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，$(\mathrm{n}-0.5)\Delta t[s]$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

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

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 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 1 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．

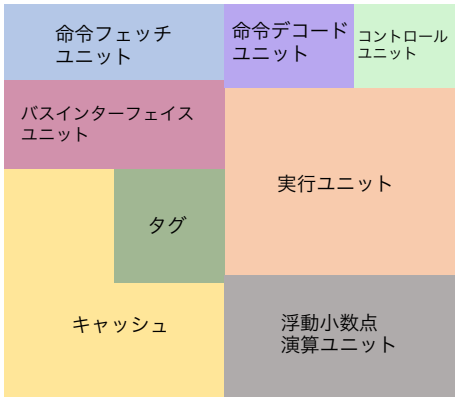

図 1　CPU の内部構造の略図

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．この CPU は電卓メーカーのビジコンが電卓の制御用に依頼したものである．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．CPU は，加法，減法命令，メモリの読み書きなど，各種命令を実行するのが目的である．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．1985 年に Sun Microsystems が SPARK を開発し,1986 年には ARM 社から低消費電力を重視した ARM2 が登場した..

その後，パソコンにおいて従来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．家庭用の CPU は 32 ビットへの以降が進んでいた．

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．

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．

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 年に周波数の向上に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し，命令の長さを減らした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．

図 2　グラフィックボード

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

1999 年に NVIDIA から 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，Fermi の登場により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しこの数を 1 つの単位とみなすものをワープとよぶ．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．

5 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．

5.1 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．

5.2 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．

5.2.1 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．

5.2.2 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 のしようが多く盛り込まれた．

5.3 DOM

5.4 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．

5.4.1 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．

5.4.2 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．

5.4.3 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

5.4.4 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．

5.5 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．

5.5.1 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;
};
```

5.6 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 Android Brower，Chrome for Android，Firefox，Opera と Opera for Android が OffscreenCanvas に対応している．

5.6.1 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．

```
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．

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

データを postMessage を用いてワーカーに送る

```
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．

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

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．

また，ワーカーと同様に window オブジェクト，Document オブジェクトや DOM の操作を行うことができない．

6 開発したシミュレータ

本研究では，OffscreenCanvas を利用し，FTDT 法によって動作するシミュレータ教材の軽量化とそれによって得られた開発効率の検証を行った．そこで，本研究では昨年の本研究室の卒業論文である「GPU 利用によるシミュレータ教材の演算速度」を基に開発を行った．先行研究では CPU と GPU の速度比較を行うという研究だった．以降本研究では，昨年開発されたシミュレータで CPU を対象とした物を CPU,GPU で動作する物を GPU と呼ぶ．同様に本研究で開発した OffscreenCanvas を用いたシミュレータを Offscreen，WebWorker を用いて制作したシミュレータを Worker と呼ぶことにする．HTML でキャンバスの大きさを制作した場合，その大きさも送信されるが，OffscreenCanvas 上でキャンバスの大きさを指定することもできる．

本章では実際に開発したシミュレータと理論の開設を図を交えて説明，実験の方法まで論じる．

6.1 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．

また，昨年度 CPU 向けに開発されたプログラムを処理 1，GPU 向けに開発されたプログラムを処理 2，本研究で開発する WebWorker を利用したプログラムを処理 3，OffscreenCanvas を利用したプログラムを処理 4 とする．

6.1.1 アルゴリズム

処理 1 では

6.2 実験方法

本章では計測を実施した環境とその方法を記載する．

6.2.1 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)

6.3 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 はより計測時間を必要としたキャンバスを参照する．

$$\text{時間比} = \frac{\text{各手法 } 10,000 \text{ 回の処理速度 } [s]}{\text{基準 } 10,000 \text{ 回の処理速度 } [s]} \tag{10}$$

7 実験結果

8 結言

9 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．

10 参考文献

[1] https://arui.tech/es2018-new-features/

[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] https://it-words.jp/w/SIMD.html

11 付録　制作したプログラム

