修 士 論 文 の 和 文 要 旨

研究科・専攻	大学院　情報理工学研究科　　情報・ネットワーク工学専攻　博士前期課程		
氏　　名	荒生 太一	学籍番号	2031008
論 文 題 目	レイトレーシング法に特化した GPU による電波伝搬シミュレーションの高速化		

要　旨

近年，第 5 世代移動通信システム(5G)のサービス開始や ITS(Intelligent Transport Systems: 高度道路交通システム)での車車間・路車間無線通信技術の発展など無線通信技術の利用がますます拡大している．

これらの通信システムを設計する際は受信する電波の変動特性を正しく把握しモデル化することが重要である．実世界での測定には多くのコストが必要であるため電波の伝わり方(電波伝搬)をコンピュータで計算する電波伝搬シミュレーションが盛んに行われてきた．

他方で，近年従来の汎用プロセッサに加えて特定の問題に特化させた専用プロセッサを搭載した GPU が普及し始めている．2018 年に発売された NVIDIA 社製の GPU，NVIDIA RTX シリーズではレイトレーシング法の計算を高速化するために RT コアと呼ばれる専用プロセッサを搭載している．レイトレーシング法に近い計算問題であれば同様に高速化することが可能であるため，GPU の高い計算性能が画像処理以外の分野に応用されてきたように，RT コアの高い計算性能を 3DCG 分野以外の計算問題に転用する動きが存在している．

このような背景から，本研究では，レイトレーシング法に特化した GPU である RTX を用いて電波伝搬シミュレーションの高速化を行った．実装の際は，CPU と GPU の間のデータ転送時間がボトルネックとなることからなるべくこれを避けるようなシステムを構築した．

作成したシステムを評価した結果，システムは CPU だけで処理を行った場合と同等の精度で計算を行うことが可能であり，かつ低負荷時でもおよそ 30 倍，高負荷時ではおよそ 5,000 倍高速であった．また，RT コアを搭載していない従来 GPU との比較により，RT コアがあることでレイトレーシング法の計算がより高速に行われていることが確かめられた．

令和3年度 修士論文

レイトレーシング法に特化したGPUによる電波伝搬シミュレーションの高速化

電気通信大学大学院 情報理工学研究科
情報・ネットワーク工学専攻 コンピュータサイエンスプログラム

学籍番号　2031008
氏名　荒生 太一
指導教員　成見 哲
副指導教員　沼尾 雅之

令和4年1月28日

概要

近年，第 5 世代移動通信システム (5G) のサービス開始や ITS(Intelligent Transport Systems:高度道路交通システム) での車車間・路車間無線通信技術の発展など無線通信技術の利用がますます拡大している．これらの通信システムを設計する際は受信する電波の変動特性を正しく把握しモデル化することが重要である．実世界での測定には多くのコストが必要であるため電波の伝わり方 (電波伝搬) をコンピュータで計算する電波伝搬シミュレーションが盛んに行われてきた．

他方で，近年従来の汎用プロセッサに加えて特定の問題に特化させた専用プロセッサを搭載した GPU が普及し始めている．2018 年に発売された NVIDIA 社製の GPU，NVIDIA RTX シリーズではレイトレーシング法の計算を高速化するために RT コアと呼ばれる専用プロセッサを搭載している．レイトレーシング法に近い計算問題であれば同様に高速化することが可能であるため，GPU の高い計算性能が画像処理以外の分野に応用されてきたように，RT コアの高い計算性能を 3DCG 分野以外の計算問題に転用する動きが存在している．

このような背景から，本研究では，レイトレーシング法に特化した GPU である RTX を用いて電波伝搬シミュレーションの高速化を行った．実装の際は，CPU と GPU の間のデータ転送時間がボトルネックとなることからなるべくこれを避けるようなシステムを構築した．

作成したシステムを評価した結果，システムは CPU だけで処理を行った場合と同等の精度で計算を行うことが可能であり，かつ低負荷時でもおよそ 30 倍，高負荷時ではおよそ 5,000 倍高速であった．また，RT コアを搭載していない従来 GPU との比較により，RT コアがあることでレイトレーシング法の計算がより高速に行われていることが確かめられた．

目次

1　はじめに

本章ではまず本研究の背景と目的について述べる．その後，本論文の構成とその概要を述べる．

1.1　研究背景

近年, 第 5 世代移動通信システム (5G) のサービス開始や ITS(Intelligent Transport Systems: 高度道路交通システム) での車車間・路車間無線通信技術の発展など無線通信技術の利用がますます拡大している．これらの通信システムを設計する際は受信する電波の変動特性を正しく把握しモデル化することが重要である [1] が，実世界での測定には多くのコストが必要であることから電波の伝わり方 (電波伝搬) をコンピュータで計算する電波伝搬シミュレーションが盛んに行われてきた．都市部での電波伝搬シミュレーションの様子を図 1 に示す．赤いエリアは電波の強度が高く，青いエリアは電波の強度が低いことを表している．

電波伝搬シミュレーション手法としては実験的モデルやマクスウェル方程式を直接解く FDTD(Finite Difference Time Domain) 法等があるが，近年の無線通信の大容量・高周波化および解析領域の大規模化に伴いレイトレーシング法が良く用いられている [3]．レイトレー

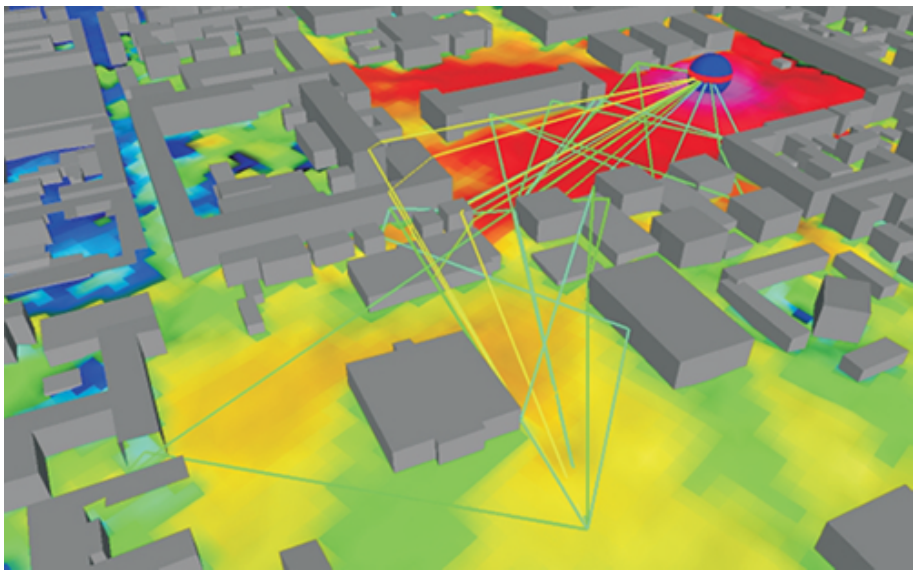

図 1　電波伝搬シミュレーション ([2] より引用)

シング法は幾何光学理論に基づき電波を直進する光とみなし，その軌跡を追跡することで電波伝搬推定を行う手法である．

他方で，画像処理を高速化するために開発されたハードウェアである GPU(Graphics Processing Unit) の高い並列計算能力を汎用的な数値計算に利用する GPGPU(General Purpose computing on GPU) が盛んに行われてきた [4]．GPU はより多くのプロセッサを搭載するように進化してきたが，近年では従来の汎用プロセッサに加えて特定の問題に特化させた専用プロセッサを搭載した GPU が普及し始めている．2018 年に発売された NVIDIA 社製の GPU，NVIDIA RTX シリーズではレイトレーシング法の計算を高速化するために RT コアと呼ばれる専用プロセッサを搭載している．RTX は 3DCG におけるレイトレーシング法を従来 GPU よりも高速に計算できるため，レイトレーシング法に近い計算問題であれば同様に高速化することが可能である．GPU の高い計算性能が画像処理以外の分野に応用されてきたように，RT コアの高い計算性能を 3DCG 分野以外の計算問題に転用する動きが存在している [5].

1.2 目的

このような背景から，本研究ではレイトレーシング法に特化した GPU である RTX を用いた電波伝搬シミュレーションの高速化を目的とする．RTX で電波伝搬シミュレーションを計算するシステムを作成し，RTX が従来 GPU よりもシミュレーションを高速に行えるかどうか検証した．また，CPU と計算精度なども合わせて比較することで実用性を検討した．

1.3 本論文の構成

本論文の構成を以下に示す．

1. はじめに
   本研究の背景と目的について述べる．
2. 電波伝搬シミュレーション
   電波伝搬シミュレーションの概要とレイトレーシング法で計算を行う方法について述べる．
3. NVIDIA RTX
   レイトレーシング法を高速に計算するための GPU である RTX と，それを使用するための API について述べる．

4. 既存研究
   関連する研究について説明し，本研究との差異を述べる．
5. システム概要 (CPU 版)
   CPU で電波伝搬シミュレーションを行うシステムの実装について述べる．
6. システム概要 (GPU 版)
   GPU で電波伝搬シミュレーションを行うシステムの実装について述べる．
7. 評価
   作成したシステムの評価を行う．
8. おわりに
   本研究のまとめと今後の展望について述べる．

2　電波伝搬シミュレーション

本章では電波伝搬シミュレーションの概要について述べる．

2.1　電波伝搬シミュレーションの種類

電波伝搬特性の研究として実際のフィールド実験より得られたデータを解析・モデル化する実験的アプローチと，モデル化した伝搬環境を電磁界理論で解析する理論的アプローチの 2 つが存在する．

前者の代表的な例としては都市部における伝搬環境をモデル化した奥村-秦式 [6] 等がある．実験的アプローチは計算が簡単で高速であるが，モデルを作成する際に用いたデータと同じような伝搬環境にのみ適用可能であり，様々な伝搬環境に対応できないという欠点がある．

後者の代表的な例としては FDTD 法やレイトレーシング法が存在する．FDTD 法はマクスウェルの方程式を数値的に解く手法であり高い精度で計算を行うことができるが，解析する空間のサイズが波長に比べて大きい場合必要なメモリサイズが増大し計算時間がかかるという欠点がある．レイトレーシング法は電波を光線 (レイ) とみなして解析を行う手法であり，高周波近似を前提とすることから FDTD 法よりも精度は低いが，計算時間とメモリサイズの観点から一般的な計算機でも実行可能であるため良く用いられている．

2.2　レイトレーシング法

図 2 にレイトレーシング法の概要図を示す．レイトレーシング法は送信点から発射された電波を無数の直進するレイとみなし，それらが解析領域内で反射や回折，透過を繰り返しながら受信点に到達するまでの経路をトレースする．反射や回折，透過といった現象はレイが持つ電界に影響を与えるため，レイを追跡する過程でこれらの現象が起きた場合電界を更新していき，最終的に受信点に到達した時点での各レイの電界を合計することで受信点での電界強度等を計算する．従って，レイトレーシング法による電波伝搬シミュレーションでは①レイが受信点に到達するまでの経路の探索手法と②反射・回折・透過を考慮した電界計算の 2 つが重要となる．ただし，今回作成するシステムでは簡単のためレイの反射現象のみを取り扱い，回折や透過については考慮しない．

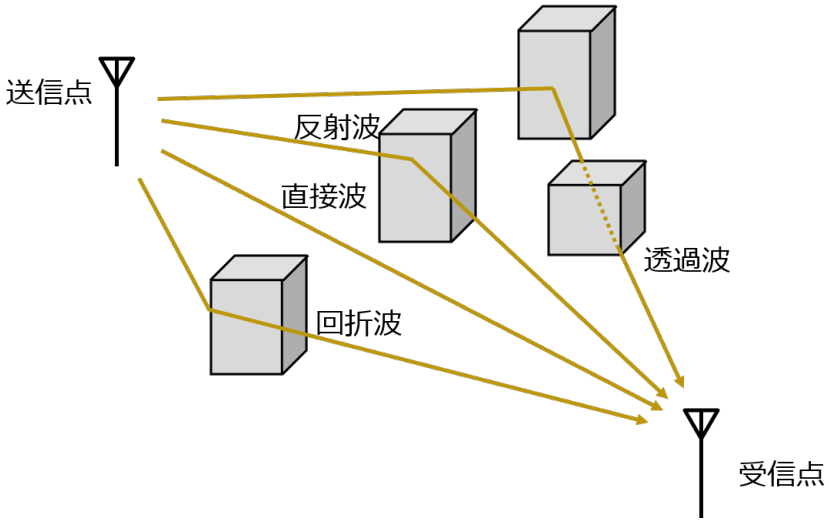

図 2　レイトレーシング法の概要図

2.2.1　イメージング法

レイの経路探索手法として，大きく分けてイメージング法と SBR(Shooting and Bouncing Rays) 法の 2 つが存在する．イメージング法は反射面の組み合わせからイメージ点を求めることでレイの軌跡を厳密に求める手法である．

イメージング法でレイの軌跡を求める方法を，送信点 (Tx)→ 反射面 1→ 反射面 2→ 受信点 (Rx) という経路をレイが辿る場合 (図 3) を例に説明する．

1. 送信点 Tx の反射面 1 に対する対称点 Tx’ を計算．以後このような点をイメージ点と呼ぶ．
2. Tx’ の反射面 2 に対するイメージ点 Tx” を計算
3. Tx” と Rx を結んだ線分と反射面 2 の交点 (Qr2) を計算
4. Tx’ と Qr2 を結んだ線分と反射面 1 の交点 (Qr1) を計算

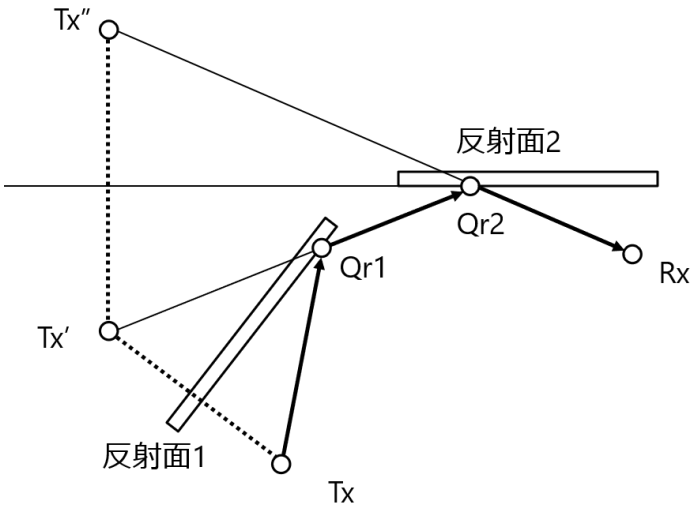

図 3　2 つの面で反射する経路をイメージング法で求める際の概要図

ここで，手順 3 や手順 4 で線分と反射面の交点が存在しない場合は送信点 (Tx)→ 反射面 1→ 反射面 2→ 受信点 (Rx) という経路を辿って受信点に到達するレイは存在しないとして他の面を通る組み合わせを計算する．

このようにイメージング法は「どの面を」「どの順番で」通って反射するかの情報を与えられれば幾何学的に反射点を求めることで厳密にレイの軌跡を計算できるが，計算対象とする領域を構成する面の数を M，レイの最大反射回数を N とすると考慮すべき組み合わせ数はおよそ M^N となる．従って，イメージング法の計算量は $O(M^N)$ となり，反射回数が増えると指数的に計算時間が増大してしまう．ただし，次の図 4 のような場合では計算量を抑えることが可能である．反射面 1 および反射面 2 が直交している場合，反射面 1，反射面 2 と反射する場合のイメージ点 Tx12 と反射面 2，反射面 1 と反射する場合のイメージ点 Tx21 は重なる性質を持つため，考慮する組み合わせ総数を削減することができる．特に屋内空間では部屋の壁が直交していることが多いため，計算量を大幅に削減することが可能である [7].

