平成30年度　卒業論文

シミュレータ教材開発に関する一提案

指導教員　須田　宇宙　准教授

千葉工業大学　情報ネットワーク学科
須田研究室

1532040　岡本　悠佑

提出日　2019年2月6日(火)

目次

図目次

表目次

1 緒言

近年 e-Learning の普及とともに様々なシミュレータ教材の需要が増加している. さらに, パソコンだけでなく，タブレットやスマートフォンといった様々な端末での利用が見込まれ, その利用は学校だけでなく，塾や家庭など幅広い場での活躍が考えられる.

しかし，スマートフォンの性能は PC と比較して決して処理速度が速いとは言えず，一部の塾や通信講座では独自のハードウェアやアプリケーションの開発を行っていることから，従来のシミュレータ教材をそのまま適用するのは容易ではないのが現状である．したがって，従来のシミュレータ教材の処理速度を改善する必要がある．しかし，全てのシミュレータ教材を新規で開発した場合その労力は計り知れない．そのため，ただ処理速度を向上させるだけでなく，いかに変更点を少なく．処理速度を上げるのかが重要となってくる．現在処理速度を向上させる手法として主にシェーダや Worker を用いる方法がある.

シェーダは処理速度が大幅に向上するが, 記述内容が複雑，増加するため手軽に実装することができないといった問題点がある．Worker は JS をマルチスレッド化させる API である．しかし，描画処理を行うことができないため worker で描画に必要なデータを計算で解き，別のファイルで描画を行う必要がある．そのため従来のソースコードを処理と描画で分ける必要がある．それらのデータの送受信を実装する必要があり，決して手軽に実装できないという問題点がある.

本研究室では主に音響のシミュレータ教材を制作し配布を行っていることを受け，音響のシミュレータで広く普及している FDTD 法を用いたシミュレータ教材を利用する.

本研究では FDTD 法により制作されたシミュレータ教材を手軽に処理速度を向上させる手法を考じ，制作，有用性を示すことを目的とする.

2 e-Learning

e-Learning とは electronic Learning の略称である．その名の通りコンピュータやネットワークなどの情報技術を扱う学習形態である．e-Learning は 1999 年 11 月，アメリカフロリダ州にて開催された TechLearn1999 において初めて使用された．現在 CAI やオンラインラーニングなど様々な学習形態が存在する．

2.1 e-Learning の定義

米国の組織学習・人材開発に関する世界最大の会員制組織である ASTD(American SocietY for Training & Development) によると「e-Learning とは, 明確な学習目的のために, エレクトロニクス技術によって提供, 可能とされた伝達されるあらゆるものである.」と, 定義されている．

一般的に情報技術を用いているもの全てを e-Learning と「広義」でとらえる場合と，非同期型オンライン方式を想定した「狭義」である場合とさまざまであり、一概に定義通りではない．テクノロジを活用した学習形態の幅が広がる現在の傾向では「情報技術によるコミュニケーション・ネットワークなどを活用した主体的な学習」を総称して e-Learning と定義するのが一般的である．

2.2 e-Learning に含まれる学習形態

e-Learning は IT 技術を用いた様々な学習形態を総称して呼ぶ．したがって一言で e-Learning と呼んでも複数の種類が存在する．そこで本章では e-Learning の種類と特徴・開発の経緯などを紹介する．

2.2.1 CAI

CAI(Computer-Assisted Instruction) は 1950 年代後半，米国で兵員教育の目的として発足．1970 年代から 80 年代にかけて，「講師と受講者が長時間同じ場所に居なければならないという問題を，コンピュータを用いることにより軽減できないだろうか」という，パソコンを教育に活用する学習形態として注目された．CAI はスタンドアローン環境が前提であり，受講者はマニュアル通りの単純な，決められた学習を行う形式だった．そのため受講者のレベルに応じた教育，柔軟なカリキュラムの提示を行えず，効率的な学習を十分に行えなかった．また，ネットワークの利用には至らなかったため，受講者の進捗の確認などは人間が行う必要があった．

1980 年代に入るとハードウェアの開発が進み，従来の大型コンピュータで実行されていた CAI がパーソナルコンピュータでも実行可能となり，モデムやスキャナーなどの周辺機器の開発によりメディアとしての能力が向上した．

1990 年代にはコンピュータネットワークが発達，個々人の情報リテラシー能力の向上を受け，CBT や WTB が考案，2000 年代以降には，これらの概念を活用した e-Learning が盛んに用いられるようになった．

2.2.2 CBT(Conputer-Based Training)

CBT(Conputer-Based Training) は，1986 年に教育，調査，測定分野の非営利団体である ETS(Esucational Testing Service) が，大学生の能力別クラスの編成用のテストとして利用したことから始まった．当時はットワークが普及していなかったため，CD-ROM の大容量な特性を活かし，動画や音声などを利用したインタラクティブなコンテンツが効果的に利用された．また，受講者管理やコンテンツ配信を行うサーバを必要としないため，比較的容易に導入することが可能だった．しかし．CD-ROM を制作するためのコストと，配布後の修正が難しく，各個人の進捗状況を一括して管理することが困難であった．

その後，1990 年代に入り，IT 系企業の認定資格試験に利用されたことから大きく発展し，現在では公的要素の強い試験など様々な分野で CBT 化が進んでいる．日本国内においても国家試験の一つである情報処理技術者試験で 2003 年から導入されているように，現在でも学校教育や企業内教育の現場で，幅広く取り入れられている．

2.2.3 WBT

1990 年代中盤から後半にかけて，ソフトウェアの進化とネットワークの普及に伴い，インターネットやイントラネットなどのネットワークを通して教育コンテンツ学習者に提供する WBT(Web-Based Training) という学習形式である．従来の WBT と CBT との大きな違いは，ネットワークを介していることである．ネットワークを用いることによる恩恵は以下の 3 つが挙げられる．

1. 知識の変化に合わせて教育内容の更新が容易なこと
2. 双方向通信が可能になり，受講者と講師，もしくは受講者同士間のインタラクティブなコミュニケーション，及び受講者の進捗状況の把握が可能になったこと
3. Web は世界標準のプロトコルが存在するため，教材互換性，汎用性，操作性が向上したこと

2.2.4 EPSS(Electronic Performance Support System)

EPSS(Electronic Performance Support System) とは，IT 技術を活用して，業務中に必要な知識やツールの提供を行うことで，パフォーマンスの向上を支援するシステムのことである．

例としてコールセンターやビデオマニュアルが挙げられる．インタラクティブ学習やパフォーマンス・サポート・システムの分野を専門にしているグロリア・ゲーリー氏によれば「他社による最小限のサポートと介入で，ジュブパフォーマンスが生まれることを可能にするモニタリングシステム，情報，ソフトウェア，支援，データ，画像，ツール，評価の全ての内容に働く人々が簡単にアクセスできる．また，それらを個々人が手元で見ることができ，直ちに必要な内容を入手できる仕組みを持つ，使いやすく統合された電子的環境」と定義している．

CBT や WBT といった，業務から一度離れて学習する形態と比較し，より業務遂行に直結した知識を会得できるシステムであることが特徴であり，近年はより効率化を図るために，

これまで活用していた紙のマニュアルを電子化する動きがある．米国においてはユーザのパフォーマンスに合わせた設計を行うことを PCD(Performance Centered Design) と呼び，EPSS と同義として扱われる場合がある．

2.2.5 Knowledge Management System

Knowledge Management とは，個人の持っている知識，情報を組織で共有し，コミュニティ全体の創造性を向上させるための手法である．個人の持つデータだけでなく，経験やノウハウを含んだ幅広い物を指す．

Knowledge Management と e-Learning は独立したものと扱われることが多かった．しかし，米国で e-Learning の分野においても，マニュアルの理解や受講者間での知識の共有や生成の場を設けることの重要性を訴えられる様になった．そのことが実現した場合 e-Learning と Knowledge Management の区別がなくなるため，両者を融合し，e-Learning の一環として捉えるようになった．

2.2.6 Blended Learning

Blendes Learning は，e-Learning と集合研修を組み合わせた形態ことである．

それらの組み合わせは，他者からの刺激を受けることで学習意欲の向上，相互作用をによって理解促進，知識の整理が行えるという利点がある．

e-Learnig + 集合研修
: 事前学習を済ませた後に，教室でインタラクティブな学習を行う

e-Learnig + 集合研修
: 事前学習と教室研修を行い，その後にバーチャルクラスで学習を行う

集合研修 + e-Learnig
: 集合研修後にフォローアップのためにセルフ学習を行う
: ディスカッションバーチャルクラス，チュータからのメンタリングを含む

このように様々な形態が存在するが，e-Learning に適した内容化を見極める必要である．

2.3 e-Learning の効果

e-Learning には様々な形態が存在するが，学習形態ごとに違いがあるが，e-Learnig という大きな枠組みで考えたときのメリットとデメリットをまとめる．

2.3.1 e-Learning のメリット

受講者のメリット

1. 場所と時間に制限がない
2. 受講者のレベルや習熟度に合わせたコンテンツの提供できる
3. 研修終了後に受講者間や講師との間でインタラクティブなコミュニケーションが可能
4. 世界標準のプロトコルを用いることで，教材の互換性，汎用性，及び操作性が向上
5. 予め筋道を立ててシステムを組むため，全体の筋道が決まっており，一括して見やすい
6. 場所の移動を省くことが可能なため，時間とコストを削減できる
7. 最新の情報を安価で早く入手し，学習することができる

制作側のメリット

1. 集団教育と比較すると安価に提供できる
2. マルチメディアを用いたインタラクティブな教材の制作が可能
3. 学習の進捗状況をリアルタイムで把握することができるため，管理が用意
4. 知識の変化に合わせ，最新の内容に更新した教材の把握が容易

2.3.2 e-Learning のデメリット

(1) コンピュータ，またはインターネット環境が整っているところでしか学習できない

本などの資料は，持ち運びが容易なため，電車の中などで資料を持ち込むことや書き込むことで学習することが可能だが，e-Learning では難しい．しかし，小型の端末などで学習を行えるソフトがリリースされており，以前と比べるとその欠点は薄れつつあるといえる．

(2) 就業時間中の学習を公認する必要がある

集合研修は，会社がその時間の学習を公認しているが，書籍での学習は就業時間外として扱われることが多く，自主学習形態をとることが多い．しかし，e-Learning はインターネットが使えるコンピュータで学習することが一般的であり，セキュリティ上社外からのアクセスは禁じられていることが多く，学習するには社内で行う必要がある．その場合就業中か，仕事を終えて自分の席で学習することになる．自分の席で学習する場合，他者の目にはばかられ，思うような学習は難しいと思われる．就業時間中の学習は多くの企業で公認されにくいため，教材をリリースしても誰も学習しないという状況も多い．

(3) システム投資コストがかかる

e-Learning を使用する環境を構築するには教材を制作するだけでなく，サーバ管理コ

ストやコンテンツ維持コストや補助的サーバなどが挙げられる．また，レポート提出や質疑応答システムと統合した環境を整えるとそれなりに費用がかかってします．システムの投資コストが従来の教育コストより高くなってしまう場合，e-Learning のメリットが無くなってしまう．

(4) コニュニケーションや，体で覚える技術教育には向かない

実際に対話を行う集合研修に対し，e-Learning は成約が多い．しかし，誰かが一方的に話すタイプの集合研修と比較すれば，この弱点は解消されたといえる．

(5) シミュレータ搭載が困難

現在実用化されている e-Learning 教材のなかで，文系科目に該当する大学などで多く実用化されている．その背景に容易に制作が可能なことが挙げられる．容易に作成できることからコンバータを用いることが多いが，コンバータはシミュレータやアニメーションに対応されておらず，実験などの補修学習が要の理系科目において十分な効果を期待することはできない．

3 シミュレータ教材

3.1 シミュレータ教材の定義

シミュレータ教材とはシミュレーションを行う教材のことである.

