2008年度　卒　業　論　文

指向性による処理軽減とGPUによる分散処理を用いた音のリアルタイム合成の研究

指導教員：三上 浩司 講師

メディア学部 ゲームサイエンス
学籍番号　M0105423
松永 崇宏

2008 年度　　卒　業　論　文　概　要

論文題目

指向性による処理軽減と GPU による 分散処理を用いた音のリアルタイム合成の研究

メディア学部 **学籍番号：M0105423**	**氏名**	松永 崇宏	**指導教員**	三上 浩司 講師

キーワード	音、リアルタイム合成、CUDA、 指向性、GPGPU

ゲームのグラフィック技術の向上につれ、ゲームサウンドも高品質なものが求められてきている。多チャンネル化や多彩なエフェクト処理、同時発音数の増加といった傾向は CPU 処理に更なる負担をもたらしている。ゲーム機や PC には高機能なサウンド用ハードウェアはコストの関係上搭載されていない場合が多い。また、サウンドハードウェアには同時発音数に上限が存在し、設定以上の数の音が鳴らせないという問題もある。

本研究ではサウンドカードの同時発音数の上限を音をインタラクティブに合成して出力することにより解決し、その処理を GPU で分散処理をすることを目的とした。同時に、音の合成方法として指向性の概念を用いて三次元空間をエリア分けしてエリア毎に音を合成するという方法を提案した。同時発音数を増やすために予め GPU で音をリアルタイム合成してまとめ、それを再生することにより同時発音数をサウンドカードの限界以上鳴らす事に成功した。音の合成には波の基本原理である「重ね合わせの原理」を用い、複数の音を加算してひとつにまとめた。

これらの理論を用いて、提案手法を CPU と GPU で処理する場合の計算速度の差を実験した。2 個の音を 1 つにまとめる実験では GPU で処理する方が CPU で処理するより 30 倍高速で、32 個の音を 1 つにまとめる実験では GPU で処理する方が CPU で処理するより 60 倍高速であった。また、提案手法を用いたリアルタイム合成により生成されたサウンドデータは、シーケンサー等で合成したサウンドデータと全く同じ品質であることが分かった。

これらの結果、CPU 負荷が低減されると同時に、GPU でのサウンド処理が有効である事を証明し、品質も実用に耐える事が分かった。

目 次

図 目 次

第 1 章

はじめに

1.1 研究背景

最近のテレビゲームのサウンドには、多チャンネル化や多彩なエフェクト処理、同時発音数の増加といったような傾向が見られる [1][2]。そのような中、熱心なゲームユーザーはハイクオリティなサウンド環境を求めて性能の良いサウンドボード [3][4] を搭載するようになってきている。グラフィックがよりリアルな表現に近づいて来ているため、サウンドにもよりリアルな表現が求められるようになってきている。近年高品質なサウンドを生み出すために多チャンネル化の一環としてサラウンドと呼ばれる技術が使われるようになった。ドルビーデジタルサウンドのようにステレオサウンドを 5.1ch サウンドに出力するような技術 [5] もあり、新たなハードを増設せずともドルビーデジタル対応のスピーカーがあればサラウンドが実現出来るため、テレビゲームではよく採用されている。テレビゲームは、インタラクティブに周囲から音が鳴らすことにより、より高いサウンドの表現力を手に入れている。

多彩なエフェクト等を発音している音すべてにかけようとすると計算コストが増える。そのため CPU パワーを使わずにサウンドを処理するために DSP（デジタルシグナルプロセッサ）が使われている。高度なエフェクト処理等は計算コストがかかるため、DSP で処理することにより CPU を別の処理に利用できるという利点がある。また、ハードウェアで処理をする事によりハードならではの高速

なレスポンスを得ることも出来る。DSP を搭載する事は生産コストが増加するため、ソフトウェアで処理する機能を搭載するゲーム機も多い。PC は Creative 社の SoundBlaster という製品が事実上の標準サウンドカードとして定着していたが、初期の頃の PC は音の入出力機能はもってはいなかった。近年高性能なサウンド機能を搭載した PC が増えてはきているが、ゲーム用途として PC を使用する場合にはサウンドカードを搭載したほうがゲームの描画速度が高速化する。Windows Vista では，ゲーム内のサウンド再生に当たって，DSP を用いたハードウェアアクセラレーションが廃止されている。

1.2 問題点

現状のゲーム機や PC は同時発音数に限りがある [6] という問題がある。そして、3D サウンド等の高機能で高コストなサウンド処理をゲームで利用しようとすると他の処理の足かせになる。たとえ 256 音同時発音が出来たとしても、ソフトウェアで一つ一つにエフェクト処理をかける事は計算コストが高い。現在はより高度なグラフィック表現のゲームが多くなってきているため、CPU の処理能力もよりグラフィックの処理に取られる傾向にある。サウンドの処理も重くなり同時発音数が多くなるゲームも増えてきているが、サウンドのハードウェアの仕様はあまり見直されていないのが現状である。

同時発音数の上限以上の音数はならせないため、複数の音源をあらかじめ一つにまとめたものを状況に応じて再生しなければならない。現状のゲームサウンドはそのような方針で製品開発が行われているが、ゲーム内の状況が変わるたびにその状況専用の音源を用意せねばならない。また、ゲームでは遠くで鳴っている音等は鳴らさないといった処理を行うことにより同時発音数をなるべく抑えるといった方法 [1] がとられたりしている。

1.3 研究目的

本研究ではハードウェアの同時発音数の限界以上の音を鳴らし、それを CPU 以外で分散処理をすることを目的とする。同時発音数を上げるためにインタラクティブに音を合成してまとめた上でサウンドデバイスに送ることにより同時発音数を増やし、インタラクティブな環境音を作成する。GPGPU[7][8] という GPU[9] を汎用プログラミングで使うことが出来る技術を利用し、GPU で音の合成処理を行うことで CPU の負荷を軽減する。

GPGPU を利用した研究には現状では処理が複雑な HPC（ハイパフォーマンスコンピューティング）分野 [10] や、既存手法の高速化 [11] といったものが多い。GPU を使ったサウンドの研究は、音場の研究であったり音声認識に関する研究がほとんどで、ゲームサウンドと直接関係のある GPU を使用した研究は少ない。3DCG から風切り音等を生成するという研究が西田ら [12] によって行われていたが、ゲームで使うには計算量が膨大で効果も限定的であった。本研究ではゲームで使える音声合成手法と合成方法の提案を行う。

音声合成をするとき、本研究では 3 次元空間上にある複数のオブジェクトが出す音をある程度まとめサウンドデバイスで再生する事になるが、「どの音をまとめるのか」という問題も出てくる。前方の音と後方の音を一つにしたのではリアリティが損なわれる。本研究ではサラウンド制作ハンドブック [13][14] に記載されている音声制作の方法を参考にしながら、ゲーム向けに指向性の概念を用いた 3 次元空間上の音の合成方法を提案する。ガンマイクと呼ばれる、ある一定の方向からの音を集中的に拾うマイクがありテレビの撮影現場等で広く使われている。その概念を使用して、そのようにある一定の方向や地域の音をまとめる事によりサラウンド対応をしていく。

音の再生には OpenAL[15] といわれる API を使う。OpenAL とはクロスプラットフォームのオーディオ API で、マルチチャンネル 3 次元定位オーディオを効率よく表現するように設計されている。本研究ではそういった API を利用できるも

のを作る。API との親和性を重視することにより既存のゲームに利用できるようにする。

第 2 章

従来の手法と調査

本研究では同時発音数を上げるために予め複数の音声ストリーミングを一つにまとめる。それについて必要な技術の従来手法や問題点等について挙げていく。

2.1 音の合成

音声の合成の基本は「重ね合わせの原理」である。音とは波の性質を持っており、二つの波が重なったときの振幅は単に二つの波の振幅の和になる。逆位相の音同士を合成すると音は消去される。鈴木ら [16] の研究によると、さまざまな方向からでも逆位相の音波であれば音が打ち消される事も実験されている。重ね合わせの原理を使い、二つの音を単純に加算する事によって合成した音を作る事とする。

あらかじめ複数の音楽ファイルを一つにまとめるが、ただ単純に加算していったのでは再生機器の再生できる音量を超えてしまうので音声のボリュームが量子化ビット数に応じた上限を超えないように飽和処理をする必要がある。これらの処理はサウンドカードの DSP がもともと処理をする分野なのだが、GPU で処理をする以上そのような機能は GPU にはついていない。

重ね合わせの原理の公式は式 2.1 のように表す。

$$y = y_1 + y_2 = f_1(x;t) + f_2(x;t) \tag{2.1}$$

波 $y_1 = f_1(x;t)$ と波 $y_2 = f_2(x;t)$ が同時に存在するときにそれらの波の合成波は式 2.1 のような単純な代数和で表すことができる。

