内容梗概

レイトレーシング法は、視点から出た光線を追跡し、交差する物体を調べ、その物体の輝度値をスクリーン上に投影することで画像を得る手法である。レイトレーシング法では、光線と物体の交差判定が処理の大部分を占めるので、交差判定を高速化することが重要である。

本論文では、この研究を改善するために GPU(Graphics Processing Unit)を用いた画面分割と空間分割を検討する。

GPU を用いた画面分割の並列化では、画像が生成されるスクリーンを複数のブロックに分割し、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し、32 個のストリーミング・プロセッサ（SP）を用いて 32 個のでスレッドを並列実行する。すなわち、複数 SM によるブロックレベルの並列処理と、32 個の SP によるマルチスレッドの並列処理を実現している。

実験では、画面分割の手法として、サイクリック分割を用いた。サイクリック分割では、SM を任意の画素に割り当てる必要があるが、任意の画素に割り当てることで、速度向上を得ることができなかった。

空間分割では、等空間分割では空間の分割数を変更し、占有率の測定を行った。多くのシーンデータでは、分割数が2^3の場合、空間の占有率は 50%を上回り、分割数が16^3あたりから占有率が 15%前後で収束傾向となった。

目次

図目次

表目次

1. はじめに

近年、天体、流体、自然現象から映画などの様々な分野でコンピュータグラフィックスが浸透してきている。コンピュータグラフィックスの分野で、実写性の高い画像を生成するための技術としてレイトレーシング法がある。レイトレーシング法は、光の屈折、反射など光学の法則を忠実にシミュレートした手法の１つであり、非常にリアルな画像を生成できるので、コンピュータグラフィックスの分野ではよく用いられている。一方で、レイトレーシング法は１画素ずつ光の挙動を計算するため、計算量が非常に膨大になるという欠点がある。

レイトレーシング法のアルゴリズムは、視点からスクリーン上の各画素を通過する光線を発生させ、すべての物体に対して交差する点を調べる。次に、視点とすべての交点との距離を調べ、最も視点に近い交点を可視点とする。また、その交点での反射、屈折など物体の質感に応じて、光線がどのように変化するかの計算を行い、この処理を順次行うことで、最終的に探索した可視点の輝度をスクリーン上に投影し、画像生成を行う。また、交差する物体が存在しない場合は、画素値を背景の輝度値とする。光線と物体との交点を探索する処理を交差判定と呼び、レイトレーシング法において、交差判定が処理時間の大部分を占める。

GPU とは、従来、画像処理のみを行うものだったが、2007 年頃から一般に普及し、画像処理以外の数値計算にも汎用されている。また、大規模な科学技術計算を主要目的としているスーパーコンピュータでも、GPU の活用が進んでいる。、GPU の構造は、SM(ストリーミングマルチプロセッサ)の中に SP(スカラプロセッサ)を持つ２レベルの並列演算ユニットとなっている。SP は算術演算など非常に限られた性能しか持たないが、SM は命令デコード機能などを持つ。

本研究では、この研究を改善するために GPU(Graphics Processing Unit)を用いた画面分割と空間分割を用いて高速化を図ることを目的とする。これらのプロセッサをスレッド、ブロックに分割し、並列に動作させることで、処理時間の高速化を図り、また、空間分割を適用することで交差判定の回数を削減する。現在、各画素の計算を GTX480,GTX580 を用いて画面分割で実現した。画面分割では、４つのシーンデータにおいて平均で 100 倍程度の高速化を実現し、最大で 116 倍の高速化を実現できている。

空間分割では、全体の空間をボクセルに分割しボクセル内の物体情報を管理する。通常のレイトレーシング法では１本の光線に対して、全ての物体と交差判定を行ってきた。しかし、空間分割を用いることで、光線が通過するボクセル内の物体とのみ交差判定を行い、交差判定回数の減少を実現する。ボクセルの分割方法として、等空間分割、適応型空間分割があり、等空間分割では、ボクセルの大きさが一定であるが、適応型空間分割では、ボクセルに属する物体数に応じて、ボクセルの大きさが変化する。レイトレーシング法に GPU を用いて画面分割を行い、等空間分割、適応型空間分割のアルゴリズムを適用することで処理時間の高速化を図る。

本論文では２章にレイトレーシングのアルゴリズム、光線の種類、交差判定、GPU、CUDA について述べ、３章では等空間分割、適応型空間分割の手法について述べる。４章で空間分割における実験結果と考察について述べる。５章では、GPU を用いた適応型空間分割手法を述べる。６章では本研究の成果と今後の課題を述べる。

2.ＧＰＵによるレイトレーシング

2.1　レイトレーシング法のアルゴリズム

現実の世界では、太陽や様々な光源から出た光物体に当たり反射し、その反射した光が人間の目に入ることでものを見ることが可能となる。つまり、光源から出る光線を１本ずつ辿って行き、目に入る光線を見つけだすと、像を再現することが可能となる。また、目に入る光線の数はごく少なく、ほとんどが目に見えない方向に進む。これを光源から辿るのは非常に効率が悪くなる。これらを考慮し、レイトレーシング法では、視点からある方向に視線を伸ばして、その伸ばした方向に物体があれば視点と物体との交点における色を計算する。つまり、現実の世界とは逆の方向に光をシュミレーションしている。これらの計算には、視点、物体、光源などの位置が最低限必要となっている。図 1 に示すように、以下の手順でレイトレーシングを行う。

（1）　視点からスクリーン上の画素に向けて光線を発生させ、光線探索を行う。
（2）　光線とすべての物体との交差判定を行い、光線と物体との交点を求める。複数の物体と交差する場合は全ての物体との交点を求め、交点が存在しない場合にはその画素は背景の輝度値とする。
（3）　光線と交差した物体のうち最も距離の近い物体を抽出する。
（4）　抽出した物体の輝度の計算をする。
（5）　反射や屈折が起これば，それを新たに探索する光線とみなして（2）～（4）の処理を繰り返し、見ることが可能な物体を抽出する。

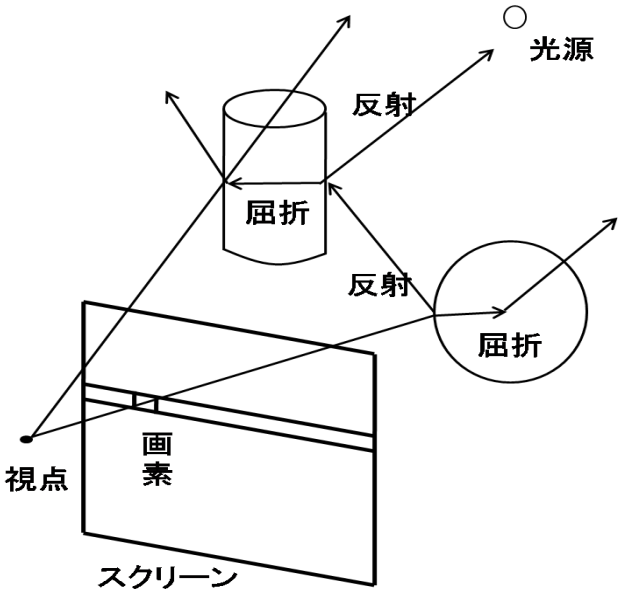


図 1　レイトレーシング法のアルゴリズム

レイトレーシング法では画像に立体感を出すために、光の種類、物体の材質などで定まる物体表面の色の変化や影を求める。影は光源から見えない場所に発生する。そこで、可視点を決定した後、その点から光源が見えるかどうかを判断する。図 2 に示すように、可視点から光源に向けて光線を放射し、すべての物体との交差判定を行う。もし、物体と交差すれば、その可視点はその光源に対して影の領域となり、輝度値はゼロとなる。レイトレーシング法におけるシェーディングの計算は以下の式で行う。I_aは環境光、K_aは環境光に対数反射率、I_lは点光源の入射光の強さ、K_d, K_sは拡散反射率、鏡面反射率、k_r, k_tは反射係数、透過係数となっている。

$$I = I_a K_a + \sum_{i=1}^{nL} I_l \{K_d (N \cdot L) + K_s (R \cdot V)^n\} + k_r I_r + k_t I_t$$

