2022 年度 I 類 (情報系) コンピュータサイエンスプログラム 卒業研究最終発表会

GPU を用いた電波伝搬シミュレーションの改良

I 類 (情報系) 成見研究室 学籍番号:1710084 植田 武

1 背景

近年，5G 技術の発展等，無線通信技術の利用は更に拡大している．通信システムの設計において，モデル化は重要である．近年の計算機の能力の進歩や地形のデータの入手が容易になったことから，コンピュータを用いた電波伝搬のシミュレーションが多く使用される．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解析にはレイトレーシング法が良く用いられている．

NVIDIA 社製の GPU(Graphics Processing Unit) である RTX シリーズでは従来の汎用プロセッサに加えてレイトレーシング法の計算に特化した RT コアと呼ばれる専用プロセッサを搭載している．成見研究室の荒生は RTX を用いて電波伝搬シミュレーションを高速化させた [1]．本研究ではこれを改良し，RTX を用いた電波伝搬シミュレーションの更なる高速化を行う．

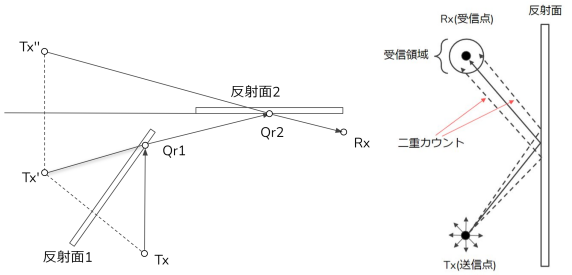
2 既存技術・研究

2.1 レイトレーシング法

レイトレーシング法は送信点から発射された電波が反射や回折を繰り返し，受信点に到達するまでの経路を探索する手法であり，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 (図 1)．イメージング法は反射面の組み合わせから鏡像点を求めることで経路を求める手法で，高精度だが反射面や反射回数が増えるほど計算量が指数的に増加してしまう．SBR 法は送信点からレイを一様に発射しその軌跡を追跡する手法である．イメージング法よりも計算量は少ないが，精度が低い．特に同一経路のレイを 2 回カウントしてしまう場合があり，これを除去しない場合、更に精度が低下してしまう．本研究では両者を組み合わせた SBR-image 法 [2] を用いた．SBR 法でレイが通る反射面を計算した後，その情報をもとにイメージング法で補正を行うことで SBR 法と同等の計算量で精度の高い計算を行うことができる．

2.2 NVIDIA OptiX

RT コアを使用できる API はいくつか存在しているが，本研究では NVIDIA OptiX を用いた．これは，OptiX 以外の API はフレーム毎の処理の方が適しているためである．OptiX はレイの振る舞いをユーザーが自由に記述できる他，オブジェクトを木構造に格納することで交差判定を行う回数を大幅に削減することができる．また，RT コアはこの交差判定を高速に行うことができる。



(a) イメージング法　　(b) SBR 法

図 1: レイトレーシング法

2.3 既存研究

OptiX を用いて電波伝搬シミュレーションを行った研究として，屋内環境 [3] や大規模トンネル内を対象としたもの [4] が存在する．これらは主に精度に着目しており，高速化に関しては余り言及されていないという点で本研究とは異なる．

2.4 荒生によるプログラム [1]

荒生は OptiX を用いて SBR-image 法を計算するプログラムを作成した．大まかな処理の流れを以下に示す．

1. シミュレーションに必要なパラメータ (レイの最大反射回数，解析領域情報，周波数，送信機と受信機の位置等) を入力．
2. それらの情報を GPU に転送し，OptiX で SBR 法の計算を行う．このときレイがどの反射面と衝突したかの経路情報を配列に格納しておく．
3. 得られた経路情報から GPU でイメージング法の処理を行う．
4. 同一経路を辿ったレイを除去し，総電界を計算して CPU に転送し結果を表示する．

二重カウント除去の手法としてイメージング法で後処理を行う際に計算される受信機に到達する直前の鏡像点 (最終鏡像点) を用いている．この座標が同じ場合，二重カウントとして扱う．

3 高速化

3.1 高速化手法 1

手法 1 の概要図を図 2 に示す．元のプログラムでは GPU の計算呼び出しが反射回数分ある．そこで，すべての反射回数分のレイを同時に計算させることで GPU の呼び出しのオーバーヘッドが削減できると考えた．特にレイの数が少ない場合は元のプログラムと比べて GPU のリソースを多く使用できるため，余るコアが少なくなり高速化できると考える．

3.2 高速化手法 2

手法 2 の概要図を図 3 に示す．元のプログラムでは各反射回数ごとに OptiX で計算を行っているが，例えば 5 回反射の場合，実際には 5 回反射のレイのみで 0〜4 回反射のレイの情報も取得することができる．このように一度レイを射出するだけで計算できるようにすることで計算回数が減り高速化できると考えた．反射回数を r，レイの数を N とすると元のプログラムでは計算回数は最大で $1 \times N + 2 \times N + ... + r \times N = \frac{r(r+1)}{2} N$ 回 となるが，手法 2 では最大でも rN 回となる．よって反射回数が多いほど高速化が期待できる．