現実世界の問題はシステムが複雑で予測が困難な場合が多く，実システムで試すにはコストや時間がかかるといった場面が多く存在する．しかし，コンピュータ上に現実世界を再現することにより，実際のシステムの制作，変更することなく様々な条件下の環境を再現，システムの挙動を調べることができる．このコンピュータ上に現実世界を仮想的に制作するシステムをシミュレータと呼ぶ.

したがって，シミュレータ教材とは様々な条件下の現象を実際に生成することなく再現することで，対象を視認，理解させるための教材である.

一般的には，航空シミュレータや音響シミュレータなど，シミュレーションの結果を体験させるためのシステムを指すことが多いが，本研究では，コンピュータ上で動作し，ブラウザ上でパラメータを変更させ，その結果を同じブラウザ上で確認できるソフトウェアのことを指す.

3.2 FDTD 法

音場のシミュレーションは大きく分けると定常解析手法と非定常解析手法の 2 つに分類される．定常解析手法とは対象が定常状態に達したときの音場や位相を計算する手法である．音波は時間とともに変動するが定常状態のため時間に依存することなく空間分布が求められる.

一方，非定常解析手法は時間によって変化する場を取り扱う．したがって音圧などの物理量が時間と共に変化する場を扱うため，波形そのものの計算が得意である.

本章では本研究に使用し，音響のシミュレータにおいて度々用いられる非定常手法の一つである FDTD 方について論じていく.

3.2.1 FDTD 法の概要

FTDT(Finite Difference Time Domain) 法は，1966 年に K.S.Yee によって提案された．K.S.Yee はマクスウェル方程式が電解と磁界の連立方程式であることに着目し，中心差分に電磁界の時間及び空間配置を考案した．後に多くの科学者によって発展し，現在では電磁波の支配式を求めるだけでなく，流体力学や熱工学だけでなく波動光学にも用いられることが多くなった.

3.2.2 計算方法

物体に力を加えると物体は移動する，それと同様に空気も力が加わると動きが生じる．それはニュートンの第 2 方式，すなわち，運動方程式で記述される．空気には物体に動きが加わることによって生じる圧力と，運動に依存しない圧力が存在し，音響において前者を音圧，後者を大気圧と呼ぶ．音波の伝搬する空間を音場と呼ぶ．

ある空間において X，Y の直交座標の寸法がそれぞれ ΔX，ΔY とし，密度を ρ，音圧を p とし，それぞれの変異を u_X，u_Y とする．ΔX が微小であると仮定したときの X 方向の運動方程式は

$$\rho \frac{\partial t^2}{\partial^2 u_x} = -\frac{\partial p}{\partial x} \tag{1}$$

Y 方向も同様に

$$\rho \frac{\partial t^2}{\partial^2 u_y} = -\frac{\partial p}{\partial y} \tag{2}$$

と表される．体積弾性率を k[N/m^2] としたときの音圧に関する連続方程式は

$$p = -k\left(\frac{\partial u_x}{\partial x} + \frac{\partial u_y}{\partial y}\right) \tag{3}$$

となる．式 (1)，(2)，(3) を物理現象を方程式化した式，すなわち，支配式として音場の解析を行う. これらの式を時間微分した式を以下に示す．

$$\rho \frac{\partial v_x}{\partial t} = -\frac{\partial p}{\partial x} \tag{4}$$

$$\rho \frac{\partial v_y}{\partial t} = -\frac{\partial p}{\partial y} \tag{5}$$

$$\frac{\partial p}{\partial t} = -k\left(\frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y}\right) \tag{6}$$

となる．

音圧は時間と共に連続的に変化するが，コンピュータは連続の関数を扱うことができない．したがって，空間や時間を細かく区切ることで，連続な関数に見せる必要がある，このことを離散化と呼ぶ．また，空間に関する区切り幅を空間離散化幅，時間に関する区切り幅は時間離散化幅と呼ぶ．区切りごとの離散的な値を参照点と呼ぶ．

FTDT 法における音圧の離散化は，隣り合う参照点がそれぞれ ΔX，ΔY の空間離散幅を持ち，Δt の時間離散幅を持つ．参照点から X，Y 方向に関して何番目の音圧参照点かを I，J を空間ステップと呼ぶ．何番目の時間参照点であるか n を用いて表すと，$(n-0.5)\Delta t$[s] であり，この n を時間ステップと呼ぶ．

X に関する偏微分係数を

$$-\frac{\partial p}{\partial x}|_{x=x_0} = \lim_{\Delta x \to 0} \frac{p|_{x=x_0+\frac{\Delta x}{2}} - p|_{x=x_0} - \frac{\Delta x}{2}}{\Delta x} \sim \frac{p|_{x=x_0+\frac{\Delta x}{2}} - p|_{x=x_0} - \frac{\Delta x}{2}}{\Delta x} \tag{7}$$

のように近似する．式 7 を式 4 に適用し，空間ステップを $i+0.5$,j の位置，空間ステップ N の時刻としたとき

$$\rho \frac{v_x^{n+0.5}(i+0.5,j) - v_x^{n-0.5}(i+0.5,j)}{\Delta t} = -\frac{p^n(i+1,j) - p^n(i,j)}{\Delta x} \tag{8}$$

と近似できる．空間微分にも時間微分にも式 7 と同様の近似を用いることが可能となり，これが FTDT 法の特徴の一つである．式 5 も同様に近似値を求めることができる．

$$\rho \frac{v_y^{n+0.5}(i+0.5,j) - v_y^{n-0.5}(i+0.5,j)}{\Delta t} = -\frac{p^n(i+1,j) - p^n(i,j)}{\Delta y} \tag{9}$$

時間ステップ N-0.5 を過去，N を現在，N+0.5 を未来の状態と考えれば，過去の X 方向の速度 v_X の値と現在の値から未来の v_X の値を求めることができる．最も時間ステップが大きい項のみ左辺に変形し，初期の粒子速度分布と音圧分布の 1 組がわかれば，全空間ステップにおいて変形した式を交互に計算することで求めることが可能である．初期時刻の場の業態を表す条件のことを初期条件と呼ぶ．時間が進むごとに次々と変化することを時間発展と呼び，初期状態から時間的な順序を追って場の上程を求めることを逐次計算と呼ぶ．

3.3 シミュレータ教材のメリット

シミュレータを使用することに得られるメリットを以下に示す．

1. コンピュータで疑似体験
   シミュレータ教材とは，現実の物理現象をモデル化し，コンピュータ上で計算することで，模擬体験できる藻である．仮想環境の中での学習により，従来では考えられない程の学習効果を得ることができ，学習者の意欲向上に繋がると考えられる．
2. 等しく均一の条件で学習が可能
   学習者自身が，実験の条件や変数を変えながら実験に参加することが可能である．実験変数の変更を安全に，何度でも変更することが可能なため，繰り返し学習が可能となり，学習者は全員均一の条件で学習できる．
3. 安全性
   シミュレータ教材により学習者は，教室内で費用，時間のかかるものや．危険で簡単に実施できない実験を疑似体験することができる．

3.4 シミュレータ教材のデメリット

シミュレータ教材は通常，全ての現象を思考要素とせず，対象要素を絞り込むことにより，要素が現象に与える影響を検証することが目的である．したがって，結果が不確定な事象を

検証することは極めて困難である．

コンピュータを用いた積算によるシミュレートは，基本線形近似による計算のため，非線形を含む自然現象をシミュレートする場合は必ず誤差が生じる．良好な結果が得たければ，誤差見積もりが重要になる．

そして，シミュレータ教材を使うにはシミュレータ教材を作る必要がある．開発者はシミュレータを開発するためのソフトウェアを保持した状態で，プログラミングの知識や主題についての専門的な知識が要求されるため，開発者が限定される．

4 プロセッサ

本章ではプロセッサ，CPU や GPU の歴史や相違点といった内容を解説する．

4.1 CPU

CPU とは Central Processing Unit の略称である．人間でいう頭脳と例えられることが多いパーツである．本章ではその概要と歴史について記す．

4.1.1 CPU の概要

CPU の外見は大きさが数 [cm], 厚さ数 [mm] で，容器に覆われている．裏面には外部との接続のため，数多くのピンが設けれらている．このピンを介して，CPU はプログラムを読み込み実行するのである．

CPU の役割はメモリやハードディスク，マウスなどからデータを受け取り，計算，判断をし，その結果をディスプレイやメモリ，ハードディスクに送り出すことである．それらの命令を実行するためには図 7 のように複数の演算ユニットが必要になる．CPU は，外部から供給されるクロック信号に合わせて内部動作を進め，その信号は数 [GHz] と非常に高い．

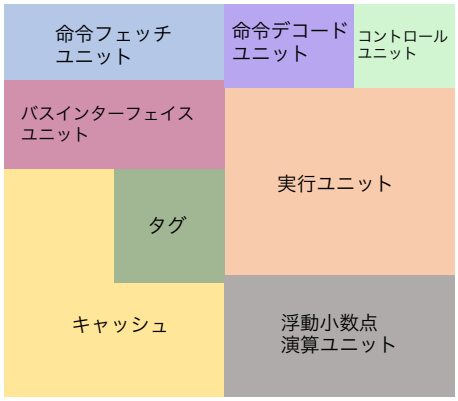

図 1　CPU の内部構造の略図

4.1.2 CPU の歴史

パソコンが誕生する前のコンピュータは非常に高価なため，個人が所有できる代物ではなかった．しかし，1971 年にインテルが 4 ビット CPU4004 の開発に成功したことにより，以前より手軽にコンピュータを入手できるようになった．1974 年には更に性能が向上した 8 ビット CPU を開発した．同年 MITS(マイクロ・インスツルメンテーション・テレメトリ) が本 CPU を利用したコンピュータ ALTAIR8800 を販売した．8080 は汎用性が高く，様々な製品に組み込まれた．

1978 年に 16 ビット CPU の 8086 が登場した．その後，80286，80386，80486 そして，Pentium へと進化していった．これらの特徴として上位互換性があることである．つまり，以前に作られた OS やアプリケーションを動かすことができる．この一連の CPU は CISC（CompleX Instruction Set Computer）タイプの CPU と分類される．実行可能な命令は CPU

ごとに異なり，CPU が実行可能な命令を全部合わせて命令セットと呼ぶ．CISC は複雑命令セットコンピュータと呼ばれ，それぞれの命令の機能が複雑かつ強力に作られている．

1980 年 RISC(Reduced Instruction Set Computer) が発表された．RISC は命令を使用頻度の高いものに限りられ，基本設計がシンプルなのが特徴である．

RISC は数多くのメーカの CPU に採用されており，1985 年に Sun Microsystems が SPARC を開発し,1986 年には ARM 社から低消費電力を重視した ARM2 と PA-RISK の最初の実装である T1 を搭載した HP 3000 が登場した．その後 1990 年に IBM が 16 ビットの RISC マイクロプロセッサの POWER を発売した．

一方，CISC タイプの CPU の pentium を開発していた Intel も　RISC ベースの CPU である i960 を発売していた．

その後，パソコンにおいて従来 CPU とのの互換性を保持しつつ，RISC 技術を取り入れたインテルが優位に立ち，ワークステーションにおいては RISC CPU に軍配が上がった．

1990 年代に入ると業務用向けに 64 ビットの CPU MIPS R4000 が登場した．家庭用では 1994 年にソニーから発売された PlayStation に R3000 が搭載されるなど 32 ビットへの移行が進んでいた．

64 ビット向けワークステーションの市場は RISC が占め，32 ビットの市場も AMD や Cyrix などが競い合っていた．そこで Intel は家庭用向けの 64 ビット CPU の命令セットを発表しが，開発が遅延したことや，POWER などの競合プロセッサの性能向上により普及することはなかった．

1993 年にインテルが Pentium シリーズを発売した．初期のクロック周波数は 60MHz だった．その後 200MHz まで向上する．

1990 年代後期にはサーバ向け CPU の 64 ビット化が一区切りし，高クロック数，マルチプロセッシングへと変化していった．

2000 年に入り AMD が Athlon を発表し，最大周波数 1GHz になった．同年インテルは Athlon に対抗して Pentium 4 を発表し，動作周波数は 1.3GHz まで向上した．