物体の陰影は、物体の質感や位置、光源の種類により変化する。光の種類は図 3 に示すように、拡散反射光、鏡面反射光、屈折光に分けられる。

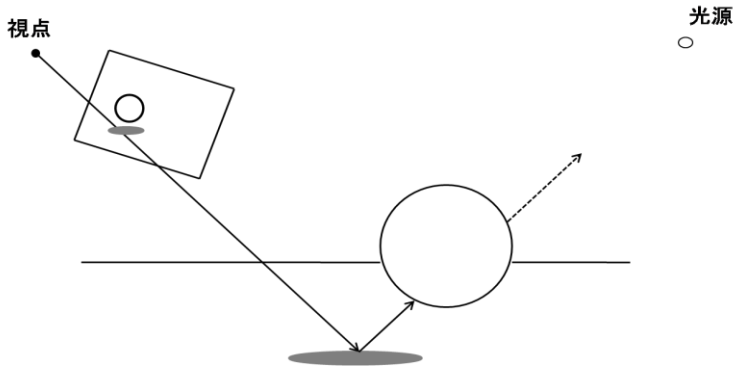


図 2　影の判定

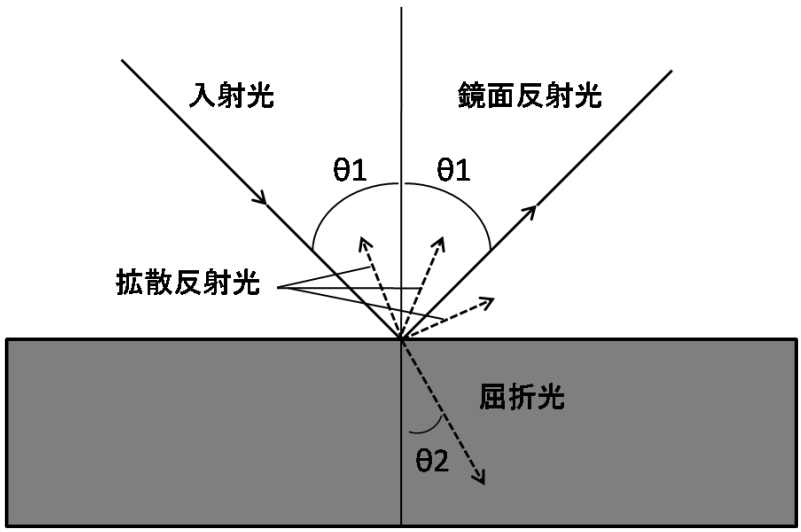


図 3　光の種類

拡散反射光とは、ざらざらした物体に対して行われる反射である。物体の内部に入った光が再び光の入射位置から外部に放射される考えである。また、散乱により拡散反射光では全ての方向に均等に放射される。拡散反射光は視線とは関係なくどの角度から見ても同じ輝度となる。図 4 に拡散反射光の原理を示す。

$$\mathbf{I_d = K_d I_l \cos\theta}$$

拡散反射光の輝度I_dは以下の式で求めることができる。K_dは拡散反射係数であり、I_lは反射光を求める点への入射光である。

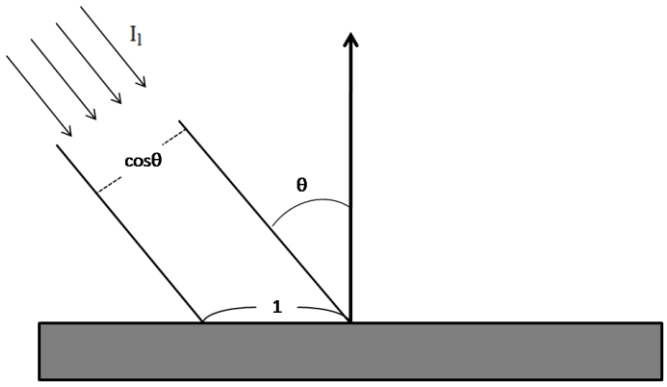


図 4　拡散反射光

鏡面反射とは、滑らかな物体に対して行われる反射である。プラスチック、金属などは、光源によって照らされた場合、鏡面反射によって、表面の一部にハイライトが生じる。図 5 に鏡面反射の原理を示す。鏡面反射光の輝度I_sは以下の式で求めることができる。

$$\mathbf{I_s = K_s I_l \cos^n\theta = K_s I_l (R \cdot V)^n}$$

V は視線ベクトルの逆ベクトル、R は反射光の方向余弦、γ を入射角と反射角のなす角、I_lを反射光を求める点への入射光、n をハイライト係数、K_sを鏡面反射率とする。

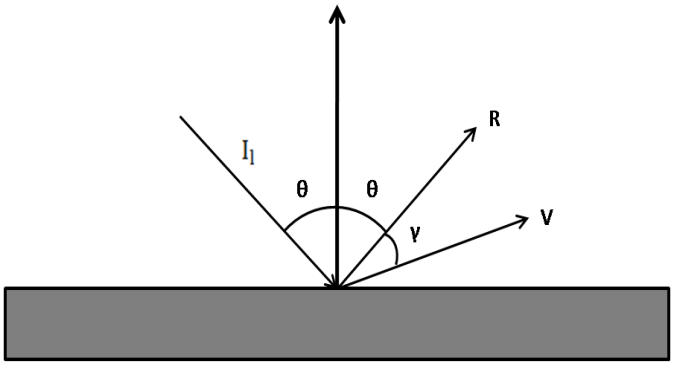


図 5　鏡面反射光

屈折光とはガラスや水面で起こる光の屈折であり、屈折率の異なる物質に入射すると、反射と屈折が起こる。屈折率は物質によって固有の値を持つ。図 6 に屈折光の原理を示す。屈折率n_1の物質の中から屈折率n_2の物質の中へ、光が入射すれば、入射角θ_1と屈折角θ_2の間にはスネルの法則が成り立つ。スネルの法則を以下に示す。