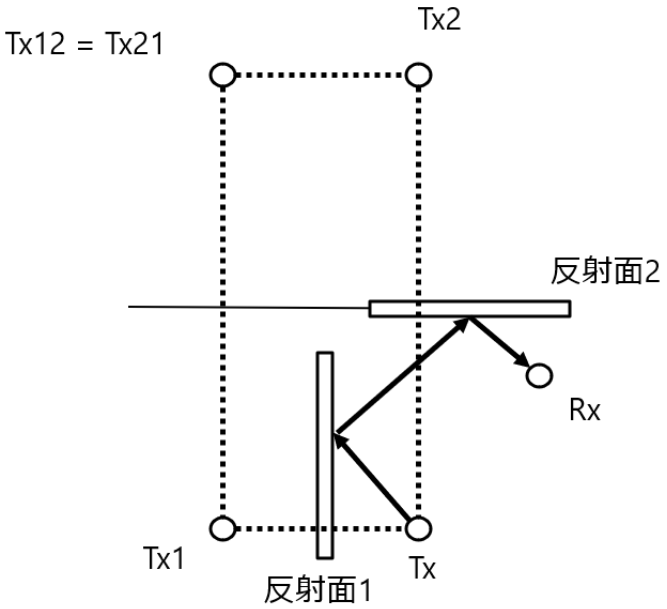

図 4　直交した面のイメージ点

2.2.2　SBR 法

SBR 法は送信点からレイを一定角度ごとに発射しその軌跡を逐次追跡することで受信点に到達するレイを探索する手法である．SBR 法の概要図を図 5 に示す．レイを離散的に出射するため，受信点に厳密にレイが到達する確率は非常に低くなる．そのため，受信点の周りに一定サイズの受信領域を設置しそこに入射したレイを受信点に到達したとみなす．受信領域には通常受信点を中心とする球が用いられ，受信球と呼ばれる．SBR 法では出射するレイの本数が少ないと精度が低下してしまうが，レイの本数が増えるほど計算時間が増大する．

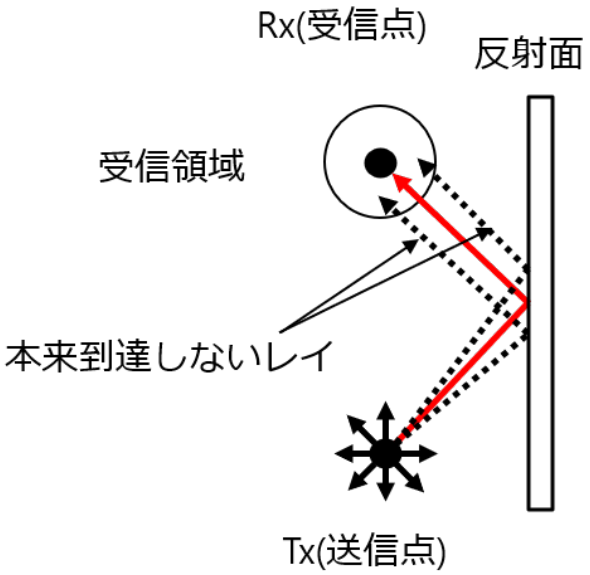

図 5　SBR 法の概要図

SBR 法の計算を行う場合，次の 3 点に注意する必要がある．

1. 一様にレイを発射する
2. 適切な受信球サイズの設定
3. 二重カウントの除去

まず SBR 法では送信点から一様な方向にレイを出射する必要がある．例えば指向性を持つアンテナであれば前方のみを考慮するだけでも良いが，等方性アンテナの場合は可能な限り均一にレイを出射しなければならない．2 次元空間の場合レイを一定角度 ($\Delta\Omega$) ごとに出射することで一様な方向に投射することができるが，3 次元空間では困難である．

3 次元空間でほぼ一様にレイを投射する方法としては正二十面体を用いた方法が考案されている [8]．正二十面体は 12 個の頂点と 20 個の正三角形の面を持つ正多面体であり，球に内接するもののなかで最も頂点数が多い．この各頂点の方向に対してレイを出射することで一様な方向にレイを出射することができる．ただし，このままでは 12 本しか出射することができないため図 6 のように各面を構成する正三角形をさらに分割することで任意の本数分レイを増やすことが可能であり，分割数を S とすれば生成されるレイの本数は $10S^2 + 2$ 本となる [9].

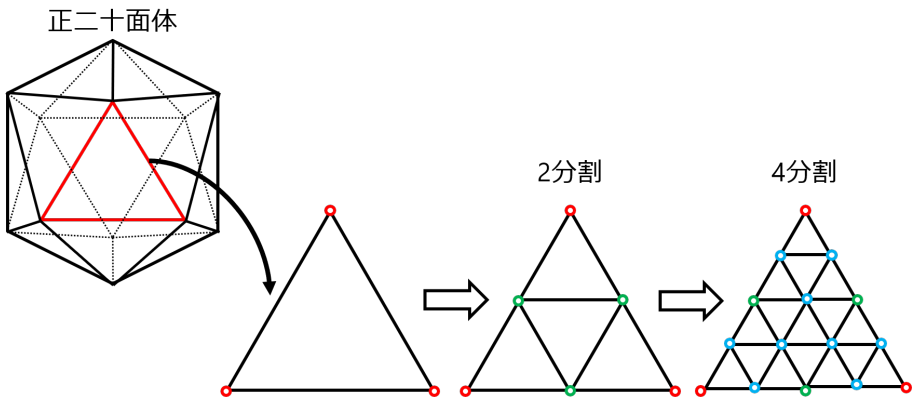

図 6　正二十面体の分割

2 つ目の受信球サイズの設定は重要な問題である．受信球の半径が小さすぎると本来受信点に到達したとしてカウントされるべきレイがカウントされず，大きすぎると本来受信点に到達しない経路のレイもカウントしてしまい誤差が増大する．従って，正確なシミュレーションを行うためにはカウントされるべきレイをカバーできる大きさよりも大きく，かつなるべく小さく設定する必要がある．前述の正二十面体によるレイの投射を行った際に，かならず 1 本以上のレイをカウントできるような受信球のサイズはレイ同士の最大分離角度を α，レイが受信球に到達するまでに辿った総距離を d とすると

$$r = \frac{\alpha d}{\sqrt{3}} \tag{1}$$

であることが既に示されている [8].

3 つ目の二重カウント除去は前述した受信球サイズの設定と合わせて計算精度に大きく影響を及ぼす部分である．例えば図 5 のような場合，黒い点線で示した 2 つのレイは同じ反射面で反射し受信点に到達している．このままでは 2 本のレイを総電界の計算に含めてしまうため，正しい電界 (赤で示したレイ 1 本分の電界) よりも大きな値が計算されてしまう．式 1 に示した最適な受信球サイズであっても隣接するレイ同士では二重カウントが起きる場合があるため，精度の良い結果を得るためには同一経路を辿ったレイを除去する必要がある．

2.2.3　SBR-image 法

SBR-image 法は SBR 法でレイが受信点に到達するまでに通る反射面を計算した後，その経路情報をもとにイメージング法の計算を行う手法である [10]．イメージング法で計算時間がかかるのは反射面の組み合わせを総当たりで求める部分であるため，SBR 法でレイの

経路が存在しそうな反射面の組み合わせを先に求め，それについてイメージング法を適用することで SBR 法よりも高精度，かつイメージング法よりも高速に計算を行うことができる．イメージング法によって正確にパスを計算するため，式 1 で示したサイズより受信球を大きくしたとしても精度よく計算を行うことが可能であるが，SBR 法と同様に二重カウント除去を行う必要がある．本研究ではこの SBR-image 法を RTX によって高速化した．

2.3 電界計算

2.3.1 各レイの電界計算

送信点から出射されたレイの電界 $\boldsymbol{E}_{out}$ は次式で与えられる．

$$\boldsymbol{E}_{out} = E_0 \frac{e^{-jkr_{total}}}{r_{total}} \boldsymbol{D}_{\mathrm{T}}(\theta_{\mathrm{T}}, \varphi_{\mathrm{T}}) (\prod_{i=1}^{N} \overline{\boldsymbol{R}}_i) \tag{2}$$

ここで E_0 は送信点における初期電界，k は $k = 2\pi/\lambda$(λ は波長) で与えられる波数，r_{total} はレイが辿った軌跡の総距離，$\boldsymbol{D}_{\mathrm{T}}(\theta_{\mathrm{T}}, \varphi_{\mathrm{T}})$ は送信アンテナの指向性利得 (θ_{T} は垂直面内の角度，φ_{T} は水平面内の角度)，N はレイが反射した回数，$\overline{\boldsymbol{R}}$ は反射係数である．

2.3.2 受信点の電界計算

受信点に N_{ray} 本のレイが到達するとき，各レイの電界を $\boldsymbol{E}_{out}^{(i)} (= E_{out}^{(i,\varphi)} \hat{\varphi}_{\mathrm{T}} + E_{out}^{(i,\theta)} \hat{\theta}_{\mathrm{T}})$ ($i = 0 \sim N_{ray}$) とする．これらの基底ベクトルはそれぞれ異なるため，受信アンテナを基準とする基底ベクトル $(\hat{\varphi}_{\mathrm{R}}, \hat{\theta}_{\mathrm{R}})$ へ変換した電界を $\boldsymbol{E}_{in}^{(i)} (= E_{in}^{(i,\varphi)} \hat{\varphi}_{\mathrm{R}} + E_{in}^{(i,\theta)} \hat{\theta}_{\mathrm{R}})$ とすると

$$\begin{bmatrix} E_{in}^{(i,\varphi)} \\ E_{in}^{(i,\theta)} \end{bmatrix} = \begin{bmatrix} \hat{\varphi}_{\mathrm{R}} \cdot \hat{\varphi}_{\mathrm{T}} & \hat{\varphi}_{\mathrm{R}} \cdot \hat{\theta}_{\mathrm{T}} \\ \hat{\theta}_{\mathrm{R}} \cdot \hat{\varphi}_{\mathrm{T}} & \hat{\theta}_{\mathrm{R}} \cdot \hat{\theta}_{\mathrm{T}} \end{bmatrix} \begin{bmatrix} E_{out}^{(i,\varphi)} \\ E_{out}^{(i,\theta)} \end{bmatrix} \tag{3}$$

となる．これを用いれば，受信点における電界は

$$\boldsymbol{E}_{in} = \sum_{i=0}^{N_{ray}} \boldsymbol{E}_{in}^{(i)} = \sum_{i=0}^{N_{ray}} E_{in}^{(i,\varphi)} \hat{\varphi}_{\mathrm{R}} + \sum_{i=0}^{N_{ray}} E_{in}^{(i,\theta)} \hat{\theta}_{\mathrm{R}} \tag{4}$$

となる．ただし，$i = 0$ の場合は受信点にレイが到達しない場合であり，$\boldsymbol{E}_{in}^{(0)}$ は 0 である．また，アンテナで受信される信号の複素振幅 a_{R} は

$$a_{\mathrm{R}} = \boldsymbol{E}_{in} \cdot \boldsymbol{D}_{\mathrm{R}}(\theta_{\mathrm{R}}, \varphi_{\mathrm{R}}) \frac{\lambda}{\sqrt{4\pi Z_0}} \tag{5}$$

で与えられる．ここで，$\boldsymbol{D}_{\mathrm{R}}(\theta_{\mathrm{R}}, \varphi_{\mathrm{R}})$ は受信アンテナの指向性，Z_0 は $\sqrt{\mu_0/\varepsilon_0}$(ただし μ_0 は真空の透磁率，ε_0 は真空の誘電率) で与えられる自由空間の固有インピーダンスである．これを用いれば，受信点における受信電力 P_{R} は

$$P_{\mathrm{R}} = |a_{\mathrm{R}}|^2 \tag{6}$$

と求めることができる．

3　NVIDIA RTX

本章では本研究で対象となる NVIDIA RTX のアーキテクチャと GPU で汎用計算を行う方法，RTX で計算を行うための API について述べる．

3.1　RTX の概要

従来の NVIDIA 社製 GPU と RTX の違いを図 7 に示す．RTX には汎用プロセッサである CUDA コアに加え，レイトレーシング法の計算に特化した RT コアと機械学習の計算に特化した TENSOR コアの 2 種類が追加されている．レイトレーシング法で一番計算時間がかかるのはレイと交差するオブジェクトを探索する処理であり，RTX の RT コアはこの交差判定処理に特化している．

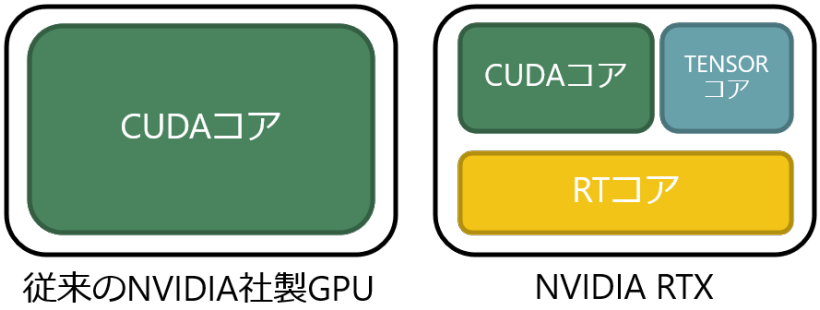

図 7　従来 GPU と RTX の違い

3.2 CUDA

NVIDIA 社製 GPU で計算を行うためのライブラリとして CUDA(Compute Unified Device Architecture) がある．CUDA によって計算を行う際のプログラミングモデルを図 8 に示す．

GPU で計算を行う場合，基本的に次のような順番で行われる．

1. ホスト (CPU) 側で計算に必要なデータを準備する．
2. 準備したデータをデバイス (GPU) に転送する．
3. 転送されたデータを用いて GPU 上で計算を行う．
4. 計算結果をホスト側に転送する．

3.3 NVIDIA OptiX

OptiX は CUDA を拡張して作成されており，レイトレーシング法を高速に計算するためのライブラリである [11]．RTX の RT コアを使用できる API は DXR(DirectX Raytracing) や Vulkan Raytracing 等が存在しているが，本研究では NVIDIA OptiX を使用した．これは，OptiX 以外の API はゲームのようなリアルタイム処理に重きを置いており，時間のかかるシミュレーションには向かないためである．OptiX のプログラミングモデルを図 9 に示す．

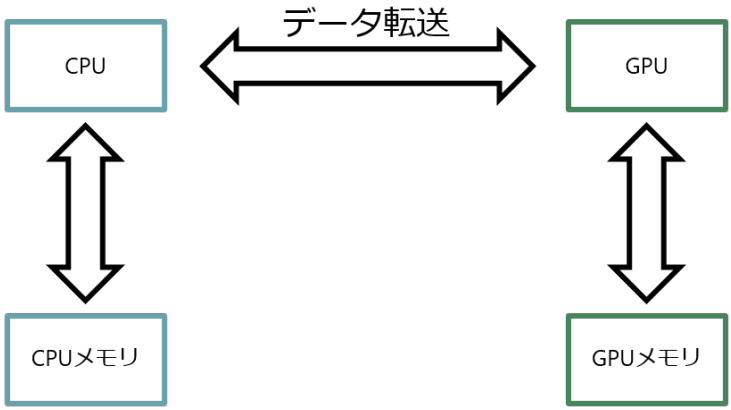

図 8　CUDA のプログラミングモデル

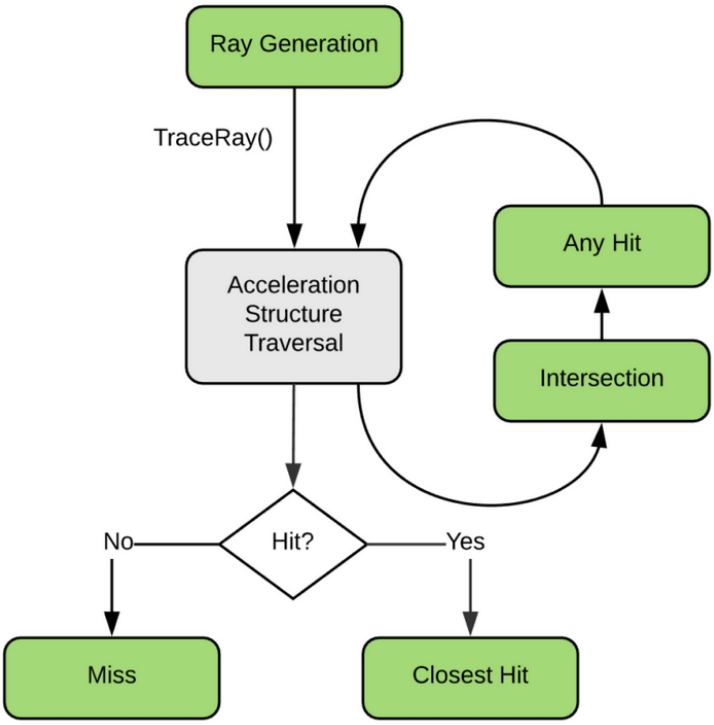

図 9　OptiX のプログラミングモデル ([12] より引用)