2002 年にサーバ分野でマルチコア CPU が導入された．翌年パソコン分野において 64 ビットの CPU が登場した．

2004 年に周波数向上の限界に突き当たった．周波数を向上させることにより，消費電力が増加した．そこでインテルは 2005 年に 1 つのパッケージに 2 つの CPU コアを実装した CPU を開発，Pentium D の登場である．この CPU がマルチコア化への先駆けとなった．同年 AMD からマルチコアの Athlon 64 X2 が発表された．AMD はクロック数が 1GHz へ到達した時点で処理効率を重視した CPU を展開した事により，デュアルコアへの拡張を意識した設計を可能とした．この頃から消費電力あたりの性能が重視されるようになった．サーバ向けの CPU ではワンチップで数十スレッドを実行する CPU が現れた．相対的に低いクロック数で性能を引き出しやすい SIMD(single instruction multiple data) の性能に重点が置かれた．

4.1.3 CPU の処理方法

パイプライン

CPU で処理を行う時プログラムはフェッチ，読み込み，実行，メモリに書き込みを繰り返している．処理を行う時，これらを順番に実行するが，同時に一つの処理しか行えない．そこでステージ数を増やすことで複数の命令を実行可能とした．Pentium4 ではステージを増やすことで高クロック化させた．

SIMD

SIMD とは（Single Instruction Multiple Data）の略称である．SIMD は 1 回の命令で複数のデータに対する処理を同時に行う演算装置設計手法である．同一の処理を大量に行わなければならないマルチメディアの処理などに向いている．

4.2 GPU

GPU は Graphics Processing Unit の略称である．GPU は画像処理，大量の計算を行うことを目的として開発されたハードウェアである．しかし，近年はその処理能力の高さからディープラーニング等画像処理以外の分野での活躍も多く見られるようになった．本章では．GPU の概要から歴史，処理の仕方について述べる．

4.2.1 GPU の概要

GPU は画像処理を行うことを目的としている並列演算ハードウェアである．本来 GPU は画面の出力やそれに付随する演算を行う．家庭用のパソコンにでは画面の出力の他にリアルタイムで大量の計算を必要とする 3D ゲームなどを行うのに役立っている．

しかし，その高い処理能力を画像以外に活かすために NVIDIA はグラフィックス処理以外の演算を行わせるようにした GPGPU(General-Purpose computing on GPUs) の開発環境 CUDA を一般公開した．GPGPU の研究が注目を浴び，現在ではその高い処理能力を活かしディープラーニングに広く普及している．

一般的に GPU とは，図 2 のようなビデオカード (グラフィックボード) と呼ばれるハードウェアとして販売されている機器を指し，ビデオカードと LSI チップを区別しないで呼ばれることが多い．

4.2.2 GPU の歴史

1970 年代に入り大型のコンピュータのサイズダウンが行われ，ミニコンという種類が登場，低価格帯の CPU による一般向けのコンピュータ，パーソナルコンピュータ（パソコン）が登場した．世界最初のパソコンと呼ばれているアルテアにはモニタがなく，コンピュータの状態を LED で表示するにとどまった．モニタに対応したのアップルの Apple Ⅰ，Apple Ⅱだった．当時のパソコンには，メモリ上にビデオ表示用の領域があり，後に Video RAM(VRAM) と呼ばれるようになった．VRAM の領域を書き変えることで画面に出力ができた．

1990 年代に入るとパソコンでも GUI の環境が整ってきた．GUI が発展すると以前の 640 × 400 ドットより広い描画範囲を必要とした．そのため，ユーザの間ではビデオカードを用

図 2　グラフィックボード

いて描画領域を拡張させる動きがあった．Windows95 が発売のころには，ビデオカードが含まれるパソコンが標準となった．

1995 年に 3dfX から 3D の処理に特化した Voodoo が発売された．当時は家庭用のパソコンとアーケードゲームのグラフィックに差が存在したが，Voodoo の登場により家庭用のパソコンもアーケードゲームに匹敵する品質を実現させた．

1997 年には CYriX からオーディオチップとグラフィックチップが 1 つのチップセットに統合された MesiaGX が発売されるなど，複数のチップを統合した製品が登場する．グラフィックス機能が貧弱であったり，低価格帯をターゲットとしてたため利用できる CPU の性能にも限度があり，普及しなかった．しかし，1999 年に Intel から発表された Intel 810 チップセットは，2D 描画性能は十分な性能を有したため，オフィス処理やブラウジングなどには十分な性能であった．また，同設計で多様な CPU を採用した製品を開発しやすかった．グラフィックカードが不要なことから小型なパソコンの設計が可能となり，グラフィックチップが組み込まれたチップセットが普及した．

1999 年に NVIDIA から高い 3D の演算能力を有する GeForce256 が発売された．GeForce256 の特徴は 3D 演算用のハードウエアが搭載されたことである．それまで 3D 専用ワークステーションでしか行えなかった分野がパソコンでも行えるようになった．CPU が通常の演算を行うことから，グラフィックの演算処理を行うハードウエアを GPU と呼ぶと NVIDIA が提案，その呼び名が定着した．

2001 年には GPU の固定機能パイプラインであるシェーダ処理がプログラム可能になった．しかし，アセンブラが基本として扱われたため敷居は非常に高かった．その後，シェーダのプログラムを書くための高級言語が登場，3D グラフィック以外の演算も行われるようになった．演算ユニットを汎用化させる統語柄シェーダーアーキテクチャが登場，GPU における汎用演算 GPGPU の発展，普及へとつながっていった．

2010 年に入ると CPU 分野ではマルチコアを搭載するだけでなく，画像出力専用回路として GPU コアを統合した製品の提供を行った．それまで行われていた画像データを外部メモリ空間へ転送するといった処理は不要になった．

4.2.3 GPU の処理方法

GPU を構成するプロセッサの最小単位はスカラプロセッサである．スカラプロセッサは CPU と比較して，算術計算やロジック処理等，限られた機能しか持ち合わせていない．スカラプロセッサは CPU のプロセッサと違い命令デコードが存在しない．

ストリーミングマルチプロセッサは複数のスカラプロセッサを内蔵させたプロセッサである．従来は 8 個であったが，現在では数が一定ではなくなった．ストリーミングマルチプロセッサには複数のスカラプロセッサを制御するユニットが搭載されている．また，命令デコード機能が含まれている．そのときマルチプロセッサに伝達される内容は同じ内容となっている．

スカラプロセッサ上で動作するプログラムの単位として，スレッドがある．スレッドは 32 スレッドごとに動作する．GPU は 1 つの命令で複数のスカラプロセッサが動作するが，開始のタイミングは同時でなく、プログラムごとに 1 クロックのズレが生じる非同期の動作である．したがって，プログラム上で同期を行うことが重要となる．

4.2.4 プログラマブルシェーダー

プログラマブルシェーダーは Microsoft が開発したゲーム及びマルチメディア処理用の API である Microsoft DirectX 8 から導入された技術である．CG において光の当たり具合などを調整して各画素の色を変化，陰影をつける処理であるシェーディングを GPU 上のプログラムとしてリアルタイムに実行する技術である．

DirectX 7 までは内部で行われた描画処理の順番や処理は固定であったが，DirectX8 移行はプログラムで自由に設定を変更できるようにした．

プログラムでシェーディングが可能になったことにより，ハードメーカーが開発されるグラフィック表現に対応したハードウェアを逐一実装する必要がなくなった．また，プログラムを利用することで画面効果を試験的に GPU 上で再現することが容易になったことにより表現力が向上した．

プログラマブルシェーダー向けの言語のことをシェーディング言語と呼ぶ．

主なシェーディング言語を以下に示す．

GLSL

GLSL は OpenGL Shading Language の略称である．GLSL は C 言語をベースとして開発された言語である．2003 年に OpenGL1.5 の拡張機能として導入され，翌 2004 年に OpenGL ARB によって策定され,OpenGL2.0 に導入されることとなった．た．

DirectX

DirectX は Microsoft が開発したゲーム・マルチメディア処理用の API である．1995 年に Windows 95 用に Windows Game SDK として登場したのが始まりである．定期的にバージョンアップされており現在は 2015 年にリリースされた DirectX 12 が最新である．

CUDA

2006 年に NVIDIA から発表された並列コンピューティングアーキテクチャである．CUDA(Compute Unified Device Architecture) はそれまで画像処理や出力が主な目的だった GPU に汎用処理をさせることを目的に開発された．専用のコンパイラや API などが提供されている．

5 HTML

HTML とは HYper TeXt Markup Languege の略称であり，ウェブ上のドキュメントを記述するためのマークアップ言語である．HTML で制作されたドキュメントは異なるドキュメントへのハイパーリンクを設定でき，リストや表などの作成を行うことができる．また，CSS や JavaScript などを別ファイルから読み出すことが可能である．

5.1 特徴

HTML の特徴はハイパーテキストを利用した相互間文章参照のフレームワークである．文章中に URL を用いて他の文章へリンクを記載することで，指定された他の文章を表示させることが可能である．

HTML はマークアップ言語であるが，マークアップとは文章を要素で括り，意味付けを行うことである．それにより文字の大きさや色などの変更，表示を可能とするのである．文字だけでなく，画像や音を添付することができる．

5.2 HTML5

HTML5 は，以前標準となっていた HTML4 や XHTML1.X の後継にあたる仕様である．以前はマークアップの仕様が主だが，HTML5 から DOM や API の仕様が多く盛り込まれた．

5.3 DOM

DOM は Document Object Model の略称である．DOM は Web 技術の標準化を行う団体である W3C(World Wide Web Consortium) から勧告されている HTML や XML(EXtensible Markup Language) ドキュメントのための API である．DOM はドキュメントを階層構造で表し DOM によって表された階層構造を DOM ツリーと呼ぶ．また，DOM は Web ページと JavaScript などのプログラミング言語を繋ぐ役割を持つ．

プログラム 1 の DOM ツリーを図 3 に示す．

プログラム 1　HTML の例

```
1  <!DOCTYPE HTML>
2  <html lang="ja-JP">
3  <head>
4    <meta charset="UTF-8">
5    <title次元の音波>1</title>
6    <script src="manY_source.js"></script>
7  </head>
8  <bodY>
9    <canvas class="mYCanvas" id="graph0" width="512" height="512"></canvas
         >
10
11 </bodY>
12 </html>
```

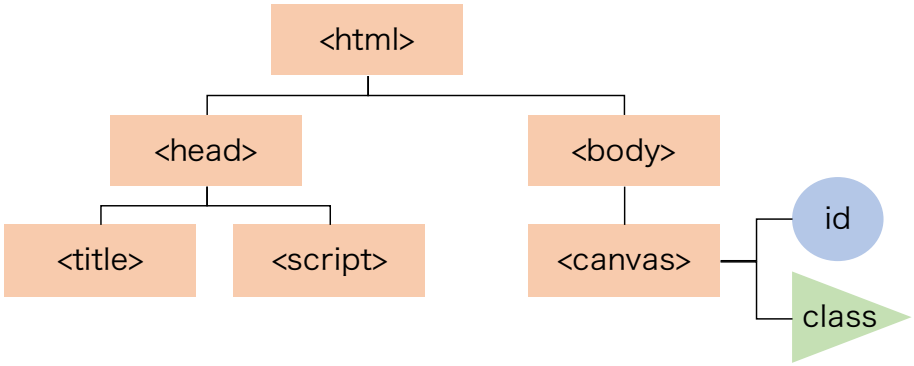

図 3　DOM ツリーのイメージ

5.4 Canvas

canvas タグとは，HTML5 で新しく追加された要素の一つであり，図を描画するために策定された仕様である．従来 HTML 上で図を表現するには画像に置き換えるか，条件に応じて表示する図を変化させる必要があった．アニメーションを実現するには Flash や Java アプレットが使用されてきた．