$$n_1 \sin\theta_1 = n_2 \sin\theta_2$$

よって、光の反射方向 R と屈折方向 T は以下の式で得ることができる。L は屈折光を求める点への入射光、N は面に対する単位法線ベクトルである。

$$T = \frac{1}{n}\{ L + (c - g) N \}, \quad R = L + 2cN$$

ここで $n=\frac{n2}{n1}$は相対屈折率であり、

$$c = \cos\theta_1 = -(L \cdot N), \quad g = \sqrt{n^2 + c^2 - 1}$$

屈折光での輝度 I は以下の式で求めることができる。Kt は透過率、It は透過した光の明るさ、Kr は反射率、Ir は反射した光線の明るさである。

$$I = Kt \times It + Kr \times Ir$$

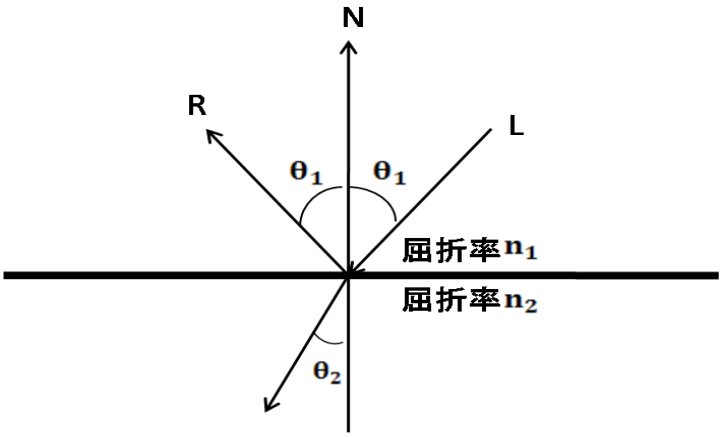


図 6　屈折光の原理

2.2 GPU アーキテクチャ

GPU(Graphics Processing Unit)とは 3D グラフィックスの表示に必要な計算処理を行う半導体チップである。従来 GPU は画像処理のみを行うプロセッサであったが、近年、浮動小数点演算能力などが搭載されたことにより、演算の幅が広がり、画像処理以外の大規模なデータの処理にも適用されている。GPU は多くのデータに対して同一の処理を行う SIMD
(Single Instruction Multiple Data)演算を用いている。また、GPU 上でのプログラミング環境として CUDA(Compute Undefined Device Architecture)を用いる。CUDA とは、NVIDIA 社が提供する GPU 向けの C 言語の統合開発環境である。

本研究で使用する GPU は、NVIDIA 社の fermi アーキテクチャの Grforce GTX480 である。Geforce GTX480 の構成を図 7 に示す。Grforce GTX480 はギガスレッドスケジューラ、L2 キャッシュ、4 つの GPC を持つ。ギガスレッドスケジューラはブロック、スレッドを SM(ストリーミングマルチプロセッサ)のスレッドスケジューラと分配する機能を持ち、L2 キャッシュは 1024K バイトである。各 GPC は 4 つの SM とラスターエンジンを持ち、ラスターエンジンはポリゴンを画面上に対応させる役割を持つ。よって、GPU 全体で 16 個の SM を持つ。SM は演算器部の集まりであり、内部に SP(スカラプロセッサ)、SFU(special Function Unit)、レジスタファイル、共有メモリ/L1 キャッシュ、ロードストアユニットなどを持つ。

SM の構成を図 8 に示す。SFU では、正弦関数や余弦関数を計算する役割を持つ。 共有メモリは 64K バイト、レジスタファイルは 128K バイトである。また、16 個のロードストアユニットがあるため、1 クロックに 16 個の SP を呼び出すことができる。GPU では、32 スレッドをまとめて 1Warp と呼ぶ。また、32 スレッドを前半部、後半部に分ける考え方を Harf Warp と呼ぶ。Geforce GTX480 では、16 個の SP が 2 クロックで動作する。Geforce GTX480 では、SM を 16 個つが、NVIDIA 社の仕様により 1 つの SM を無効にしている。したがって、GPU 全体の SP の個数は 15×32 = 480 個となる。Geforce GTX580 では、SM が 16 個動作し、GPU 全体の SP の個数は 16×32 = 512 個となる。

