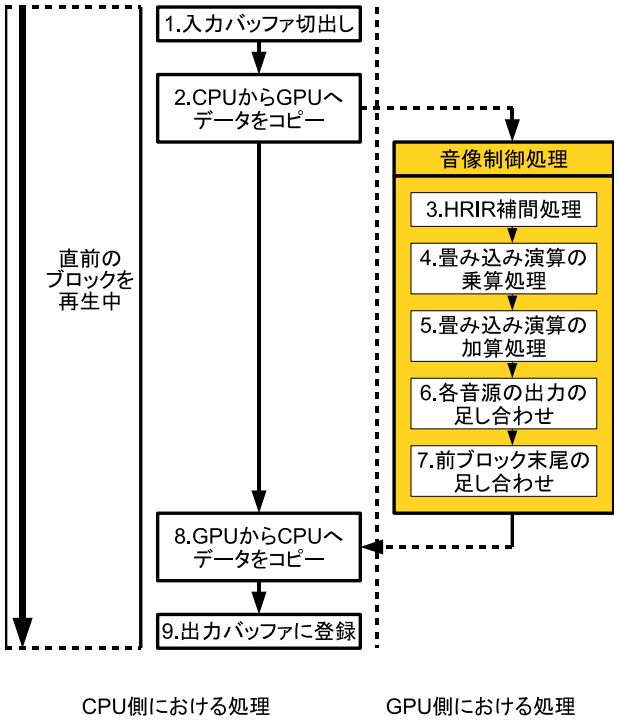

図 1 聴覚ディスプレイシステムにおける 1 ブロック内の CPU と GPU の処理の流れ

2.2 GPU による音像制御処理の実装

2.1 節で述べた処理を，図 1 のように分けてプログラムを開発する．図 1 は，ある 1 ブロックの演算において CPU，GPU が行っているそれぞれの処理の流れを示したものである．CPU は入力信号の切り出しと転送，出力信号の再生を行う．音像制御処理は GPU が行い，その過程は 5 つの段階に分けられる．図 1 中の 3 は，離散的な音源方向の HRIR から任意の音源方向の HRIR を算出するための補間処理である．4 は畳み込みの乗算，5 は畳み込みの加算である．6 は各音源に対する畳み込み処理の結果を足し合わせることによる出力の算出，7 は，畳み込みによって発生する直前のブロックの

* 秋田県立大学大学院 システム科学技術研究科
† 秋田県立大学 システム科学技術学部

末尾を現在のブロックに足し合わせることで，最終的な出力信号の算出を行う．畳み込み処理の加算，乗算は順不同であり，さらに音源ごとの処理も独立して扱えばよいので，各段階内の処理は並列に行うことができる．一方，3〜7 の段階同士は，前段の出力が確定しなければ演算を始められないため，並列化を行うことはできない．これらの処理を行っている間，CPU は直前のブロックで処理したデータを再生し続けているため，これを再生し終えるまでに現在のブロックに対する処理を完了できれば，途切れることなく音を再生し続けることができる．

3. GPGPU による音像制御処理の処理時間の測定

3.1 測定方法と条件

本システムは，複数音源に対する処理を行ってもリアルタイム性を損なわないことを目標としている．そこで本節では，処理する音源数に対する処理時間を計測し，現状のシステムにおいて処理可能な音源数を明らかにする．

また，音源数に対して処理時間が一定となっているか，すなわち演算が並列に行われているかどうかを確認する．なお，本実験における “音源数” とは，仮想音源のことであり，1 つの仮想音源は，音源信号に左右耳の HRIR を畳み込んで得られる 2ch の信号からなる．実験は，図 1 の 2〜8 の処理に対して，処理の開始時刻と終了時刻を 1 ms の精度で取得し，それらの差を取ることで処理にかかる時間を計測した．なお，実際には，一回の処理に対する時間が上記の精度と比べて短すぎるため，2〜8 の処理を 1000 回繰り返して実行し，1 回あたりの処理時間を求めた．測定を行った PC は，表 1 に示すスペックを持つものを用いた．また，実験条件を表 2 に示す．

3.2 計測結果

図 2 に処理時間の計測結果を示す．図の横軸は処理する音源数，縦軸は処理時間を表す．本システムは，理想的には並列演算が行われて処理時間が一定時間になるように設計されているが，図 2 からは，音源数の増加に伴う処理時間の単調増加が見られる．この原因を明らかにするため，GPU が関わる処理のうち，特に処理に時間を必要とすると考えられる，