それに対し，canvas タグは HTML 5 に対応するブラウザ内でプラグインを利用することなく，JavaScript ベースで図を描画することが可能である．

6 プログラミング言語

プログラミング言語とは，コンピュータのプログラムを記述する言語であり，ソフトウェアのアルゴリズムを記述するための言語である．

本章では本論文で使用したプログラミング言語について述べる．

6.1 プログラミング言語の種類

プログラミング言語はいくつかに分類することができる．

ソースコードを機械語へ翻訳する方法は 2 種類ある．1 つはソースコードを実行する前に機械語へ変換を行うコンパイラ型である．特徴として始めにコンパイルを行うため，コンパイルする時間を有するが，コンパイル後は行う必要性がないことからプログラムの実行速度が速くなる．

もう 1 つはプログラムを 1 行ずつ機械語に翻訳して実行するインタプリタ型である．インタプリタ型の特徴は直ぐに実行できるため，動作を確認しながら開発を行えることである．しかし，実行する都度機械語に翻訳するためコンパイラ型と比べ処理が遅くなる．

プログラムの構築方法によって，プログラミング言語を分類することができる．手続きを順番に記述する手続き型言語，関連するデータと手続きを 1 つのまとまりとして捉えるオブジェクト指向言語，プログラムを関数の組み合わせで実行する関数型言語，データ間の関係や理論を記述する記述型言語に分けられる．

6.2 JavaScript

本研究では JavaScript を用いた開発を行った．そこで本章では JavaScript の特徴と歴史について紹介する．

6.2.1 JavaScript の特徴

インタプリタ型

JavaScript はインタプリタ言語である．近年の Web ブラウザの多くは実行時にソースのコンパイルを行う JIT（Just In Time）コンパイラを導入したことにより，実行速度を高めている．

動的なオブジェクト指向言語

JavaScript はプロトタイプベースのオブジェクト指向言語である．JavaScript のオブジェクトは後からプロパティやメソッドを動的に追加，削除することができる．

動的型付けの言語

C++ や Java は，変数の型が実行前に決まる静的型付けの言語である．一方，JavaScript は変数に型がなく，プログラムの実行と共に変数に納めるデータの型が動的に変更する．

関数が第一級オブジェクト

JavaScript の関数はオブジェクトであり，関数の引数を渡すことができる．

関数はクロージャを定義する

JavaScript の関数は自分を囲むスコープにある変数を参照することができるため，隠蔽や永続化など様々な機能を実現できる．

6.2.2 JavaScript の技術的要素

JavaScript の中核となる技術は，ECMAScript によって規定されている．TC-39 委員会によって標準化され ECMA-262 という文章で公開されている．

Web ブラウザで動作する JavaScript をクライアント JavaSctipt という．ECMAScript によって規定されたコア言語と，Web ブラウザ特有の API(Application Program Interface) から構成されている．また，HTML5 で規定されている API で様々な API を利用することができる．

Web サーバーでは，PHP や RubY などのプログラミング言語が広く利用されているが，近年サーバーサイド JavaScript の利用も増加してきた．サーバーサイド JavaScript の実行環境には Node.js や Rhino などがある．

6.2.3 ECMAScript2018 の新機能

2015 年に公表された ECMAScript6 から毎年使用を改定することとなったため，ECMAScript2015 とも呼ばれ，現在は ECMAScript2018 が最新である．そこで定義された新機能を表 1 に示す．

表 1　ECMAS2018 にて改定された主な内容

オブジェクトの Rest/Spread プロパティ	オブジェクト内に別のオブジェクトを展開する構文を実装
Promise.prototYpe.finallY	プロミスの成否に関わらず処理を行うメソッド
テンプレートリテラルの改修	タグ付きのテンプレートリレラルの場合エラーが出ない
正規表現	s オプションを付けることで改行文字に対応など
AsYnchronous Iterators	非同期のイテレータやジェネレータが利用可能

6.2.4 JavaScript の歴史

JavaScript は，1995 年に Netscape Communications 社の Brendan Eich によって開発され Netscape Navigator 2.0 にて実装された．1996 年，Microsoft 社の Internet EXploer 3.0 に搭載され急速に普及した．

JavaScript の黎明期には各々が独自の拡張を行ったため，ブラウザ間での互換性が低く，各ブラウザに対応したコードを書く必要があった．1997 年から ECMAScript による標準化が進められ，各ブラウザはその使用を実装したため，ブラウザ間の互換性の問題は解消している．

JavaScript 本来の言語仕様を解説した本がなかったことや，当時のブラウゼ機能の処理が低いことから，プログラマが本気で取り組む言語ではないと考えられていた．しかし，AjaX という非同期通信技術を使い，デスクトップアプリケーションと遜色ないソフトウェアの登場によりその認識が薄れ，理解されるようになった．2008 年から HTML 5.0 の策定が始まり，JavaScript による Web アプリケーションを作るための様々な API が規定され，ブラウザの高品質化が進み，JavaScript が普及していった．

6.2.5 実行方法

setInterval()

setInterval() は一定時間間隔を開けてコードを繰り返し行うメソッドである．経過する時間はミリ秒 [ms] で指定される．

setInterval() は前の処理が完了したか確認せずに次の処理を開始するため重たい処理を行うため予期せぬ動作をする場合がある．

```
1 setInterval( 関数，遅延時間指定[ ，引数] )
```

requestAnimationframe()

requestAnimationFrame() はブラウザにアニメーションを行うことを知らせ，指定した関数を呼び出してアニメーションを更新するメソッドである．

画面上でアニメーションの更新準備が整ったときに呼び出される．つまり，ブラウザの再描画が実行される前に呼び出されることを要求する．通常毎秒 60 回だが，W3C の勧告に従い，ディスプレイのリフレッシュレートに従う．そのため，タブに隠れたりすると描画の回数が少なくなる．

```
requestAnimationFrame( 関数)
```

6.3 WebWorker

WebWorker とは HTML 5.0 で規定された API の一つである．JavaScript はシングルスレッドの言語のため処理が実行されているときは，他の処理は保留される．WebWorker は特定の処理をマルチスレッドで実行する API である．これによって高負荷の処理によって処理が中断される問題が解消される．WebWorker で並列に実行されるスレッドをワーカーと呼び，メインスレッドとワーカーとは異なるグローバルオブジェクトを持ち互いのグローバルオブジェクトを参照することはできない．

6.3.1 WebWorker の実装方法

WebWorker を利用するにはワーカーの定義，データの送信，ワーカーでデータを受信．ワーカーでデータを送信，ワーカーのデータを受信の 5 つの作業が必要となってくる．ワーカーを定義する JavaScript のファイルが必要となる．そこでワーカーオブジェクトを生成する．

```
var worker = new Worker('worker.js');
```

ワーカーのファイルは相対 URL と絶対 URL 両方用いることができる．絶対 URL の場合は同一生成元である必要がある．

postMessage を使うことでワーカーにメッセージを送ることができる．Window オブジェクトや Document オブジェクトは送信できない．

```
worker.postMessage("message");
```

ワーカーでメッセージを受信するにはグローバルオブジェクトに onmessage イベントハンドラを登録しておく．送信されたデータはイベントオブジェクトの data プロパティに格納される．

```
addEventListerner('message', fuction(e) {
      var message = e.data;
};
```

ワーカーで処理したデータをメインスレッドに送信するには，グローバルオブジェクトの postmessage を呼び出す．

```
postmessage('message');
```

メインスレッドでメッセージを受信するには，Worker オブジェクトに message イベントハンドラを登録しておき，送信されたデータは data プロパティに格納されている．

```
worker.onmessage = function(e) {
      var message = e.data;
};
```

6.4 OffscreenCanvas

OffscreenCanvas とは，従来行えなかった WebWorker での描画処理を可能とする API である．

従来は JavaScript で動的に描画を行うかメインとなるスレッドで描画するしか方法がなかった．WebWorker の登場により，別スレッドに計算処理を分離することで処理を高速化させていた．しかし，メインスレッドの描画速度は上がらないため，3D など描画対象が多くなるとイベントが大量に発生し，他の処理が中断し結果として動作が遅いということがあった．しかし，OffscreenCanvas の登場でそれらの現象の改善が期待されるようになった．また，OffscreenCanvas は処理と描画を同一のスレッドで行うことで，WebWorker では処理後に必要としたメインへデータの送信が要らなくなった．

Web ブラウザにおいて本機能が標準で行えるようになったのは 2018 年 9 月にリリースされた Chrome 69 からであり，対応しているブラウザは，2019 年 1 月の時点で Chrome の他に Chrome for Android，Opera と Opera for Android が OffscreenCanvas に対応している．また，実験的かつ，機能が限定的ではあるが，FirefoX，FirefoX for Android と Android Brower も対応している．

6.4.1 OffscreenCanvas の実装法方

まず，WebWorker と同様にワーカーを定義する必要がある．

```
1 var worker = new Worker('worker.js');
```

次に Canvas 要素の描画コントロールを OffscreenCanvas に移譲する．

```
1 var offscreenCanvas = canvas.transferControlToOffscreen();
```

データを postMessage を用いてワーカーに送る

```
1 worker.postMessage({canvas: offscreenCanvas}, [offscreenCanvas]);
```

最後にワーカーで送信されたデータを受け取る．ワーカーと同様に data に送信されたデータが格納されている．

```
1  addEventListerner('message',fuction(e) {
2   const offscreenCanvas = event.data.canvas;
3 }
```

これら一連の処理を行った後は，ワーカーであることを意識せずに，演算，描画を行う事ができるため，1 つのファイルで requestAnimationFrame や setInterval を用いたアニメーションを制作することが可能である．また，HTML の canvas 要素で使用されるメッソドを用いることは可能なため，従来のキャンバスの操作と変わらない操作性を保持している．

7 開発したシミュレータ

本章では実際に開発したシミュレータについて論じる．

7.1 開発したシミュレータ

本研究では昨年度本学卒業論文である「GPU 利用によるシミュレータ教材の演算速度」で用いられたソースコードを拡張することで開発を行う．

7.2 開発したシミュレータ教材

図 4 は複数の音源から発せられる音場を可視化するシミュレータ教材である．昨年度本学の卒業研究である「GPU 利用によるシミュレータ教材の演算速度」を基に開発を行った．

本研究では Worker などを使用せず，素の JavaScript で開発を行った手法を手法 1 とし，シェーダで GPU に移植を行った手法を手法 2 とする．WebWorker を利用し開発を行った手法を処理 3 とする．OffscreenCanvas を利用し開発を行う手法を手法 4 とする．

図 4 は開発を行ったシミュレータの画像である．手法 3，手法 4 は開発したシミュレータ教材の Canvas 要素を 2 分割した．

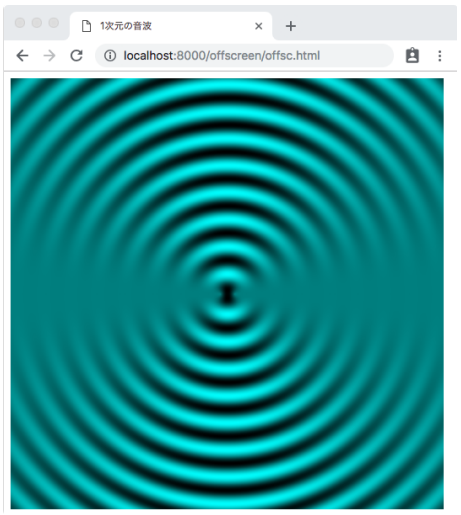

図 4　開発したシミュレータ

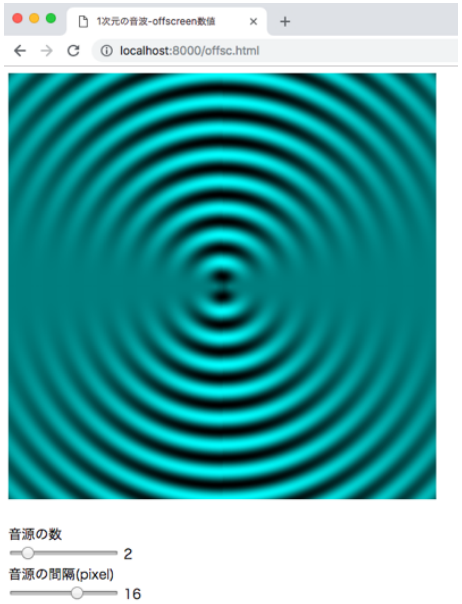

図 5　開発したシミュレータ教材

図 5 は手法 4 をリアルタイムで数値の変更を可能としたシミュレータ教材である．

手法 1 では HTML から Canvas 要素を受信し，音場の計算と描画を行った．手法 3 では手法 1 の音場の計算をサブスレッドで行い描画をメインスレッドで行った．手法 4 では手法 1 の音場の計算と描画をサブスレッドで行うことで，演算と描画 2 つの負荷を分散させた．

7.3 計測方法

本章では計測を実施した環境とその方法を記載する．

7.3.1 計測環境

本研究では以下のスペックのパソコンを用いて計測を行った.