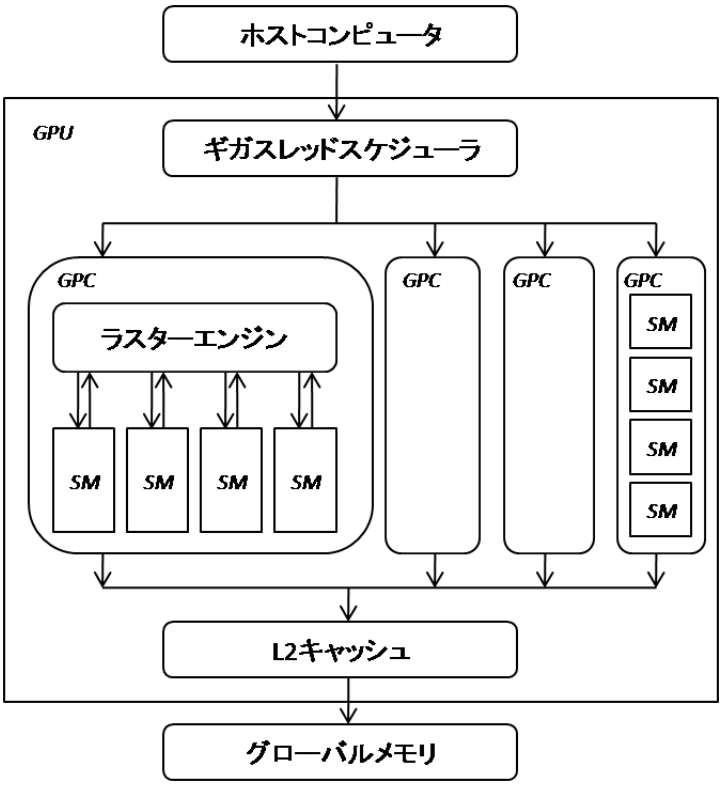


図 7　GeforceGTX480 の構成

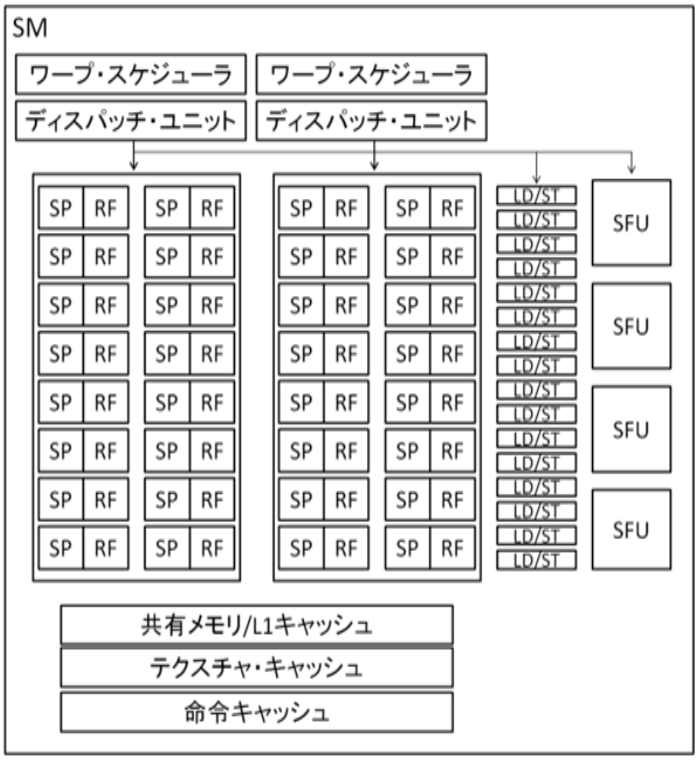


図 8　SM の構成

2.3 CUDA プログラミング

CUDA は C 言語、C++の一部の構文のみが対応しており、変数の型に GPU 特有の型しか使えないなど、汎用的なプログラムは困難である。CUDA はコンピュータのマザーボード上にあるビデオカード内で動作する。このとき、マザーボードには CPU があり、メモリも装着されている。また、ビデオカードには GPU が搭載されており、VRAM（Video RAM）も搭載されている。マザーボード及び CPU 側をホスト、GPU 及び VRAM 側をデバイスと呼ぶ。ホスト側のプログラムは、ホスト上の CPU で動作し、ホスト上のメモリを利用する。また、デバイス側で動作するプログラムをカーネルプログラムと呼び、カーネルプログラムは、GPU が処理を行い、VRAM のメモリを利用する。ホスト、デバイスのデータの移動例を図 9 に示す。(1)から(5)までの番号は、プログラムとデータの流れが対応している。(1)では CPU 側にメモリ確保、(2)で GPU 側にメモリ確保、(3)(4)(5)でそれぞれ CUDA におけるデータの移動を行っている。

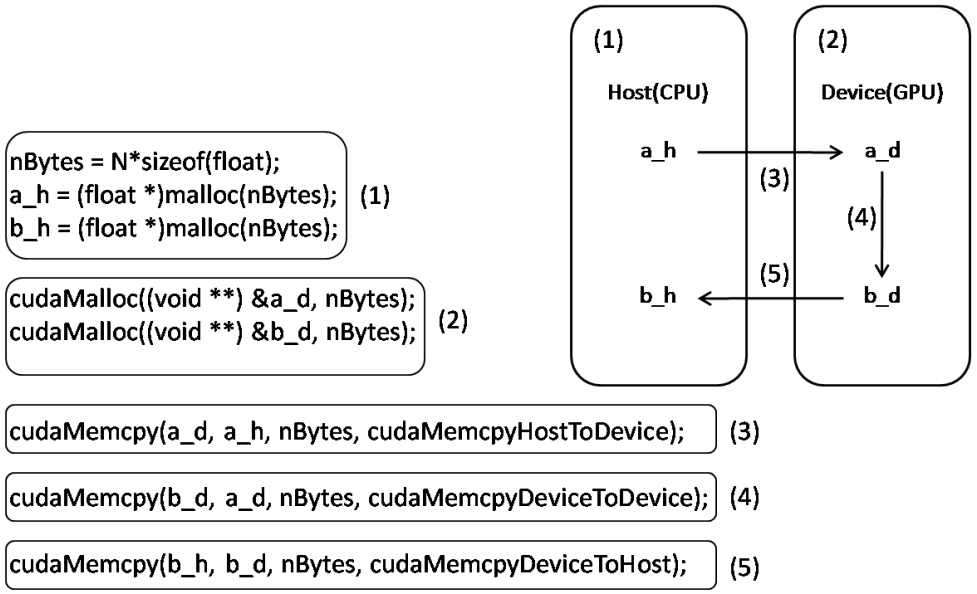


図 9　ホスト、デバイスのデータの移動

CUDA の並列実行の単位としてグリッド、ブロック、スレッドがある。これらの階層関係を図 10 に示す。グリッドはブロックの集合であり、現在 x 方向あるいは y 方向に配置できる最大ブロック数は 65535 個となっている。ブロックはスレッドの集合となっており、ブロックとは CUDA におけるプログラムを分割する並列単位で、1 つのブロック当たり最大 512 スレッド格納することができる。スレッドはカーネルを起動させたときの多数のプログラムの最少単位となっている。1 つのブロックは、1 つの SM 上で処理され、各ブロックはそれぞれの SM 上で動作する。SM 内には 32 個の SP があるので、32 スレッドが 1Warp で実行される。ブロック、スレッドを番号で管理することでどの SM、SP がどの部分を処理するかを判別する。図 11 にインデックスによる処理位置の判定を示す。

threadIdx.x はブロック内で動作しているスレッドの場所であり、blockDim.x はグリッド内でのブロックで、動作しているスレッドの個数であり、blockIdx.x はグリッド内でのブロックで、現在動作しているブロックの場所が格納されている。

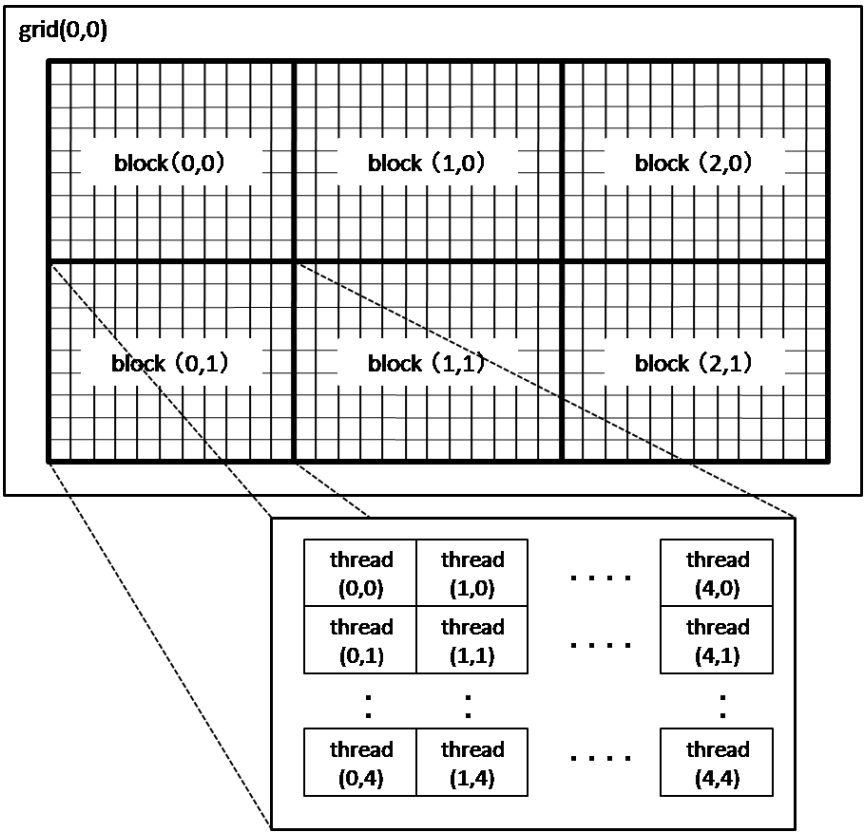


図 10　グリッド、ブロック、スレッドの階層関係

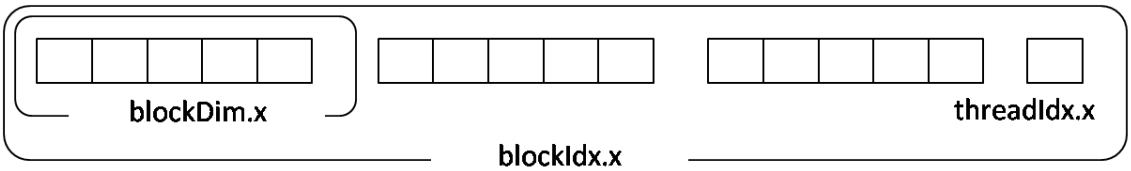


図 11　インデックスによる処理位置の判定

3. 空間分割による交差判定回数の削減

3.1 等空間分割と適応型空間分割

等空間分割とは、物体が存在する空間を決定し、それ以外の空間は考えないものとする。空間の決定は物体の物体の最大値、最小値から求めることができる。等空間分割では、ボクセルと呼ばれる一定の大きさに空間を分割し、その各空間に属する物体情報を得る。次に、レイトレーシングを行うが、基本的なレイトレーシングでは物体の個数分すべての物体と交差判定を行ってきた。しかし、空間分割では光線がボクセルに入った場合のみ交差判定を行い、交差判定の回数は光線の入ったボクセル内の物体の個数だけとなる。図 12 に等空間分割のイメージを示す。