- CPU から GPU への入力信号の転送
- 畳み込みの乗算処理
- 畳み込みの加算処理

の 3 つの処理について，音源数に対する処理時間の計測を行った．なお，補間やその他の処理については，簡易的に演算時間を計測した予備実験の結果，全体の処理時間に大きな影響が見られなかったため，計測対象から除外することとした．図 3 に，システム全体の処理 (“All”)，CPU から GPU への入力信号の転送 (“Memory Copy”)，畳み込みの乗算処理 (“Multiplication”)，畳み込みの加算処理 (“Summation”) の音源数に対するそれぞれの処理時間を示す．横軸は処理する音源数，縦軸は処理時間である．図 3 の “Memory Copy” の結果から，CPU から GPU への入力信号の転送にかかる時間は，他の処理と比較し無視できるほど小さいことが確認できる．一方，“Multiplication”，“Summation” の結果において，音源数の増加に伴う処理時間の単調増加が見られる．このことから，畳み込みの乗算処理及び加算処理が処理時間の大部分を占めているといえる．この 2 つの処理には並列演算を適用しているので，理論的には一定時間での処理となるはずであるが，実際には演算時間の単調増加が確認された．

表 1　実験に用いた PC の主なスペック

CPU	Intel Core2 Duo E8600 3.33 GHz	
メモリ	2 GByte	
グラフィックボード	型番	MSI N295GTX M2D1792
	GPU	NVIDIA GeForce GTX 295 1242 MHz
	メモリ	GDDR3 1792 MB
	バス	PCI-Express × 16
OS	Windows XP Professional SP2	

表 2　実験条件

ブロック長	2000 ポイント (1ch)
HRIR 長	512 ポイント (2ch)
サンプリング周波数	44100 Hz
1 ポイントのサイズ	4byte(float 型)

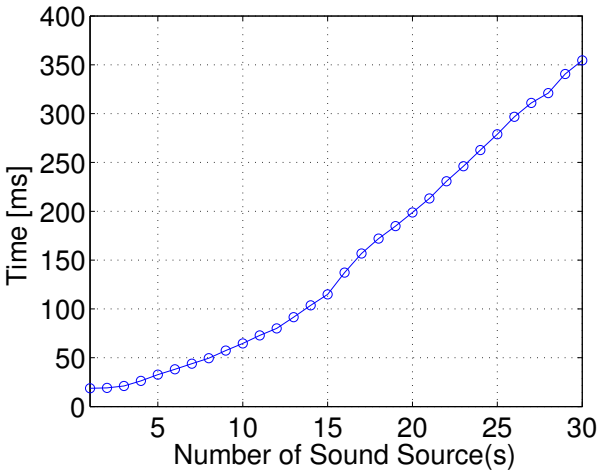

図 2　音源数に対するシステム全体の処理時間の変化

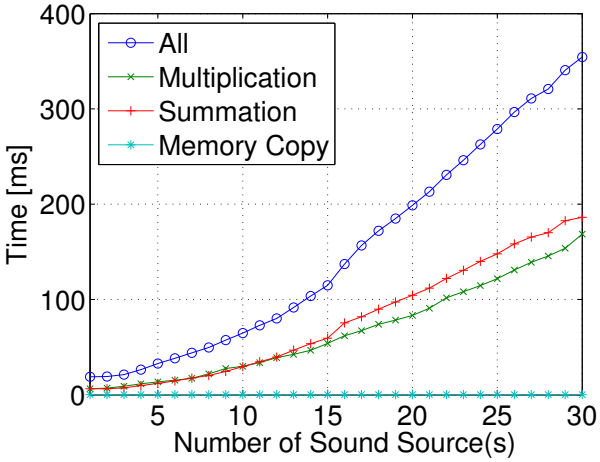

図 3　音源数に対する主要な処理の処理時間の比較

3.3 畳み込みの乗算処理における並列処理の検証とメモリアクセス時間の測定

GPU 上の処理は厳密には演算だけではなく，演算に必要なデータを保持しているメモリへのアクセスが発生する．このことをふまえて 3.2 節の結果を詳細に検討するために，並列演算が行われているかを確かめる実験と，メモリアクセスの処理時間に対する影響の検証を行った．

図 4 は，NVIDIA 社の GPU におけるメモリモデルの概念図である [7]．図中のグレーの矩形がメモリを表す．各矢印は，そのメモリに対してどこからアクセス可能かを示している．まず，GPU による並列演算が行われているかを検証する