緑で塗られた部分がユーザーが定義する部分で，灰色の部分が OptiX 側の組み込み関数で処理される部分である．それぞれについて説明する．

3.3.1　Ray Generation プログラム

OptiX を起動したときに最初に呼ばれるプログラムである．レイの始点，方向といった初期条件を設定し，TraceRay() 関数を呼び出すことでレイトレーシングを行う．

3.3.2　Acceleration Structure Traversal

レイと交差するオブジェクトがあるかどうかの交差判定を行う部分である．Acceleration Structure(AS) とはシーンに存在する 3D オブジェクトをまとめた木構造のことである．AS の概要図を図 10 に示す．なお，シーンのオブジェクトは全て三角形ポリゴンで構成されているとする．

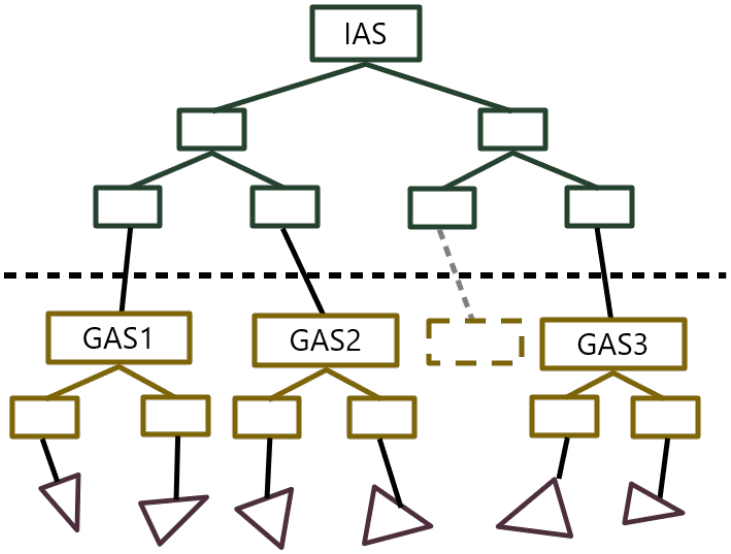

図 10　AS の概要図

AS は Geometry Acceleration Structure(GAS) と Instance Acceleration Structure(IAS) の二種類で構成されている．GAS にはオブジェクトを構成するポリゴンが木構造に分割して格納されている．この GAS に対して座標変換を行い，実際にシーンに配置したオブジェクトを IAS が木構造で保持している．レイとの交差判定を行う際は，まず IAS に対して交差判定を行いレイと交差が起きる可能性のある GAS を探索し，その後 GAS 内のポリゴンについてレイとの交差判定を行う．このようにすることでレイとオブジェクトの交差判定回数を $O(n)$ から $O(\log n)$(n はポリゴン数) に削減し，交差判定の高速化を行っている．

3.3.3　Intersection プログラム

Intersection プログラムはユーザーが定義した構造とレイの交差判定を行う際に呼び出される．例えば球とレイの交差判定を行う際はこのプログラムに交差判定処理を記述する必要がある．なお，三角形との交差判定は OptiX 側にあらかじめ組み込まれているため定義する必要がなく，RT コアによって高速に計算される．

3.3.4　Any Hit プログラム

Any Hit プログラムはレイと交差したオブジェクトがあった場合に呼び出されるプログラムである．そのポリゴンとの交差を確定するかどうかを決定することができる．例えば，交差したオブジェクトがガラスのような透過する物体であった場合はそのポリゴンとの交差を

破棄してレイの探索を続けることが可能である．

3.3.5 Closest Hit プログラム

Closest Hit プログラムは，レイと交差するポリゴンが確定した場合に呼び出されるプログラムである．ポリゴンの法線情報等を用いて反射方向の計算等を行う．

3.3.6 Miss プログラム

Miss プログラムはレイと交差するポリゴンが存在しなかった場合に呼び出されるプログラムである．3DCG においては背景色を返すことが多い．

4　既存研究

本章では本研究と類似した先行研究について説明し，本研究との差異を述べる．

4.1　Electromagnetic wave propagation in the millimeter wave band using the NVIDIA OptiX GPU ray tracing engine

4.1.1　研究の概要

Robert らは電波伝搬シミュレーションを OptiX を用いて実装し，Paray と呼ばれる OptiX とは別のレイトレーシングソフトウェアで実装した電波伝搬シミュレータと性能の比較を行った [13]．実装には SBR-image 法を用いている．60GHz 帯の周波数を使用し，図 11 に示す部屋でシミュレーションを行った際の計算速度や精度について実測値との比較を行っている．

比較の結果，OptiX で実装した電波伝搬シミュレータは Paray で実装したものよりも約 100 倍高速であり，より多くの反射回数に対応できることを確認している．

また，部屋の 3D モデルの粒度を変えた場合の計算精度についても述べられている．部屋を構成するポリゴン数を約 2 万，9 万，50 万として計算したところ，計算精度に大きな違い

図 11　解析に使用された部屋 ([13] より引用)

は見られなかったとしている．従って，レイトレーシング法で電波伝搬シミュレーションを計算する際は必要以上の細かさで解析領域を表現しても計算精度にはあまり影響を及ぼさないため，大まかな特徴さえ捉えていれば良いことが述べられている．

4.1.2　本研究との差異

Liu らの研究は，OptiX で SBR-image 法を実装しているという点で本研究と同様である．しかし，RTX ではない従来 GPU を使用しており，また速度の比較も CPU で計算する別のシミュレータとのもののみである．本研究では従来 GPU と RTX の計算速度の違いについても述べる．

4.2　GPU accelerated cone based shooting bouncing ray tracing

4.2.1　研究の概要

Troksa は電波伝搬シミュレーションを OptiX を用いて実装し，精度について商用電波伝搬シミュレーションソフトウェアと比較することで有用性の検証を行った [14]．実装には SBR 法を用いている．単純な直方体のトンネルや，図 12 のような 3D スキャンによって作成されたトンネルについてシミュレーションを行い，精度を実測値や FDTD 法等と比較している．

比較の結果，作成したシステムは大規模トンネル内における電波伝搬を商用ソフトウェアと同等の精度かつ高速に計算できることが示されている．また，出射するレイの本数と反射回数の収束についても述べられており，実装された SBR 法では数百万本程度のレイで十分

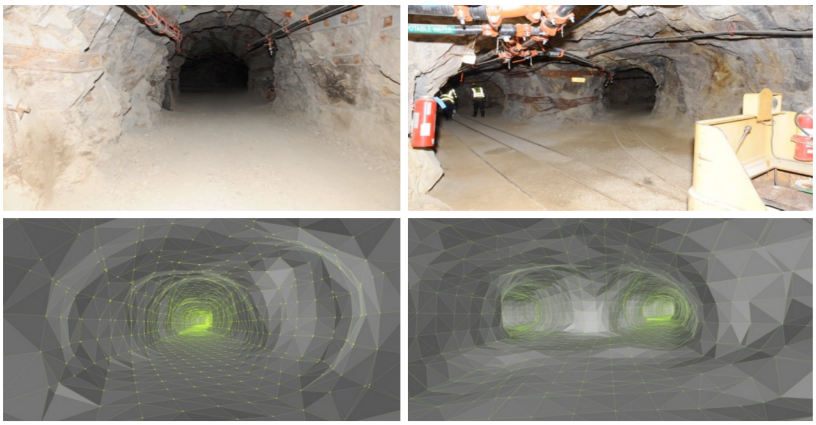

図 12　3D スキャンされたトンネル ([14] より引用)

な計算精度が達成されることが示されている．

4.2.2 本研究との差異

Troksa らの研究は，OptiX で電波伝搬シミュレーションを実装しているという点で本研究と同様である．しかし，SBR-image 法ではなく SBR 法を実装しているという点，商用ソフトウェアとの精度差に関する議論が中心であり速度に関する議論は従来 GPU と CPU のみの比較にとどまっているという点で本研究とは異なっている．

4.3 Opal: An open source ray-tracing propagation simulator for electromagnetic characterization

4.3.1 研究の概要

Esteban らは汎用ゲームエンジンである Unity で電波伝搬シミュレーションを行うためのプラグインとして Opal というオープンソースソフトウェアを開発した [15]．Opal は Unity 上で定義された 3D シーンデータを OptiX に転送し，OptiX 上で SBR 法による電界計算を行うプログラムである．様々な 3D シーンを手軽に生成できる Unity と組み合わせることでより柔軟なシーンに対応したシミュレーションを行うことが可能である．また，前述の 2 つの論文では実装されていなかった回折の計算も単純な場合については実装されており，より実用性を重視したソフトウェアになっている．Opal を使用している様子を図 13 に示す．利用者は解析を行いたいモデルを用意し，右側のパネルに周波数や反射回数といったシミュレーションの初期条件を設定するだけで簡単に電波伝搬シミュレーションを行うことが可能である．実装したシステムで計算した結果をトンネルや都市部での測定結果と比較している．

評価の結果，Opal は様々な環境について十分な精度でシミュレーションを行うことが可能であることが示されている．

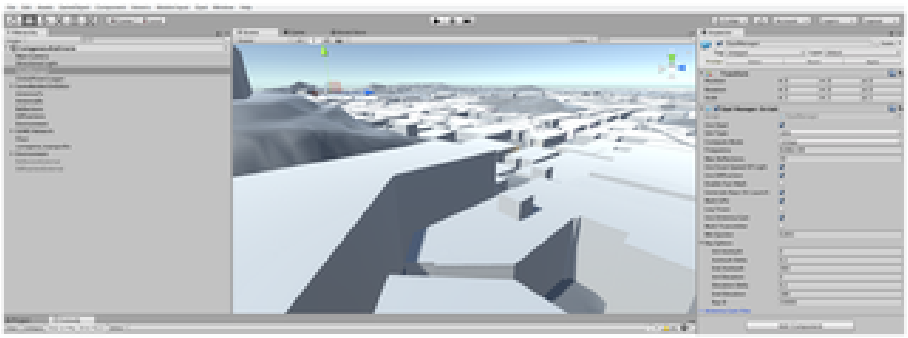

図 13　Opal によるシミュレーションの設定 ([15] より引用)

4.3.2　本研究との差異

Esteban らの研究は OptiX を用いて電波伝搬シミュレーションを行っている点で本研究と同様であるが，SBR-image 法ではなく SBR 法を実装しているという点，精度の比較が中心であり，速度に関する議論は著者の環境における実行時間が述べられたのみであるという点で本研究とは異なっている．

5　システム概要 (CPU 版)

本章では，CPU で実装した SBR 法及び SBR-image 法の概要と，それらの手法における二重カウント除去手法について述べる．

5.1　解析領域の表現

レイトレーシング法によるシミュレーションは解析領域のモデル化の精度に大きく依存するため，解析領域を正しく計算機上で表現する必要がある．今回作成したシステムでは解析領域をポリゴンで表現された 3D モデルで表現する．3D モデルを表現する形式はいくつか存在するが，今回は汎用性および簡便性から Wavefront obj 形式を使用する．3.3 節で述べたように，OptiX はレイと三角形との交差判定を高速に計算することができる．従って，三角形ポリゴンで解析領域の部屋や建物を表現することで RTX の性能を活かすことができると考えられる．また，obj はマテリアルファイル (mtl) を設定することができる．3DCG においてはテクスチャや色を設定するために用いられるが，電波伝搬では壁の材質に沿った反射係数等のシミュレーションに用いるパラメータを設定することで多様な解析領域に柔軟に対応することができる．さらに，近年では 3D スキャン技術が発展し詳細な環境データを 3D ポリゴンで得ることができるようになっているため，3D ポリゴンに対応したシミュレーションであればこれらの技術から得られたデータを容易に扱うことができる．なお，4.1.1 節でも述べたように多くのポリゴンを用いて解析領域の全てを精密にモデル化してもシミュレーション精度にはあまり影響を及ぼさないため，本研究では大まかな形だけを表現したモデルを用いた．

5.2　レイの初期方向の生成

5.2.1　初期方向生成手順

SBR 法を精度良く計算するためには，2.2.2 節で述べたようにレイを一様な方向に出射する必要がある．従って SBR 法や SBR-image 法の実装を行う前にまずレイを出射する初期方向を正二十面体の分割を繰り返すことで計算するプログラムを作成した．処理の流れとしては以下のようになる．

1. 頂点を格納する配列 icoVertex，三角形を格納する配列 triIndex を用意する．
2. icoVertex に初期座標として正二十面体の 12 個の頂点 $(\pm 1, \pm\phi, 0)), (0, \pm 1, \pm\phi), (\pm\phi, 0, \pm 1)$

を登録する．ここで ϕ は黄金比で $\phi = (1 + \sqrt{5})/2$ である．

3. triIndex に正二十面体の各面を登録する．icoVertex の要素を指し示すインデックス 3 つで三角形を表す．
4. 図 14 のように各三角形の頂点 2 つの中点を新しい座標として登録する．その後，(v0, a, c), (v1, b, a), (v2, c, b), (a, b, c) の 4 つの三角形を分割後の三角形を格納する配列である newTriIndex に登録する．
5. triIndex の中身がなくなるまで手順 4 を繰り返す
6. newTriIndex の中身を triIndex にコピーし，newTriIndex の中身を空にする．
7. 設定した最大分割回数 (maxSplitNum) に到達するまで手順 4 から手順 6 を繰り返す．

大きな分割数になると計算時間がかかるため，最終的に生成された頂点座標はテキストファイルに格納した．今後説明する SBR 法や SBR-image 法はこのテキストファイルを読み込むことでレイの初期方向を得る．

5.2.2　重複回避方法

手順 4 で中点を新しい座標として登録する際，そのまま登録してしまうと問題が発生する．例えば図 15 で青の三角形を処理した後黄色の三角形を処理する際，v0 と v2 の中点である頂点 a は既に登録されているため，このままでは頂点の重複が発生する．

重複を防ぐために，中点を生成する三角形のインデックス (32 ビット整数型) の小さい方を 64 ビット整数型の上位 32 ビット，大きい方を下位 32 ビットに格納したものをキーとした連想配列 middlePointIndex を用意する．中点を登録する際はキーを計算し，それが既に middlePointIndex に登録されていればペアになっているインデックスを，そうでなければ新

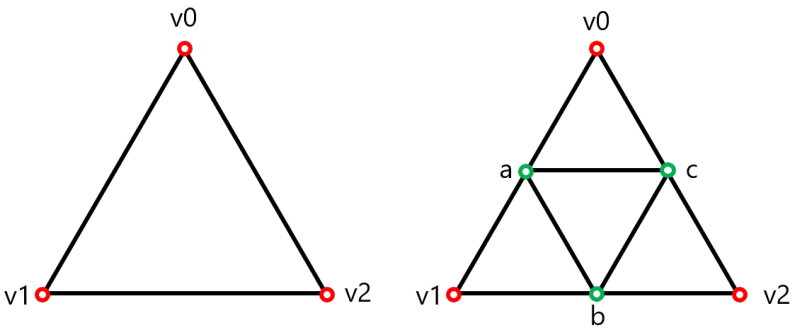

図 14　正三角形の分割

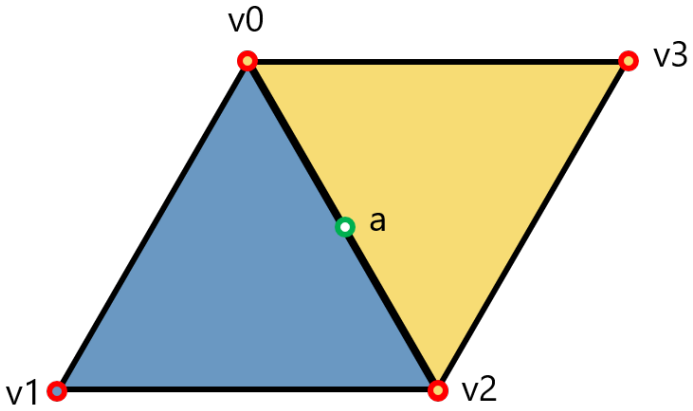

図 15　分割時の重複

しく中点を登録する．

5.2.3　生成される初期方向の数

最後にこのプログラムが生成する初期方向の数について述べる．2.2.2 節で既に述べたように，正二十面体の三角形を S 分割する際に生成される初期方向の数は $10S^2+2$ である．このアルゴリズムでは maxSplitNum が 1 であれば三角形は 2 分割，2 であれば 4 分割 $\cdots$ となるので，このプログラムから生成される初期方向の数は $10\times(2^{\mathrm{maxSplitNum}})^2+2$ となる．