2.2 音楽制作ソフトについて

シーケンサと呼ばれる音楽制作の現場で使われるソフトウェアがあり、図 2.1 や図 2.2 のような UI をしている。図 2.2 の Cubase や ProTools といったようなソフトは今日の楽曲制作には欠かせない存在で、ゲームサウンド制作の現場でも使われている。これらのソフトウェアには様々な機能がついていて複数の音声ファイルや MIDI ファイル等の合成が行えるのだが、リバーブやエコー等のエフェクトやボリュームの変更、さらにはピッチとよばれる音高の調節まで出来る。

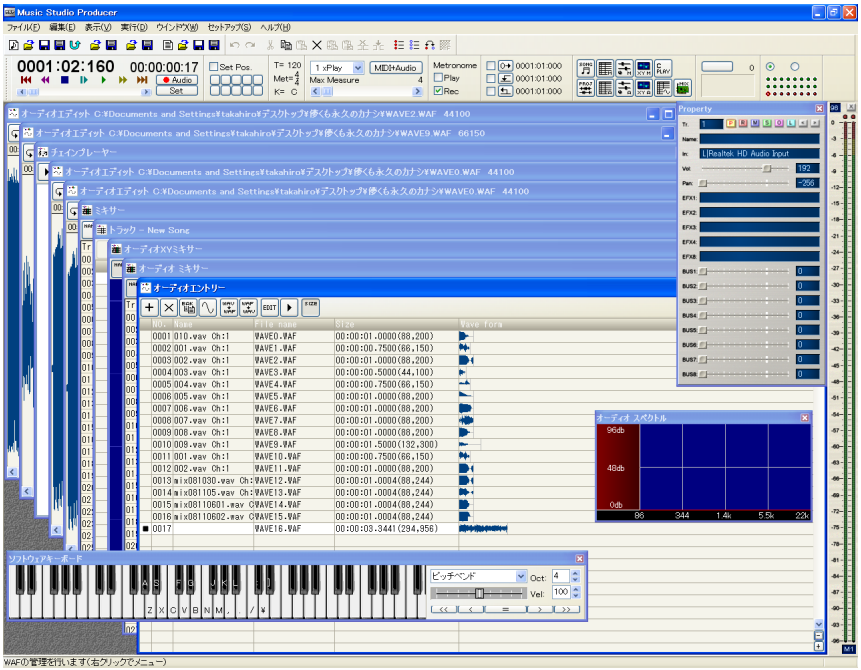


図 2.1: MusicProducer（Free）

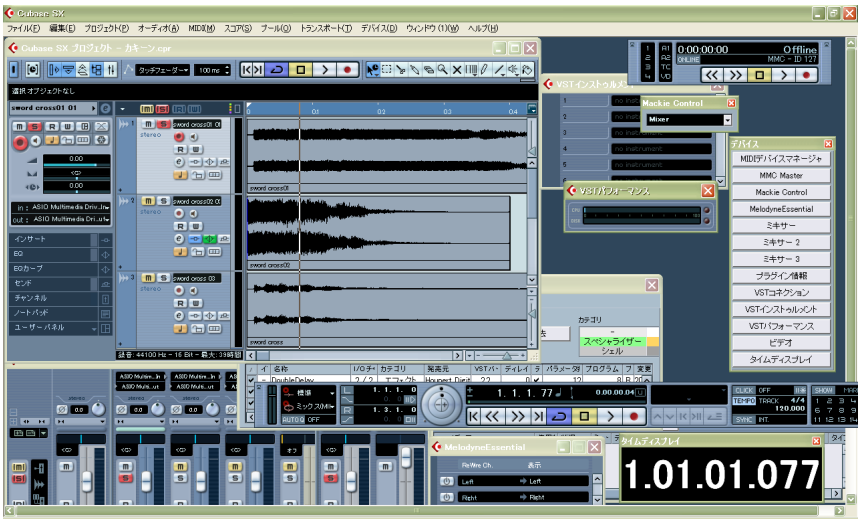


図 2.2: Cubase SL 3

これらの音声を最終的に一つの音楽ファイルとして出力することを「ミックスダウン」と言う。これも基本的には各トラックを重ね合わせの原理により合成している。

今回の研究で制作する音楽合成技術は、この単純なミックスダウン機能となる。

2.3 ゲームサウンドについて

例えばシューティングゲームにおいて画面上に埋め尽くされた敵が一斉に倒された時に音を鳴らそうとすると、同時発音数の限界を超える数のサウンドエフェクトを鳴らさなければならない。そのようなときにゲームにおいてはある程度の数の音だけ鳴らして残りは省略するといった手法がとられる。本研究ではすべての音を鳴らせるようにすることで、音を省略する事なく臨場感のある音場を形成する。

第 3 章

研究手法

3.1　合成のアルゴリズム

3.1.1　扱う音のフォーマットについて

本研究では、サウンドを鳴らすために OpenAL というサウンド API を使用する。そこでサラウンドをするためには音声のフォーマットがある程度限られるため、ここに扱う音声フォーマットを記述する。

- PCM ファイル（wav ファイル）
- サンプリングレート　44.1khz（44100hz）
- チャンネル数　モノラル（1 チャンネル）
- ビットレート　705kbps（88125Byte/sec）
- 長さ　2 秒
- 量子化ビット深度　16bit

サラウンド対応するにあたり、モノラルサウンドしかサラウンド化できない仕様が OpenAL にあるためこのようなフォーマットとする。サウンドエフェクトをオブジェクトのアクションごとに鳴らして環境音をインタラクティブに作るというシチュエーションが本研究でもっとも想定している状況であるため長さを 2 秒と決めている。

量子化ビット深度を 16 ビットと設定しているので、0〜65535 の 65536 段階で音量を表わす。実際には波の計算なので-32768〜32767 の範囲で数字を扱うようにする必要がある。

3.1.2 PCM ファイルとアナログ信号

音の合成をするにあたって PCM ファイル [17] と音声のアナログ信号について説明する。PCM とは Pulse code modulation（パルス符号変換化変調）といい、アナログ信号から、信号に含まれる周波数成分の二倍以上のサンプリング周波数でサンプル値を取り出し、この標本値を時系列的に連続した 1,0 のパルス列によって量子化して表現する方式である。図 3.1 の左側はアナログ信号データを標本化するイメージ図で、右側は標本化されたデータを量子化するイメージ図である。

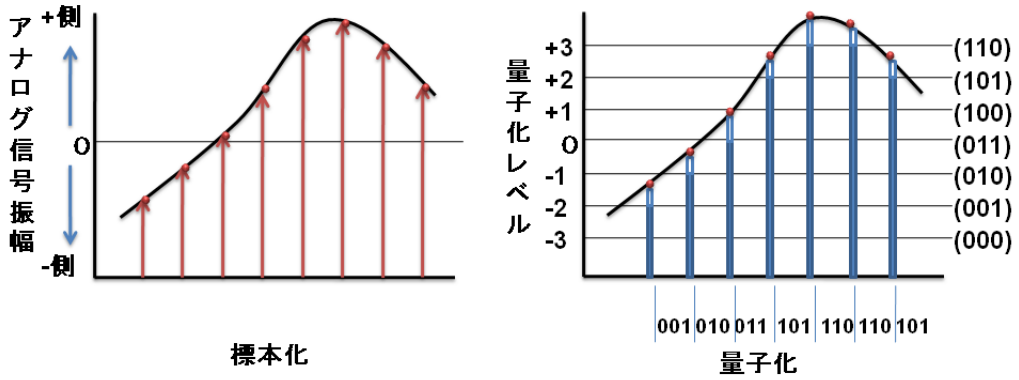


図 3.1: PCM（Pulse Code Modulation）の説明

PCM ファイルをスピーカーから出力する際には量子化されたデータをアナログ信号に変換して出力する。そのためコンピュータ上では量子化しているファイルを加工する。wav ファイルはバイナリファイルであるため、そのまま加工すればよい。

3.1.3 音のアルゴリズム

音の合成は前章で挙げたとおり「重ね合わせの原理」を使用する。本研究では実験的に扱う音のフォーマットを予め決める事とする。合成する際は一度音声をメモリー上にロードした後にバイナリデータを数値化し、タイムライン毎に加算をする。上が 32767 大きい場合は 32767 を代入、下が-32768 より小さい場合は-32768 を代入するという飽和処理をかける。このように飽和処理をかけないと、数字が反転してしまい、32768 は-32767 となってしまい音が割れてしまう。

PCM ファイルを読み込んでそれを再生する場合ファイルの音の構成部分をまず読み込み、その後波形データ部分を時間に合わせて再生していく。音の構成部分はファイル情報が記述されているので合成は行わず、波形データ部分のみを合成することとする。図 3.2 は研究のアルゴリズムを簡略化した図である。図 3.2 は同じタイミングで音が鳴った場合であるが、音が鳴るタイミングが異なる場合には重なっている部分のみを加算して図 3.3 のように合成する。