```
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>
```

```
1  var theta = 0;
2  var i = 0;
3
4
5  function calc0() {
6    console.time('timer1');
7    let canvas = document.getElementById('graph0');
8    let ctx = canvas.getContext( "2d" );
9
10
11   let lambda = 8.6; // 波長は8.6mm （音で言えば40）kHz
12   let c_size = 512; // のサイズ（単位：ピクセル）canvas
13   let w_number = 16; // フィールド内の波の数
14   let s_number = 2 // 音源はつ  2で文字列を整数型にしているparseInt
15   let interval = 16// 音源間隔（単位：ピクセル）
16
17   ctx.clearRect( 0, 0, 512, 512);
18
19   let imageData = ctx.createImageData( 512, 512 );
20   let pixelData = imageData.data;
21
22
23   let k = 2.0 * Math.PI / lambda;
24   let m_size = ( lambda * w_number ) / c_size;
25
26   i++;
27
28   for( let y=0; y<c_size; y++ ) {
29     for( let x=0; x<c_size; x++ ) {
30       let sx = -interval * s_number/2.0 + interval/2.0;
31       //if( sx!=-8 ) console.log("Err("+x+","+y+")");
32       let amp = 0;
33       for( let n=0; n<s_number; n++ ) {
34         let px = c_size / 2.0 - x - sx;
```

```
35          let py = c_size / 2.0 - y;
36          let r = Math.sqrt( ( px * m_size * px * m_size ) + ( py * m_size *
37          amp += Math.sin( - k * r + Math.PI * 2.0 / 360.0 * theta );
38          sx += interval;
39        }
40        let wh = Math.floor( 127 + 126.0 * amp / s_number );
41        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  });
```

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

```
1  \caption{many_source.js}
2  let theta = 0;
3  var i = 0;
4  console.time('timer1');
5  function calc() {
6    let canvas = document.getElementById( 'graph0' );
7    let gl = canvas.getContext( "webgl" );
8
9    let lambda = 8.6; // 波長は8.6mm （音で言えば40）kHz
10   let c_size = 512; // のサイズcanvas
11   let w_number = 1; // フィールド内の波の数
12
13   i++;
14
15   gl.clearColor( 1.0, 1.0, 1.0, 1.0 );
16   gl.clear(gl.COLOR_BUFFER_BIT);
17
18   let imageData = gl.createBuffer();
19   gl.bindBuffer( gl.ARRAY_BUFFER, imageData );
20
21   // バーテックス頂点シェーダー()
22   var vSource = [
23     "precision mediump float;",
24     "attribute vec2 vertex;",
25     "attribute float theta_n;",
26     "attribute float distance_n;",
27     "varying float theta;",
28     "varying float distance;",
29     "void main(void) {",
30     "gl_Position = vec4(vertex, 0.0, 1.0);",
31     "theta = theta_n;",
32     "distance = distance_n;",
33     "}"
34   ].join("\n");
```

```
35
36    var vShader = gl.createShader(gl.VERTEX_SHADER);
37    gl.shaderSource(vShader, vSource);
38    gl.compileShader(vShader);
39    gl.getShaderParameter(vShader, gl.COMPILE_STATUS);
40
41    // フグメントシェーダー
42    var fSource = [
43      "precision mediump float;",
44      "varying float theta;",
45      "varying float distance;",
46      "void main(void) {",
47      "const float PI = 3.14159265358979323846264338327 95;",
48      "const float lambda = 8.6;",
49      "const float c_size = 512.0;",
50      "const float w_number = 16.0;",
51      "const float s_number = 2.0;",
52      "const float interval = 16.0;",
53
54      "const float k = 2.0 * PI / lambda;",
55      "const float m_size = ( lambda * w_number ) / c_size;",
56      "float sx = -interval * s_number/2.0 + interval/2.0;",
57      "float amp = 0.0;",
58      "for( int n=0; n<int(s_number); n++ ) {",
59      "float px = c_size / 2.0 - gl_FragCoord.x - sx;",
60      "float py = c_size / 2.0 - gl_FragCoord.y;",
61      "float r = sqrt( ( px * m_size * px * m_size ) + ( py * m_size * py *
62      "amp += sin( -k * r + PI * 2.0 / 360.0 * theta );",
63      "sx += interval;",
64      "}",
65      "amp /= 2.0;",
66      "gl_FragColor = vec4( 0, (amp+1.0)/2.0, (amp+1.0)/2.0, 1.0 );",
67      "}"
68    ].join("\n");
```