ボクセルに物体が属していれば、ボクセル内のすべての物体と交差判定を行い、もし交点が存在すれば、その光線に対する追跡を終了する。交点が存在しない場合、ボクセル内に物体が存在しない場合は、現在のボクセルから次のボクセルへと移動し、再びボクセル内の物体との交差判定を行う。物体と交差し反射が起こった場では、反射光を新たに追跡する光線とみなし同様の処理を行う。光線が全ての空間を抜けた場合は、その光線に対しての追跡を終了とする。

実際に交差判定回数の変化を図 13 に示す。左図は基本的なレイトレーシングであり、右図は空間分割を用いたレイトレーシングである。物体４つの空間に対しての基本的なレイトレーシングでは、各光線に対して物体数回分の４回の交差判定を必要とする。空間分割を用いたレイトレーシングではボクセルに属する物体とのみ交差判定を行う。最も左の光線では２つのボクセルを通過するが、ボクセルに物体が属していないため、交差判定の回数は０回となる。よって図 13 の例では、基本的な(a)のレイトレーシングの交差判定の回数は４×５＝２０回となっているが、空間分割を行っている(b)交差判定の回数を６回まで削減することができる。

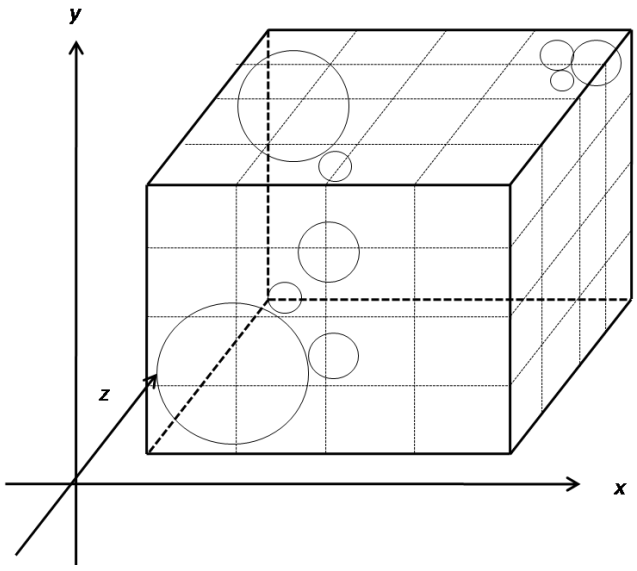


図 12　等空間分割

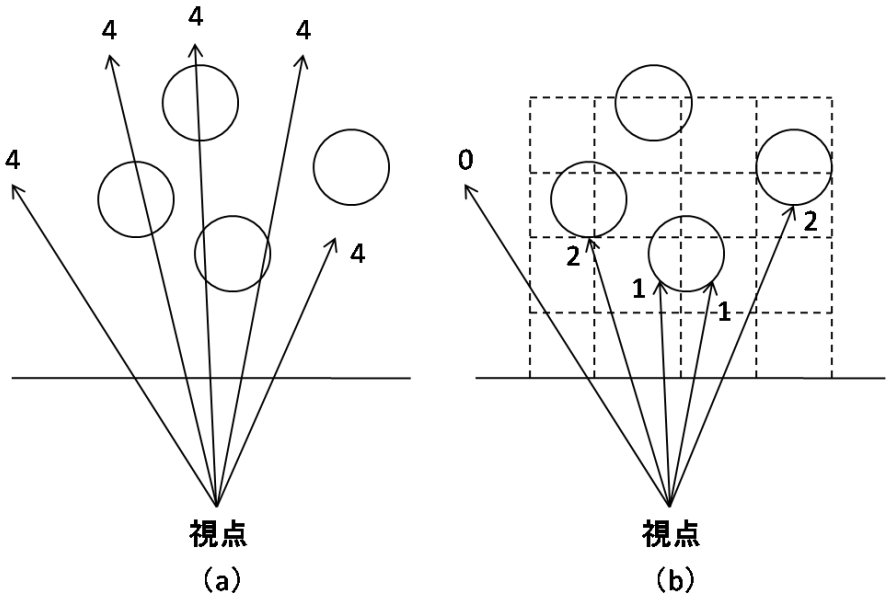


図 13　交差判定の回数

適応型空間分割とは、等空間分割では一定の大きさでボクセルの分割を行った。そのため、全体の交差判定の回数は削減できるが、各ボクセルに属する物体数は異なるので、各ボクセル内で行われる交差判定の回数にばらつきが生じる。適応型空間分割では、あらかじめ、空間を細かく分割して、物体数の少ない物体数の少ない周囲の空間同士を結合することにより、各ボクセル間で物体数を均等にするようにボクセルを決定する。この結果、どのボクセルに光線が入っても、ほぼ同じ回数の交差判定が行うよう、空間を最適化する。図 14 にボクセルの適応型空間分割の手順を示す。(a)では、全体空間を(x,y,z)それぞれに対し 128 等分に分割し、分割された空間に属する物体数を調べている。(b)では、(a)の各ボクセルの情報を元に空間を適応型に分割している。(b)では左手前の空間に物体が少なかった例である。

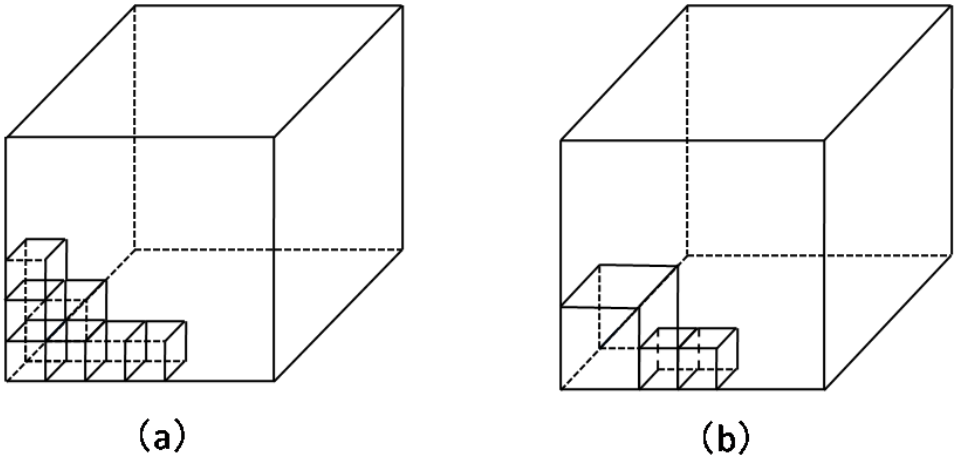


図 14　適応型空間分割

3.2　空間分割のアルゴリズム

空間分割を実装するために、各シーンデータにおける、物体情報の読み込み、全体ボクセルの作成、空間の分割、空間内の物体の調査、空間の最適化を行う。図 15 に空間分割の実装手順を示す。本研究では、SPD(Standard Procedural Databases)を用いて空間分割を行う。SPD はレイトレーシング用に開発されたシーンデータプログラムであり、SPD を用いて出力されたシーンデータを使用することにより、様々な画像を生成することが可能となる。出力されるシーンデータは nff 形式のファイルとなっており、nff ファイルは以下の情報で構成されている。