表 2　計測環境

OS	macOS High Sierra(10.13.6)
メモリ	32GB
CPU	Intel Core i5 3.2GHz (4 コア)
GPU	NVIDIA GeForce GT 775M
ブラウザ	Google Chrome 71.0.3578.98(64 ビット)

7.3.2 計測方法

本研究ではそれぞれ 1,000 回計算するごとに費やした時間を計測，10,000 回計算を行い，1,000 ごとの時間と，10,000 回計算を行うのに有した時間を測定した.

処理に費やした時間の比を式（10）に当てはめて求めた.

WebWorker と OffscreenCanvas はキャンバスを分割し表示させているため，左右で計測時間が異なる．したがって，WebWorker と OffscreenCanvas はより計測時間を費やしたキャンバスを参照する.

$$\text{時間比} = \frac{\text{各手法が } 10,000 \text{ 回演算するのに費やした時間 } [s]}{\text{手法 } 1 \text{ が } 10,000 \text{ 回演算するのに費やした時間 } [s]} \tag{10}$$

手法 3，手法 4 は Canvas 要素を分割したため，左右の演算速度の差の計測を行う.

8 実験結果

本章では 7 章で示した計測方法を基に得られた結果を記す．

本章では演算時間の他に手法 3，手法 4 の Canvas ごとに生じた演算時間の差や演算回数による演算時間の変化を述べる．また，手法ごとに生じたソースコードの追加箇所についても言及する．

8.1 演算時間

setInterval() を用いて実行した．各手法 10,000 回演算実行するのに費やした時間と時間比を表 3 に示す．

表 3　演算時間と時間比

手法	演算時間 [s]	時間比
手法 1	206.6	1
手法 2	128.7	0.622
手法 3	98.1	0.796
手法 4	164.4	0.475

図 3 より，手法 2 〜 4 全て手法 1 を上回る処理速度を実現できた．ディスプレイのリフレッシュレートより高い処理速度を実現できたため，速度面において有用であると言える．

8.2 Canvas 要素の分離による左右の処理速度の差

8.1 により速度面の有用性は証明できた．しかし，手法 3 と手法 4 においては Canvas 要素を 2 分割しているため，左右のズレが大きくてはシミュレータとしての体を成すことはできない．したがって，本章ではキャンパスを分割したことによるズレの検証を行う．検証方法は各手法 10,000 回計測した際に費やした時間をそれぞれのキャンバスごとに計測し，左右のズレを検証する．

表 4　手法 3, 手法 4 の Canvas ごとの演算時間の差

手法	手法 3(左)	手法 3(右)	手法 4 (左)	手法 4(右)
演算時間 [s]	164.4	164.4	98.0	97.3
差 [s]	0		0.7	

表 4 より手法 4 の差は大きいと言える．本研究では生産性の検証を主としているため，手法 4 においては各サブスレッドごとに同期処理を行わなかったことが原因であると推測する．

したがって，描画結果をメインスレッドに送り同期処理を行うことで差を埋めることができると考える．また，本研究では setInterval を実行に用いたが，requestAnimationFrame を用いることにより Canvas ごとの演算時間の差を埋めることができると考える．

8.3 1000 回ごとの演算時間

図 6 に各種法が 1000 回の演算に費やした時間の変化を示す.

各手法に演算回数による処理時間の変化は見られなかった.

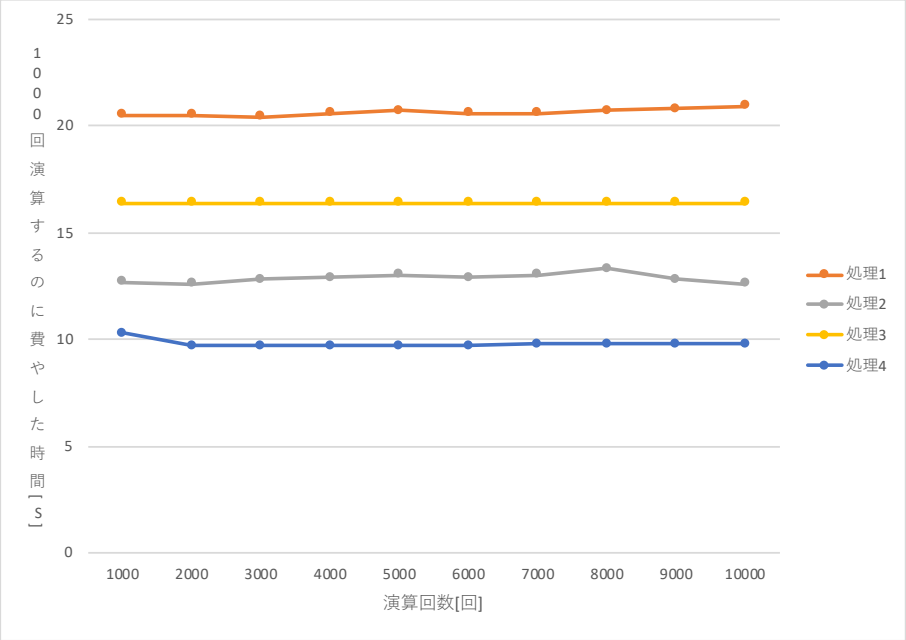

図 6　演算回数によるパフォーマンスの変化

8.4 各手法による実装の違い

本章では各手法の生産性の観点から Offscreencanvas の有用性の検証を行う．

8.4.1 手法 1

図 7 は手法 1 のフローチャートである．HTML から Canvas 要素を取得した後，演算部分では，複数の音源から発せられる音の音場の計算を行う．計算は 1 フレーム，Canvas のサイズだけ行う．演算で得られた情報を基に描画を行う．この手順を繰り返し行うことでアニメーションさせている．

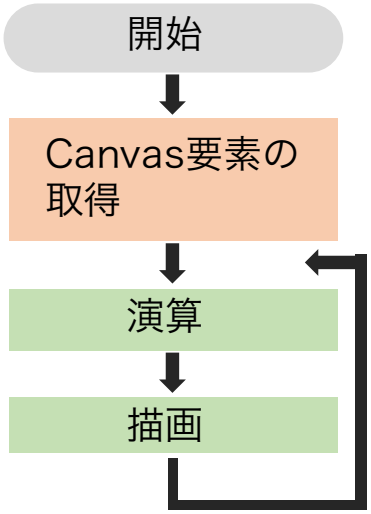

図 7　手法 1 のフローチャート

8.4.2 手法 3

図 8 は手法 3 のフローチャートである．WebWorker では描画を行うことができないため図 7 の演算と描画の処理が分断される．

したがってまず，サブスレッドへ Canvas 要素などのパラメータを送信を行う．その後，サブスレッドで受信したデータを基に演算を行い結果を再度メインスレッドへ送信する．最後にメインスレッドでは各サブスレッドの同期を行い，描画を実行する．この処理を繰り返し行うことで，処理を継続することが可能となった．ソースコードは処理と描画が分断されるため，存在していたコードを分ける必要があった．

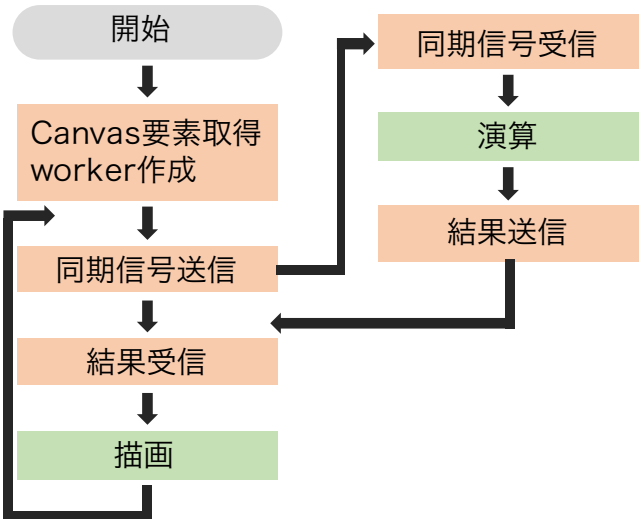

図 8　手法 3 のフローチャート

8.4.3 手法 4

図 9 は手法 4 のフローチャートである．OffscreenCanvas は WebWorker を利用し，従来サブワーカーでは行えなかった描画を可能とする API である．

したがって，手法 3 と同様にサブスレッドへ Canvas 要素などのパラメータの送信を行う．サブスレッド内で描画まで行えるため手法 1 の演算の結果をメインスレッドへ送る必要がなかった．サブワーカー内で演算と描画が行えるため，処理 1 を大きく変更することなく移植を可能とした．

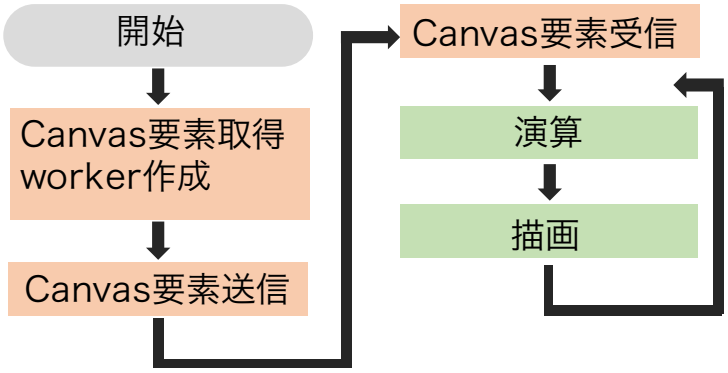

図 9　手法 4 のフローチャート

9 結言

近年の e-Learning の普及により高い処理速度と高い生産性を両立させた開発手法が求められている．

本研究室ではシミュレータ教材を対象とした高速化手法の提案を行っている．それらの研究では処理速度の向上は得られたが，設計やデバックなど実装が容易でなかった．

そこで WebWorker を利用しながら描画もサブスレッドで完結されることのできる OffscreenCanvas を用いた開発を行い処理速度と生産性の観点から検証を行った．

OffscreenCanvas を利用することで高い処理速度が得られる一方で，新たに書き足す必要性があることがわかった．

既存のシミュレータ教材へ OffscreenCanvas を適用する場合は一連の処理をサブスレッドに移植するだけで実装が可能なため，ソースコードを大きく変更する必要がなく，実装が容易であると結論づける．

欠点として対応しているブラウザの

今後は OffscreenCanvas に対応したブラウザが増加するにあたり，OffscreenCanvas を利用したシミュレータ教材が増加していくことが考えられる．

10 謝辞

本研究の遂行及び本論文の制作に当たり，助言をくださった須田研究室の仲間に深く感謝の意を表します．そして，本論文の制作に当たり多大なるご指導及びご助言をいただきました須田宇宙准教授に深く感謝いたします．

11 参考文献

[1] ecma INTERNATIONAL:"EECMAScript ® 2018 Language Specification, urlhttps://www.ecma-international.org/ecma-262/9.0/indeX.html(参 照:2019 年 1 月 15 日)

