RC-002

GPU を用いたリアルタイムレイトレーシングの並列化

Parallelization of Real-time Ray Tracing Using GPU

孟　林†　上野　謙二郎‡§　山崎　勝弘†
Lin Meng　Kenjiro Ueno　Katsuhiro Yamazaki

1．まえがき

レイトレーシングでは、視点からスクリーン上の各画素を通過する光線が物体と交差する点を探索し（交差判定と呼ぶ）、その交点で反射、屈折など物体の表面の質感に応じて、光線がどの方向に変化するのかを計算する。この変化した光線が、物体と交差する点を改めて探索し直す。この処理を順次行い、最終的に探索した点の輝度をスクリーン上に投影することで画像を生成する。

交差判定は物体の数だけその処理が行われるので、描画した画像が複雑になればなるほど物体の数が膨大になり、それだけ交差判定が繰り返される。また、物体が反射、屈折の質感をもつものがあれば、交差判定の回数が増えるので、計算量が非常に大きい。

一方、GPU は汎用 CPU と比べて桁違いの演算性能・メモリ転送性能を備えており、高い演算性能を必要とする問題への GPU の適用が進んでいる。リアルタイムレイトレーシングの研究も進められている[4]－ [6]。

本論文では、GPU を用いてリアルタイムレイトレーシングを実現するために、画面分割による並列化について検討し、処理系を実現して評価する。

GPU を用いた画面分割による並列化では、画像が生成されるスクリーンを均等にブロック分割する。分割された各ブロックをストリーミング・マルチプロセッサ（SM）に割り当てる。各 SM ではブロック内のスレッドを 32 個単位のワープに分割する。各 SM ではストリーミング・プロセッサ（SP）を用いてワープ単位でスレッドを実行する。スクリーンをブロック分割することで、各スレッドが並列で演算処理を行い、レイトレーシングを高速化する。

2．レイトレーシングの各種手法

2.1　アルゴリズム

光源から出た光は、様々な物体に衝突して、反射や屈折を繰り返す。光線は物体に衝突するごとに物体の輝度をもち、その光線が目に入ることで人は物体を認識することができる。目に入ってくる光線のみを逆に辿り、像を生成するのがレイトレーシング手法である。図 1 に示すように、レイトレーシングは次の手順で行う。

（1） 視点からスクリーン上の画素を通過する光線を発生させ、光線と物体との交差判定を行う。

（2）交差判定により、光線と交差する物体が存在するなら、光線と物体との交点を求める。交差する物体が複数ある場合には、すべての物体について交点を求める。

†立命館大学理工学部, College of Information Science and Engineering ,Ritsuemikan University
‡立命館大学大学院理工学研究科, Graduate School of Science and Engineering , Ritsumeikan University
§IHI, IHI

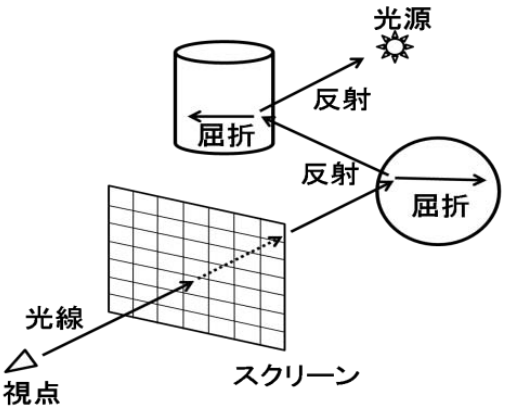

図１　レイトレーシングの原理

交点がない場合は、その画素を背景の輝度とする。

（3）光線と交差する物体の交点までの距離を求め、最も近い物体を抽出する。

（4）抽出物体の輝度の計算をする。また、反射、屈折があるならその方向を求め、その方向を光線とみなして（2）、（3）の処理を行い、屈折、反射して見える物体を抽出する。

2.2　拡散反射光と鏡面反射光

拡散反射光とは、物体に陰影をつけることができる手法である。拡散反射光を求める点への入射角を θ、光線の光度を Iq、拡散反射係数を Kd とすると、拡散反射光の輝度 Id は、$Id = Kd \times Iq \times \cos\theta$ と表せる。鏡面反射光とは、物体にハイライトなどテカリを入れることができる手法である。鏡面反射を求める点への入射光を Ip、入射光の正反射光と視線のなす角を α、面反射係数を Ks とすると、鏡面反射光の輝度 Is は、$Is = Ks \times Ip \times \cos^n\alpha$ と表せる。