・物体の形状の定義(三角形、球、円柱)
・物体の位置の定義
・物体の材質の定義
・スクリーンの位置、大きさの定義
・視線ベクトルの定義
・光源の位置の定義

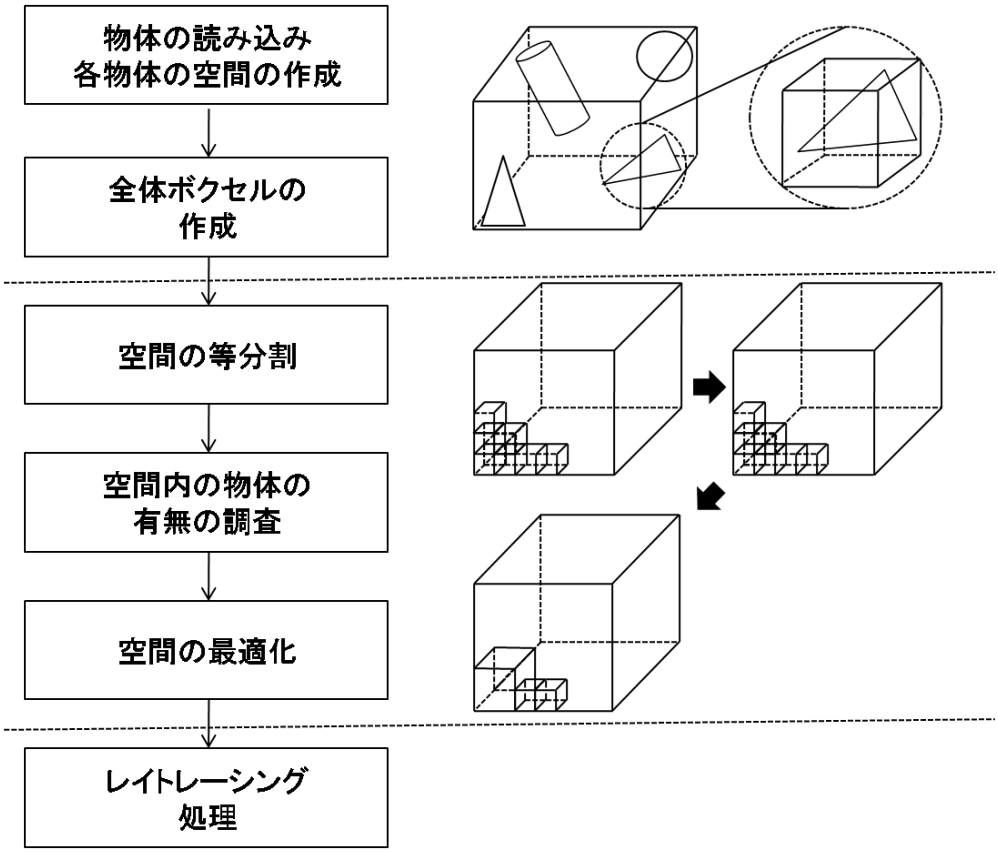


図 15　空間分割の実装手順

物体情報の読み込みでは、SPD シーンデータの物体に対して、ファイル処理を行うことで定義されている全ての物体情報を得る。レイトレーシング法などの CG では、同一物体に対し、異なる座標を定義することができる。座標の定義方法は、視点を中心に物体座標を定義するスクリーン座標系と、空間を中心に物体の座標を定義するワールド座標系がある。スクリーン座標系では視点、スクリーンを任意の位置に定義することにより、視点から物体をどのように眺めるかにより、物体の座標を決定する。ワールド座標系では、視点、スクリーンの位置に関係することなく、空間に属する物体の座標が決定される。よってワールド座標系とスクリーン座標系では、同じ物体であっても座標の値は異なる。本研究では、空間分割を行うにあたり、すべてワールド座標系で行っている。図 16 にワールド座標系、スクリーン座標系を示す。

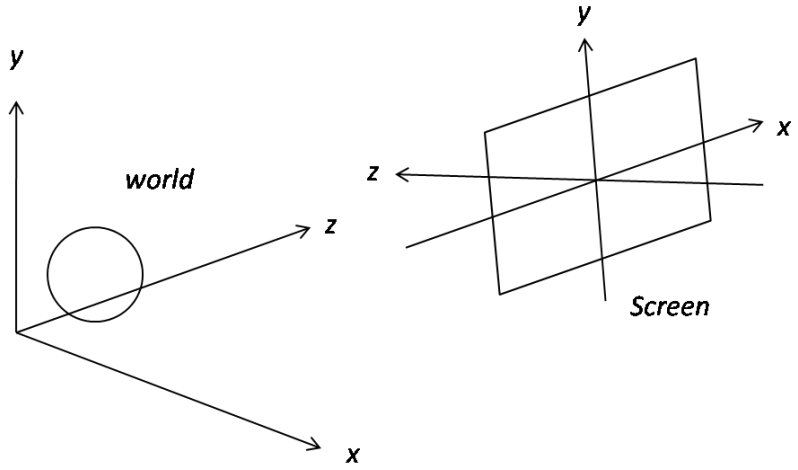


図 16　空間の座標系

3.3 全体の空間の作成

空間分割を行うためには、分割される全体の空間の作成を行う。全体の空間は物体の座標値をもとに作成する。本研究で扱う SPD シーンデータでには、三角形、球、円柱が含まれており、それぞれ座標の定義の仕方が異なっており、定義された物体の座標値のみからは物体の最大値、最小値を判断することができない。座標値から判断できない、球、円柱における座標の定義の例を以下に示す。s は球の定義であり、球における 4 つの座標は中心(x,y,z)と半径 r である。また、c は円柱であり、下底の中心(x,y,z)と半径 r1、上底の中心(x,y,z)と半径 r2 である。よって、座標から半径を考慮した値を求め、物体をすべて立方体のボクセルで囲い、ボクセルの最大値、最小値を用いて物体の最大値、最小値を決定する。図 17 に空間の作成のイメージ、

・s　-2.22045e-16　3.10258　0.791951　0.07412
・c　-0.930995　2.61313　0　0.07412　-0.287693　3.0091　-0.791951　0.07412