5.3　CPU での SBR 法の実装

5.3.1　概要図

GPU でシミュレーションを実装する前段階として CPU で SBR 法と SBR-image 法を計算するプログラムを作成した．まず CPU で実装した SBR 法の概要図を図 16 に示す．

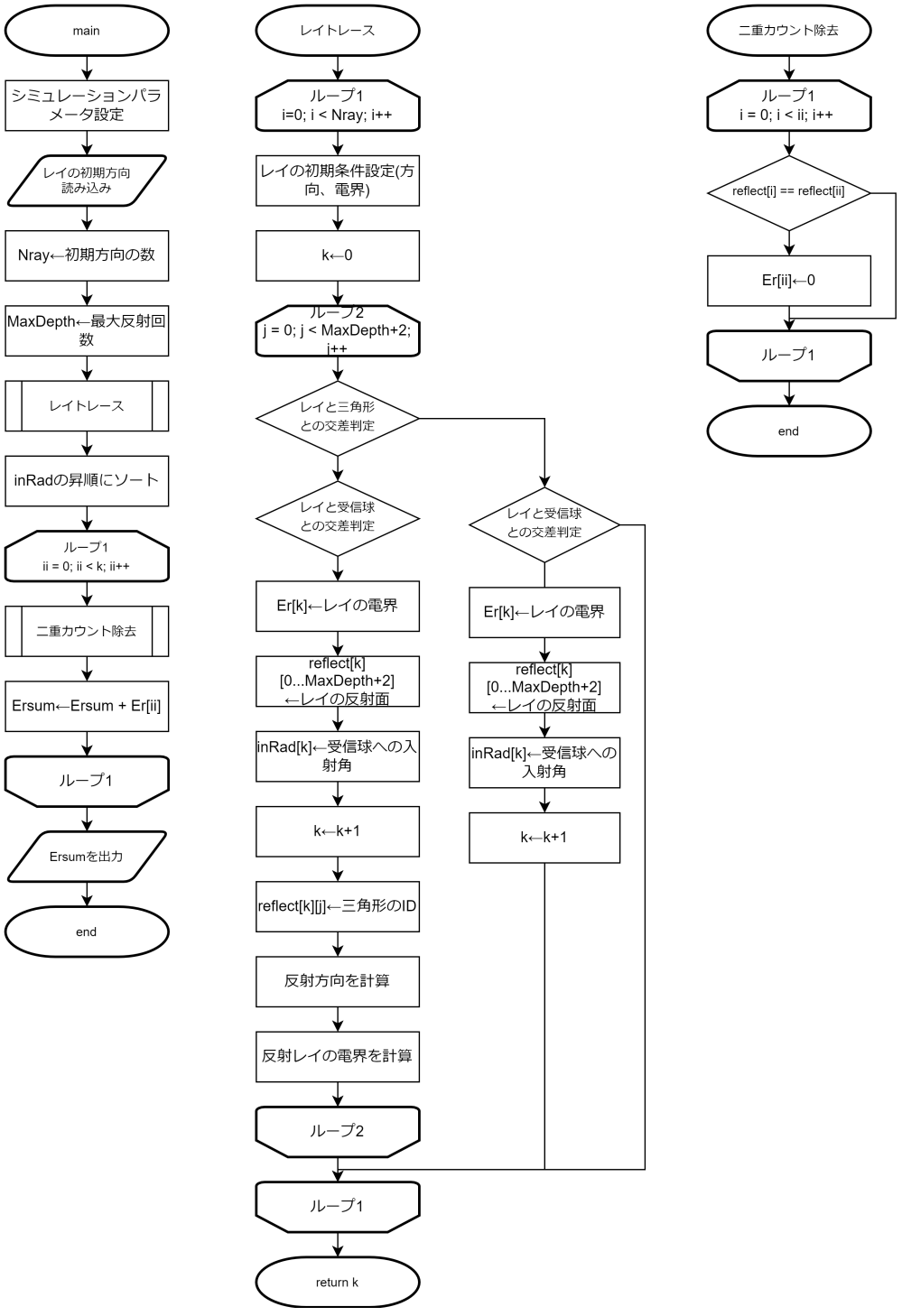

図 16　CPU で実装した SBR 法で計算を行うプログラムの概要図

5.3.2　二重カウント除去手法

reflect は二重カウント除去を行うためにレイがどの反射面で反射したかを記録する二次元配列である．図 17 に概要図を示す．reflect は 0 から (MaxDepth+2) 個の整数型を受信球に到達したレイの数 (k) 個分持つ．MaxDepth より 2 つ多く用意するのは，送信機 (id:-2) と受信機 (id:-1) を登録するためである．二重カウント除去はこの配列の 0 ~ (MaxDepth+2) までが完全に一致しているかどうかで判定する．例えば図 17 であれば 0 番目と 2 番目のレイは一致しているため，同一経路を辿ったレイとなる．この 2 つのうち inRad が大きい方 (= 受信球から遠いレイ) の電界 (Er) を 0 にすることで二重カウントを除去できる．

5.4　CPU での SBR-image 法の実装

5.4.1　概要図

次に CPU で実装した SBR-image 法の概要図を図 18 に示す．

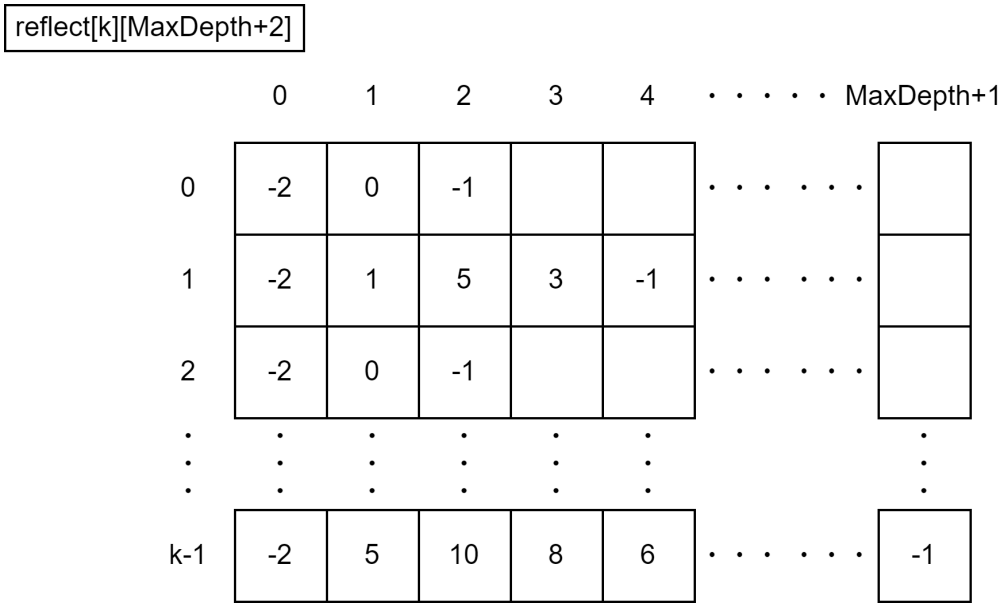

図 17　二次元配列 reflect の概要図

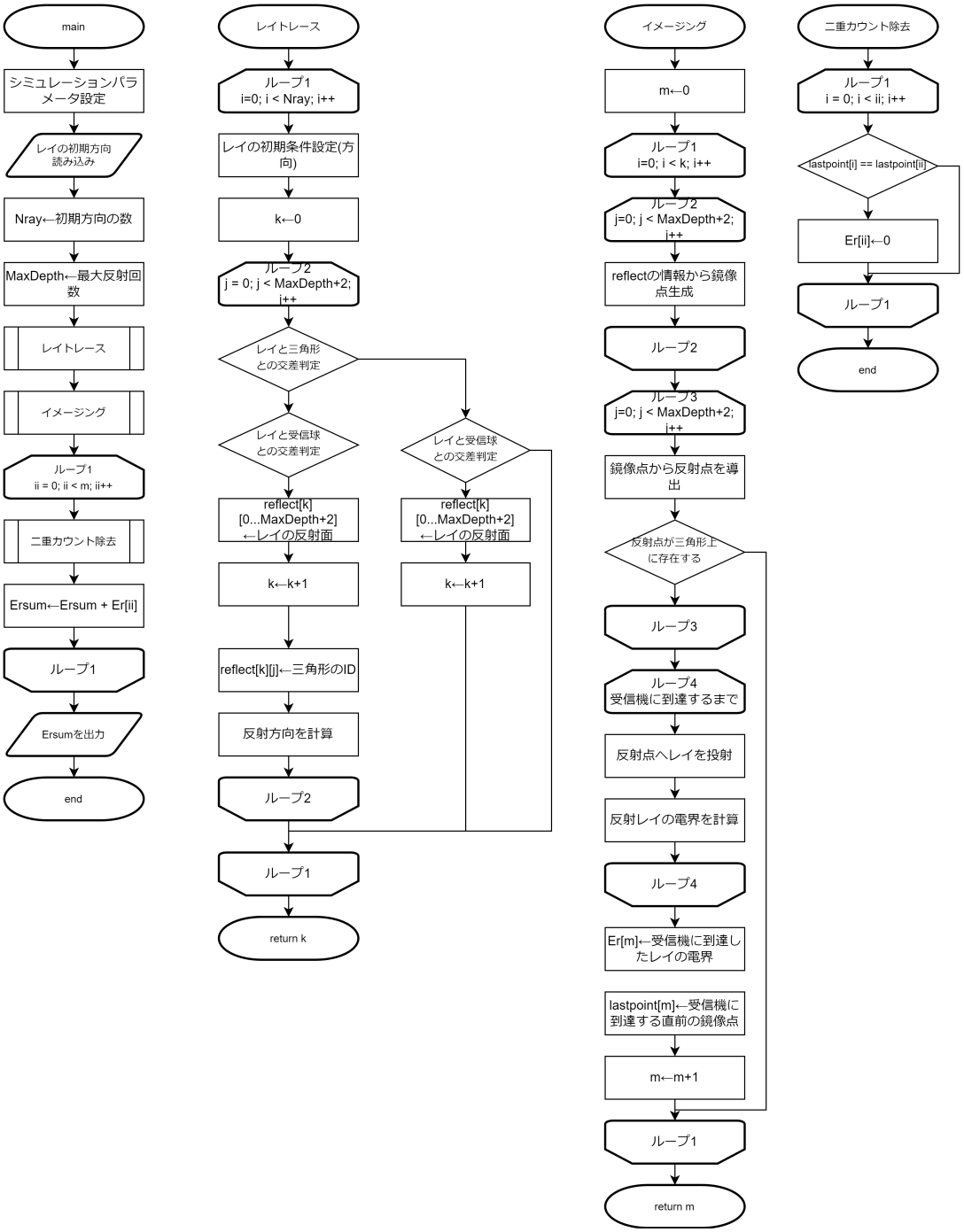

図 18　CPU で実装した SBR-image 法で計算を行うプログラムの概要図

SBR 法と違いレイトレーシング部分では配列 reflect を更新するだけで電界計算を行わない．レイトレーシング部分の終了後，reflect の情報をもとにイメージング法の計算を行う．このとき計算された反射点が reflect に登録された該当の三角形上に存在するかの判定を行うことで，実際には存在しないパスを除去することができる．

5.4.2 二重カウント除去手法

二重カウント除去はイメージング法の計算を行う際に計算される最終イメージ点を用いる．例えば図 3 の場合，Tx” が最終イメージ点である．同じ反射面を通ったレイはイメージング法で同じ経路が計算されるため，最終イメージ点の座標も同一になる．従って，最終イメージの座標が同じ場合片方の電界を 0 にすることで二重カウントを除去できる．

SBR 法で行った単純な反射面の比較と違い，受信球に近いレイを残すためのソート処理が必要ない．これは同じ反射面を辿ったものは同じ電界が計算されるため受信球から近い，遠いといった区別を行う必要がないためである．

また，SBR 法では反射面の比較を最大で MaxDepth 回行う必要があるため，MaxDepth が増えるにつれて比較するべき配列の要素数が増えていたが，今回の手法は 2 つの点の座標を比較するだけであるため計算量は反射回数によらないという利点がある．

さらに，SBR 法で行ったような単純な比較では除去しきれない場合でも正しく判定することができる．図 19 に具体例を示す．今，レイ i とレイ j が直交する 2 つの面 1 と 2 の境目の部分で反射を行い受信球に入射している．このような場合，イメージング法では直交する面のイメージ点が重なる性質を利用することでレイを 1 つだけ追跡する．しかし，SBR 法で行ったような単純な面情報の比較では i と j は別のレイとして計算されてしまう．ここで，i と j の最終イメージ点は Tx’12 と Tx’21 であり，両者の座標は等しい．従って，最終イメージ点が同一であれば重複とみなす手法は単純な反射面の比較よりも高精度かつ高速に二重カウント除去を行うことができる．

reflect				
i	-2	1	2	-1
j	-2	2	1	-1

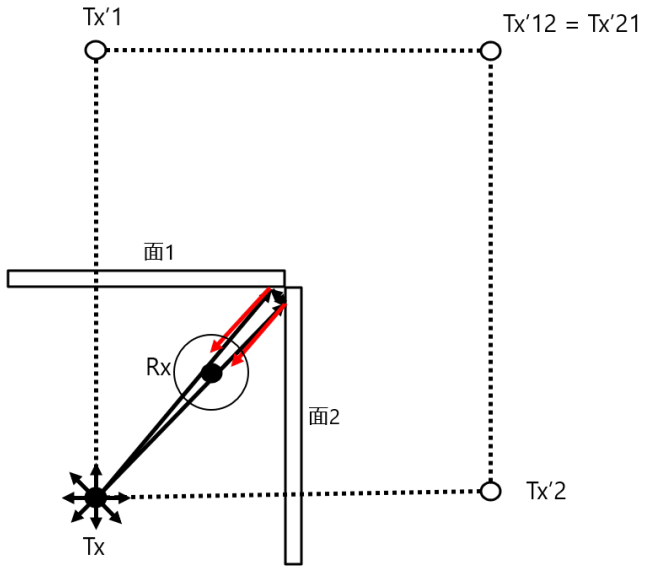

図 19　隅で反射するケースでの二重カウント除去

6　システム概要 (GPU 版)

本章では作成した GPU 版電波伝搬シミュレーションシステムの概要と，実装の詳細について述べる．

6.1　概要図

CPU で SBR-image 法を計算するプログラムを元に，GPU で SBR-image 法を計算するプログラムを OptiX を用いて実装した．プログラムの概要を図 20 ~ 図 23 に示す．図 20 は全体の制御を行う部分であり，主に CPU で処理を行う．図 21 および図 22 は OptiX(GPU) で処理を行う部分で，図 23 は二重カウント除去等の後処理を GPU 上で CUDA により行う部分である．