3. 画面分割による並列化の設計と実現

3.1　GPU の構成

本研究で使用する GPU は、NVIDIA 社の Fermi アーキテクチャの Geforce GTX 480 である。図 2 に Geforce GTX 480 の構成を示す。Geforce GTX 480 は、Giga スレッドスケジューラと 4 個の GPC（Graphics Processing Cluster）から構成されている。Giga スレッドスケジューラはスレッドブロックを SM のスレッドスケジューラへと分配する機能を担当する。各 GPC には 4 個のストリーミング・マルチプロセッサ（SM）、及びポリゴンを画面上のピクセルに対応づけるラスターエンジンが搭載されている。つまり、GPU 全体では 16 個の SM をもつことになる。Fermi アーキテクチャは第 3 世代の GPU で、メモリ階層もグローバルメモリ、L2 キャッシュ、共有メモリ/L1 キャッシュの 3 階層に強化されている[7]。

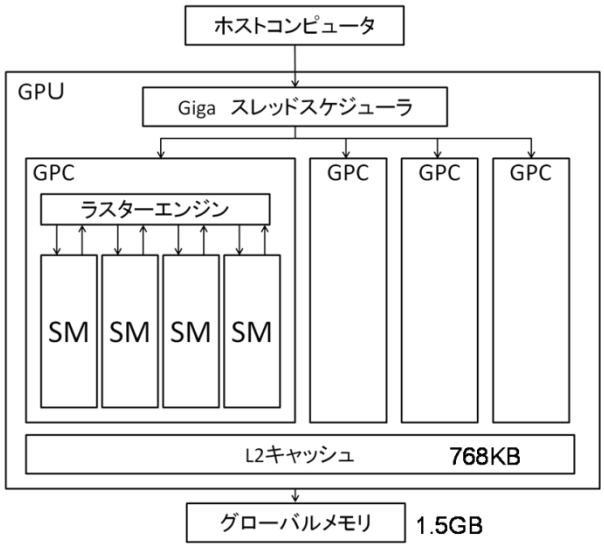

図 2　GTX480 の構成

図 3 に SM の構成を示す。各 SM は 32 個のストリーミング・プロセッサ（SP）、正弦関数や余弦関数を計算する SFU（Special Function Unit）を 4 個、ワープ・スケジューラとディスパッチ・ユニットがそれぞれ 2 個ある。また、64KB の共有メモリ/L1 キャッシュ、128KB のレジスタファイル、16 個のロード・ストアユニットをもつ。ロード・ストアユニットが 16 個あるため、1 クロックあたり 16 スレッド分のソースアドレスと宛先アドレスを計算することができる。

CUDA 対応 GPU では、ある計算処理をスレッドと呼ばれる単位に分割して並列計算を行う。まず、スレッドスケジューラが GPU のプログラムをスレッドに分割して、SM の中の SP に割り当てる。SP はクロックサイクル毎に複数のスレッドに対して同一の命令を実行する。これを SIMT（Single Instruction Multiple Thread）と呼ぶ。

SM は 32 個のスレッドをグループ化したワープを単位として、スレッドのスケジューリングを行う。SM にはワープ・スケジューラとディスパッチ・ユニットがそれぞれ 2 個ずつあるため、2 クロックで 2 ワープを実行することができる。各ワープ・スケジューラは、1 つのワープを選択し、1 命令で 16 スレッドを実行する。ディスパッチ先となるのは、16 個の SP、16 個のロード・ストアユニット、4 個の SFU のいずれかのグループとなる。

Geforce GTX 480 には 16 個の SM があるが、NVIDIA 社の仕様により 1 個の SM を無効にしている。すなわち、SM は 15 個、SP は 480 個が並列に動作している。

3.2　ストリーミング・マルチプロセッサによる画面分割

レイトレーシングは処理時間が膨大であり、その多くは交差判定の処理に費やされている。つまり、交差判定の高速化がリアルタイムレイトレーシングには必要不可欠であり、そのためには、交差判定の回数をアルゴリズムの改良により減らす、あるいは交差判定自体を高速化することが必要である。ここでは交差判定を高速化するために GPU を用いた画面分割による並列化を行う。