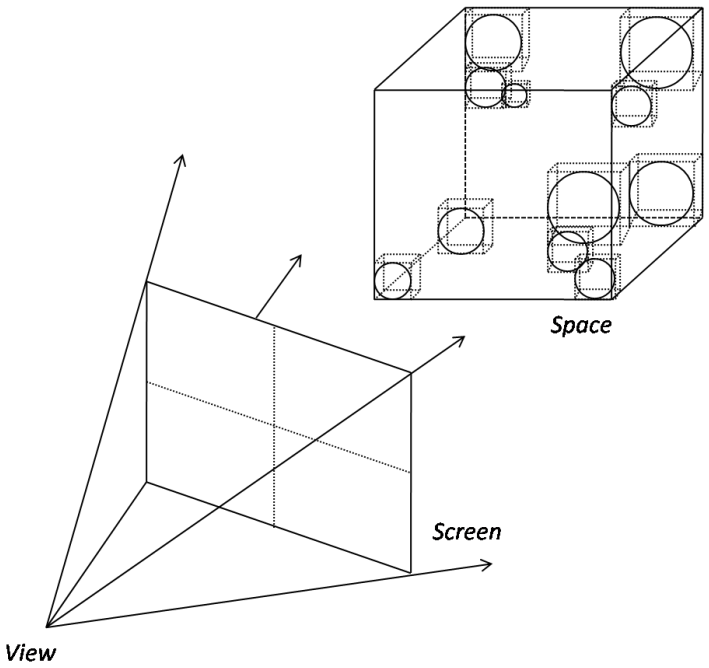


図 17　空間の作成

3.4 空間の分割

空間の分割では、物体から作成した全体の空間を等分割する。各分割された空間は、８つ頂点(x,y,z)、ボクセル番号で管理する。１つの分割されたボクセルにおいて、(x,y,z)はそれぞれ異なる２つの数値を持つ。この数値は分割されたボクセルに対する(x,y,z)の始点と終点となる。図 18 に作成した全体の空間の分割を示す。図 18 は全体の空間を(x,y,z)それぞれ 128 等分したものである。(x,y,z)の最大値、最小値がそれぞれ S,-S の場合の各頂点が持つを座標を以下に示す。

・ボクセル番号(0,0,0)の各点における座標値は、A(-S, -S+S/64, -S+S/64)、B(-S+S/64, -S+S/64, -S+S/64) 、C(-S+S/64, -S+S/64)、D(-S+S/64, -S+S/64,-S)、E(-S,-S, -S+S/64)、F(-S+S/64,-S,-S+S/64)、G(-S,-S,-S)、H(-S+S/64,-S,-S)である。
・ボクセル番号(64,64,64)における座標値は、I(0,0,0)である
・ボクセル番号(128,128,128)における座標値は、J(1,1,1)である。

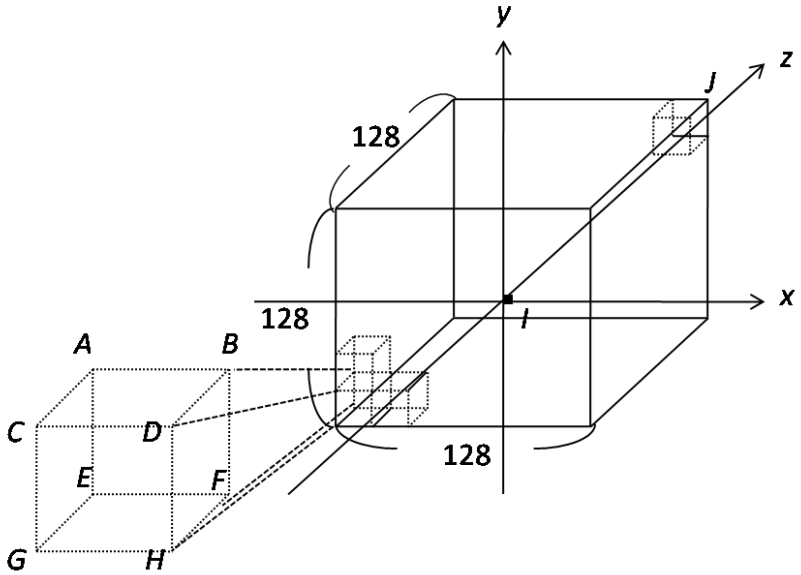


図 18　空間の分割

3.5 物体と空間の判別

空間分割を実装するために、分割された各ボクセルと物体を囲うボクセルの交差判定を行う必要がある。分割されたボクセルを空間ボクセル、物体を囲うボクセルを物体ボクセルとして示す。空間のボクセルと物体ボクセルの交差の関係は大きく分類すると、以下の３通りが考えられる。図 19 に物体と空間の交点の判別を示す。また、(c)においては、物体ボクセルの頂点が空間ボクセルに２点含まれる場合と１点含まれる場合が考えられる。よって、(c)はさらに分類すると１２通りの判定が必要となる。

(a)物体ボクセルが空間ボクセルの内部に存在する。
(b)物体ボクセルが空間ボクセルを含む。
(c)物体ボクセルの頂点が空間ボクセルの内部に一部存在する。

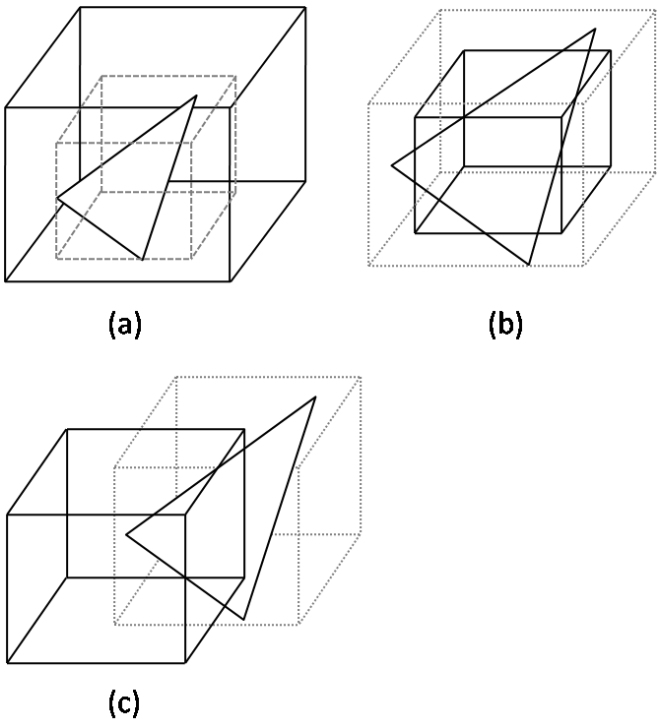


図 19　物体と空間の交点の判別

4. GPU 上での画面分割と等空間分割の実現

4.1　PU を用いたレイトレーシングの処理方式

図 20 に GPU を用いたレイトレーシングの処理方式を示す。本プログラムでは CPU(ホスト)側と GPU(デバイス)側から構成されている。

まず、ホスト CPU 側でレトレーシングの実験用シーンデータである SPD ファイルの読み込みを行う。読み込んだ SPD ファイルに対し、シーンデータにおける全体空間を作成し、空間の分割を行い、ボクセル単位で空間情報を管理する。空間データの登録を終えると、GPU 側でスクリーンの画面分割の単位である、ブロック数、スレッド数を指定し、SPD 情報を GPU 側に転送する。

デバイス側では、指定されたブロック数、スレッド数をもとに、スクリーンを SM 数分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。このとき、１スレッドはスクリーンの１画素の処理を行う。

各スレッドでは、１画素分のレイトレーシングの結果を得るために、光線探索、空間の参照、交差判定、反射、輝度計算を行い、スクリーンにレイトレーシングの計算結果を書き込み、CPU 側に転送する。CPU 側でスクリーン情報を受信し、受信した結果を出力する。