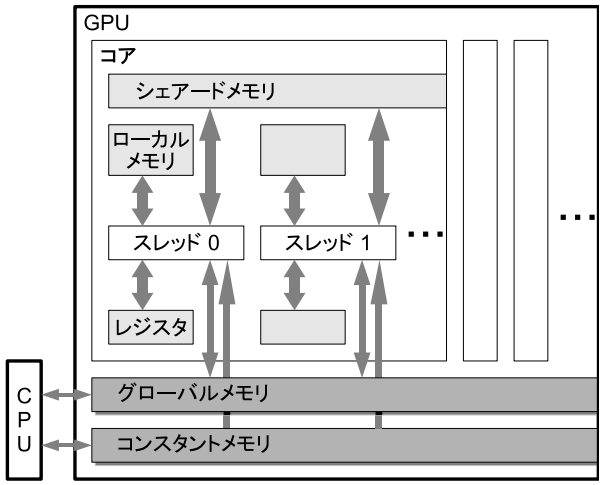

図 4　NVIDIA 社の GPU のメモリモデル [7]

ため，畳み込みの乗算処理の演算からメモリアクセスを除き，3.1 節と同様の条件で計測する．ただし，メモリアクセスを発生させずに畳み込みの乗算と同等の演算負荷をかけるため，図 4 に示されている各スレッドがレジスタのみにアクセスするように乗算を行わせる．次に，メモリアクセスを高速にすることで処理速度が改善するかを確かめるために，図 1 の 3 で生成した補間済み HRIR のデータを，GPU のコンスタントメモリに移して，同様に処理時間を計測する．コンスタントメモリとは，GPU に搭載されているメモリの一種で，図 4 に示すように，CPU 側からのみ書き込み制御が可能である．グローバルメモリと比較して容量が小さいという制限があるが，レジスタと同等な速度でコアとのアクセスが可能である [8] ため，HRIR のデータをコンスタントメモリに置くことにより処理速度が改善することが予想される．なお，ここで乗算処理のみを扱っているのは，コンスタントメモリの容量が小さく，大きな実装の変更をしない限り補間済み HRIR のデータしか格納できなかったためである．また，コンスタントメモリにデータを転送する処理が加わるため，図 1 の構成とは異なったシステムとなるが，追加された処理の処理時間に対する影響は十分に小さいことを予備実験で確認している．

図 5 は，メモリアクセスが発生する場合としない場合の音源数に対する処理時間の比較である．図の “Multiplication” は畳み込みの乗算処理の処理時間を，“Multiplication(No Memory Access)” はそれと同等の演算回数を持つ乗算処理を GPU 上で動かした場合の処理時間を示している．図 5 では，同等回数の乗算を行いながらも，メモリアクセスを除いた場合に処理時間が下回ったことから，乗算演算そのものは並列に行われていることが確認できる．なお，処理時間が 8 音源ごとに階段状に上昇しているが，これは，並列演算と言っても完全に同時に行われるものではなく，厳密には GPU 内部では一定の処理単位ごとに処理が行われているためであると思われる [9]．また，図 6 は，コンスタントメモリを用いた場合と用いない場合の音源数に対する処理時間の比較である．図の “Global Memory”はグローバルメモリを用いた場合の処理時間，“Constant memory” は高速メモリを用いた場合の処理時間を示している．図 6 の結果から，高速なコンスタントメモリを用いた場合に明らかな処理時間の改善が見られる結果となった．