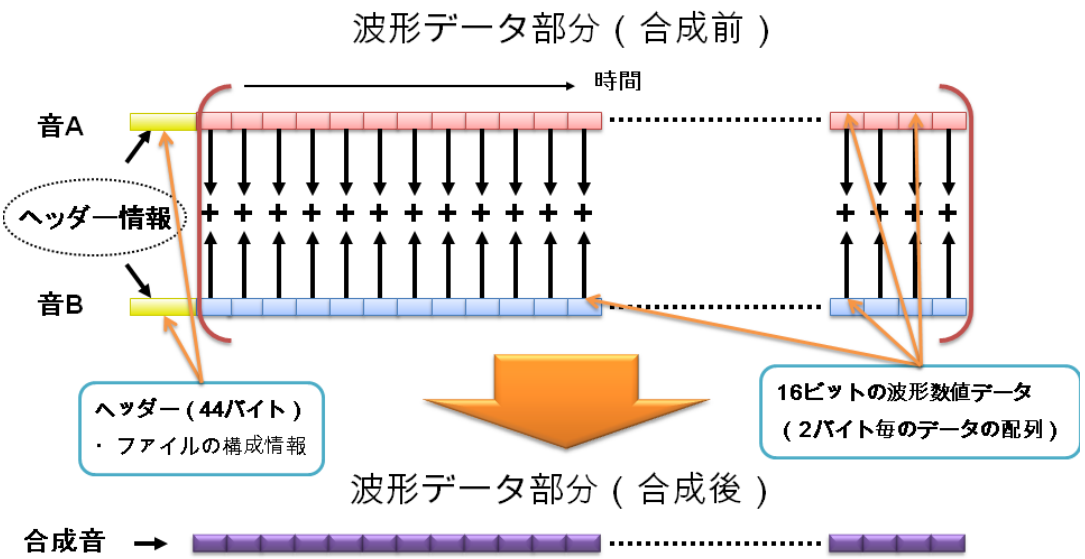


図 3.2: 音の合成方法図 1

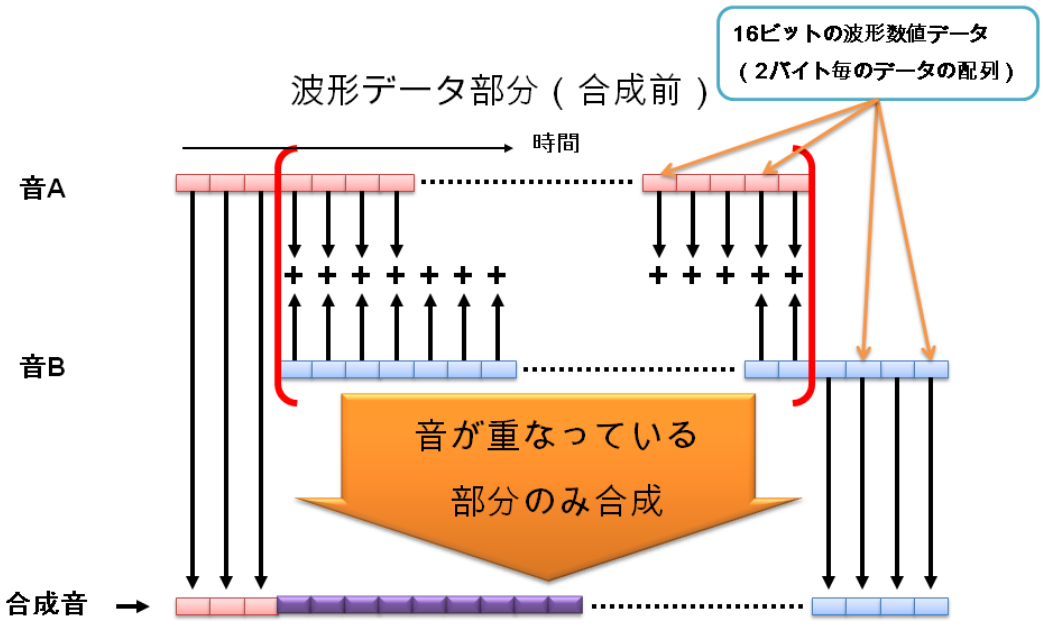


図 3.3: 音の合成方法図 2

3.2　合成のための空間のエリア分け

音を合成するために空間のエリア分けをするのだが、2つルールに従って合成をおこなう。1つ目のルールはBGMは合成をせずに独立して再生する。BGMには空間のある一定の場所から鳴っているという概念がないので合成を行わない。2つ目は、音を受け取るレシーバーからの距離が近いところでなっている音は任意で合成する。提案手法で音を合成する場合、音が遠くから鳴っているか近くから鳴っているかが曖昧になる。方向に関しても厳密ではなくなるため、クリエイターがどうしても聞いてほしい音や重要だと思う音は合成しないことで逆に聞かせたい音が明確にできる。

合成のための空間のエリア分けは、エリアをグリッドに分けて合成する方法と方角によって分ける方法の2種類を提案する。

1つ目の方法は高さを含まない平面上をグリッドで分ける方法である。図3.4のようにエリアを一定の間隔で分け、その中に入っている音を出すオブジェクトが出した音すべてを合成し再生する。近くの音や個別に慣らしたい音はたとえエリアの中に入っていても合成せず再生する。この方法であれば方向、距離による音の減衰のどちらもある程度正確に再現することができる。しかし、フィールドがあまりに広い場合にはより多くのグリッド分けをするか一つ一つのグリッドを大きくしなければならない。グリッドを多くすると同時再生数の上限を超える可能性があるし、グリッドを大きくすると方向や距離による減衰も不正確になる。

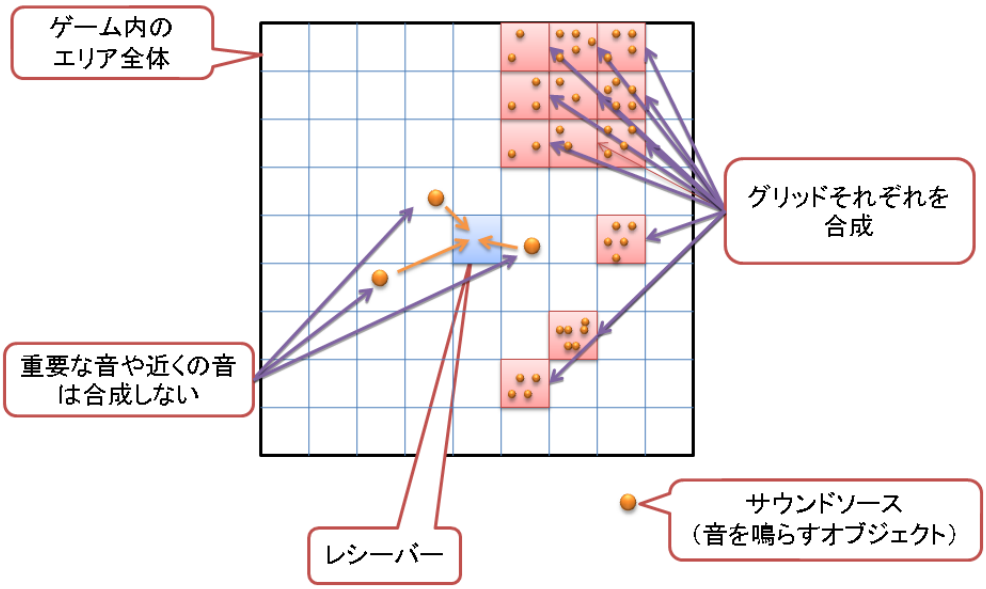


図 3.4: 音の合成エリア分けの図　グリッド分け

2 つ目の方法は、レシーバーを中心に角度でフィールドを放射状に分ける方法である。図 3.5 はこれをより分かりやすくした図で、このように角度の割合はユーザーが自由に選択しても良い。レシーバーから後方の音はすべてまとめ、前方を細かく分けるといった方法も可能である。こちらも近くの音や個別に鳴らしたい音はたとえエリアの中に入っていても合成せず再生する。この方法であれば方向の再現性はかなり正確になる。しかし、距離による減衰の再現が若干不正確になる可能性がある。本研究ではゲームが想定する状況によってエリア分けの方法を変える事を提案する。