[2] 磯 博:"プログラミングの教養から言語仕様，開発技法までが正しく身につく徹底マスター JavaScript の教科書",SB Creative,2017 年 3 月 25 日

[3] 大島 篤:"見てわかるパソコン解体新書",SB Creative,2001 年 4 月 30 日

[4] 大島 篤:"見てわかるパソコン解体新書 vol3",SB Creative,2001 年 4 月 30 日

[5] 豊田　政弘 :"音響サイエンスシリーズ 1 4　FTDT 法で見る音の世界", コロナ社,2015 年 12 月 16 日

[6] Hisa Ando:"GPU を支える技術　超並列ハードウェアの快進撃 [技術基礎]", 技術評論社,2017 年 7 月 13 日

[7] 小山田　耕二，岡田　賢治:"CUDA 高速 GPU プログラミング入門", 秀和システム,2010 年 3 月 25 日

[8] 菅原　愛子:"Web Worker を利用したシミュレータ教材の開発", 平成 25 年度卒業論文,2014 年 1 月 30 日

[9] 中嶋　大貴:"GPU 利用によるシミュレータ教材の演算速度", 平成 29 年度卒業論文,2018 年 1 月 29 日

12 付録　制作したプログラム

12.1 手法 1

プログラム 2　CPU.html

```
<!DOCTYPE HTML>
<html lang="ja-JP">
<head>
  <meta charset="UTF-8">
  <title次元の音波>1</title>
  <script src="many-source.js"></script>
</head>
<body>
  <canvas class="myCanvas" id="graph0" width="512" height="512"></canvas
      >

</body>
</html>
```

プログラム 3　manY-source.js

```
var theta = 0;
var i = 0;


function calc0() {
  console.time('timer1');
  let canvas = document.getElementById('graph0');
  let ctx = canvas.getContext( "2d" );


  let lambda = 8.6; // 波長は 8.6mm （音で言えば 40）kHz
  let c_size = 512; // のサイズ（単位：ピクセル）canvas
  let w_number = 16; // フィールド内の波の数
  let s_number = 2 // 音源はつ　2で文字列を整数型にしている parseInt
  let interval = 16// 音源間隔（単位：ピクセル）

  ctx.clearRect( 0, 0, 512, 512);

  let imageData = ctx.createImageData( 512, 512 );
  let pixelData = imageData.data;


  let k = 2.0 * Math.PI / lambda;
  let m_size = ( lambda * w_number ) / c_size;

  i++;

  for( let y=0; y<c_size; y++ ) {
    for( let x=0; x<c_size; x++ ) {
      let sx = -interval * s_number/2.0 + interval/2.0;
      //if( sx!=-8 ) console.log("Err("+x+","+y+")");
      let amp = 0;
      for( let n=0; n<s_number; n++ ) {
        let px = c_size / 2.0 - x - sx;
        let py = c_size / 2.0 - y;
        let r = Math.sqrt( ( px * m_size * px * m_size ) + ( py * m_size
            * py * m_size ) );
        amp += Math.sin( - k * r + Math.PI * 2.0 / 360.0 * theta );
        sx += interval;
      }
      let wh = Math.floor( 127 + 126.0 * amp / s_number );
      if( wh<0 || 255<wh ) console.log( ？？？ "" );
```

```
42        pixelData[ y * c_size * 4 + x * 4 + 0 ] = 0; //R
43        pixelData[ y * c_size * 4 + x * 4 + 1 ] = wh; //G
44        pixelData[ y * c_size * 4 + x * 4 + 2 ] = wh; //B
45        pixelData[ y * c_size * 4 + x * 4 + 3 ] = 255; //a
46      }
47    }
48    ctx.putImageData( imageData, 0, 0 );
49
50    if(i%1000 == 0)
51    {
52      console.log( i );
53      console.timeEnd('timer1');
54      console.time('timer1');
55    }
56    theta += 4.5; // フレームで位相を゜進める 14.5
57    requestAnimationFrame( calc0 );
58  }
59
60
61  window.addEventListener( 'load', function() {
62    calc0();
63  });
```

12.2 手法 2

プログラム 4　GPU.html

```
1  <!DOCTYPE HTML>
2  <html lang="ja-JP">
3  <head>
4    <meta charset="UTF-8">
5    <title次元の音波>1</title>
6    <script src="many-source-gpu.js"></script>
7  </head>
8  <body>
9    <canvas class="myCanvas" id="graph0" width="512" height="512"></canvas
         >
10 </body>
11 </html>
```

プログラム 5　manY-source-gpu.js

```
let theta = 0;
var i = 0;
console.time('timer1');
function calc() {
  let canvas = document.getElementById( 'graph0' );
  let gl = canvas.getContext( "webgl" );

  let lambda = 8.6; // 波長は 8.6mm （音で言えば 40）kHz
  let c_size = 512; // のサイズ canvas
  let w_number = 1; // フィールド内の波の数

  i++;

  gl.clearColor( 1.0, 1.0, 1.0, 1.0 );
  gl.clear(gl.COLOR_BUFFER_BIT);

  let imageData = gl.createBuffer();
  gl.bindBuffer( gl.ARRAY_BUFFER, imageData );

  // バーテックス頂点 ()シェーダー
  var vSource = [
    "precision mediump float;",
    "attribute vec2 vertex;",
    "attribute float theta_n;",
    "attribute float distance_n;",
    "varying float theta;",
    "varying float distance;",
    "void main(void) {",
    "gl_Position = vec4(vertex, 0.0, 1.0);",
    "theta = theta_n;",
    "distance = distance_n;",
    "}"
  ].join("\n");

  var vShader = gl.createShader(gl.VERTEX_SHADER);
  gl.shaderSource(vShader, vSource);
  gl.compileShader(vShader);
  gl.getShaderParameter(vShader, gl.COMPILE_STATUS);

  // フグメントシェーダー
  var fSource = [
    "precision mediump float;",
```

```
43    "varying float theta;",
44    "varying float distance;",
45    "void main(void) {",
46    "const float PI = 3.14159265358979323846264338327950;",
47    "const float lambda = 8.6;",
48    "const float c_size = 512.0;",
49    "const float w_number = 16.0;",
50    "const float s_number = 2.0;",
51    "const float interval = 16.0;",
52
53    "const float k = 2.0 * PI / lambda;",
54    "const float m_size = ( lambda * w_number ) / c_size;",
55    "float sx = -interval * s_number/2.0 + interval/2.0;",
56    "float amp = 0.0;",
57    "for( int n=0; n<int(s_number); n++ ) {",
58    "float px = c_size / 2.0 - gl_FragCoord.x - sx;",
59    "float py = c_size / 2.0 - gl_FragCoord.y;",
60    "float r = sqrt( ( px * m_size * px * m_size ) + ( py * m_size * py
          * m_size ) );",
61    "amp += sin( -k * r + PI * 2.0 / 360.0 * theta );",
62    "sx += interval;",
63    "}",
64    "amp /= 2.0;",
65    "gl_FragColor = vec4( 0, (amp+1.0)/2.0, (amp+1.0)/2.0, 1.0 );",
66    "}"
67  ].join("\n");
68
69  var fShader = gl.createShader(gl.FRAGMENT_SHADER);
70  gl.shaderSource(fShader, fSource);
71  gl.compileShader(fShader);
72  gl.getShaderParameter(fShader, gl.COMPILE_STATUS);
73
74
75  // プログラムオブジェクトの生成
76  var program = gl.createProgram();
77  gl.attachShader(program, vShader);
78  gl.attachShader(program, fShader);
79  gl.linkProgram(program);
80  gl.getProgramParameter(program, gl.LINK_STATUS);
81  gl.useProgram(program);
82
83
84  // シェーダー側の変数を側から設定する js
```

```
85   var vertex = gl.getAttribLocation(program, "vertex");
86   gl.enableVertexAttribArray(vertex);
87   gl.vertexAttribPointer(vertex, 2, gl.FLOAT, false, 0, 0);
88
89   var theta_n = gl.getAttribLocation(program, "theta_n");
90   gl.vertexAttrib1f( theta_n, theta );
91
92   var distance_n = gl.getAttribLocation(program,"distance_n");
93   gl.vertexAttrib1f( distance_n, 1 );
94
95   // 点の座標をセット 4
96   var vertices = [
97     -1.0, 1.0,
98     1.0, 1.0,
99     -1.0, -1.0,
100    1.0,-1.0
101  ];
102
103  if(i%1000 == 0)
104  {
105    console.log( i );
106    console.timeEnd('timer1');
107    console.time('timer1');
108  }
109
110  // 描画する
111  gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices), gl.
         DYNAMIC_DRAW);
112  gl.drawArrays(gl.TRIANGLE_STRIP, 0, vertices.length/2);
113
114  theta += 4.5; // フレームで位相を゜進める 14.5
115  setInterval( calc,0 );
116 }
117
118
119
120
121
122
123
124
125 function calc1() {
126   let canvas = document.getElementById( 'graph1' );
```

```
127  let gl = canvas.getContext( "webgl" );
128
129  let lambda = 8.6; // 波長は 8.6mm （音で言えば 40）kHz
130  let c_size = 512; // のサイズ canvas
131  let w_number = 1; // フィールド内の波の数
132
133  i++;
134
135  gl.clearColor( 1.0, 1.0, 1.0, 1.0 );
136  gl.clear(gl.COLOR_BUFFER_BIT);
137
138  let imageData = gl.createBuffer();
139  gl.bindBuffer( gl.ARRAY_BUFFER, imageData );
140
141  // バーテックス頂点 ()シェーダー
142  var vSource = [
143    "precision mediump float;",
144    "attribute vec2 vertex;",
145    "attribute float theta_n;",
146    "attribute float distance_n;",
147    "varying float theta;",
148    "varying float distance;",
149    "void main(void) {",
150    "gl_Position = vec4(vertex, 0.0, 1.0);",
151    "theta = theta_n;",
152    "distance = distance_n;",
153    "}"
154  ].join("\n");
155
156  var vShader = gl.createShader(gl.VERTEX_SHADER);
157  gl.shaderSource(vShader, vSource);
158  gl.compileShader(vShader);
159  gl.getShaderParameter(vShader, gl.COMPILE_STATUS);
160
161  // フグメントシェーダー
162  var fSource = [
163    "precision mediump float;",
164    "varying float theta;",
165    "varying float distance;",
166    "void main(void) {",
167    "const float PI = 3.14159265358979323846264338327950;",
168    "const float lambda = 8.6;",
169    "const float c_size = 512.0;",
```

```
170     "const float w_number = 16.0;",
171     "const float s_number = 2.0;",
172     "const float interval = 16.0;",
173
174     "const float k = 2.0 * PI / lambda;",
175     "const float m_size = ( lambda * w_number ) / c_size;",
176     "float sx = -interval * s_number/2.0 + interval/2.0;",
177     "float amp = 0.0;",
178     "for( int n=0; n<int(s_number); n++ ) {",
179     "float px = c_size / 2.0 - gl_FragCoord.x - sx;",
180     "float py = c_size / 2.0 - gl_FragCoord.y;",
181     "float r = sqrt( ( px * m_size * px * m_size ) + ( py * m_size * py
          * m_size ) );",
182     "amp += sin( -k * r + PI * 2.0 / 360.0 * theta );",
183     "sx += interval;",
184     "}",
185     "amp /= 2.0;",
186     "gl_FragColor = vec4( 0, (amp+1.0)/2.0, (amp+1.0)/2.0, 1.0 );",
187     "}"
188   ].join("\n");
189
190   var fShader = gl.createShader(gl.FRAGMENT_SHADER);
191   gl.shaderSource(fShader, fSource);
192   gl.compileShader(fShader);
193   gl.getShaderParameter(fShader, gl.COMPILE_STATUS);
194
195
196   // プログラムオブジェクトの生成
197   var program = gl.createProgram();
198   gl.attachShader(program, vShader);
199   gl.attachShader(program, fShader);
200   gl.linkProgram(program);
201   gl.getProgramParameter(program, gl.LINK_STATUS);
202   gl.useProgram(program);
203
204
205   // シェーダー側の変数を側から設定する js
206   var vertex = gl.getAttribLocation(program, "vertex");
207   gl.enableVertexAttribArray(vertex);
208   gl.vertexAttribPointer(vertex, 2, gl.FLOAT, false, 0, 0);
209
210   var theta_n = gl.getAttribLocation(program, "theta_n");
211   gl.vertexAttrib1f( theta_n, theta );
```

```
212
213   var distance_n = gl.getAttribLocation(program,"distance_n");
214   gl.vertexAttrib1f( distance_n, 1 );
215
216   // 点の座標をセット 4
217   var vertices = [
218     -1.0, 1.0,
219     1.0, 1.0,
220     -1.0, -1.0,
221     1.0,-1.0
222   ];
223
224   if(i%1000 == 0)
225   {
226     console.log( i );
227     console.timeEnd('timer1');
228     console.time('timer1');
229   }
230
231   // 描画する
232   gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices), gl.
          DYNAMIC_DRAW);
233   gl.drawArrays(gl.TRIANGLE_STRIP, 0, vertices.length/2);
234
235   theta += 4.5; // フレームで位相を゜進める 14.5
236   requestAnimationFrame( calc1 );
237 }
238
239
240
241 window.addEventListener( 'load', function() {
242   calc();
243   calc1()
244 });
```