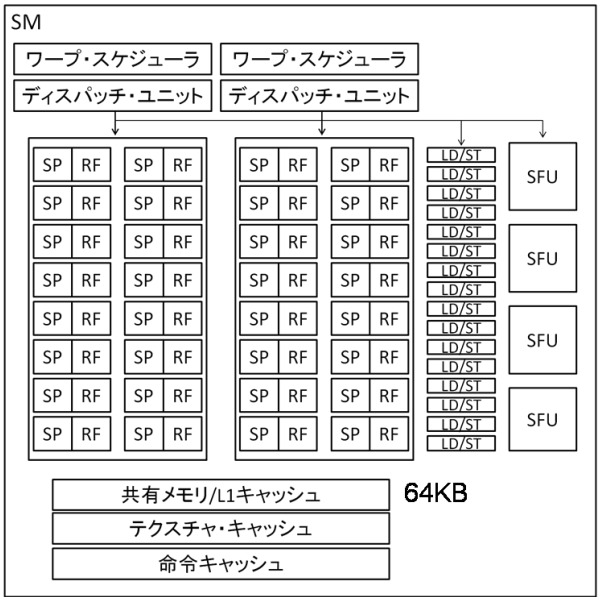

図 3　ストリーミング・マルチプロセッサ（SM）

画面分割による並列化は、レイトレーシングの特徴を利用して行う。レイトレーシングでは、スクリーン上のある画素の輝度値を求める処理が、他の画素の輝度値を求める処理に依存していないので、各画素の輝度値を独立に計算できる。

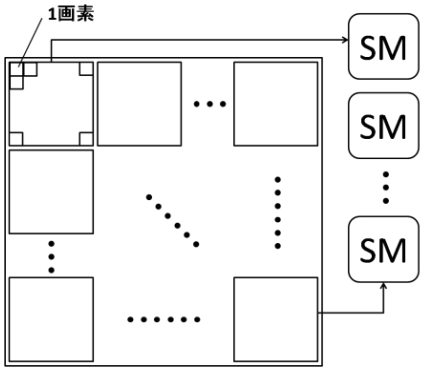

図 4　画面のブロック分割による並列実行

本研究では 512*512 の解像度のスクリーンを用いる。SM による画面分割では、図 4 に示すようにスクリーンを複数のブロックに均等に分割し、各ブロックをそれぞれ SM に割り当てる。各 SM ではブロック内のスレッドを 32 個単位のワープに分割する。

4. GPU 上での並列処理方式

図 5 に作成した処理系の構成を示す。本プログラムは CPU 側と GPU 側の 2 つの処理から成り立っている。

まず、CPU 側でレイトレーシングの実験用のシーンデータである SPD（Standard Procedural Databases）ファイルを読み込む。SPD には視点ベクトル、スクリーンの背景色、照明、物体素材の性質、球、三角形などの定義が含まれている。文献[3]では SPD 情報をリスト構造により管理しているが、リスト構造では GPU 側に正しくデータを渡すこ

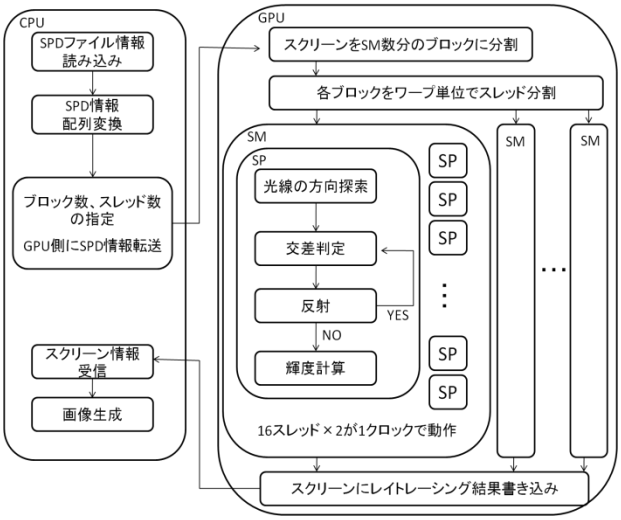

図 5　モジュールの構成

とができない。そのため、本プログラムでは SPD 情報をリストから配列構造に変換して管理する。本実験で用いた 3 つのシーンデータ（図 6～図 8）のポリゴン数とサイズを表 1 に示す。CPU 側でブロック数とスレッド数を指定し、SPD 情報を GPU 側に転送する。

表 1　SPD シーンデータ

	teapot	tetra	mount
ポリゴン数	2328	4096	8192
サイズ（KB）	370	249	617

GPU 側では受け取ったブロック、スレッド情報を元にスクリーンを複数のブロックに分割する。次に、各ブロックをワープ単位の 32 スレッドに分割する。この時 1 スレッドはスクリーンの画素 1 つの処理を行う。各スレッドでは 1 画素分のレイトレーシング結果を得るために、光線の方向探索、交差判定、反射、及び輝度計算を行い、スクリーンにレイトレーシング結果を書き込む。