```
69
70      var fShader = gl.createShader(gl.FRAGMENT_SHADER);
71      gl.shaderSource(fShader, fSource);
72      gl.compileShader(fShader);
73      gl.getShaderParameter(fShader, gl.COMPILE_STATUS);
74
75
76      // プログラムオブジェクトの生成
77      var program = gl.createProgram();
78      gl.attachShader(program, vShader);
79      gl.attachShader(program, fShader);
80      gl.linkProgram(program);
81      gl.getProgramParameter(program, gl.LINK_STATUS);
82      gl.useProgram(program);
83
84
85      // シェーダー側の変数を側から設定するjs
86      var vertex = gl.getAttribLocation(program, "vertex");
87      gl.enableVertexAttribArray(vertex);
88      gl.vertexAttribPointer(vertex, 2, gl.FLOAT, false, 0, 0);
89
90      var theta_n = gl.getAttribLocation(program, "theta_n");
91      gl.vertexAttrib1f( theta_n, theta );
92
93      var distance_n = gl.getAttribLocation(program,"distance_n");
94      gl.vertexAttrib1f( distance_n, 1 );
95
96      // 点の座標をセット4
97      var vertices = [
98        -1.0, 1.0,
99        1.0, 1.0,
100       -1.0, -1.0,
101       1.0,-1.0
102     ];
```

```
103
104     if(i%1000 == 0)
105     {
106       console.log( i );
107       console.timeEnd('timer1');
108       console.time('timer1');
109     }
110
111     // 描画する
112     gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices), gl.DYNAMIC_DR
113     gl.drawArrays(gl.TRIANGLE_STRIP, 0, vertices.length/2);
114
115     theta += 4.5; // フレームで位相を°進める14.5
116     setInterval( calc,0 );
117 }
118
119
120
121
122
123
124
125
126 function calc1() {
127     let canvas = document.getElementById( 'graph1' );
128     let gl = canvas.getContext( "webgl" );
129
130     let lambda = 8.6; // 波長は8.6mm （音で言えば40）kHz
131     let c_size = 512; // のサイズcanvas
132     let w_number = 1; // フィールド内の波の数
133
134     i++;
135
136     gl.clearColor( 1.0, 1.0, 1.0, 1.0 );
```

```
137     gl.clear(gl.COLOR_BUFFER_BIT);
138
139     let imageData = gl.createBuffer();
140     gl.bindBuffer( gl.ARRAY_BUFFER, imageData );
141
142     // バーテックス頂点シェーダー()
143     var vSource = [
144       "precision mediump float;",
145       "attribute vec2 vertex;",
146       "attribute float theta_n;",
147       "attribute float distance_n;",
148       "varying float theta;",
149       "varying float distance;",
150       "void main(void) {",
151       "gl_Position = vec4(vertex, 0.0, 1.0);",
152       "theta = theta_n;",
153       "distance = distance_n;",
154       "}"
155     ].join("\n");
156
157     var vShader = gl.createShader(gl.VERTEX_SHADER);
158     gl.shaderSource(vShader, vSource);
159     gl.compileShader(vShader);
160     gl.getShaderParameter(vShader, gl.COMPILE_STATUS);
161
162     // フグメントシェーダー
163     var fSource = [
164       "precision mediump float;",
165       "varying float theta;",
166       "varying float distance;",
167       "void main(void) {",
168       "const float PI = 3.14159265358979323846264338327 95;",
169       "const float lambda = 8.6;",
170       "const float c_size = 512.0;",
```

```
171     "const float w_number = 16.0;",
172     "const float s_number = 2.0;",
173     "const float interval = 16.0;",
174
175     "const float k = 2.0 * PI / lambda;",
176     "const float m_size = ( lambda * w_number ) / c_size;",
177     "float sx = -interval * s_number/2.0 + interval/2.0;",
178     "float amp = 0.0;",
179     "for( int n=0; n<int(s_number); n++ ) {",
180     "float px = c_size / 2.0 - gl_FragCoord.x - sx;",
181     "float py = c_size / 2.0 - gl_FragCoord.y;",
182     "float r = sqrt( ( px * m_size * px * m_size ) + ( py * m_size * py *
183     "amp += sin( -k * r + PI * 2.0 / 360.0 * theta );",
184     "sx += interval;",
185     "}",
186     "amp /= 2.0;",
187     "gl_FragColor = vec4( 0, (amp+1.0)/2.0, (amp+1.0)/2.0, 1.0 );",
188     "}"
189   ].join("\n");
190
191   var fShader = gl.createShader(gl.FRAGMENT_SHADER);
192   gl.shaderSource(fShader, fSource);
193   gl.compileShader(fShader);
194   gl.getShaderParameter(fShader, gl.COMPILE_STATUS);
195
196
197   // プログラムオブジェクトの生成
198   var program = gl.createProgram();
199   gl.attachShader(program, vShader);
200   gl.attachShader(program, fShader);
201   gl.linkProgram(program);
202   gl.getProgramParameter(program, gl.LINK_STATUS);
203   gl.useProgram(program);
204
```