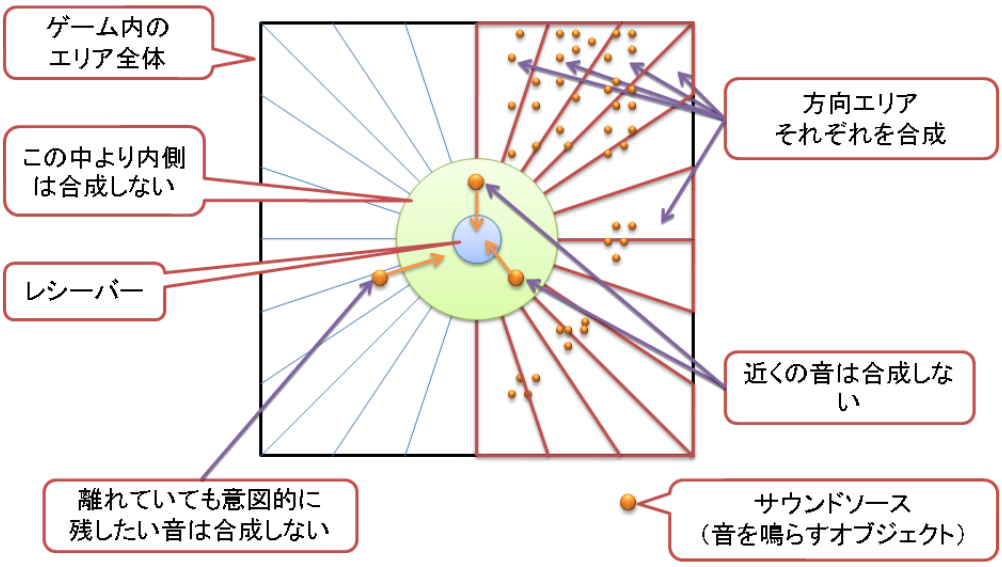


図 3.5: 音の合成エリア分けの図　放射状にエリア分け

複数の音源を合成した場合、その音源の座標をどこに設定するかが問題である。そこで、グリッドで分ける方法ではグリッドの中心座標とし、方角によって分ける方法では合成するオブジェクトの中で一番近い音の座標とする。後者が前者と方法が違う理由は、後者の場合非常に遠くの音とある程度近くにある音を一緒にする場合があり、この時エリアの中心とした場合に実際になっている音よりも音量が小さくなる可能性があるためである。

3.3 GPUでの分散処理

3.3.1 CUDAについて

CUDA[18][19] とはnVidia社が提供するGPUの汎用プログラミング言語で、GPUをグラフィック以外の要素に使用できるようにする事が出来る言語である。GPUは単純なデータを一度に大量に処理することに特化しているため、今回の単純な加算の場合は向いているため採用した。繰り返し処理を沢山やらなければならないため、GPUの方がCPUよりも向いている。

汎用プログラミングをGPUで行うことをGPGPUと言い、nVidia以外にもAMDがATI Stream[20]というGPGPU開発環境の提供を行っている。また、OpenCL[21]という並列処理に特化されたCプログラミング言語ベースの並列コンピューティングのためのフレームワークもある。こちらはGPUだけでなくマルチCPUやPLAYSTATION3のCPUであるCell[22]等の並列コンピューティングに利用される。OpenCLは後々のスタンダードになるであろうから、このシステムで仕組みを作る事により再利用に役立つことが期待出来る。

3.3.2 GPUの担当部分

CUDAは繰り返し処理は得意ではあるが分岐予測などが貧弱であるため、この点を考慮してプログラミングをしなければならない。今回は以下の処理のみを担当させる。

- 波形数値データの加算
- 飽和処理

3.4 実装

3.4.1 開発環境

開発環境は以下の通りである

- OS: Microsoft Windows XP Professional Service Pack 3
- CPU: Intel Core2Duo E8500 @3.16Ghz
- Memory: PC2-6400 4GB
- GPU:GeForce 9800 GTX （memory 512MB）
- サウンド:オンボード（ALC8XX シリーズ）
- 開発ソフト:Microsoft Visual Basic 2005
- 開発言語:C++ CUDA
- 最大同時発音数　32 音

3.4.2 CPU での実装

C++で作ったプログラムの処理の流れは以下のようになる。ユーザのサウンド再生命令が入力されたら、サウンドファイルをメインメモリに読み込む。その時、前のサウンドの再生が終了していないか、ユーザで二つ以上の音を再生する命令が出ているかを判定し、Yes なら更に前のサウンドファイルの再生が終了しているかを判定する。終了していなければ新規入力サウンドファイルと前のサウンドファイルの未再生部分を合成して再生する。前のサウンドファイルの再生も終了していて、2 つ以上同時に再生命令が出されていなければそのままサウンドファイルを再生する。図 3.6 は処理の流れを図解したものである。

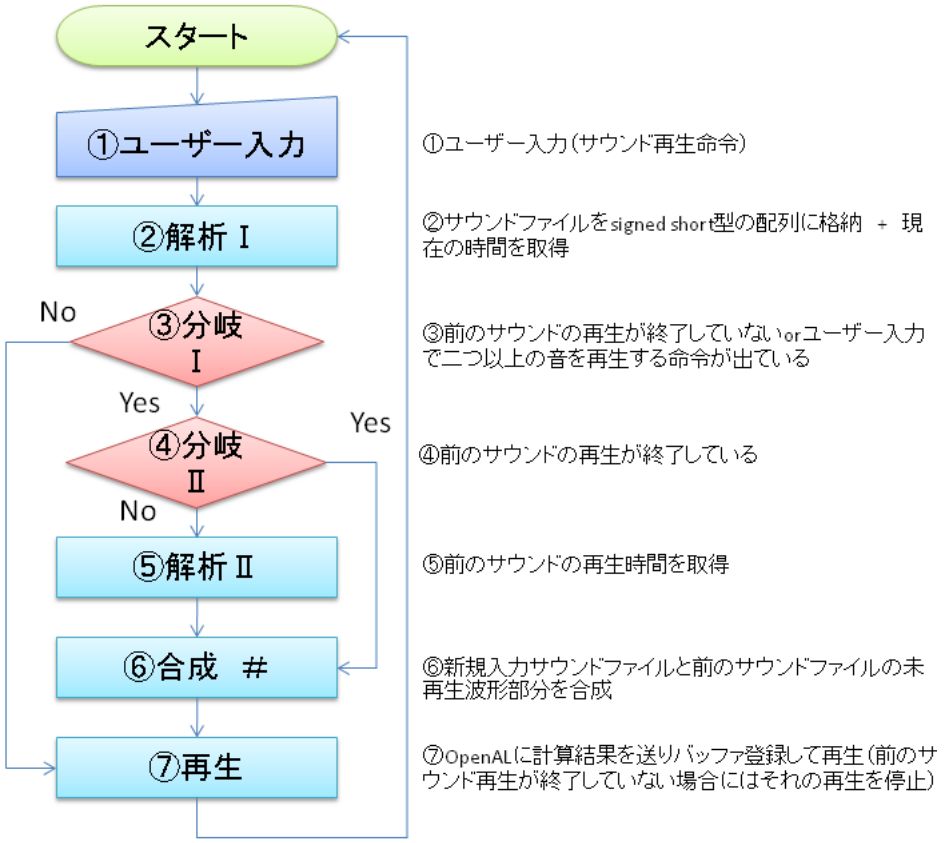


図 3.6: CPU での合成図　フローチャート

3.4.3　GPU での実装

CUDA を使用して GPU で処理した場合には、新たに GPU へのサウンドデータ転送とメインメモリへのデータ転送が新たに加わる。それ以外は CPU でのメインのフローチャートとあまり変わらない。図 3.7 は GPU で合成をした場合を図解したものである。