CPU 側でスクリーン情報を受信する。受信した結果を ppm 形式のファイルに変換することで画像を得る。

GPU のメモリは外部のグローバルメモリ、GPU 内の L2 キャッシュ、SM 内の共有メモリ/L1 キャッシュの 3 つの階層から構成されている。本研究ではスクリーンをグローバルメモリ上に確保している。

SPD 情報は最初グローバルメモリに転送される。各 SP による 1 画素分のレイトレーシング計算では、SPD 情報全体を必要とするので、SPD 情報は L2 キャッシュを介して、L1 キャッシュに転送される。共有メモリ/L1 キャッシュは（48KB,16KB）、または（16KB、48KB）を選択でき、本研究では後者を採用している。L1 キャッシュは 48KB であるので、SPD 情報がすべて入らず、L2 と L1 間の転送が必要に応じて発生すると考えられる。

5　実験と考察

5.1　実験環境

図 6～図 8 に示す 3 種類のシーンデータ（teapot, tetra, mount）に対して、GPU を用いた画面分割による並列化を行った。各シーンデータに対して、ブロック数とスレッド数を変化させて画面分割を行い、最適な分割パターンを検討した。実験は以下に示す 5 通りのブロック数とスレッド数で行った。

- ブロック数 256*256、スレッド数 2*2
- ブロック数 128*128、スレッド数 4*4
- ブロック数 64*64、スレッド数 8*8
- ブロック数 32*32、スレッド数 16*16
- ブロック数 16*16、スレッド数 32*32

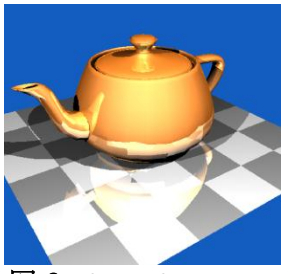

図 6　teapot

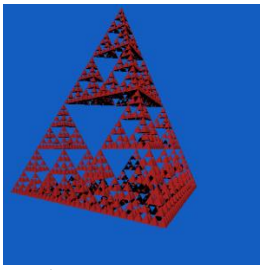

図 7　tetra

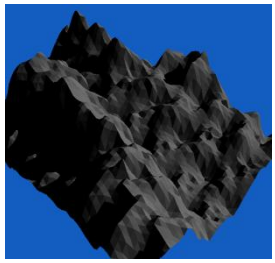

図 8　mount

例えば、ブロック数 64*64、スレッド数 8*8 の場合は、スクリーンを 64*64 ブロックに分割し、各ブロックは 8*8 の画素で構成されている。このとき 1 ブロック内には 64 スレッドがあるため、ワープが 2 つ存在することになる。1 つのブロックは 1 つの SM で処理され、1 つのスレッドは 1 つの SP で処理される。

CPU1 個（4 コア）の実験環境は、OS：Windows7 Ultimate、プロセッサ：Intel(R) Core(TM) i7 CPU 950 @3.07GHz、メモリ容量：6.00GB である。

GPU を用いた画面分割による並列化の実験環境は、CPU1 個の環境に、GPU：Geforce GTX480、グラフィッククロック：700MHz、メモリクロック：1848MHz、メモリ容量：1536MB、並列処理環境 CUDA を加えたものである。

5.2　実験結果

CPU1 個の実行時間は図 6 の teapot が 198(秒)、図 7 の tetra が 93.2(秒)、図 8 の mount が 263(秒)である。並列実行環境での実行時間と、CPU1 個に対する速度向上を表 2 に示す。

表 2 GPU を用いた画面分割の並列化の実験結果

ブロック数 スレッド数	teapot		tetra		mount	
	実行時間(秒)	速度向上(倍)	実行時間(秒)	速度向上(倍)	実行時間(秒)	速度向上(倍)
256*256 2*2	13.9	14.2	7.75	12.0	20.7	12.7
128*128 4*4	3.60	55.0	2.00	46.6	5.22	50.4
64*64 8*8	1.71	116	0.92	101	2.41	109
32*32 16*16	1.83	108	0.94	99.1	2.45	107
16*16 32*32	2.17	91.2	0.99	94.1	2.56	103