```
205
206     // シェーダー側の変数を側から設定するjs
207     var vertex = gl.getAttribLocation(program, "vertex");
208     gl.enableVertexAttribArray(vertex);
209     gl.vertexAttribPointer(vertex, 2, gl.FLOAT, false, 0, 0);
210
211     var theta_n = gl.getAttribLocation(program, "theta_n");
212     gl.vertexAttrib1f( theta_n, theta );
213
214     var distance_n = gl.getAttribLocation(program,"distance_n");
215     gl.vertexAttrib1f( distance_n, 1 );
216
217     // 点の座標をセット4
218     var vertices = [
219       -1.0, 1.0,
220       1.0, 1.0,
221       -1.0, -1.0,
222       1.0,-1.0
223     ];
224
225     if(i%1000 == 0)
226     {
227       console.log( i );
228       console.timeEnd('timer1');
229       console.time('timer1');
230     }
231
232     // 描画する
233     gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices), gl.DYNAMIC_DR
234     gl.drawArrays(gl.TRIANGLE_STRIP, 0, vertices.length/2);
235
236     theta += 4.5; // フレームで位相を°進める14.5
237     requestAnimationFrame( calc1 );
238 }
```

```
239
240
241
242 window.addEventListener('load', function() {
243     calc();
244     calc1()
245 });
```

```
\caption{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"></canv
  <canvas class="myCanvas" id="graph1" height="512px" width="512px"></canv
</body>
</html>
```

```
1  \caption{main.js}
2  class Main {
3    constructor() {
4      let canvas = document.getElementById('graph0');//要素の取得
   canvas
5      let canvas1 = document.getElementById('graph1');
6      this.theta = 0;
7      this.worker1 = new Work( "worker1", canvas, 0 );
8      this.worker2 = new Work( "worker2", canvas1, 256 );
9
10     //let nl = new nylon();
11     //nl.on( 'worker2', () => {
12     setInterval( () => {this.bar()}, 0 );
13     //})
14   }
15
16   bar() {
17     this.worker1.execute( this.theta );
18     this.worker2.execute( this.theta );
19     this.theta += 4.5;
20   }
21 }
22
23 class Work {
24   constructor( name, canvas, left ) {
25     this.name = name;
26     this.left = left;
27     this.worker = new Worker('worker.js');
28     this.ctx = canvas.getContext('2d');
29
30     this.image = this.ctx.getImageData( 0, 0, 512, 512 );
31     //this.nl = new nylon();
32
33     this.worker.addEventListener('message',(e) => {
34       let imageData = e.data.image;
```

```
35          let count1 = e.data.i;
36
37          if (count1 %1000 == 0){
38            console.log(count1);
39            console.time('time1');
40            console.timeEnd('time1');
41          }
42          this.ctx.clearRect(0,0,512,512);
43          this.ctx.putImageData( imageData, 0, 0 );
44          //this.nl.emit( this.name, null );
45        },false);
46      }
47
48       execute( theta ) {
49          this.worker.postMessage({image:this.image, theta:theta, left:this.le
50          //console.log(this.name);
51
52      }
53  }
54
55  window.addEventListener('load',()=>{
56      let hoge = new Main();
57      hoge.bar();
58      console.time('time1');
59  });
```

```
\caption{worker.js}

let i = 0;
function calc(image, theta,left) {
  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）

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

  i++;
//console.log(theta);
  for( let y=0; y<c_size; y++ ) {
    for( let x=left; x<=left+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 *
        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( ??? "" );
      image[ y * c_size * 4 + x * 4 + 0 ] = 0;  //R
      image[ y * c_size * 4 + x * 4 + 1 ] = wh; //G
      image[ y * c_size * 4 + x * 4 + 2 ] = wh; //B
      image[ y * c_size * 4 + x * 4 + 3 ] = 255;    //a
    }
  }
```