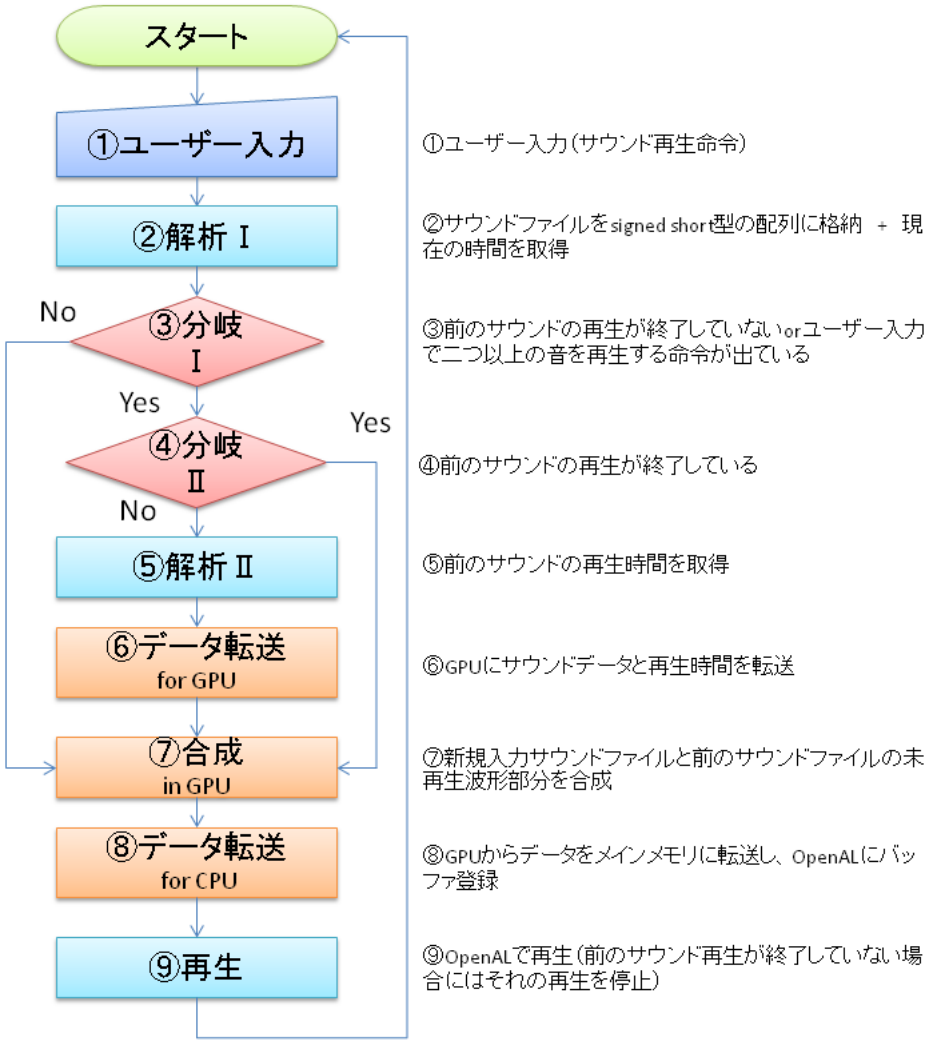


図 3.7: GPU での合成図　フローチャート

第 4 章

検証・評価

CPU と GPU それぞれの合成方法で実験と検証をする。

4.1 再生実験

グリッドによる 3D 空間分けの合成手法を使用して音声を合成し再生した。開発環境は第三章の実装の項目で記した環境で実装を行った。図 4.1 は実際に作成したツールの GUI である。

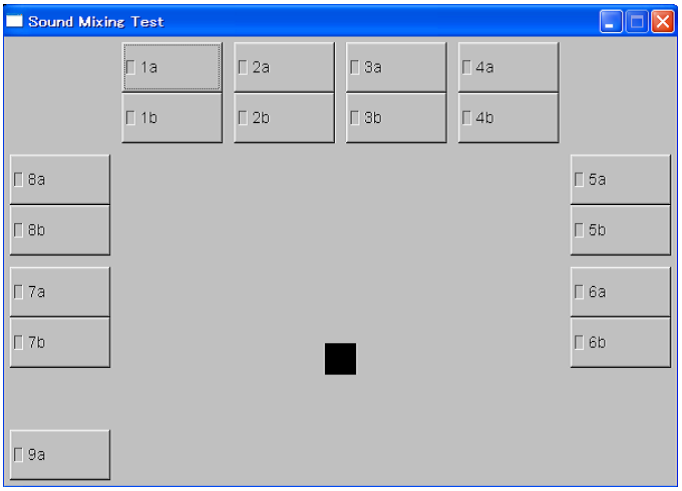


図 4.1: 実験用音声合成再生ツール

本実験では、80 個の音源を 3D 空間上に配置をし、グリッド分けの合成方法を使用して音を合成するプログラムを作成した。図 4.2 は今回の実験での音源の配置のイメージ図である。10 と書かれているエリアの中に音源を 10 個配置し、図 4.2 のようにそれを 8 エリアに配置した。本ツールでは、この音源配置図を模して図 4.1 のようにボタンを配置した。黒の正方形をリスナーポジションに見立て、合成エリアの位置にボタンを配置した。80 個の音源を配置することによって、従来のように音をそのまま再生しようとすると音は再生できず、音を合成して再生すれば 80 個すべての音が再生できる仕様である。

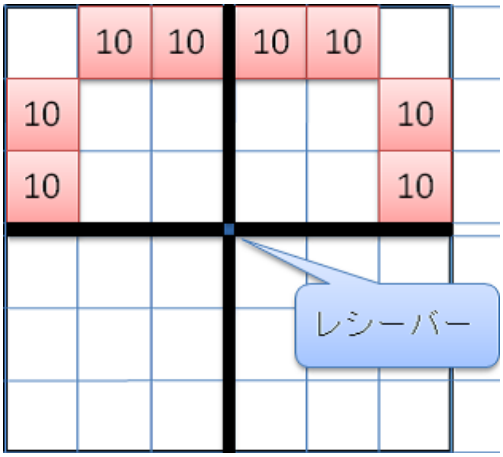


図 4.2: ツールの音源の配置図

合成せず再生をするボタンには小文字の a をつけ、合成をして再生するボタンには小文字の b を付与した。例えば 1a を押すと合成せず 10 個の音を再生し、1b を押すと 10 個の音を合成して再生をする。他のボタンも同様の法則でボタンを配置している。音再生時にはチェックボックスが黄色に塗りつぶされる仕様である。図 4.3 は音再生時のツールの画像である。

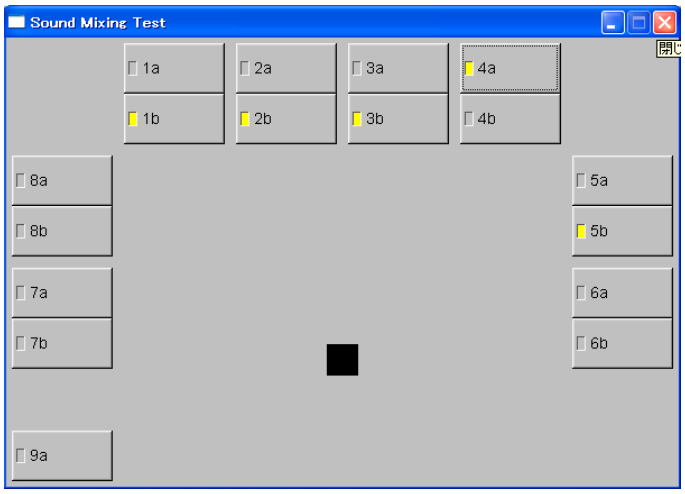


図 4.3: 再生時のツールの画像

このツールを使用して、同時発音数が増えたかどうかを、1a から 8a までをすべて押した場合と、1b から 8b までをすべて押した場合とで比べてみた。1a から 8a までのボタンを押すと、40 個以上再生しようとしたところで同時発音数に変化が見られなくなった。1b から 8b までのボタンを押した場合、同時発音数は 80 個まで達した。このことから、提案手法を使用すると同時発音数が増加することが証明された。

4.2 計算速度の CPU と GPU の差

提案手法での計算速度を測定した。合成後の音が実際に発音される音の数ではあるが、複数の音をまとめているため実際には合成前の音数分聞こえていることになる。図 4.4 は合成単位とその単位を用いた合成前後の音数をまとめたものである。この条件で計算をし計算時間を測定した。100 回の計算時間の平均を算出する。

合成方法	合成前		合成後				
2⇒1	2	⇒	1	8⇒1	8	⇒	1
	4	⇒	2		16	⇒	2
	8	⇒	4		32	⇒	4
	16	⇒	8		64	⇒	8
	32	⇒	16		128	⇒	16
	64	⇒	32		256	⇒	32
4⇒1	4	⇒	1	16⇒1	16	⇒	1
	8	⇒	2		32	⇒	2
	16	⇒	4		64	⇒	4
	32	⇒	8		128	⇒	8
	64	⇒	16		256	⇒	16
	128	⇒	32		512	⇒	32
				32⇒1	32	⇒	1
					64	⇒	2
					128	⇒	4
					256	⇒	8
					512	⇒	16
					1024	⇒	32

図 4.4: 実験表

全合成結果は図 4.5 に示す。合成前の音が多ければ多いほど計算量も増えている。実験結果は CPU で処理するよりも GPU で処理したほうが約 30 倍〜60 倍程高速な事が分かり、計算量が多い程高速であった。1024 個を 32 個にする実験でも 1 ミリ秒以下と非常に高速で、CPU で合成するとフレームレート 60FPS 換算で 3 フレーム程かかるのに対し GPU だと 1 フレーム未満と実用に耐えうる。

	単位 ミリ秒		
合成方法	CPU (ms)	GPU (ms)	倍率(CPU/GPU)
2⇒1	0.224138	0.007619	29.41829636
4⇒2	0.392721	0.013222	29.70208743
8⇒4	0.785587	0.024316	32.30741076
16⇒8	1.576676	0.045975	34.29420337
32⇒16	3.141828	0.090929	34.55254099
64⇒32	6.283145	0.17928	35.0465473

	CPU (ms)	GPU (ms)	倍率(CPU/GPU)
4⇒1	0.247536	0.008639	28.65331636
8⇒2	0.493411	0.017144	28.78038964
16⇒4	0.984461	0.028489	34.55582857
32⇒8	1.969546	0.055265	35.63821587
64⇒16	3.939115	0.108125	36.43112139
128⇒32	7.880421	0.214925	36.66591136