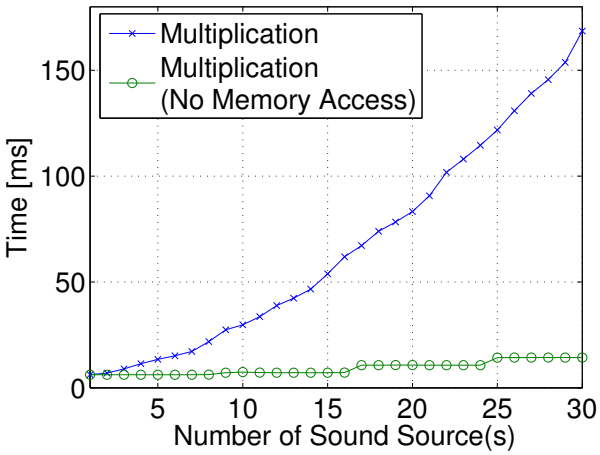

図 5　音源数によるメモリアクセスを伴う乗算処理および乗算演算のみの処理時間の変化

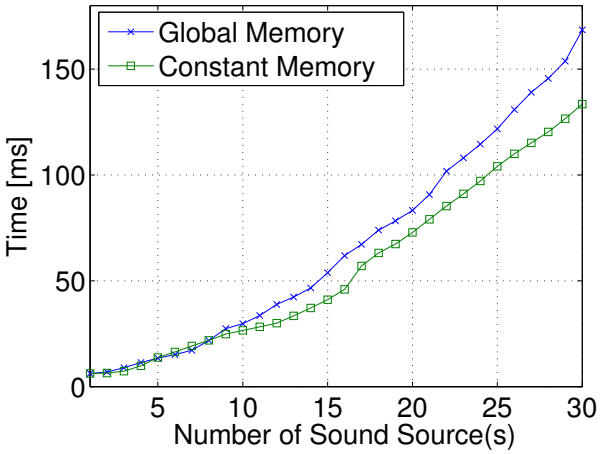

図 6　音源数によるグローバルメモリおよびコンスタントメモリを用いた畳み込みの乗算処理時間の変化

4. 考察

表 2 から，本実験において 1 ブロックの時間長は $2000/44100 \approx 45 \times 10^{-3} = 45$ ms となる．また，図 2 から，音源数 7 個で約 45 ms となることから，7 音源以下であれば直前のブロックのデータを再生し切る前に畳み込みを終えられることがわかる．よって，本システムは現状で最大で 7 音源程度の同時処理を行えると言える．

3.3 節では各処理における処理時間の計測を行った．図 3 から，畳み込みの乗算処理と加算処理に時間を要していたことが確認された．現在のシステムでは，1 音源に対する処理に，左右のチャンネルに対応する 2 コアを与えて処理しているため，7 音源であれば 14 コアを稼動させている．ここで，表 1 より GPU クロックが 1242MHz であることから，理論値としては 2×1242 MHz $\times 14$ コア ≈ 35 GFLOPS 程度の処理が可能であるが，結果から実際の処理能力を計算すると，(2000 ポイント $\times$ 512 ポイント $\times$ 2 ch $\times$ 7 音源$\times$ 13 ステップ)/45 ms ≈ 8.2 GFLOPS となり，CUDA の処理能力を最大まで出し切れていない．そこで，より詳しく検証した結果，図 5 から乗算演算そのものは並列に演算できているが，図 6 に示すようにメモリのアクセス時間によって処理時間が増加していると思われる結果となった．この原

因として，メモリアクセスが並列的にできなくなる箇所が多数発生していることが考えられる． 図 7 は，メモリアクセスが並列に行えなくなるような場合の例を示している． 通常の場合，GPU 上でのデータの転送は，メモリのある範囲に対して図 7(a) のスレッド A，B のようにまとめて行われる． ところが，図 7(b) は，○印のメモリアドレスにアクセスを求める場合に，スレッド B’ はスレッド A と同時に転送できる範囲を超えた領域にアクセスを行っているため，スレッド A と B’ のアクセスが逐次的になる．本システムの畳み込み演算においては，表 2 から 1 ループの処理内で最大 $512 \times 2000 \times 4$[byte] $= 4096000$[byte] のデータを扱うことになる．NVIDIA 社製 GPU において同時に転送できる単位は 128 byte であるから [10]，最低でも $4096000/128 = 32000$ 回の逐次的な処理が行われていることとなる．また，図 7(c) のような，複数箇所からの同時アクセスによっても，アクセスの順番待ちが発生し，やはり処理が逐次的になる．本システムの畳み込みは以下の式に基づいて行っている．

$$y(n) = \sum_{k=0}^{N-1} x(k)h(n-k) \tag{1}$$

ここで，$y(n)$ は出力信号，$x(n)$ は入力信号，$h(n)$ は HRIR を表している．つまり

$$\begin{aligned} y(0) &= x(0)h(0) \\ y(1) &= x(0)h(1) + x(1)h(0) \\ y(2) &= x(0)h(2) + x(1)h(1) + x(2)h(0) \\ &\cdots \end{aligned} \tag{2}$$

で示される乗算と加算を並列に行っている．しかしながら，この式から明らかなように実際には同じ値 (例えば $x(0)$) を格納したメモリへのアクセスが同時に発生することになる．このようなメモリに関する局所的な逐次処理が多く発生しているために，音源数の増加にともなってアクセスにかかる時間も増加していると考えられる．

図 7(b) の問題に対する解決法として，処理中のバッファを小さくしメモリ境界をまたぐ確率を減らすことが考えられる．図 7(c) の問題に対しては，近年の GPU では最低でも数百 MB 単位のメモリを搭載していることから，あえて同一のデータを複数の領域に持つようにすることでアクセスの衝突を避けることなどが考えられる．さらに，メモリアクセス速度全体を底上げするものとして，3.3 節で用いたような GPU 独自の高速なメモリを活かす設計を行うことが考えられる．