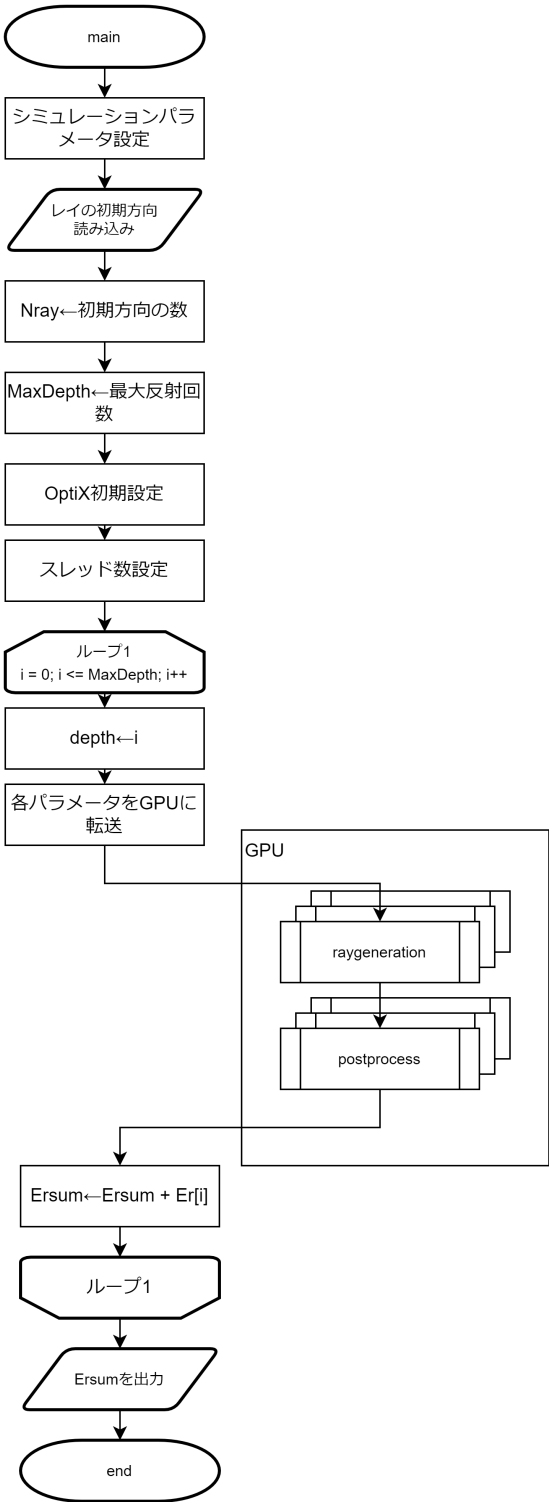

図 20　GPU で実装した SBR-image 法で計算を行うプログラムの概要図 (CPU 部分)

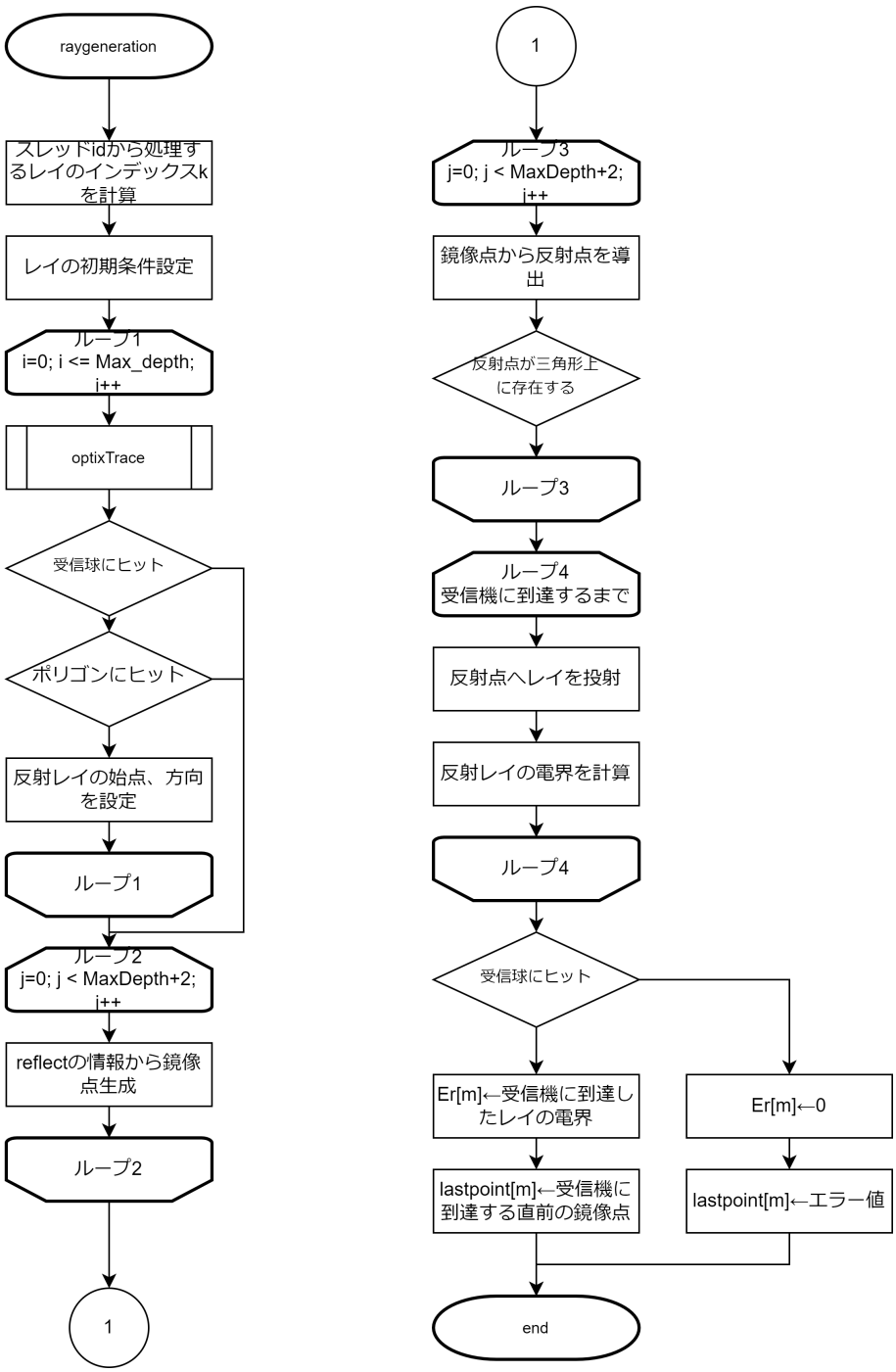

図 21　GPU で実装した SBR-image 法で計算を行うプログラムの概要図 (OptiX 部分:raygeneration)

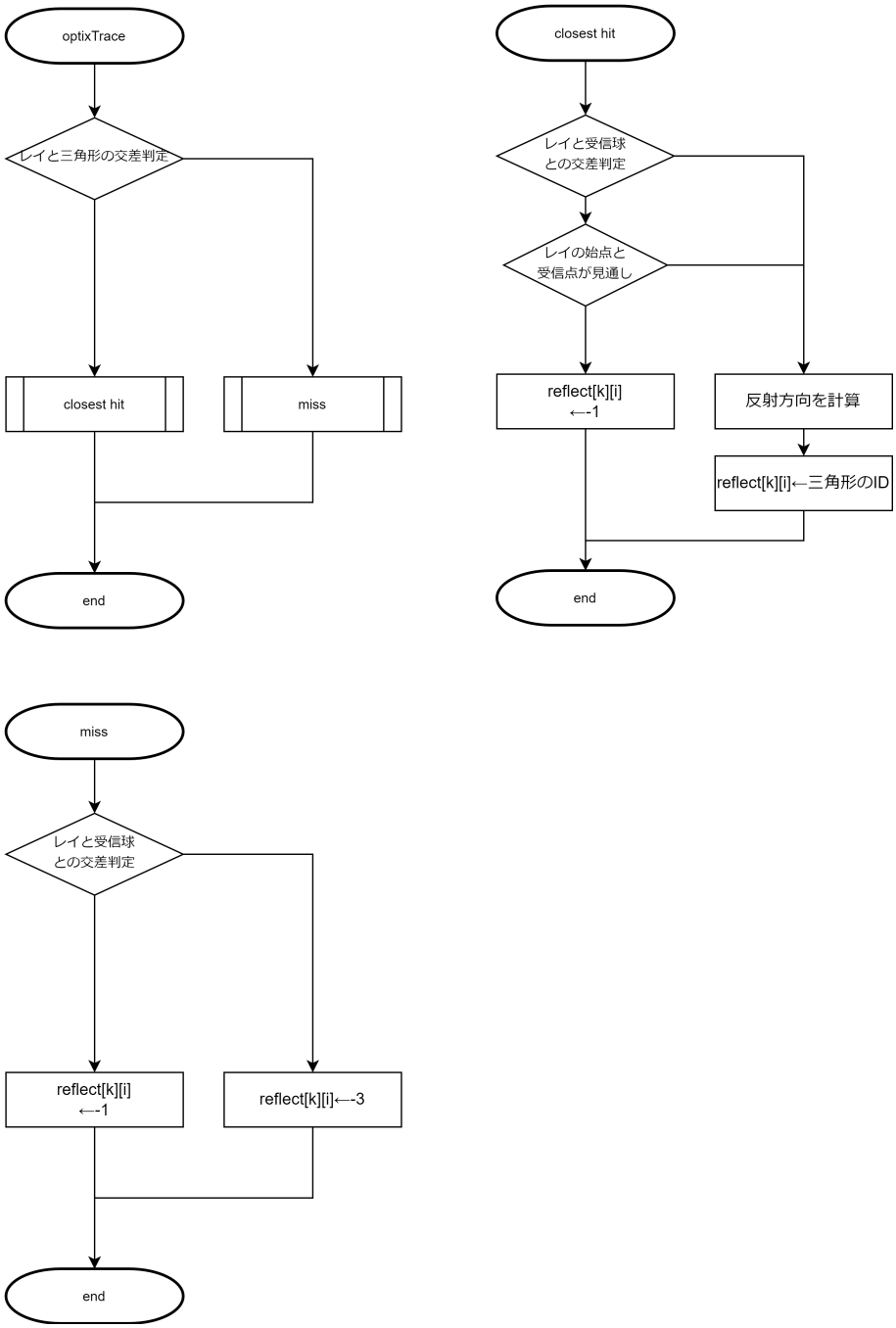

図 22　GPU で実装した SBR-image 法で計算を行うプログラムの概要図 (OptiX 部分:その他)

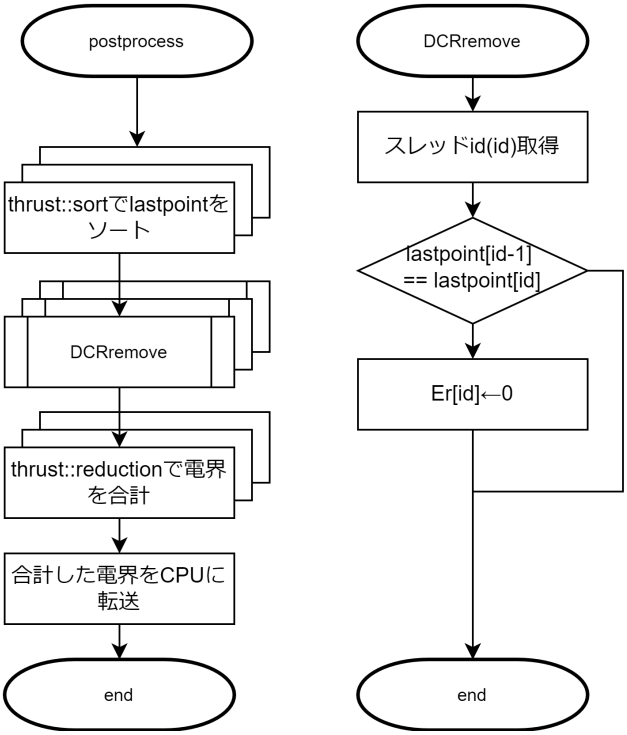

図 23　GPU で実装した SBR-image 法で計算を行うプログラムの概要図 (CUDA による後処理部分)

CPU 上でまずシミュレーションパラメータやレイの初期方向のロード，対象とする解析領域 (3D モデル) のロードを行い，その後 OptiX の起動に必要な初期設定を行う．3.3.2 節で述べた Acceleration Structure もここで作成する．読み込んだ 3D モデルの頂点数やポリゴン数，バッファサイズ等を OptiX に渡すことで作成できる．

OptiX ではまず raygeneration プログラムが呼び出され，レイの始点や方向といった初期条件を設定した後 optixTrace 関数を呼び出すことでレイトレーシングを行う．optixTrace 関数はレイと AS の交差判定を行い，レイとヒットするポリゴンが存在する場合 closest hit プログラムを，存在しない場合は miss プログラムを呼び出す．3.3 節で述べたようにこの交差判定は OptiX による組み込み処理となっており，RTX の RT コアを用いることで高速に計算される．呼び出された closest hit プログラムや miss プログラムでは受信球との交差判定を行いレイの経路情報を更新して終了する．

6.2 ペイロード

OptiX での各プログラム間 (raygeneration, Closest hit, miss 等) でのデータのやり取りはペイロードと呼ばれるレジスタを用いて行われる．デフォルトでは 8 つの 32 ビット整数レジスタしか使用できないため，多くのデータをやり取りする際は次のような工夫が必要である．以下に概要を示す．

1. 受け渡すデータを構造体にまとめ GPU 上に確保する．
2. GPU メモリ上のデータは 64 ビットのアドレスで表現されるため，リスト 1 に示した packPointer 関数に構造体のアドレスを渡し上位 32 ビット (i0) と下位 32 ビット (i1)

リスト 1　ペイロードのやり取りをするための関数

```
void  packPointer(void* ptr, uint32_t& i0, uint32_t& i1)
{
    const uint64_t uptr = reinterpret_cast<uint64_t>(ptr);
    i0 = uptr >> 32;
    i1 = uptr & 0x00000000ffffffff;
}

void* unpackPointer(uint32_t i0, uint32_t i1)
{
    const uint64_t uptr = static_cast<uint64_t>(i0) << 32 | i1;
    void* ptr = reinterpret_cast<void*>(uptr);
    return ptr;
}
```

に分割する．

3. 分割した 2 つの 32 ビット整数 (i0, i1) をレジスタ 0 とレジスタ 1 に設定し，各プログラム間で受け渡す．
4. 渡された 2 つの 32 ビット整数からリスト 1 に示した unpackPointer 関数を用いて元のアドレスを復元し，データにアクセスする．

このように受け渡したいデータのアドレスを分割して渡すことで多くのデータをプログラム間でやり取りすることができる．

6.3 二重カウント除去

レイトレーシングが終了した後，反射面情報からイメージング法の処理を行う．受信球に到達したレイは計算した電界と最終イメージ点を記録し，到達しなかったレイは電界として 0 を，最終イメージ点として通常の探索ではあり得ないエラー値を設定する．このようにすることで各レイの電界計算を独立に行うことができる．

その後二重カウント除去をした後総電界を計算するが，これらの並列実行には工夫が必要である．

6.3.1 二重カウント除去を並列実行する際の問題点

3DCG におけるレイトレーシング法において，各レイの計算結果は他のレイに依存せず独立しているため GPU での高速化に適している．電波伝搬シミュレーションにおけるレイトレーシング法も各レイの電界を計算する部分までは独立しているため，同様に並列化を行うことができる．既に述べたように電波伝搬シミュレーションは各レイの電界をまとめる必要があり，この際に二重カウント除去をおこなうことでシミュレーションの精度を向上させることができるが，これを行うにはレイ同士で反射面情報や最終イメージ点座標といった値を参照する必要があるため独立性が損なわれてしまう．従って，電波伝搬シミュレーションを GPU で並列化する際は二重カウント除去の処理に工夫が必要である．

単純な解決方法としては各レイの電界を GPU で計算した後，CPU に反射面や最終イメージ点のデータを転送し CPU 側で二重カウント除去を行うという手法が考えられる．しかし，実際にこのように実装したところかえって計算時間が増大してしまった．

この理由について述べる．まず GPU でプログラムを実装する際にネックとなるのは CPU と GPU の間のデータ転送である．従って，このデータ転送をなるべく避ける，かつあまり大きなデータを転送しないようにプログラムを組む必要がある．SBR-image 法ではイメー