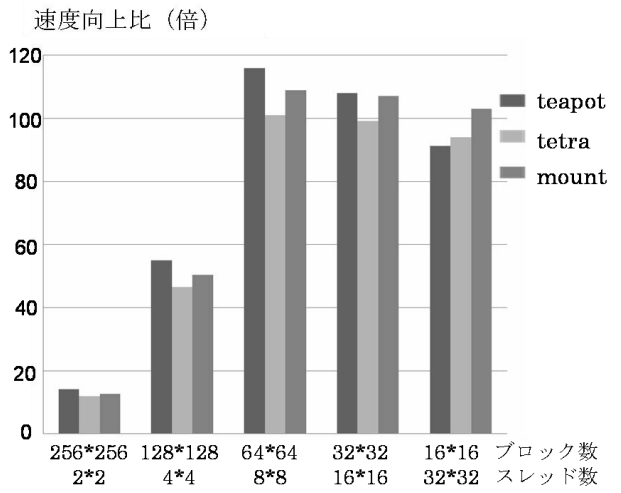

図 9　teapot、tetra、mount 速度向上比

図 9 は teapot, tetra, mount の 3 つのシーンの速度向上比を表したものである。このグラフからどのシーンも共通して、ブロック数が 64*64、32*32、16*16 の場合は速度向上比が大きく、ブロック数が 64*64、スレッド数が 8*8 の場合が最も高速であると確認できる。ブロック数が 256*256、128*128 の場合は速度向上があまり得られていない。

5.3　考察

ブロック数 64*64、32*32、16*16 の場合は、それぞれ 1 ブロック内のスレッド数が 64、256、1024 である。これらはワープ内のスレッド数 32 の倍数であるので、各 SP が常に動作することができる。そのため、速度向上比が大きくなっている。ブロック数 64*64、32*32、16*16 の 3 つの場合を比較すると、ブロック数が少なくなるほど、速度向上比が小さくなっている。これは、ブロック数が少なくなると 1 ブロック内に含まれるスレッド数が増加し、スレッドあたりの使用レジスタ数が減少したためである。

また、SM は 15 個が並列に動作しているので、64*64 の場合が、他の場合に比べて、使用効率が良い。したがって、ブロック数 64*64 の場合が最速で、速度向上比が最も高いと考えられる。

ブロック数 256*256、128*128 の場合は、それぞれ 1 ブロック内のスレッド数が 4、16 であり、32 よりも小さいので、各 SP に均等に処理を振り分けることができず、速度向上比が大幅に低下している。

表 2 の実行時間は、表 1 の SPD シーンデータのサイズと相関が認められる。1 画素のレイトレーシング計算で SPD 全体を参照するので、SPD のサイズにほぼ比例した時間がかかっていると考えられる。

6　おわりに

本論文では、GPU を用いたリアルタイムレイトレーシングの画面分割による並列化について述べた。レイトレーシングの各画素の輝度値を求める処理が、他の画素の計算と独立であることを用いて、画面を均等にブロック分割し、各ブロックを各ストリーミング・マルチプロセッサで並列実行した。

実験では Geforce GTX 480 を用い、SPD の 3 つのシーンデータ（teapot, tetra, mount）に対して、ブロック数とスレッド数を変化させて実行時間を測定した。その結果、teapot はブロック数 64*64、スレッド数 8*8 のときに、実行時間 1.71 秒で、CPU 1 個による実行と比べて 116 倍の速度向上を達成できた。これはスレッド数が 64 であり、ワープ内のスレッド数 32 の 2 倍になっているので、全ての SP を同時に使用できているためである。また、後継の GTX580 は 16 個の SM が同時動作するので、ブロック数 64*64 との適合性が良く、さらなる速度向上が期待できる。

本研究の最終目標は、レイトレーシング法での画像生成を 100ms 以下で実現することである。今後、適応型空間分割を用いて交差判定回数を減少させ、現在の 10 倍程度の速度向上を達成することが課題である。

参考文献

[1] James D. Foley, et al：コンピュータグラフィックス理論と実践, オーム社, 2005.
[2] 小山田耕二, 岡田賢治：CUDA 高速 GPU プログラミング入門, 秀和システム, 2010.
[3] Patrickomatic.com：http://patrickomatic.com/c-ray-tracer
[4] Gourmel, O., Pajot, A. and Paulin, M.: BVH for Efficient Raytracing of Dynamic Metaballs on GPU, SIGGRAPH2009, 2009.
[5] Kashyap, S., Goradia, R., Chaudhuri, P. and Chandran S.: Real Time Ray Tracing of Point-based Models, 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games, 2010.
[6] Singh, J.M. and Narayanan P.J.: Real-Time Ray Tracing of Implicit Surfaces on the GPU, IEEE Transactions on Visualization and Computer Graphics, Vol.16, No.2, pp.261-272, 2010.
[7] NVIDIA：ホワイトペーパー　NVIDIA の次世代 CUDA コンピュートアーキテクチャ：Fermi、2009.