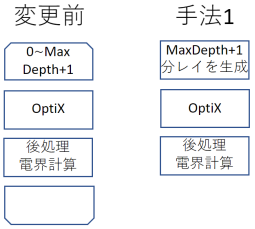


図 2: 手法 1 の概要

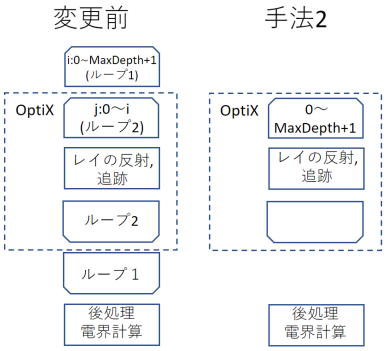


図 3: 手法 2 の概要

3.3 ソートの分割

計算時間の測定を行った際，レイの数が多い場合後処理部分のソートが大部分を占めていることが分かった．ソートは thrust ライブラリのものを使用しているが，thrust のソートは標準的な型 (char, int 等) では高速な基数ソートが，それ以外 (ユーザ定義等) の型ではマージソートが行われる [5]．今回のプログラムでは float 型 3 つを持つ構造体 (座標) であるためマージソートが実行される．マージソートの計算量はデータの数を n とすると $O(n \log n)$ であることから，各反射回数ごとにソートを行う方が高速になると考えた．

4 評価

4.1 手法 1 と手法 2

測定は RTX3070 を用い，屋外環境の 3D モデルを解析領域として行った．以下の図 4 は反射回数を 10 回として，レイの数を変えた場合のシミュレーション全体の計算時間の比較である．どちらの手法も速くなっているが，基本的に手法 2 の方が高速である．10 万本以下では 3，4 倍程度高速であるが，それ以上の場合本数が多くなるほど効果が減っており，1000 万本では約 1.3 倍である．また，図 5 はこの手法で変更した OptiX 部分のみの計算時間の比較である．手法 1 について，70 万本から変更前との差が非常に小さくなっているが，今回の実行環境での並列処理の限界が 700 万本 (70 万× 10) 程度であることが分かる．手法 2 について，レイの本数が変わっても基本的に変更前と比べ約 5 倍であり，予想に近い値となっている．また，シミュレーション全体に対し OptiX 部分の計算時間は変更前でも 2 割以下であり，OptiX 部分以外の方が影響が大きいことが分かった．詳しく測定を行ったところ，最も影響が大きい処理は二重カウント除去のためのソート処理であることが分かった．

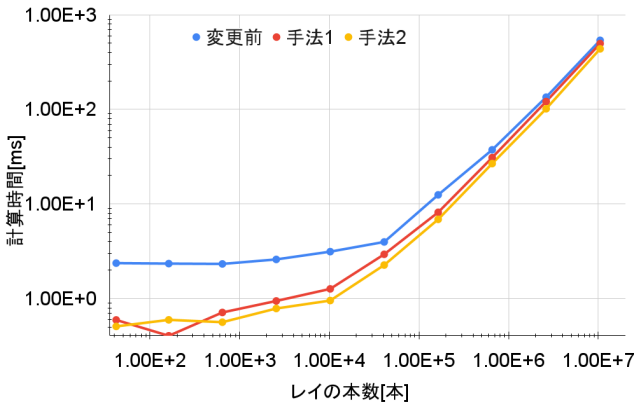


図 4: 計算時間の比較

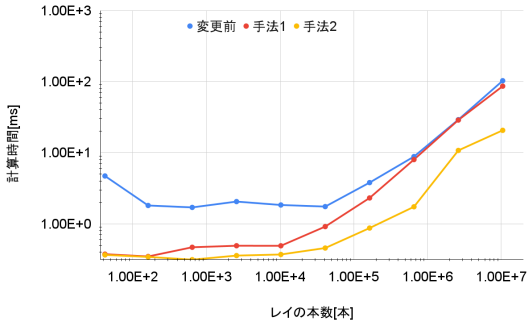


図 5: 計算時間の比較 (OptiX 部分)

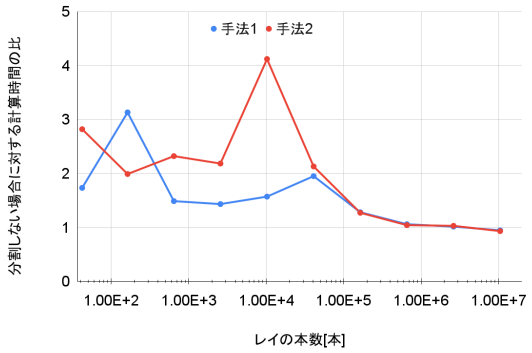


図 6: 分割しない場合に対する計算時間の比

4.2 ソート分割の有無

ソート分割を行わない場合に対する計算時間の比を図 6 に示す．約 300 万本まででではかえって遅くなってしまった．これはソートにかかる時間よりも処理の呼び出しにかかる時間の合計の方が大きいことが考えられる．一方，約 1000 万本ではソートにかかる時間が大きいため，分割した方が高速になったと考える．

5 まとめ

レイの数が少ない場合は約 4 倍，レイの数が多い場合約 1.3 倍高速化した．OptiX 部分だけでみると反射回数に比例して高速化できることが分かった．現状の実装ではソートにかかる時間がシミュレーションの大半を占めており，ソートの高速化や二重カウント除去方法の見直しが必要であることが分かった．

参考文献

[1] 荒生太一．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，電気通信大学情報・ネットワーク工学専攻修士論文，2022.

[2] 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.

[3] 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.

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

[5] Natham Bell, Jared Hoberock. Thrust: A Productivity-Oriented Library for CUDA. GPU Computing Gems, Jade Edition, Edited by Wen-mei W. Hwu, pp. 359-371, 2011.