ジング法による補正を行うために各レイの反射経路を保存しておく必要がある．一つの方向について (反射回数 +1) 回のレイが受信球を通過する可能性があるため，受信球を通る最大のレイの本数は Nray × (MaxDepth+1) である．これらすべての経路情報を保存するために一つの反射面について int 型 (4 バイト) を使用した場合，必要なメモリサイズは (レイの本数)×(反射回数 +1)×(反射回数 +2)×4Byte である．これは例えばレイの本数が 2,621,442 本，反射回数が 10 回の場合約 1.3GB となり，データ転送時間が大きなボトルネックとなってしまう．これらの理由から，大きなサイズのデータ転送が起きないように二重カウント除去や電界計算を可能な限り GPU 上で計算する必要がある．

この問題について [16] ではレイをグループ分けすることで二重カウントが起きそうなレイを事前に絞り込んでいる．これにより参照するレイの数を削減し高速に二重カウント除去を行っている．また [17] ではブルームフィルタを用いることで二重カウント除去が確率的になる代わりに高速化を達成している．

6.3.2 実装した二重カウント除去手法

本研究では異なる反射回数のレイは異なる経路を持つという性質に着目し二重カウント除去を行った．例えば 1 回反射して受信球に到達するレイと 2 回反射して受信球に到達するレイが同じ経路を辿ることはないため，これらのレイは別々に計算することができる．従って，計算すべき総電界は (0 回反射して受信球に到達するレイの総電界) + (1 回反射して受信球に到達するレイの総電界) + ··· + (MaxDepth 回反射して受信球に到達するレイの総電界) となる．これにより反射面情報を保存するのに必要なメモリサイズは (レイの本数)×(反射回数 +2)×4Byte となり，メモリサイズを減らすことができる．

各反射回数におけるレイの二重カウント除去は図 23 の postprocess において行われる．まず各レイの最終イメージ点 (lastpoint) をソートする．なお，ソートには thrust ライブラリ [18] の sort 関数を用いた．これにより GPU 上で高速にソートを行うことができる．

次に DCRremove 関数をスレッド数を指定して並列に実行する．lastpoint はソート済みであるため，自分と同じ最終イメージ点を持つレイがあるかどうかは自分の一つ前のスレッドが持つ最終イメージ点を見ることで判断することができる．自分の最終イメージ点と同じであれば自分が持つ電界を 0 にする．この際スレッド間で参照が起きるのは最終イメージ点のみであり書き込みは行われないため並列に実行することが可能である．

最後に，全ての電界の総計を thrust ライブラリの reduction 関数で行う．reduction 関数は図 24 のような処理を行うことで配列の合計を高速に求めることができる．なお，スレッド

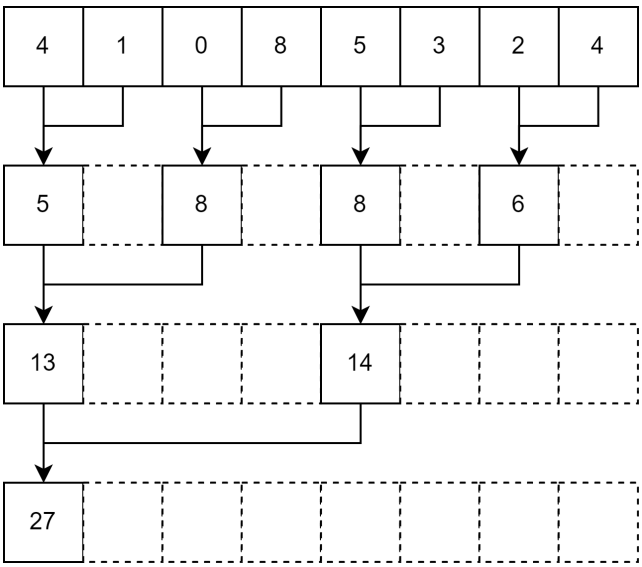

図 24　reduction の概要

間の同期はライブラリ内で自動で行われる.

7　評価

本章では作成した CPU 版 SBR 法，CPU 版 SBR-image 法，GPU 版 SBR-image 法について評価を行う．

7.1　実験環境

評価を行う前に，データの測定環境について述べる．使用した GPU の詳細を表 1，GPU を使用するホスト側の環境を表 2 に示す．

表 1　使用した GPU の詳細

GPU 名	RTX A6000	RTX3070	RTX2080	GTX1660Ti
CUDA コア数	10,752	5,888	2,944	1,536
RT コア数	84	46	46	0
TENSOR コア数	336	184	368	0
ブーストクロック [MHz]	1,800	1,730	1,710	1,770
ベースクロック [MHz]	1,410	1,500	1,515	1,500
メモリ速度 [Gbps]	16	14	14	12
標準メモリ構成	48GB DDR6	8GB DDR6	8GB DDR6	6GB DDR6
メモリインターフェイス [bit]	384	256	256	192
メモリ帯域幅 [GB/s]	768	448	448	288
電力 [W]	300	220	215	120
アーキテクチャ	Ampere	Ampere	Turing	Turing

表 2　GPU を使用するホスト側の環境

OS	Windows10 Education, 64-bit バージョン 20H2
CPU	Intel Core i7-10700K 3.80GHz
CPU メモリ	16GB DDR 3
CUDA バージョン	11.3
OptiX バージョン	7.3
グラフィックスドライババージョン	472.84(RTX A6000) 496.49(それ以外)

7.2　シミュレーションの精度

7.2.1　イメージング法との比較

まず実装した CPU 版 SBR 法および SBR-image 法の計算精度を確かめるためにイメージング法との比較を行った．比較に用いた領域を図 25 に示す．

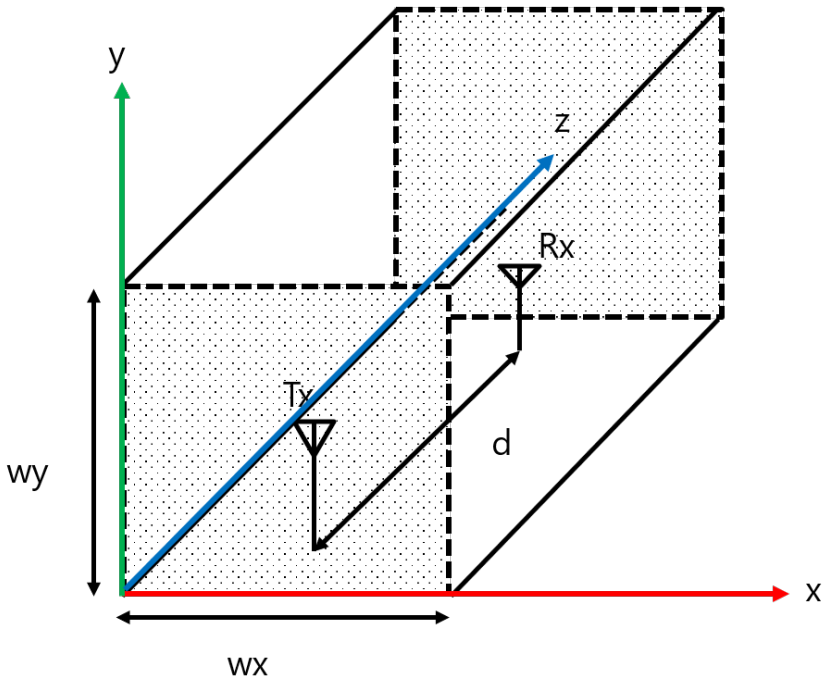

図 25　解析領域

解析領域は幅 wx 高さ wy で内部が空洞のトンネルである．シミュレーションの各パラメータは表 3 のように設定した．

Rx の z 座標を 0 から 1,000m まで 5m ずつ遠ざけながら電界を計算した．結果を図 26 に示す．

表 3　解析領域のパラメータ

wx	1.8[m]
wy	2.2[m]
Tx 位置	(wx / 2, wy / 2, 0)
Rx 位置	(wx / 2, wy / 2, 0 ~1,000)
周波数	2,200[MHz]
ポリゴン数	8
壁の比誘電率 (ϵ_r)	6.76
壁の導電率 (σ)	0.0023[S/m]
レイの初期本数	10,485,762
最大反射回数	40

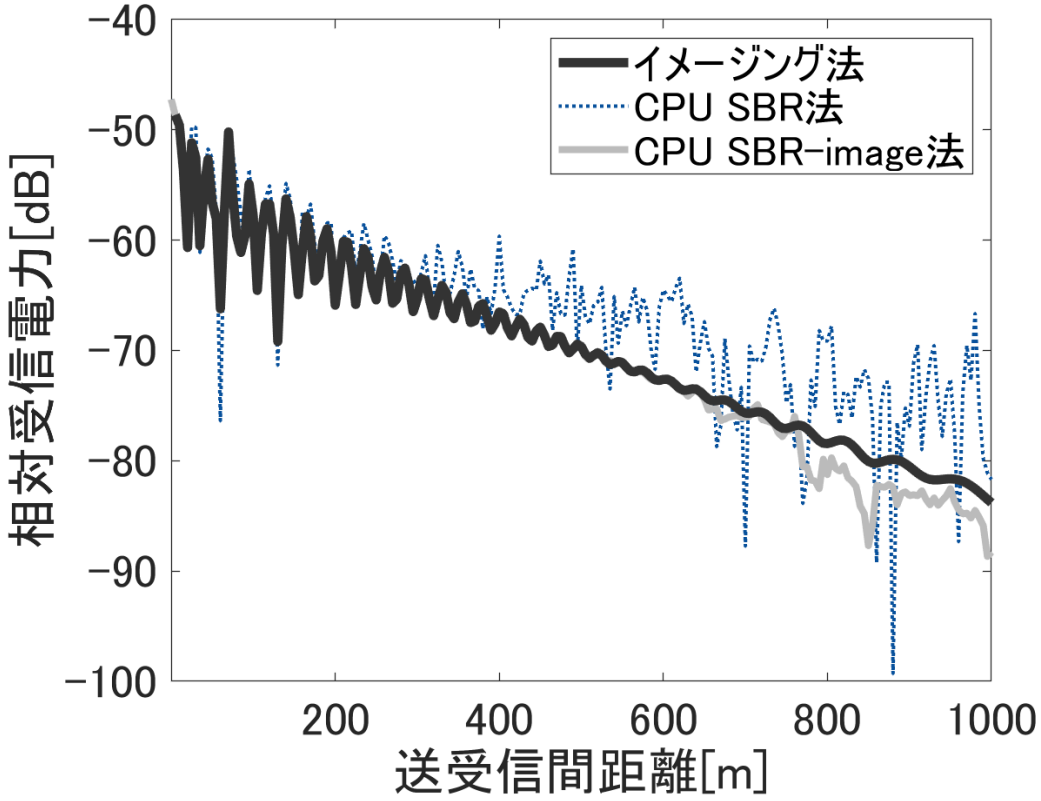

図 26　イメージング法，SBR 法，SBRimage 法の精度の比較

図 26 より，受信点が遠ざかるにつれて SBR 法と SBR-image 法はともに精度が下がっていることがわかる．今回解析に用いたトンネルは非常に長く，かつレイの半分は受信点とは逆方向に出射されるため受信球に到達するレイは初期出射本数に比べて非常に少ない．従って受信点が遠くなるにつれて送信点から出射しているレイが届きにくくなるため精度が低下していると考えられる．レイの出射本数を変えずに到達するレイを増やすためには受信球のサイズを大きくする必要がある．2.2.3 節で述べたように SBR-image 法はイメージング法による補正を行うため，SBR 法よりも受信球サイズを大きくすることができる．これにより SBR 法よりも受信球に到達するレイの本数が増えるため，精度よく計算できていると考えられる．

また，受信点が比較的近い場合でも SBRimage 法の方が誤差が少ない．この理由としては今回の計算では送信点と受信点を領域の中央に配置しているため 5.4.2 節で述べた隅での反射が発生しているためである．SBR-image 法は最終イメージ点を用いた二重カウント除

去を行っており，隅で起こる反射を正しく計算できるため SBR 法よりも精度が高いと考えられる．

7.2.2　CPU と GPU の比較

実装した CPU 版 SBR-image 法と GPU 版 SBR-image 法の精度に関して 7.2.1 節と同様の条件で比較した．結果を図 27 に示す．

CPU 版は計算に double 型を使用し，GPU 版では float 型を使用しているため，GPU 版の方が精度が悪くなると考えられるが，図 27 より GPU 版の方が高精度となっている．この理由としては，float 型を用いていることにより受信球との交差判定チェックの基準を緩く設定していることが挙げられる．これにより，CPU 版よりも多くのレイが受信球に到達するため精度が向上していると考えられる．

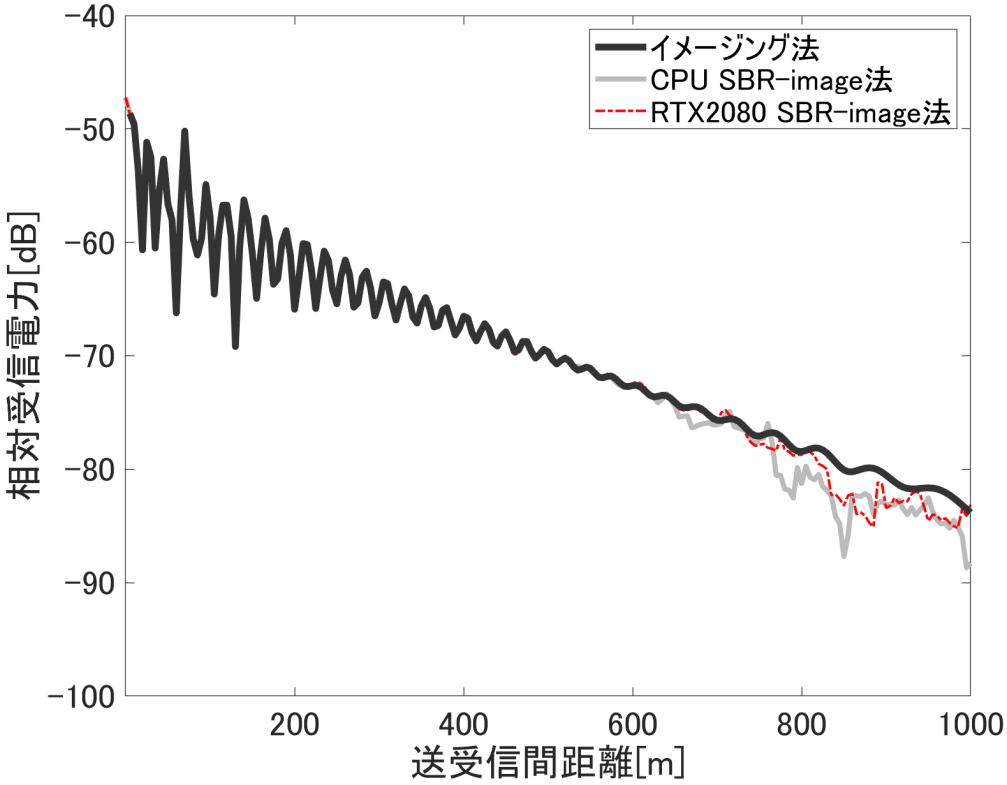

図 27　イメージング法，CPU SBRimage 法，GPU SBRimage 法の精度の比較

7.3　シミュレーションの速度

7.3.1　レイの反射回数を変えた場合

レイの反射回数を変えていった場合での各手法の速度を比較した．比較に用いた領域を図 28 に示す．

解析領域は幅 wx 高さ wy 奥行 wz で内部が空洞の部屋である．シミュレーションの各パラメータは表 4 のように設定した．

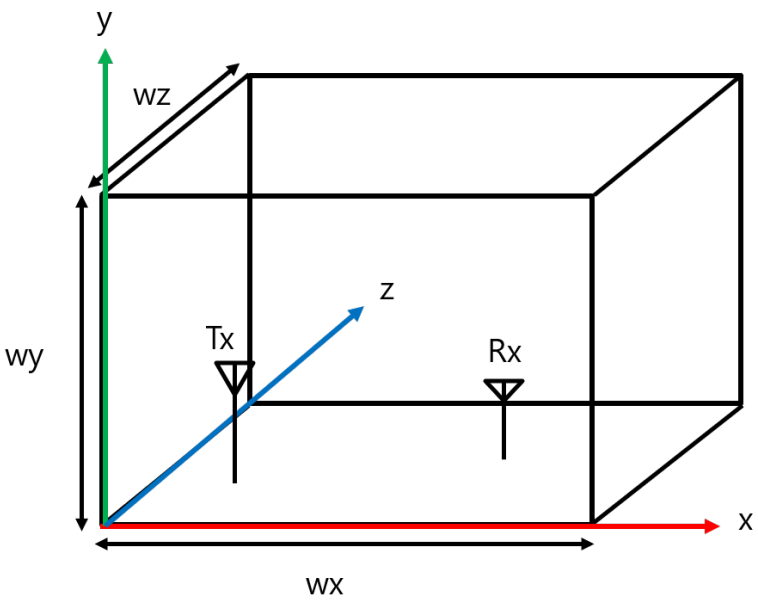

図 28　解析領域

表 4　解析領域のパラメータ

wx	50.0[m]
wy	4.0[m]
wz	50.0[m]
Tx 位置	(20.0, 3.0, 10.0)
Rx 位置	(30.0, 1.5, 40.0)
周波数	2,000[MHz]
ポリゴン数	12
壁の比誘電率 (ϵ_r)	6.76
壁の導電率 (σ)	0.0023[S/m]