上記の解決法を実装する例として，本システムでは現在使用していない，図 4 のシェアードメモリを利用する設計を行うことが考えられる．このメモリは，現在のアーキテクチャではひとつのコアにつき 16 kbyte という容量の制限があるが，理論的な性能としてはレジスタと同等の帯域を持つメモリである．そこで，1 ブロックのデータに対しシェアードメモリ内で処理できる単位に分けて演算を行うことにより，バッファを小さくする，高速メモリを用いる，という条件を満たすことができる．これに，同一のデータを複数の領域に持つことによるアクセス衝突の回避を組み合わせることで，より高速に処理を行うことができると考えられる．ただし，分割数が多すぎると，プログラム中の関数呼び出し回数増加による遅延など，他の要因が影響してくることも考えられるため，適切なブロックサイズとバッファサイズの決定には試行錯誤が必要であると思われる．

5. まとめ

GPU に基づく並列処理を用いた聴覚ディスプレイシステムを開発し，処理速度の観点から性能の評価を行った．その結果，本システムは 7 音源の同時畳み込みをリアルタイムに行えることを確認した．また，各処理における処理時間を詳細に検討することで，畳み込み演算を GPU で行う場合は，メモリアクセスが処理遅延の主な原因となることを明らかにした．並列処理に関しては，同時転送可能なメモリの範囲を超えたアクセスや畳み込み処理で発生するメモリアクセスの衝突といった，局所的な逐次処理の積み重ねを解決することで，性能の向上が期待できる．今後は，4 節で提案したメモリアクセスによる遅延 の改善法を実装し，本論文と同様の評価を行うことが課題としてあげられる，また，1 節で GPU を用いたシステムの例をあげたが，マルチコア CPU の登場によって，さらにシステム性能の向上が予想されるため，それらのシステムとの比較を行い，GPU を用いたシステムの有用性を検証する必要があると考えられる．

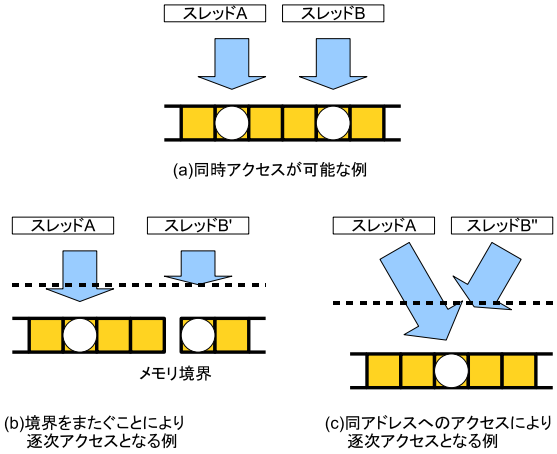

図 7　メモリアクセスが並列に行われない例

謝辞

本研究は科研費 (21700140) の助成を受け行われた．

参考文献

[1] D. R. Begault, *3-D sound for virtual reality and multimedia* (AP professional, Massachusetts, 1994).
[2] J. Blauert, *Spatial Hearing* (The MIT Press, London, 1983).
[3] 矢入聡，岩谷幸雄，鈴木陽一，“頭部運動感応型ソフトウェア聴覚ディスプレイの開発，” 日本バーチャルリアリティ学会論文誌，**11**(3), 437-446, (2006).
[4] J. W. Scarpaci, J. A. White, and H. S. Colburn, “A system for real-time virtual auditory space,” in *Proc. 11th Intl. Conf. on Auditory Display*, July (2005).
[5] 大谷真，平原達也，“ Windows 上で動作する動的聴覚ディスプレイ，” 日本音響学会 2007 年春期研究発表会 講演論文集，711-712 (2007).
[6] *NVIDIA CUDA Reference Manual Version 3.0* (NVIDIA Corporation, California, 2010).
[7] 青木尊之，額田彰，はじめての *CUDA* プログラミング (工学社， 東京， 2009).
[8] 小山田耕二 (監修)，岡田賢治 (著)，*CUDA* 高速 *GPU* プログラミング入門 (秀和システム, 東京， 2010).
[9] Jason Sanders, Edword Kandort *CUDA by Example* 汎用 *GPU* プログラミング入門 (インプレスジャパン， 東京， 2011).
[10] David B. Kirk，Wen-mei W. Hwu，*CUDA* プログラミング実践講座 超並列プロセッサにおけるプログラミング手法 (誠信書房， 東京， 2010).