12.3 手法 3

プログラム 6　Worker.html

```
<!DOCTYPE HTML>
<html lang="ja-JP">
<head>
  <meta charset="UTF-8">
  <title次元の音波>1-Worker</title>
  <style>
  .myCanvas {
    margin: 0px;
    padding: 0px;
  }
  #graph1 {
    margin-left: -512px;
  }
  </style>
  <script src="main.js"></script>
</head>
<body>
  <canvas class="myCanvas" id="graph0" height="512px" width="512px"></
      canvas>
  <canvas class="myCanvas" id="graph1" height="512px" width="512px"></
      canvas>
</body>
</html>
```

プログラム 7　main.js

```
class Main {
  constructor() {
    let canvas = document.getElementById('graph0');//要素の取得 canvas
    let canvas1 = document.getElementById('graph1');
    this.theta = 0;
    this.worker1 = new Work( "worker1", canvas, 0 );
    this.worker2 = new Work( "worker2", canvas1, 256 );

    //let nl = new nylon();
    //nl.on( 'worker2', () => {
    setInterval( () => {this.bar()}, 0 );
    //})
  }

  bar() {
    this.worker1.execute( this.theta );
    this.worker2.execute( this.theta );
    this.theta += 4.5;
  }
}

class Work {
  constructor( name, canvas, left ) {
    this.name = name;
    this.left = left;
    this.worker = new Worker('worker.js');
    this.ctx = canvas.getContext('2d');

    this.image = this.ctx.getImageData( 0, 0, 512, 512 );
    //this.nl = new nylon();

    this.worker.addEventListener('message',(e) => {
      let imageData = e.data.image;
      let count1 = e.data.i;

      if (count1 %1000 == 0){
        console.log(count1);
        console.time('time1');
        console.timeEnd('time1');
      }
      this.ctx.clearRect(0,0,512,512);
      this.ctx.putImageData( imageData, 0, 0 );
```

```
43        //this.nl.emit( this.name, null );
44      },false);
45    }
46
47     execute( theta ) {
48        this.worker.postMessage({image:this.image, theta:theta, left:this.
              left});
49        //console.log(this.name);
50
51    }
52  }
53
54  window.addEventListener('load',()=>{
55    let hoge = new Main();
56    hoge.bar();
57    console.time('time1');
58  });
```

プログラム 8　worker.js

```
1
2  let i = 0;
3  function calc(image, theta,left) {
4    let lambda = 8.6; // 波長は 8.6mm （音で言えば 40）kHz
5    let c_size = 512; // のサイズ（単位：ピクセル）canvas
6    let w_number = 16; // フィールド内の波の数
7    let s_number = 2; // 音源はつ　2で文字列を整数型にしている parseInt
8    let interval = 16; // 音源間隔（単位：ピクセル）
9
10   let k = 2.0 * Math.PI / lambda;
11   let m_size = ( lambda * w_number ) / c_size;
12
13   i++;
14 //console.log(theta);
15   for( let y=0; y<c_size; y++ ) {
16     for( let x=left; x<=left+c_size*0.5; x++ ) {
17       let sx = -interval * s_number/2.0 + interval/2.0;
18       let amp = 0;
19       for( let n=0; n<s_number; n++ ) {
20         let px = c_size / 2.0 - x- sx ;
21         let py = c_size / 2.0 - y;
22         let r = Math.sqrt( ( px * m_size * px * m_size ) + ( py * m_size
               * py * m_size ) );
23         amp -= Math.sin( - k * r + Math.PI * 2.0 / 360.0 * theta );
24         sx += interval;
25       }
26       let wh = Math.floor( 127 + 126.0 * amp / s_number );
27       if( wh<0 || 255<wh ) console.log( ？？？ "" );
28       image[ y * c_size * 4 + x * 4 + 0 ] = 0; //R
29       image[ y * c_size * 4 + x * 4 + 1 ] = wh; //G
30       image[ y * c_size * 4 + x * 4 + 2 ] = wh; //B
31       image[ y * c_size * 4 + x * 4 + 3 ] = 255; //a
32     }
33   }
34 }
35 onmessage = (evt)=>{
36   let image = evt.data.image;
37   let theta = evt.data.theta;
38   let left = evt.data.left;
39   //setInterval(() => {calc(image.data, theta);},100);
40   calc(image.data,theta,left);
41   postMessage({image:image, i:i, theta:theta});
```

```
42   //calc( image.data, theta );
43
44   //theta += 4.5; // フレームで位相を゜進める 14.5
45
46 }
```

12.4 手法 4

プログラム 9　offsc.html

```
<!DOCTYPE HTML>
<html lang="ja-JP">
<head>
  <meta charset="UTF-8">
  <title次元の音波>1</title>
</head>
<body>
  <canvas class="myCanvas" id="graph0" height="512px" width="512px"></
      canvas>
  <canvas class="myCanvas" id="graph1" height="512px" width="512px"
      style="margin-left:-516px"></canvas>

  <script src="main.js"></script>
</body>
</html>
```

プログラム 10　main.js

```
window.addEventListener('load',()=>{

  //要素を取得する canvas
  let canvas = document.getElementById('graph0');
  let canvas1 = document.getElementById('graph1');

  //要素の描画コントロールをに委譲する canvasoffscreen
  let offscreenCanvas = canvas.transferControlToOffscreen();
  let offscreenCanvas1 = canvas1.transferControlToOffscreen();
  //を作成し，を渡す workeroffscreenCanvas.
  let worker = new Worker('many-source.js');
  worker.postMessage({canvas:offscreenCanvas},[offscreenCanvas]);

  let worker1 = new Worker('many-source1.js');
  worker1.postMessage({canvas1:offscreenCanvas1},[offscreenCanvas1]);
});
```

プログラム 11　manY-source.js

```
let theta = 0;
let i = 0;
onmessage = (evt)=>{
let offsc = evt.data.canvas;
let sound = 2;
let inter = 16;
let ctx = offsc.getContext('2d');
console.time('timer');
//setInterval(() => {calc(offsc, ctx, i, theta);},100);
function calc( ) {
let lambda = 8.6; // 波長は 8.6mm （音で言えば 40）kHz
let c_size = 512; // のサイズ（単位：ピクセル）canvas
let w_number = 16; // フィールド内の波の数
let s_number = sound; // 音源はつ　2で文字列を整数型にしている parseInt
let interval = inter; // 音源間隔（単位：ピクセル）

ctx.clearRect( 0,0,512,512);

let imageData = ctx.createImageData( 512,512 );
let pixelData = imageData.data;

let k = 2.0 * Math.PI / lambda;
let m_size = ( lambda * w_number ) / c_size;

i++;

for( let y=0; y<c_size; y++ ) {
  for( let x=0; x<=c_size*0.5; x++ ) {
    let sx = -interval * s_number/2.0 + interval/2.0;
    let amp = 0;
    for( let n=0; n<s_number; n++ ) {
      let px = c_size / 2.0 - x- sx ;
      let py = c_size / 2.0 - y;
      let r = Math.sqrt( ( px * m_size * px * m_size ) + ( py * m_size *
           py * m_size ) );
      amp -= Math.sin( - k * r + Math.PI * 2.0 / 360.0 * theta );
      sx += interval;
    }
    let wh = Math.floor( 127 + 126.0 * amp / s_number );
    if( wh<0 || 255<wh ) console.log( ？？？ "" );
    pixelData[ y * c_size * 4 + x * 4 + 0 ] = 0; //R
    pixelData[ y * c_size * 4 + x * 4 + 1 ] = wh; //G
```

```
42      pixelData[ y * c_size * 4 + x * 4 + 2 ] = wh; //B
43      pixelData[ y * c_size * 4 + x * 4 + 3 ] = 255; //a
44    }
45  }
46  ctx.putImageData( imageData, 0, 0 );
47  if(i%1000 == 0)
48  {
49    console.log( i );
50    console.timeEnd('timer');
51    console.time('timer');
52  }
53
54  theta += 4.5; // フレームで位相を゜進める 14.5
55    //requestAnimationFrame(calc);
56    setInterval( calc,0);
57  }
58  //requestAnimationFrame(calc);
59  setInterval( calc,0);
60  };
```

プログラム 12　manY-source.js

```
let theta = 0;
let i = 0;
onmessage = (evt)=>{
let offsc = evt.data.canvas1;
let sound = 2;
let inter = 16;
let ctx = offsc.getContext('2d');
console.time('timer2');
//setInterval(() => {calc(offsc, ctx, i, theta);},100);
function calc( ) {
let lambda = 8.6; // 波長は 8.6mm （音で言えば 40）kHz
let c_size = 512; // のサイズ（単位：ピクセル）canvas
let w_number = 16; // フィールド内の波の数
let s_number = sound; // 音源はつ　2で文字列を整数型にしている parseInt
let interval = inter; // 音源間隔（単位：ピクセル）

ctx.clearRect( 0,0,512,512);

let imageData = ctx.createImageData( 512,512 );
let pixelData = imageData.data;

let k = 2.0 * Math.PI / lambda;
let m_size = ( lambda * w_number ) / c_size;

i++;

for( let y=0; y<c_size; y++ ) {
  for( let x=512*0.5; x<=c_size; x++ ) {
    let sx = -interval * s_number/2.0 + interval/2.0;
    let amp = 0;
    for( let n=0; n<s_number; n++ ) {
      let px = c_size / 2.0 - x- sx ;
      let py = c_size / 2.0 - y;
      let r = Math.sqrt( ( px * m_size * px * m_size ) + ( py * m_size *
           py * m_size ) );
      amp -= Math.sin( - k * r + Math.PI * 2.0 / 360.0 * theta );
      sx += interval;
    }
    let wh = Math.floor( 127 + 126.0 * amp / s_number );
    if( wh<0 || 255<wh ) console.log( ？？？ "" );
    pixelData[ y * c_size * 4 + x * 4 + 0 ] = 0; //R
    pixelData[ y * c_size * 4 + x * 4 + 1 ] = wh; //G
```

```
42      pixelData[ y * c_size * 4 + x * 4 + 2 ] = wh; //B
43      pixelData[ y * c_size * 4 + x * 4 + 3 ] = 255; //a
44    }
45  }
46  ctx.putImageData( imageData, 0, 0 );
47  if(i%1000 == 0)
48  {
49    console.log( i );
50    console.timeEnd('timer2');
51    console.time('timer2');
52  }
53
54  theta += 4.5; // フレームで位相を゜進める 14.5
55    //requestAnimationFrame(calc);
56    setInterval( calc,0);
57  }
58  //requestAnimationFrame(calc);
59  setInterval( calc,0);
60  };
```