レイトレーシング部分の計算を始めてから電界が出力されるまでの時間を実行時間として計測した．計測には C++ の時間処理用ライブラリである chrono を用いた．レイの本数を 2,621,442 本で固定とし，反射回数を 0 から 10 まで増やしながら計測を行った．結果を図 29 に示す．

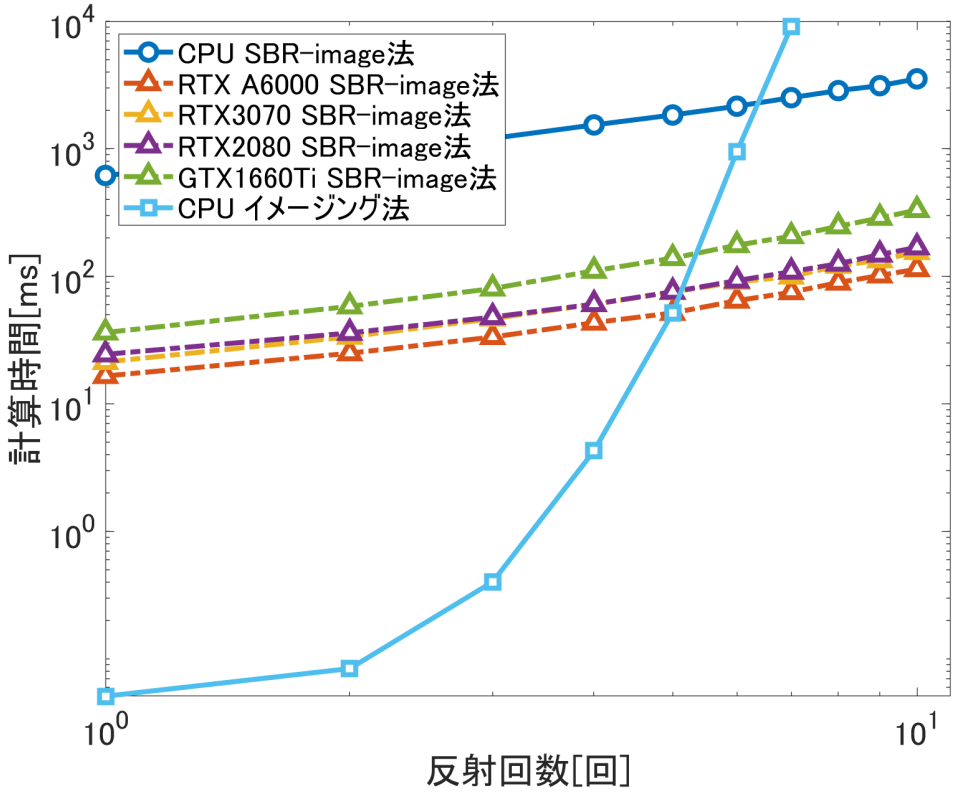

図 29　反射回数を変えた場合の CPU と GPU の速度の比較

図 29 より，イメージング法は反射回数が増えるにつれて計算時間が指数的に増加しており，そのままでは反射回数が多い場合の計算を行うのは現実的ではない．また，各 GPU 版は CPU 版 SBR-image 法よりも高速に計算できており，最も性能が高い RTX A6000 の場合は CPU に比べ約 30 倍高速であった．

7.3.2　レイの本数を変えた場合

7.3.1 節と同様の条件で，レイの本数を変えていった場合の各手法での速度を比較した．反射回数を 10 回で固定し，レイの本数を増やしながら計測を行った．結果を図 30 に示す．

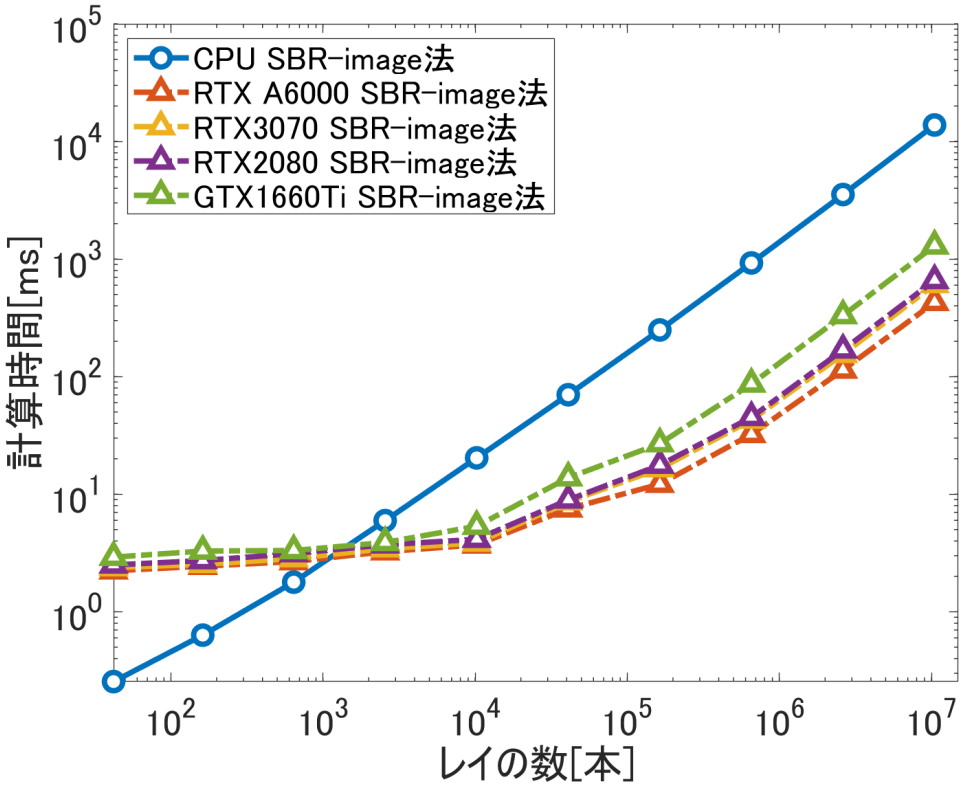

図 30　レイの本数を変えた場合の CPU と GPU の速度の比較

CPU の SBR-image 法と GPU の SBR-image 法では反射回数が少ない場合では CPU 版の方が速くなっている．これは，データ転送等のオーバーヘッドが計算時間よりも長くなってしまっているためであり，レイの本数が多くなる，すなわち負荷が高くなるほど GPU 版の方が計算時間が短くなっている．

今回の条件ではレイの本数を増やした場合 RTX A6000 が CPU よりもおよそ 30 倍高速であったが，シミュレーションに用いた解析領域である部屋はポリゴン数が 12 でありレイトレーシング部分の計算負荷が非常に小さい．従って，より大規模な領域のシミュレーションを行う際は CPU に比べてより高速に計算できると考えられる．

7.4　大規模領域における計算時間

これを確かめるためにより大きな領域で計算を行った．図 31 に解析領域の概要図を示す．解析領域は OpenStreetMap[19] から得られた地形データをもとに建物の概形を作成し

た屋外の都市である．シミュレーションの各パラメータは表 5 のように設定した．また，解析領域に Tx および Rx の位置を追加し，図 1 のように各エリアでの受信レベルを色分けして表示したものを図 32 に示す．

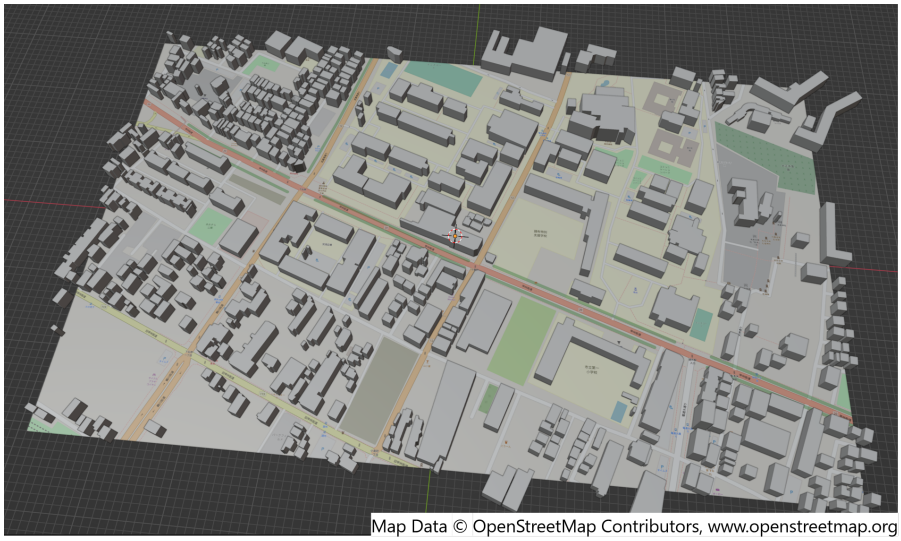

図 31　解析領域

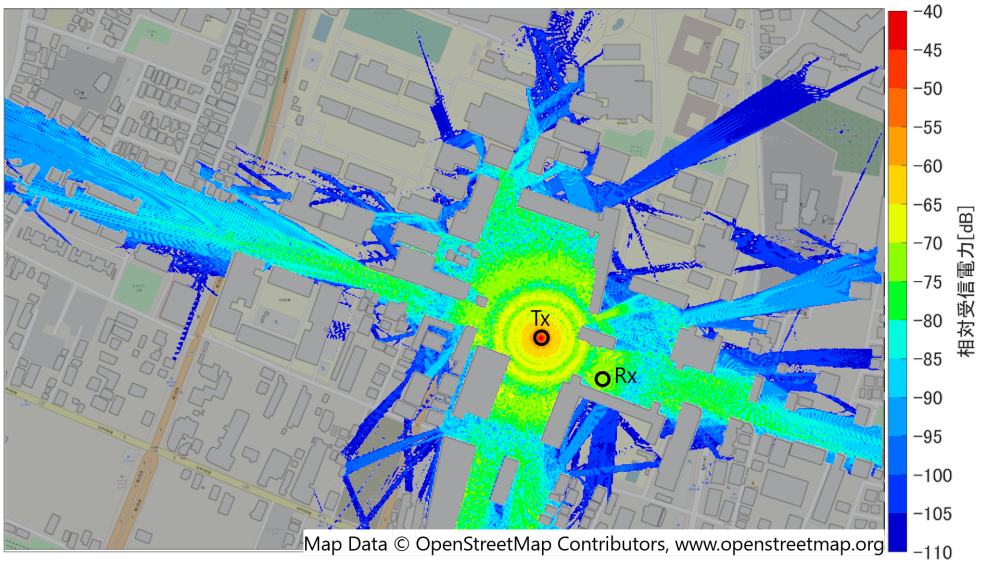

図 32　解析領域 (受信レベルと Tx，Rx の位置を追加したもの)

7.4.1　レイの反射回数を変えた場合

レイの本数を 2,621,442 本で固定とし，反射回数を 0 から 10 まで増やしながら計測を行った．結果を図 33 に示す．図 33 より，CPU の計算時間は図 29 に比べて大きく増加しているが，GPU の計算時間はあまり増加していない．これは CPU 版が毎回全てのポリゴンに対して交差判定を行うのに対し，GPU 版は AS により少ない回数で交差判定を行っているためであると考えられる．特に今回用いた解析領域は屋外であるためレイがどのポリゴン

表 5　解析領域のパラメータ

Tx 位置	(117.0, 3.0, -72.0)
Rx 位置	(195.0, 3.0, -105.0)
周波数	2,000[MHz]
ポリゴン数	7,247
壁の比誘電率 (ϵ_r)	6.76
壁の導電率 (σ)	0.0023[S/m]

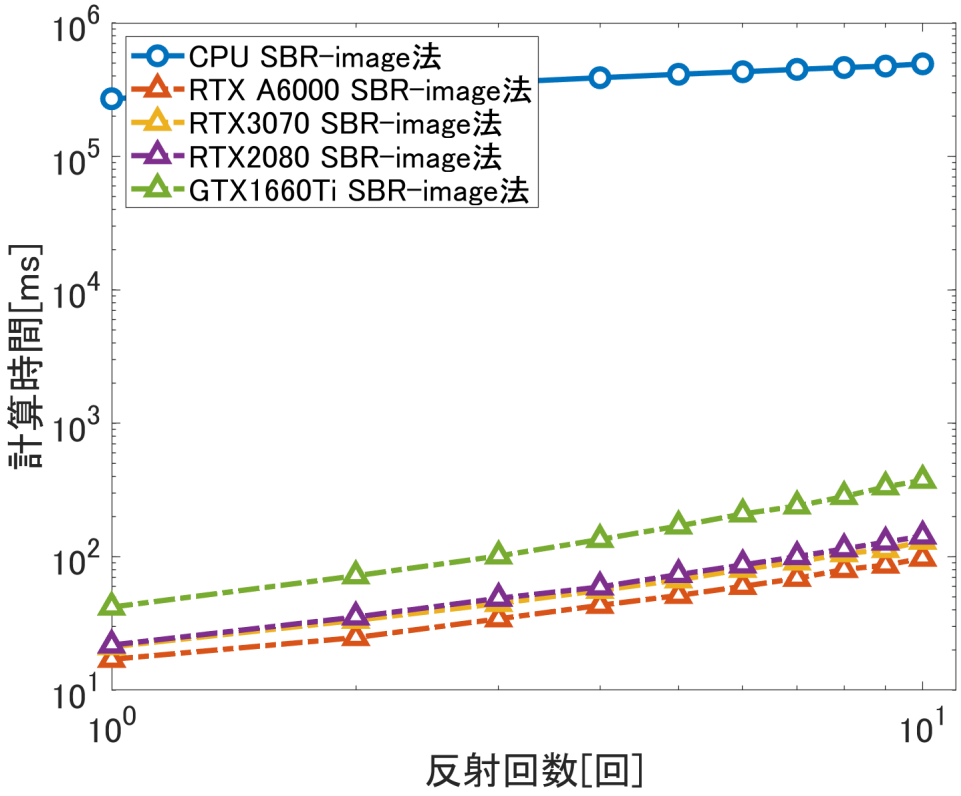

図 33　反射回数を変えた場合の CPU と GPU の速度の比較 (大規模領域)

とも交差しない可能性が高く，例えば真上方向に出射されたレイも CPU では全てのポリゴンとの交差判定を行うため，多くの計算時間がかかっていると考えられる．比較の結果，最も性能が高い RTX A6000 の場合は CPU に比べ約 5000 倍高速であった．

7.4.2　レイの本数を変えた場合

部屋の場合と同様に反射回数を 10 回で固定しレイを増やしながら計算時間を測定した．結果を図 34 に示す．既に述べたように解析領域が巨大になり CPU の計算時間が大幅に増加したため，図 30 と違いレイの本数が少ない場合でも GPU 版は CPU 版より高速に計算を行うことができている．比較の結果，最も性能が高い RTX A6000 の場合は CPU に比べ約 5000 倍高速であった．