```
35 }
36 onmessage = (evt)=>{
37   let image = evt.data.image;
38   let theta = evt.data.theta;
39   let left = evt.data.left;
40   //setInterval(() => {calc(image.data, theta);},100);
41   calc(image.data,theta,left);
42   postMessage({image:image, i:i, theta:theta});
43   //calc( image.data, theta );
44
45   //theta += 4.5; // フレームで位相を°進める14.5
46
47 }
```

```
\caption{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"></canv
  <canvas class="myCanvas" id="graph1" height="512px" width="512px" style=

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

```
1  \caption{main.js}
2  window.addEventListener('load',()=>{
3
4    //要素を取得するcanvas
5    let canvas = document.getElementById('graph0');
6    let canvas1 = document.getElementById('graph1');
7
8    //要素の描画コントロールをに委譲するcanvasoffscreen
9    let offscreenCanvas = canvas.transferControlToOffscreen();
10   let offscreenCanvas1 = canvas1.transferControlToOffscreen();
11   //を作成し，を渡すworkeroffscreenCanvas.
12   let worker = new Worker('many_source.js');
13   worker.postMessage({canvas:offscreenCanvas},[offscreenCanvas]);
14
15   let worker1 = new Worker('many_source1.js');
16   worker1.postMessage({canvas1:offscreenCanvas1},[offscreenCanvas1]);
17 });
```

```
\caption{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;
```

```
35          let r = Math.sqrt( ( px * m_size * px * m_size ) + ( py * m_size * p
36          amp -= Math.sin( - k * r + Math.PI * 2.0 / 360.0 * theta );
37          sx += interval;
38        }
39        let wh = Math.floor( 127 + 126.0 * amp / s_number );
40        if( wh<0 || 255<wh ) console.log( ??? "" );
41        pixelData[ y * c_size * 4 + x * 4 + 0 ] = 0;   //R
42        pixelData[ y * c_size * 4 + x * 4 + 1 ] = wh; //G
43        pixelData[ y * c_size * 4 + x * 4 + 2 ] = wh; //B
44        pixelData[ y * c_size * 4 + x * 4 + 3 ] = 255;     //a
45      }
46  }
47  ctx.putImageData( imageData, 0, 0 );
48  if(i%1000 == 0)
49  {
50    console.log( i );
51    console.timeEnd('timer');
52    console.time('timer');
53  }
54
55  theta += 4.5; // フレームで位相を°進める14.5
56    //requestAnimationFrame(calc);
57    setInterval( calc,0);
58  }
59  //requestAnimationFrame(calc);
60  setInterval( calc,0);
61  };
```

```
\caption{many_source1.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;
```

```
35          let r = Math.sqrt( ( px * m_size * px * m_size ) + ( py * m_size * p
36          amp -= Math.sin( - k * r + Math.PI * 2.0 / 360.0 * theta );
37          sx += interval;
38        }
39        let wh = Math.floor( 127 + 126.0 * amp / s_number );
40        if( wh<0 || 255<wh ) console.log( ??? "" );
41        pixelData[ y * c_size * 4 + x * 4 + 0 ] = 0;  //R
42        pixelData[ y * c_size * 4 + x * 4 + 1 ] = wh; //G
43        pixelData[ y * c_size * 4 + x * 4 + 2 ] = wh; //B
44        pixelData[ y * c_size * 4 + x * 4 + 3 ] = 255;    //a
45      }
46  }
47  ctx.putImageData( imageData, 0, 0 );
48  if(i%1000 == 0)
49  {
50    console.log( i );
51    console.timeEnd('timer2');
52    console.time('timer2');
53  }
54
55  theta += 4.5; // フレームで位相を°進める14.5
56    //requestAnimationFrame(calc);
57    setInterval( calc,0);
58  }
59  //requestAnimationFrame(calc);
60  setInterval( calc,0);
61  };
```

```
1  \caption{offsc.html}
```

```
1  \caption{main.js}
2
3  class Main {
4    constructor() {
5      this.i = 0; //の数sound
6      this.j = 0; 間の大きさ//
7
8      this.canvas = document.getElementById('graph0');   //の要素を取
   得canvas
9      this.canvas1 = document.getElementById('graph1');
10     this.offscreenCanvas = this.canvas.transferControlToOffscreen();//
   要素の描画コントロールをに委譲するcanvasoffscreen
11     this.offscreenCanvas1 = this.canvas1.transferControlToOffscreen();
12     this.worker = new Worker('many_source.js'); //を作成worker
13     this.worker1 = new Worker('many_source1.js');