	CPU (ms)	GPU (ms)	倍率(CPU/GPU)
8⇒1	0.583069	0.012022	48.50016636
16⇒2	1.003833	0.021158	47.44460724
32⇒4	2.004949	0.039885	50.26824621
64⇒8	4.005231	0.079485	50.38977166
128⇒16	8.014381	0.15855	50.54797225
256⇒32	16.056381	0.317062	50.64113959

	CPU (ms)	GPU (ms)	倍率(CPU/GPU)
16⇒1	1.003346	0.017554	57.15768486
32⇒2	1.992096	0.033247	59.9180678
64⇒4	4.030532	0.064497	62.49177481
128⇒8	7.942026	0.126904	62.58294459
256⇒16	15.914141	0.251921	63.17115683
512⇒32	31.870712	0.504404	63.18489148

	CPU (ms)	GPU (ms)	倍率(CPU/GPU)
32⇒1	2.018944	0.032718	61.70743933
64⇒2	4.061766	0.063245	64.22272116
128⇒4	7.880467	0.124612	63.24003306
256⇒8	15.846869	0.249287	63.56877414
512⇒16	31.526305	0.490592	64.26175926
1024⇒32	62.988389	0.984124	64.00452484

図 4.5: CPU を用いた処理と GPU を用いた処理の全計算結果の比較

まず、合成後の音数に注目してデータを検証していく。計算時間は計算量に比例して伸びていく事が分かった。2 個を一つに束ねるより 32 個を一つに束ねる方が時間がかかる。図 4.6 はこれを分かりやすく示した図で、2 個を 1 つにまとめるより 32 個を 1 つにまとめる方が計算量が多い。計算時間をわかりやすくグラフで表したものが図 4.7 である。2 個を 1 つにまとめるより 32 個を 1 つにまとめる方が時間がかかるが、計算量が多い方が高速化されている

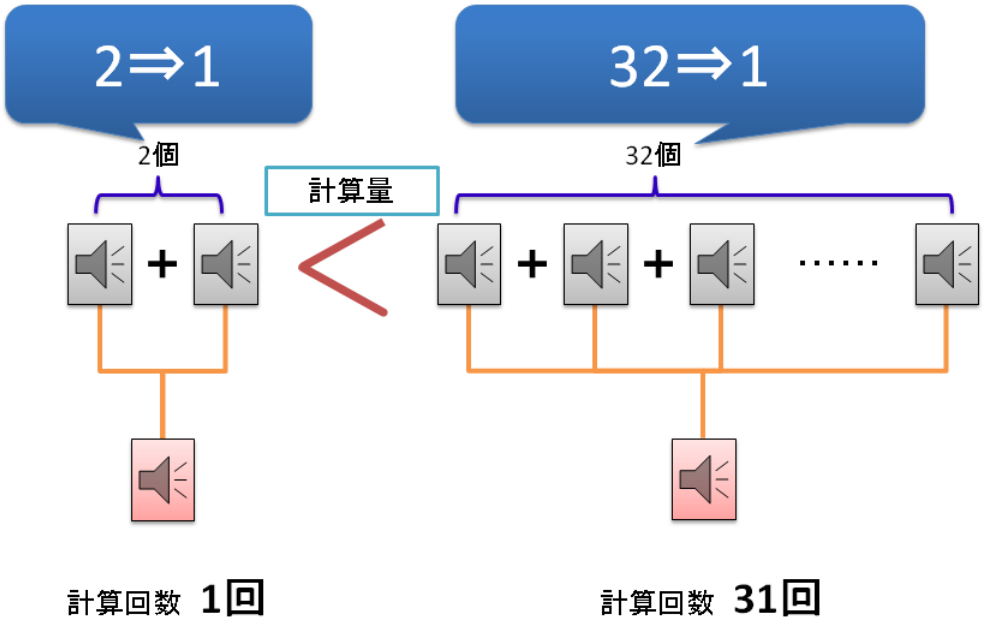
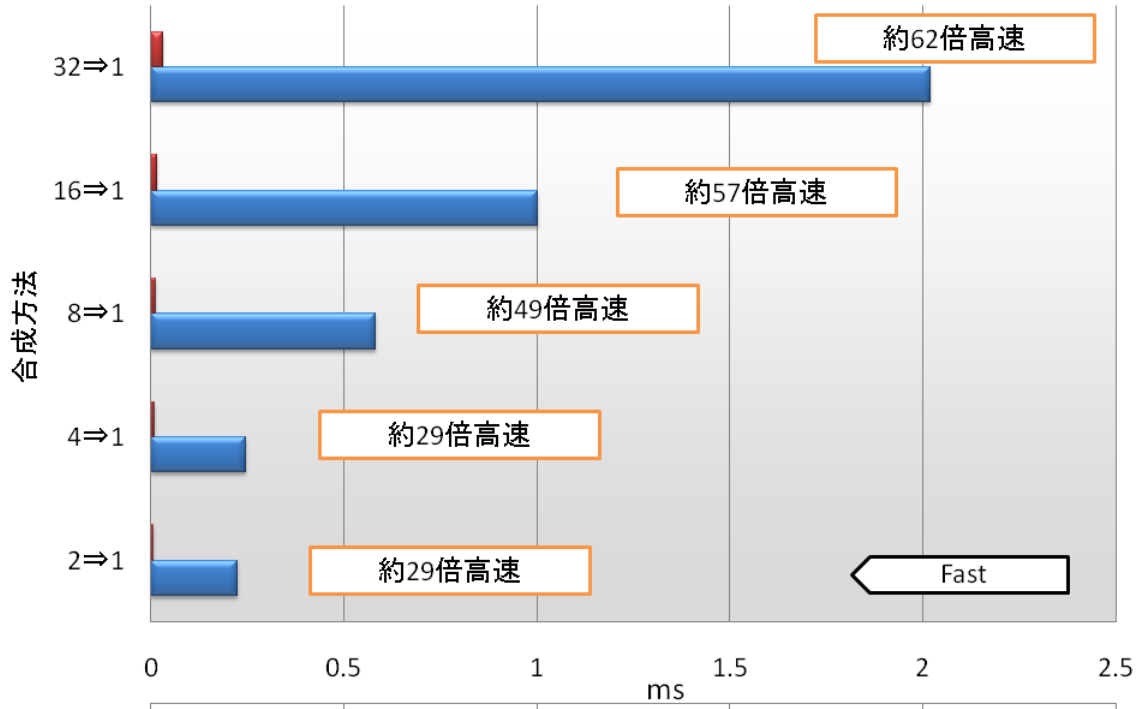


図 4.6: 合成前の音数の違いによる音を一つにまとめる際の計算時間の変化



	2⇒1	4⇒1	8⇒1	16⇒1	32⇒1
GPU (ms)	0.007619	0.008639	0.012022	0.017554	0.032718
CPU (ms)	0.224138	0.247536	0.583069	1.003346	2.018944

図 4.7: 2 ⇒ 1 から 32 ⇒ 1 までの計算時間のグラフ

次に、合成前の音数を固定して合成後の音数を変えることにより計算時間に変化が生まれるか検証する。図 4.8 はこれを分かりやすく示した図である。計算量合成前の音数が 32 個と 64 個の GPU での合成時間のデータを抜き出して図 4.9 にまとめた。32 個の音を 2 個ずつ 16 個に束ねる方が 32 個ずつ 1 個に束ねるより時間がかかった。図 4.10 はまとめたデータを棒グラフ化してみて時間を比べてみた図である。合成前の音数を倍にすると計算時間も倍かかっていることが図 4.10 を見ると分かる。