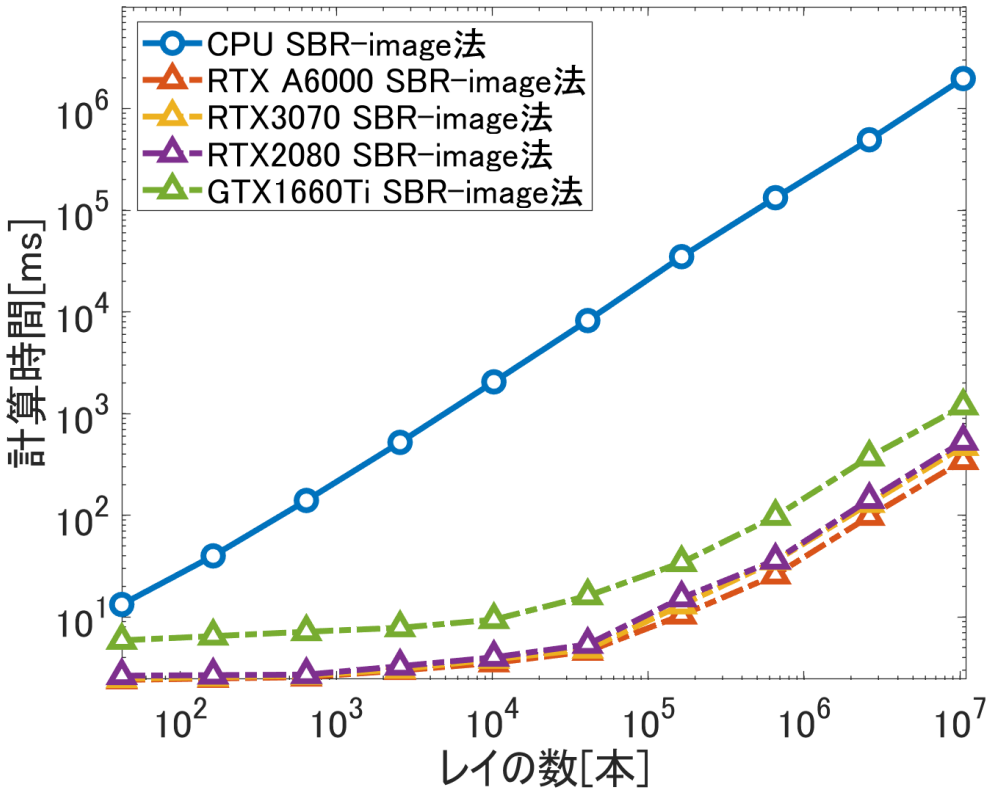

図 34　レイの本数を変えた場合の CPU と GPU の速度の比較 (大規模領域)

7.5　GPU のアーキテクチャによる違い

各 GPU によってどの程度計算時間に差があるかを確認するために，図 33 において GTX1660Ti による計算時間を 1 としたときの他 GPU による計算時間の比をプロットしたものを図 35 に示す.

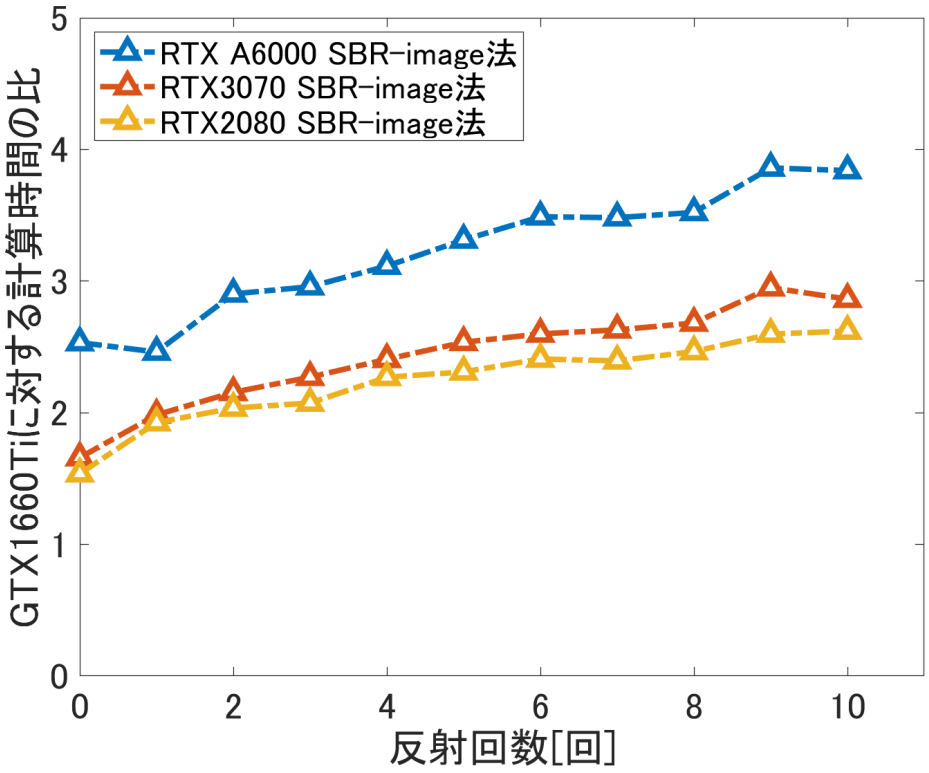

図 35　GTX1660Ti に対する実行時間の比

表 1 より，RTX2080 は GTX1660Ti よりも CUDA コアの数が約 1.9 倍多いが，図 35 より RTX2080 は GTX1660Ti よりも 2 倍以上高速に計算を行っている．これは GTX1660Ti は RT コアを持たないが RTX2080 は RT コアを持つため，より高速に計算を行うことができていると考える．

一方で，RTX A6000 は GTX1660Ti よりも CUDA コアの数が 7 倍，RTX3070 は約 3.83 倍多いが，計算時間は RTX A6000 で約 4 倍，RTX3070 で約 3 倍程度となっている．この理由について考察する．比較に用いた屋外領域は，レイがどのオブジェクトとも交差しない場合が多いため，トレースするレイの数が少なくなっている．従って，これらの GPU が持つ計算リソースを使い切れていないことが考えられる．これを確認するためには，屋外ではなく部屋のようなレイとオブジェクトとの交差が起きやすく，かつポリゴン数が多い領域を用意する，もしくはレイの出射本数や反射回数をより多くして測定を行うことが必要であると考える．

8　おわりに

本章では本研究が達成した内容をまとめ，今後の課題について述べる．

8.1　まとめ

本研究では高速な電波伝搬シミュレーションを行うために，レイトレーシング法に特化した GPU である RTX で計算を行うプログラムを作成した．作成したシステムを評価した結果，システムは CPU だけで処理を行った場合と同等の精度で計算を行うことが可能であり，かつ低負荷時でもおよそ 30 倍，高負荷時ではおよそ 5000 倍高速に計算することができていた．また，RTX ではない従来 GPU との比較により，RTX が搭載している RT コアによってレイトレーシング法の計算が高速に行われていることが確かめられた．

8.2　今後の課題

8.2.1　postprocess 関数の並列化

GPU 版 SBR-image 法において，反射回数が異なるレイは必ず異なる経路となる点に注目して独立して計算を行った．実装したシステムでは逐次実行しているが，各反射回数での計算は独立しているため並列に計算することが可能である．OptiX でレイトレーシング法を計算する際に，ある反射回数のときだけのレイを計算するのではなく，まとまった反射回数分のレイトレーシングを行い，その後 postprocess 関数を並列に実行することでより高速に計算することが可能であると考える．図 36 に概要を示す．

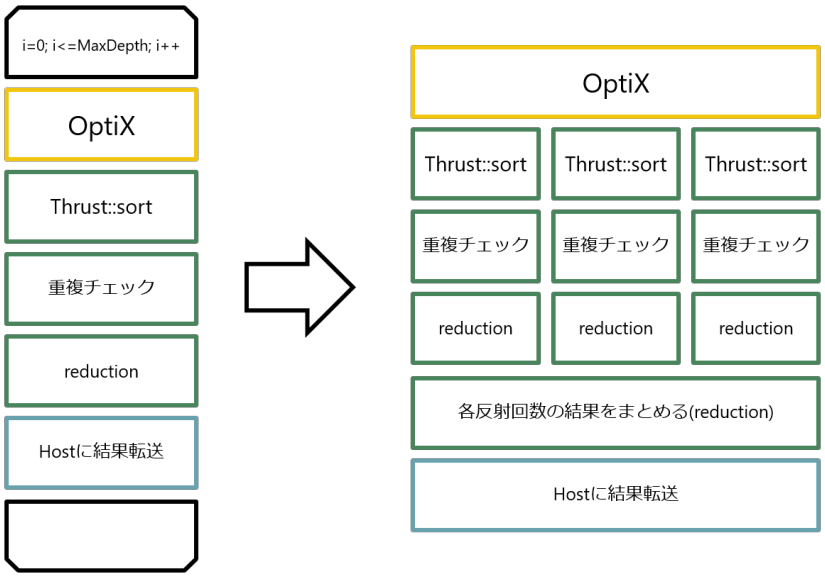

図 36　二重カウント除去の並列実行

これによりさらなる高速化が可能であると考えられるが，6.3.1 節で述べたように反射面情報を格納するためのメモリサイズが増えるため，まとめて計算する量については計算する GPU のメモリサイズに合わせたスケーリングが必要である．

8.2.2　SBR-image 法の省メモリ化

[15] では SBR 法の二重カウント除去手法として反射面にぶつかるたびにその id からハッシュ値を次々に更新していく手法を取ることで必要なメモリサイズを抑えることに成功している．しかし SBR-image 法では反射面情報をハッシュ値に格納してしまうと後処理であるイメージング法の計算が出来なくなってしまうため，反射面情報を必ず保存する必要がありメモリサイズが大きくなってしまう．

これはレイが完全鏡面反射しかしないという仮定をした場合，SBR 法を行う際に当たった反射面について受信点のイメージ点を次々に生成することで解決することができる．

最終的に生成されるイメージ点の方向が正しいレイの初期方向となるため，この方向について SBR 法と同様のレイトレーシング + 電界計算を行うことで正しく計算することができる．この手法であれば保存する情報は受信点の最終イメージ点の座標だけで良く，二重カウント除去もこの座標を用いて行うことができるためメモリサイズを抑えることができる．レイトレーシング部分の計算を 2 回行うため速度は低下するが，RTX であればレイトレーシング法を高速に計算できるため計算時間の増加を抑えることが可能であると考えられる．

8.2.3　レイの初期方向生成の高速化

5.2.1 節で述べたように，今回実装したシステムではレイの初期方向の生成は事前に計算したテキストファイルを読み込むことで行っていた．しかし，この方式ではレイの本数が増えるにつれてデータを読み込む際に必要なメモリサイズが増大してしまうという問題がある．

読み込んだ初期方向は GPU に転送するため，この部分のサイズが大きくなるとシミュレーション全体の実行時間が増加してしまう．正二十面体による出射を諦め，レイの初期方向を各スレッドの id から直接計算するようにすることでメモリサイズを減らしつつ高速にシミュレーションを行うことができると考えられる．

参考文献

[1] Danping He, Bo Ai, Ke Guan, Longhe Wang, Zhangdui Zhong, and Thomas Kürner. The Design and Applications of High-Performance Ray-Tracing Simulation Platform for 5G and Beyond Wireless Communications: A Tutorial. *IEEE Communications Surveys Tutorials*, Vol. 21, No. 1, pp. 10–27, 2019.

[2] 株式会社情報工房. WinProp. https://www.johokobo.co.jp/winprop/winprop_index.html(アクセス日 2022-01-21).

[3] 今井哲朗. 電波伝搬解析のためのレイトレーシング法—基礎から応用まで—. コロナ社, 2016.

[4] H. 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]. WEB+DB PRESS plus シリーズ. 技術評論社, 2017.

[5] Ingo Wald, Will Usher, Nathan Morrical, Laura Lediaev, and Valerio Pascucci. RTX Beyond Ray Tracing: Exploring the Use of Hardware Ray Tracing Cores for Tet-Mesh Point Location. In Markus Steinberger and Tim Foley, editors, *High-Performance Graphics - Short Papers*. The Eurographics Association, 2019.

[6] Y. OKUMURA. Field Strength and Its Variability in VHF and UHF Land-Mobile Radio Service. *Rev. Electr. Commun. Lab.*, Vol. 16, pp. 825–873, 1968.

[7] 日下美穂, 塩田茂雄. 屋内電波伝搬特性解析におけるレイトレーシング法の高速化. 電子情報通信学会論文誌 B, Vol. 98, No. 7, pp. 654–663, 2015.

[8] G. Durgin, N. Patwari, and T.S. Rappaport. An advanced 3D ray launching method for wireless propagation prediction. In *1997 IEEE 47th Vehicular Technology Conference. Technology in Motion*, Vol. 2, pp. 785–789 vol.2, 1997.

[9] S.Y. Seidel and T.S. Rappaport. Site-specific propagation prediction for wireless in-building personal communication system design. *IEEE Transactions on Vehicular Technology*, Vol. 43, No. 4, pp. 879–891, 1994.

[10] Shin-Hon Chen and Shyh-Kang Jeng. SBR image approach for radio wave propagation in tunnels with and without traffic. *IEEE Transactions on Vehicular Technology*, Vol. 45, No. 3, pp. 570–578, 1996.

[11] Steven G. Parker, James Bigler, Andreas Dietrich, Heiko Friedrich, Jared Hoberock, David Luebke, David McAllister, Morgan McGuire, Keith Morley, Austin Robison, and Martin Stich. OptiX: A General Purpose Ray Tracing Engine. *ACM Transactions on Graphics*, Vol. 29, No. 4, pp. 1–13, jul 2010.

[12] NVIDIA. How to Get Started with OptiX 7. https://developer.nvidia.com/blog/how-to-get-started-with-optix-7/(アクセス日 2022-01-21).

[13] Robert Felbecker, Leszek Raschkowski, Wilhelm Keusgen, and Michael Peter. Electromagnetic wave propagation in the millimeter wave band using the NVIDIA OptiX GPU ray tracing engine.

In *2012 6th European Conference on Antennas and Propagation (EUCAP)*, pp. 488–492, 2012.

[14] Blake A Troksa. GPU accelerated cone based shooting bouncing ray tracing. Master's thesis, Colorado State University, Electrical and Computer Engineering, 2019.

[15] Esteban Egea-Lopez, Jose Maria Molina-Garcia-Pardo, Martine Lienard, and Pierre Degauque. Opal: An open source ray-tracing propagation simulator for electromagnetic characterization. *PLOS ONE*, Vol. 16, No. 11, pp. 1–19, 11 2021.

[16] Cam Key, Blake A. Troksa, Stephen Kasdorf, and Branislav M. Notaroš. Non-Self-Adjacent Ray Classes for Parallelizable Shooting-Bouncing Ray Tracing Double Count Removal. *IEEE Journal on Multiscale and Multiphysics Computational Techniques*, Vol. 5, pp. 245–254, 2020.

[17] Roman Novak. Bloom Filter for Double-Counting Avoidance in Radio Frequency Ray Tracing. *IEEE Transactions on Antennas and Propagation*, Vol. 67, No. 4, pp. 2176–2190, 2019.

[18] Nathan Bell and Jared Hoberock. Chapter 26 - Thrust: A Productivity-Oriented Library for CUDA. In Wen mei W. Hwu, editor, *GPU Computing Gems Jade Edition*, Applications of GPU Computing Series, pp. 359–371. Morgan Kaufmann, Boston, 2012.

[19] OpenStreetMap contributors. OpenStreetMap. `https://www.openstreetmap.org` (アクセス日 2022-01-21).

謝辞

本研究を進めるにあたり，ご指導いただいた主指導教員の成見先生，副指導教員の沼尾先生並びに論文の執筆の際にご助力いただいた研究室の皆様に厚くお礼申し上げます.