14     //setInterval(() => {main(canvas,canvas1,i,j,offscreenCanvas,offscreen
15
16     this.worker.postMessage( { canvas:this.offscreenCanvas, sound:2, inter
17     this.worker1.postMessage( { canvas:this.offscreenCanvas1, sound:2, int
18
19     var nl = new nylon();
20     nl.on( 'root', ( key, params ) => {
21       this.worker.postMessage( { sound: params.root } );
22       this.worker1.postMessage( { sound: params.root } );
23     });
24     nl.on( 'kyori', ( key, params ) => {
25       this.worker.postMessage( { interval: params.kyori } );
26       this.worker1.postMessage( { interval: params.kyori } );
27     })
28   }
29
30
31 }
32
33 var guisetup = () => {
```

```
34    var nl = new nylon();
35    document.querySelector('#root').addEventListener('change', (evt) => {
36      console.log(evt.target.value);
37      nl.emit( "root", { "root": evt.target.value});
38      document.querySelector('#output1').value = evt.target.value;
39    });
40    document.querySelector('#kyori').addEventListener('change', (evt) => {
41      console.log(evt);
42      nl.emit( "kyori", { "kyori": evt.target.value});
43      document.querySelector('#output2').value = evt.target.value;
44    });
45  }
46  window.addEventListener("load",()=>{
47    guisetup();
48    x = new Main();
49  });
```

```
\caption{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;
```

```
35
36      this.i++;
37
38      for( let y=0; y<c_size; y++ ) {
39        for( let x=0; x<=c_size*0.5; x++ ) {
40          let sx = -interval * s_number/2.0 + interval/2.0;
41          //console.log(sx);
42          let amp = 0;
43          for( let n=0; n<s_number; n++ ) {
44            let px = c_size / 2.0 - x- sx ;
45            let py = c_size / 2.0 - y;
46            let r = Math.sqrt( ( px * m_size * px * m_size ) + ( py * m_size
47            amp -= Math.sin( - k * r + Math.PI * 2.0 / 360.0 * this.theta );
48            sx += interval;
49          }
50          let wh = Math.floor( 127 + 126.0 * amp / s_number );
51          if( wh<0 || 255<wh ) console.log( ??? "" );
52          pixelData[ y * c_size * 4 + x * 4 + 0 ] = 0;   //R
53          pixelData[ y * c_size * 4 + x * 4 + 1 ] = wh; //G
54          pixelData[ y * c_size * 4 + x * 4 + 2 ] = wh; //B
55          pixelData[ y * c_size * 4 + x * 4 + 3 ] = 255;     //a
56        }
57      }
58      this.ctx.putImageData( imageData, 0, 0 );
59
60      this.theta += 4.5; // フレームで位相を°進める14.5
61      //console.log(this.theta);
62      requestAnimationFrame(this.calc.bind(this));
63    }
64  }
65
66  new many_source();
```

```
\caption{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;
```

```
35
36      this.i++;
37
38      for( let y=0; y<c_size; y++ ) {
39        for( let x=c_size*0.5; x<=c_size; x++ ) {
40          let sx = -interval * s_number/2.0 + interval/2.0;
41          //console.log(sx);
42          let amp = 0;
43          for( let n=0; n<s_number; n++ ) {
44            let px = c_size / 2.0 - x- sx ;
45            let py = c_size / 2.0 - y;
46            let r = Math.sqrt( ( px * m_size * px * m_size ) + ( py * m_size
47            amp -= Math.sin( - k * r + Math.PI * 2.0 / 360.0 * this.theta );
48            sx += interval;
49          }
50          let wh = Math.floor( 127 + 126.0 * amp / s_number );
51          if( wh<0 || 255<wh ) console.log( ??? "" );
52          pixelData[ y * c_size * 4 + x * 4 + 0 ] = 0;   //R
53          pixelData[ y * c_size * 4 + x * 4 + 1 ] = wh; //G
54          pixelData[ y * c_size * 4 + x * 4 + 2 ] = wh; //B
55          pixelData[ y * c_size * 4 + x * 4 + 3 ] = 255;     //a
56        }
57      }
58      this.ctx.putImageData( imageData, 0, 0 );
59
60      this.theta += 4.5; // フレームで位相を°進める14.5
61      //console.log(this.theta);
62      requestAnimationFrame(this.calc.bind(this));
63    }
64  }
65
66  new many_source1();
```