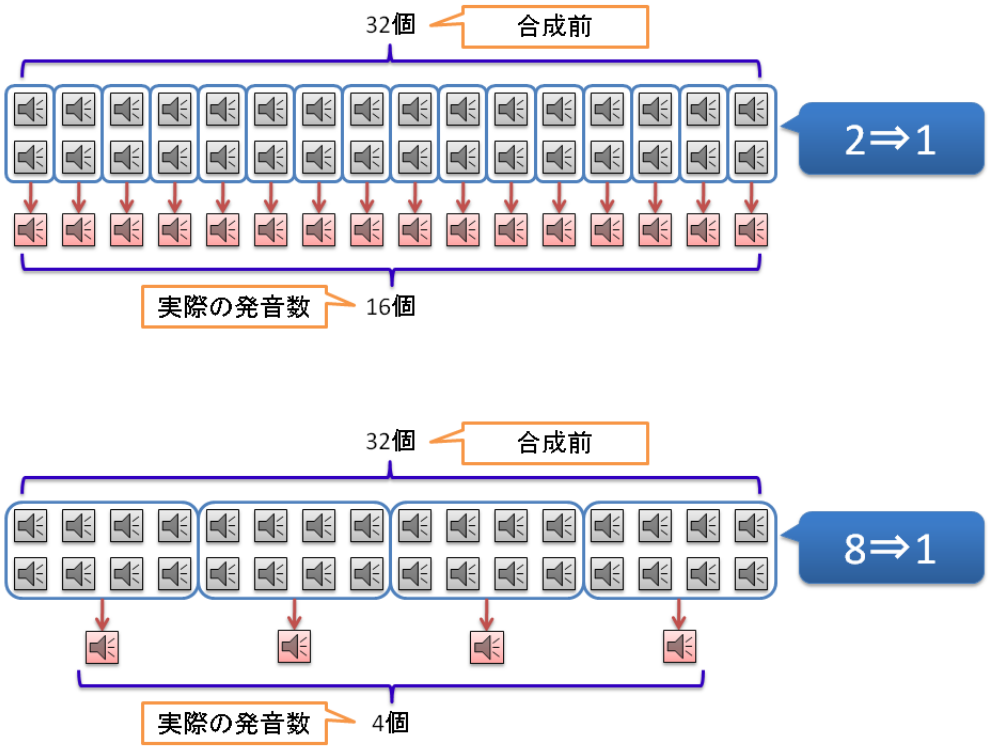


図 4.8: 合成前の音数を固定して束ねる音数を変化させる時の解説図

	合成前		合成後	計算時間（ミリ秒）	合成方法
合成前の音数を固定 32個	32	⇒	1	0.032718	32⇒1
	32	⇒	2	0.033247	16⇒1
	32	⇒	4	0.039885	8⇒1
	32	⇒	8	0.055265	4⇒1
	32	⇒	16	0.090929	2⇒1
合成前の音数を固定 64個	64	⇒	2	0.063245	32⇒1
	64	⇒	4	0.064497	16⇒1
	64	⇒	8	0.079485	8⇒1
	64	⇒	16	0.108125	4⇒1
	64	⇒	32	0.179280	2⇒1

図 4.9: 合成方法の差異による合成時間の比較表

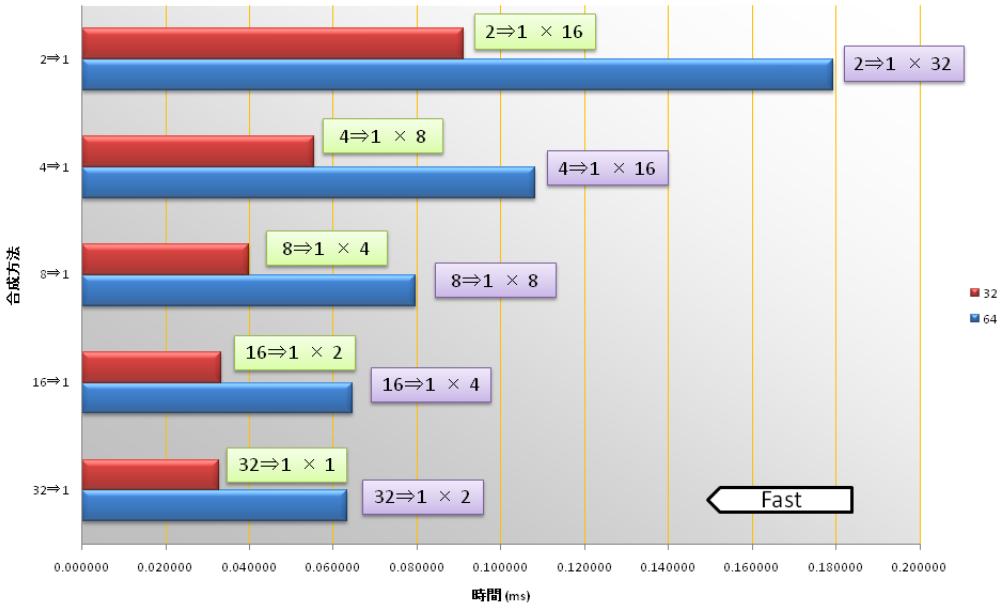


図 4.10: 合成方法の差異による合成時間の比較グラフ

4.3 品質

提案手法での合成結果とシーケンサでの合成結果の違いを、波形図として出力してみた。図 4.11 の上の波形図は GPU で合成した波形で下は Cubase SL3 を使用してミックスダウンのみをして合成した波形図である。ほとんど二つの波形に差が見られない。そこで、波形データをエクセルに出力して違いを比べてみた。図 4.12 は比較データの一部抜粋である。左の列の配列番号とは、サンプリング周波数によってサンプルされた波形データの最小単位の事で、ここでは 2 バイト分のデータが入っている。これが 1 秒間に 44100 個あり、2 秒間で 88200 個ある。1/44100 秒の間になる音のボリュームの値がそれぞれの配列番号の行に格納されている。このグラフでは元データ①と②をエクセルにて合成した結果と、GPU で合成した結果と Cubase SL3 にて合成したものを比較した。

2 秒の wav ファイルのデータをエクセルに書き出した結果、提案手法とシーケンサでは合成結果が同じであった。シーケンサには Cubase SL3 を使用した。2 つの音源データとを元に、エクセルで合成をした列と提案手法で合成した列と Cubase で合成した列の数の値がすべての行で同じであった。このことから、提案手法はテレビゲームでの使用に品質的には耐えうるものであると言える。