12.5 手法 4 の数値変更

プログラム 13　offsc.html

```
<!DOCTYPE HTML>
<html lang="ja-JP">
<head>
  <meta charset="UTF-8">
  <title次元の音波>1</title>
</head>
<body>
  <canvas class="myCanvas" id="graph0" height="512px" width="512px"></
      canvas>
  <canvas class="myCanvas" id="graph1" height="512px" width="512px"
      style="margin-left:-516px"></canvas>

  <script src="main.js"></script>
</body>
</html>
```

プログラム 14　main.js

```
1
2  class Main {
3    constructor() {
4      this.i = 0; //の数 sound
5      this.j = 0; 間の大きさ//
6
7      this.canvas = document.getElementById('graph0'); //の要素を取得 canvas
8      this.canvas1 = document.getElementById('graph1');
9      this.offscreenCanvas = this.canvas.transferControlToOffscreen();//要素
           の描画コントロールをに委譲する canvasoffscreen
10     this.offscreenCanvas1 = this.canvas1.transferControlToOffscreen();
11     this.worker = new Worker('many-source.js'); //を作成 worker
12     this.worker1 = new Worker('many-source1.js');
13     //setInterval(() => {main(canvas,canvas1,i,j,offscreenCanvas,
           offscreenCanvas1);},30);
14
15     this.worker.postMessage( { canvas:this.offscreenCanvas, sound:2,
           interval:16 }, [this.offscreenCanvas] );
16     this.worker1.postMessage( { canvas:this.offscreenCanvas1, sound:2,
           interval:16 }, [this.offscreenCanvas1] );
17
18     var nl = new nylon();
19     nl.on( 'root', ( key, params ) => {
20       this.worker.postMessage( { sound: params.root } );
21       this.worker1.postMessage( { sound: params.root } );
22     });
23     nl.on( 'kyori', ( key, params ) => {
24       this.worker.postMessage( { interval: params.kyori } );
25       this.worker1.postMessage( { interval: params.kyori } );
26     })
27   }
28
29 }
30
31 var guisetup = () => {
32   var nl = new nylon();
33   document.querySelector('#root').addEventListener('change', (evt) => {
34     console.log(evt.target.value);
35     nl.emit( "root", { "root": evt.target.value});
36     document.querySelector('#output1').value = evt.target.value;
37   });
38   document.querySelector('#kyori').addEventListener('change', (evt) => {
```

```
39      console.log(evt);
40      nl.emit( "kyori", { "kyori": evt.target.value});
41      document.querySelector('#output2').value = evt.target.value;
42    });
43  }
44  window.addEventListener("load",()=>{
45    guisetup();
46    x = new Main();
47  });
```

プログラム 15　manY-source.js

```
class many_source {
  constructor() {
    this.theta = 0;
    this.i = 0;
    this.moving = false;
    self.addEventListener('message', (evt) => {
      if( evt.data.canvas != null ) {
        this.offsc = evt.data.canvas;
        this.ctx = this.offsc.getContext('2d');
        console.time('timer');
      }
      if( evt.data.sound != null ) this.sound = evt.data.sound;
      if( evt.data.interval != null ) this.inter = evt.data.interval;
      if( this.moving == false ) {
        this.moving = true;
        requestAnimationFrame(this.calc.bind(this));
      }
    });
  }
  calc( ) {
    let lambda = 8.6; // 波長は 8.6mm （音で言えば 40）kHz
    let c_size = 512; // のサイズ（単位：ピクセル）canvas
    let w_number = 16; // フィールド内の波の数
    let s_number = this.sound; // 音源はつ　2で文字列を整数型にしている
        parseInt
    let interval = this.inter; // 音源間隔（単位：ピクセル）

    this.ctx.clearRect( 0,0,512,512);

    let imageData = this.ctx.createImageData( 512,512 );
    let pixelData = imageData.data;

    let k = 2.0 * Math.PI / lambda;
    let m_size = ( lambda * w_number ) / c_size;

    this.i++;

    for( let y=0; y<c_size; y++ ) {
      for( let x=0; x<=c_size*0.5; x++ ) {
        let sx = -interval * s_number/2.0 + interval/2.0;
        //console.log(sx);
        let amp = 0;
```

```
42          for( let n=0; n<s_number; n++ ) {
43            let px = c_size / 2.0 - x- sx ;
44            let py = c_size / 2.0 - y;
45            let r = Math.sqrt( ( px * m_size * px * m_size ) + ( py *
                  m_size * py * m_size ) );
46            amp -= Math.sin( - k * r + Math.PI * 2.0 / 360.0 * this.theta
                  );
47            sx += interval;
48          }
49          let wh = Math.floor( 127 + 126.0 * amp / s_number );
50          if( wh<0 || 255<wh ) console.log( ？？？ "" );
51          pixelData[ y * c_size * 4 + x * 4 + 0 ] = 0; //R
52          pixelData[ y * c_size * 4 + x * 4 + 1 ] = wh; //G
53          pixelData[ y * c_size * 4 + x * 4 + 2 ] = wh; //B
54          pixelData[ y * c_size * 4 + x * 4 + 3 ] = 255; //a
55        }
56      }
57      this.ctx.putImageData( imageData, 0, 0 );
58
59      this.theta += 4.5; // フレームで位相を゜進める 14.5
60      //console.log(this.theta);
61      requestAnimationFrame(this.calc.bind(this));
62    }
63  }
64
65  new many_source();
```

プログラム 16　manY-source.js

```
class many_source1 {
  constructor() {
    this.theta = 0;
    this.i = 0;
    this.moving = false;
    self.addEventListener('message', (evt) => {
      if( evt.data.canvas != null ) {
        this.offsc = evt.data.canvas;
        this.ctx = this.offsc.getContext('2d');
        console.time('timer1');
      }
      if( evt.data.sound != null ) this.sound = evt.data.sound;
      if( evt.data.interval != null ) this.inter = evt.data.interval;
      if( this.moving == false ) {
        this.moving = true;
        requestAnimationFrame(this.calc.bind(this));
      }
    });
  }
  calc( ) {
    let lambda = 8.6; // 波長は 8.6mm （音で言えば 40）kHz
    let c_size = 512; // のサイズ（単位：ピクセル）canvas
    let w_number = 16; // フィールド内の波の数
    let s_number = this.sound; // 音源はつ　2で文字列を整数型にしている
        parseInt
    let interval = this.inter; // 音源間隔（単位：ピクセル）

    this.ctx.clearRect( 0,0,512,512);

    let imageData = this.ctx.createImageData( 512,512 );
    let pixelData = imageData.data;

    let k = 2.0 * Math.PI / lambda;
    let m_size = ( lambda * w_number ) / c_size;

    this.i++;

    for( let y=0; y<c_size; y++ ) {
      for( let x=c_size*0.5; x<=c_size; x++ ) {
        let sx = -interval * s_number/2.0 + interval/2.0;
        //console.log(sx);
        let amp = 0;
```

```
42          for( let n=0; n<s_number; n++ ) {
43            let px = c_size / 2.0 - x- sx ;
44            let py = c_size / 2.0 - y;
45            let r = Math.sqrt( ( px * m_size * px * m_size ) + ( py *
                  m_size * py * m_size ) );
46            amp -= Math.sin( - k * r + Math.PI * 2.0 / 360.0 * this.theta
                  );
47            sx += interval;
48          }
49          let wh = Math.floor( 127 + 126.0 * amp / s_number );
50          if( wh<0 || 255<wh ) console.log( ？？？ "" );
51          pixelData[ y * c_size * 4 + x * 4 + 0 ] = 0; //R
52          pixelData[ y * c_size * 4 + x * 4 + 1 ] = wh; //G
53          pixelData[ y * c_size * 4 + x * 4 + 2 ] = wh; //B
54          pixelData[ y * c_size * 4 + x * 4 + 3 ] = 255; //a
55        }
56      }
57      this.ctx.putImageData( imageData, 0, 0 );
58
59      this.theta += 4.5; // フレームで位相を゜進める 14.5
60      //console.log(this.theta);
61      requestAnimationFrame(this.calc.bind(this));
62    }
63  }
64
65  new many_source1();
```

プログラム 17　nYlon.js

```
1
2  /**
3   * クライアント側の本体 nylon
4   * 関係の機能を使う場合は必ず読み込んでください nylon
5   */
6  class nylon {
7
8          /**
9           * constructor
10          * グローバル変数, nylonnylon.などを初期化します map
11          */
12         constructor() {
13                 /**
14                  * @type {hash}
15                  */
16                 this.nylon = window.nylon;
17
18                 /**
19                  * @type {window}
20                  */
21                 this.parent = window.parent;
22
23                 if( window.nylon == null ) {
24                         window.nylon = {};
25                 }
26                 if( window.nylon.map == null ) {
27                         window.nylon.map = {};
28                 }
29
30                 /**
31                  * @type {array}
32                  */
33                 this.map = window.nylon.map;
34         }
35
36         /**
37          * 受け取ったイベント情報をそのまま外部に渡すための関数
38          * nylon.で使用します iFrame
39          * @param {function} fn - イベントを処理するコールバック関数
40          */
41         setPassThrough( fn ) {
42                 if( window.nylon.passthrough == null ) {
```

```
43                      window.nylon.passthrough = fn;
44              } else {
45                      console.log( "Error by duplexy registration" );
46              }
47      }
48
49      /**
50       * キーワードと，キーワードに対する処理を登録する関数
51       * @param { string } keyword - キーワード
52       * @param { function } fn - コールバック関数
53       */
54      on( keyword, fn ) {
55              console.log( this );
56              if( this.map[ keyword ] == null ) {
57                      //if( window != parent ) {
58                      // this.parent.postMessage( { "keyword": ["on"], "
                            params": {"keyword": ["on"], "key":keyword}},
                             nylon.origin );
59                      //}
60                      console.log( "new keyword : " + keyword );
61                      this.map[ keyword ] = [ new nylonfunc( fn, this )
                            ];
62              } else {
63                      this.map[ keyword ].push( new nylonfunc( fn, this
                            ) );
64              }
65      }
66
67      /**
68       * イベントを起こす関数
69       * @param { string[] } keys - キーワードの配列
70       * @param { hash } params - パラメータ（ハッシュで与える）
71       */
72      emitByArray( keys, params ) {
73              if( window.nylon.passthrough != null ) {
74                      window.nylon.passthrough( keys, params, this );
75              }
76              if( params == null ) {
77                      params = {};
78              }
79              console.log( "-->" + params["keywords"] );
80
81              if( params["keywords"] == null ) {
```

```
82                      console.log("params.が keywordsnull");
83                      params["keywords"] = keys;
84              }
85
86              for( let key of keys ) {
87                      if( this.map[ key ] == null ) {
88                              console.log( "Invarid keyword " + key );
89                      } else {
90                              for( let element of this.map[ key ] ) {
91                                      //console.log( element );
92                                      //console.log( "-->" + element.obj
                                            );
93                                      if( element.obj != this ) {
94                                              element.fn( key, params );
95                                      }
96                              }
97                      }
98              }
99      }
100
101     /**
102      * イベントを起こす関数
103      * @param { string } keyword - キーワード. で区切ることで複数キー
           ワードを与える|
104      * @param { hash } params - パラメータ（ハッシュで与える）
105      */
106     emit( keyword, params ) {
107             var keys = keyword.split( "|" );
108             this.emitByArray( keys, params );
109     }
110 }
111
112 //export default nylon;
113 //exports = module.exports = nylon;
114
115 /**
116  * に登録する関数のクラス nylon
117  */
118 class nylonfunc {
119
120     /**
121      * コンストラクタ
122      * @param { function } func - コールバック関数
```

```
123          * @param { nylon } object - オブジェクト nylon
124          */
125         constructor( func, object ) {
126                 /** @type { function } */
127                 this.fn = func;
128                 /** @type { nylon } */
129                 this.obj = object;
130         }
131 }
132 //export default nylonfunc;
```