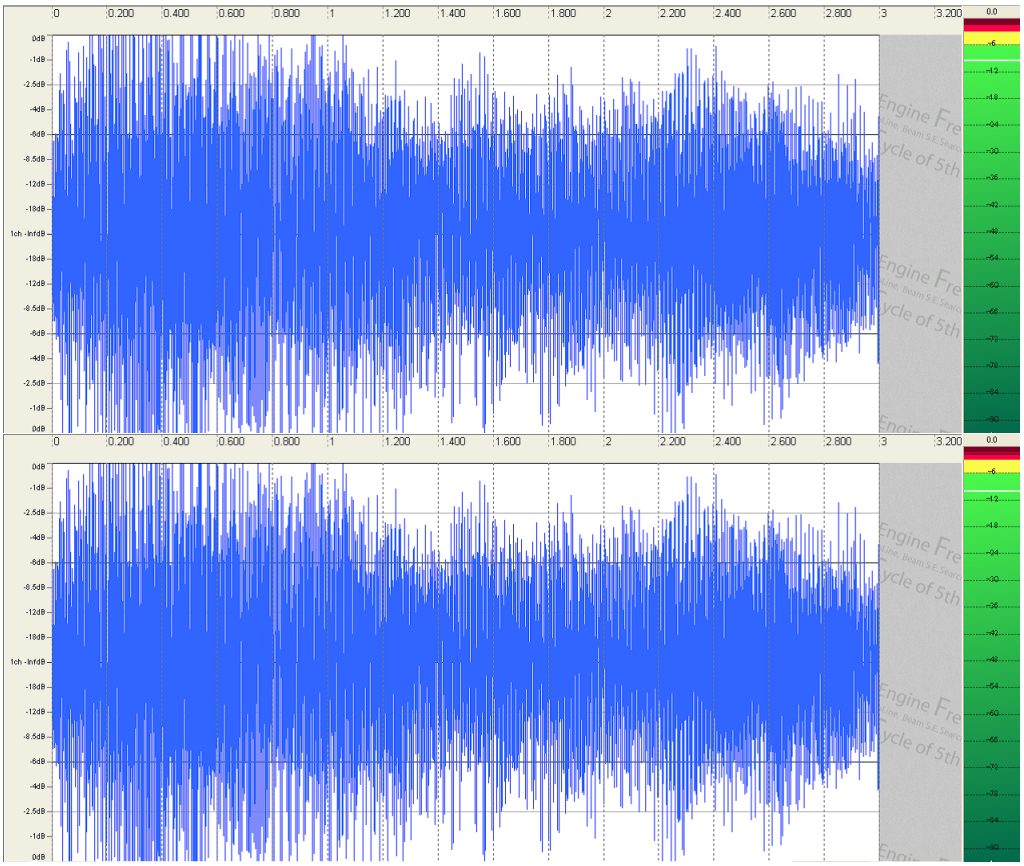


図 4.11: 上：GPU での合成結果の波形図　下：Cubase でのミックスダウンの波形図

配列番号	元データ①		元データ②		エクセルで合成	GPUで合成	cubaseで合成
1	1	+	-6277	=	-6276	-6276	-6276
2	1	+	-5803	=	-5802	-5802	-5802
3	-3	+	-5404	=	-5407	-5407	-5407
4	0	+	-5029	=	-5029	-5029	-5029
5	3	+	-4687	=	-4684	-4684	-4684
6	4	+	-4404	=	-4400	-4400	-4400
7	8	+	-4127	=	-4119	-4119	-4119
8	4	+	-3763	=	-3759	-3759	-3759
9	0	+	-3282	=	-3282	-3282	-3282
10	0	+	-2748	=	-2748	-2748	-2748
11	0	+	-2182	=	-2182	-2182	-2182
12	0	+	-1594	=	-1594	-1594	-1594
13	1	+	-1045	=	-1044	-1044	-1044
14	-3	+	-522	=	-525	-525	-525
15	0	+	11	=	11	11	11
16	1	+	529	=	530	530	530
38720	11084	+	-13728	=	-2644	-2644	-2644
38721	11297	+	-13310	=	-2013	-2013	-2013
38722	11281	+	-12792	=	-1511	-1511	-1511
38723	11063	+	-12200	=	-1137	-1137	-1137
38724	10717	+	-11518	=	-801	-801	-801
38725	10330	+	-10752	=	-422	-422	-422
38726	9891	+	-9926	=	-35	-35	-35
38727	9424	+	-8986	=	438	438	438
38728	8999	+	-7875	=	1124	1124	1124
38729	8609	+	-6629	=	1980	1980	1980
38730	8364	+	-5322	=	3042	3042	3042
38731	8484	+	-3977	=	4507	4507	4507
38732	8955	+	-2614	=	6341	6341	6341
38733	9563	+	-1253	=	8310	8310	8310
38734	10075	+	110	=	10185	10185	10185
38735	10361	+	1472	=	11833	11833	11833
88183	-8820	+	4155	=	-4665	-4665	-4665
88184	-8841	+	4567	=	-4274	-4274	-4274
88185	-8726	+	4858	=	-3868	-3868	-3868
88186	-8504	+	5066	=	-3438	-3438	-3438
88187	-8213	+	5071	=	-3142	-3142	-3142
88188	-7896	+	4899	=	-2997	-2997	-2997
88189	-7615	+	4729	=	-2886	-2886	-2886
88190	-7432	+	4540	=	-2892	-2892	-2892
88191	-7409	+	4160	=	-3249	-3249	-3249
88192	-7507	+	3640	=	-3867	-3867	-3867
88193	-7652	+	3282	=	-4370	-4370	-4370
88194	-7788	+	3050	=	-4738	-4738	-4738
88195	-7874	+	2677	=	-5197	-5197	-5197
88196	-7920	+	2253	=	-5667	-5667	-5667
88197	-7922	+	1931	=	-5991	-5991	-5991
88198	-7839	+	1568	=	-6271	-6271	-6271
88199	-7584	+	1108	=	-6476	-6476	-6476
88200	-7178	+	634	=	-6544	-6544	-6544

図 4.12: GPU での合成結果と Cubase でのミックスダウンの比較

第 5 章

まとめ

本研究ではGPUによる処理分散と高速化、指向性を用いた音の合成手法の提案をした。GPGPUを利用することによりCPUでの処理に比べ非常に高速に音の合成をすることが可能となり、エリア分けをして音を合成する事により距離と方向を保持した状態で同時発音数を上げることが出来た。今回は最大1024個まで音を再生したが、プロセッサの処理能力によってはこれ以上の数を合成しても実用に耐えうる事が出来るであろう。

合成結果は通常のサウンドデータと同じように扱えるため、既存のサウンドライブラリでも従来と同じように再生する事が出来る。本研究では合成結果をOpenALにて再生した。合成結果は通常のサウンドデータと同じようにバッファとして送り、再生することが可能であった。

問題点は、メインメモリから一度グラフィックボードのメモリにデータを送ったあともう一度メインメモリへデータを戻さなければいけない点である。この点はGPGPUを利用する以上解決しようがない。今後の展望としては、サウンドのエフェクト処理にGPUを利用する事である。こちらも今回の研究から非常に高速に処理できるもと推測できる。さらにはグラフィックボードにサウンド再生機能が搭載された場合、GPUで処理したデータを直接再生することができるようになるため非常に高速化が期待出来る。

今回は音の合成の分散処理と高速化、サラウンド対応を実現した。今後の展望

として、GPGPU でのサウンドのエフェクト処理等がある。OpenAL 等のサウンドライブラリは基本的に CPU でエフェクト処理や減衰、反射等の処理をするのでこれらを GPGPU で処理すれば非常に高速化が可能であろう。シーケンサ等の音楽制作ソフトに使われるソフトウェア音源の処理を GPGPU で行うことも非常に高速化が期待できると思われる。音楽制作ソフトでソフトウェア音源を使う時にネックになってくるのが PC の性能の問題であった。高性能な CPU がなければ良いレスポンスを得る事が出来ず、昔から DTM（デスクトップミュージック）をするには高性能な PC は必須であった。GPGPU によって処理速度が向上すれば、性能の低い PC でも GPU を搭載していれば音楽制作が出来る事となり有意義であろう。

謝辞

ゲームサイエンス卒研室の三上講師と渡辺講師には研究に対する姿勢についてご教授いただきましたこと深く感謝いたします。ゲームサイエンスの生ける神である竹内さんには本当に助けていただきまして、感謝感激雨あられです。また、同じ屋根にて泣いたり笑ったり共に努力したゲームサイエンスプロジェクトのメンバーに感謝します。研究に取り組んだ一年間は非常に長く辛い日々だったのですが、それ以上に発見や感動の多い日々であったこと、全て神と全ての生命が私に力を与えてくれた事に感謝します。

参考文献

[1] 近藤浩治, “家庭用テレビゲームにおける音響表現,” **映像情報メディア学会誌**: **映像情報メディア Vol.57 No.7**, 786–788 (2003).

[2] (株) カプコンソフトウェア技術部, “ゲームサウンドにおける音響技術について,” **日本音響学会誌 Vol.57 No.11**, 733–736 (2001).

[3] 山崎芳男, “サウンドカードにできることできないこと,” **日本音響学会誌 Vol.55 No.5**, 356–359 (1999).

[4] 山崎芳男, “パーソナルコンピュータでどんな音響処理ができるか,” **日本音響学会誌 Vol.41 No.1**, 51–55 (1984).

[5] ドルビーラボラトリーズ, ホームエンターテイメント, “ステレオからサラウンドへ.” `http://www.dolby.co.jp/consumer/home_entertainment/stereosurround.html`.

[6] 榎本 涼, “Xonar PCI Express Sound Blaster X-Fi Titanium SE-200PCI LTD.” `http://www.4gamer.net/games/029/G002923/20081224069/`.

[7] 宮田 一乗 高橋 誠史 黒田 篤, “Gpu の先駆的利用の研究動向と将来像,” **芸術科学会論文誌 Vol.6, No.3**, 167–178 (2007).

[8] Hubert Nguyen (著), 加藤 諒 (編集), 中本 浩（訳）, [*GPU Gems 3* **日本語版**], ボーンデジタル (2008).

[9] 宮田 一乗 高橋 誠史 黒田 篤, “Gpu コンピューティングの動向と将来像,” **芸術科学会論文誌 Vol.3, No.1**, 13–19 (2005).

[10] 成瀬彰; 住元, 真司; 久門, 耕一, “Gpgpu 上での流体アプリケーションの高速化手法–1gpu で姫野ベンチマーク 60gflops 超 (ハイパフォーマンスコンピューティング),” **情報処理学会研究報告 Vol.2008 No.99**, 49–54 (2008).

[11] 鈴木 和愛; 西尾 孝治; 小堀 研一, “グラフィックスボードを用いた delaunay 三角形分割の高速化手法,” **日本設計工学会 Vol.43 No.11**, 618–624 (2008).

[12] 西田 友是 , 土橋 宜典 , 山本 強, “Real-time rendering of aerodynamic sound using sound textures based on computational fluid dynamics,” *SIGGRAPH* , 786–788 (2003).

[13] 編著：沢口 真生 著：沢口 真生, [**サラウンド制作ハンドブック　第二章　放送メディアにおけるサラウンド制作**　*p102-126*], 兼六館出版株式会社, 東京 (2001).

[14] 編著：沢口 真生 著：フローリアン・カメル, [**サラウンド制作ハンドブック　第二章　テレビドキュメンタリーにおけるサラウンド音声制作**　*p142-151*], 兼六館出版株式会社, 東京 (2001).

[15] OpenAL, “OpenAL　オフィシャルウェブサイト.” `http://connect.creativelabs.com/openal/Lists/OpenAL/AllItems.aspx`.

[16] 鈴木 康太, “音場再生に適したスピーカ配置,” **高知工科大学情報システム工学科** , 20–24 (2007).

[17] 編著：城戸 健一 著：山口 公典, [**基礎音響工学**　*9,* **オーディオ音響** *p215-258*], コロナ社, 東京 (1990).

[18] NVIDIA, “CUDA ZONE.” `http://www.nvidia.co.jp/object/cuda_home_jp.html`.

[19] Tom R. Halfhill, "Parallel processing with cuda nvidia's high-performance computing platform uses massive multithreading," *Nvidia* (2008).

[20] AMD, "ATI Stream." http://ati.amd.com/technology/streamcomputing/index.html.

[21] KHRONOS GROUP, "OpenCL." http://www.khronos.org/opencl/.

[22] 後藤 弘茂, "■後藤弘茂の Weekly 海外ニュース■ ISSCC で、ついに Cell が登場〜ソニーグループ、IBM、東芝が共同発表." http://pc.watch.impress.co.jp/docs/2005/0208/kaigai153.htm.