画面分割により、１画素に対して、SM 内の SP で光線を追跡し、処理を行い、空間分割を適用することで、交差判定の回数を削減することで高速化を図る。

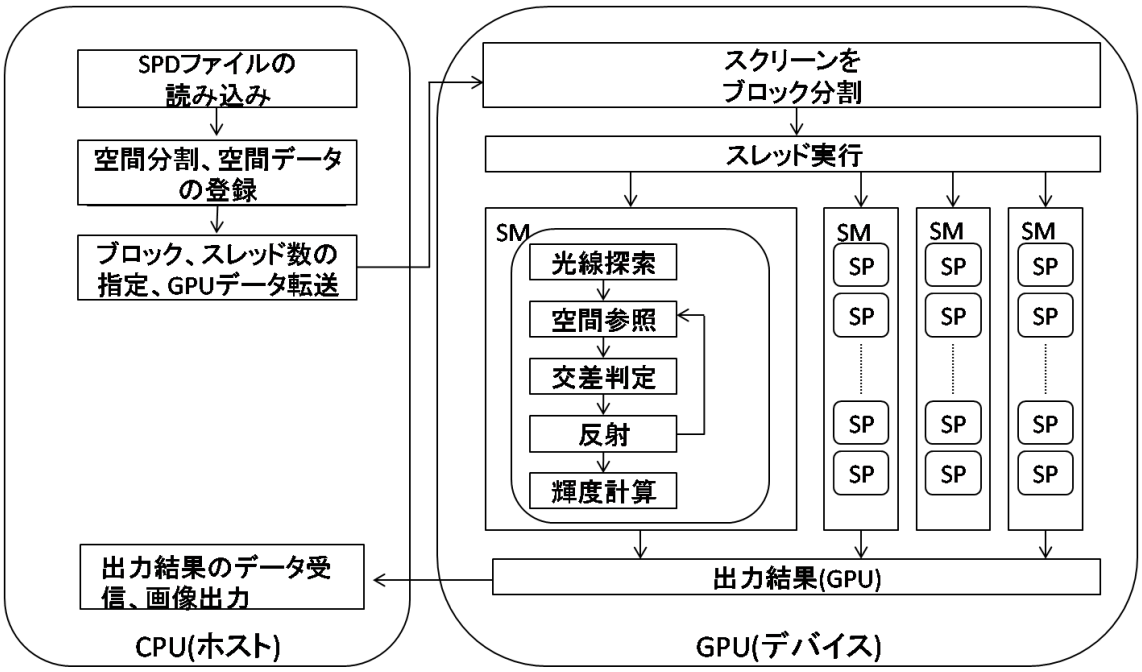


図 20　プログラムの流れ

4.2　画面のサイクリック分割

GPU を用いたレイトレーシング法では、スクリーンの画面分割を行うことで並列化を図る。レイトレーシング法では、画面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ため、並列処理の効果を得やすい。画面分割の手法はサイクリック分割である。図 21 にブロック分割とサイクリック分割を示す。(a)はサイクリック分割で、(b)はブロック分割である。スクリーンが 512*512 の解像度の場合、ブロック数 64*64、スレッド数 8*8 が最速であることが分っている。図 21 の(a)に示すようにまずスクリーンを 64*64 に２次元サイクリック分割する。SM0 は行方向で 0,4,8..番目、列方向で 0,4,8..番目を担当する。各ブロックは 8*8 の画素があるので、32 個の SP それぞれが、１画素の処理を担当し、交差判定、反射、輝度計算を行う。SM は 16 個あるので、インターリーブ分割で各ブロックを割り当てる。

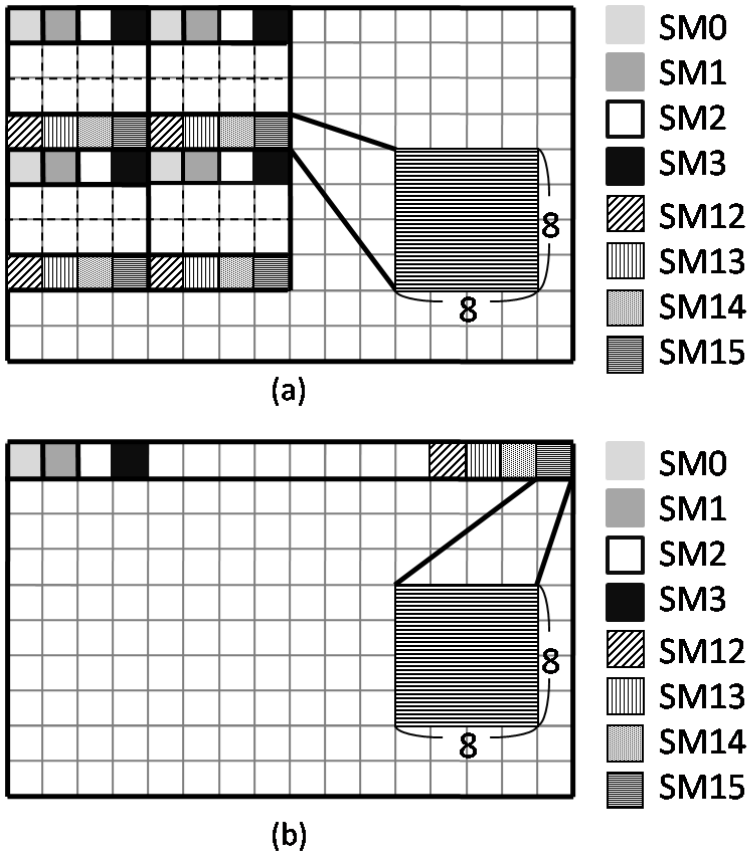


図 21　スクリーンの画面分割

4.3　等空間分割の実現

三角形、球、円柱を含む 7 種類のシーンデータに対して全体の空間の作成、空間の分割、物体と空間の判定を行う。全体の空間の作成では、NFF ファイルで定義されているシーンデータの最大値、最小値をもとに空間を作成する。空間の分割では、物体の最大値、最小値をもとに作成した空間を等分割する。分割数は、(x,y,z)の各軸をそれぞれ 2,4,8,16,32 等分で行う。2^3では、x,y,z をそれぞれ 2 等分しているので、分割されたボクセルの数は2^3=8 個となる。物体と空間の判定では、等分割された空間に対し、どの空間に物体が属しているかを判定する。図 22 に三角形の構造を示し、図 23 に空間と三角形のデータ構造を示し、図 24 に球の構造を示し、図 25 に球の構造体を示し、図 26 に円柱の構造を示し、図 27 に円柱の構造体を示す。また、球、円柱では、物体空間のデータ構造、分割された空間のデータ構造は三角形のデータ構造と同様であるが、NFF ファイルによって物体の定義の方法が異ため、物体データの構造体は異なる。

図 23 の三角形の場合、1 次元目で物体数個分の領域を確保し、2 次元目で物体の 3 頂点 top_i, top_j, top_k 分の領域を確保し、3 次元目で各頂点(x,y,z)の値を格納するための領域を確保する。

物体空間のデータ構造では、1 次元目で物体数個分の領域を確保し、2 次元目でボクセルの(x,y,z)の領域を確保し、3 次元目でボクセルの(x,y,z)の最大値、最小値を格納するための領域の確保を行っている。

分割された空間のデータ構造は、1 次元目で空間数個分の領域を確保し、2 次元目でボクセルの 8 個の頂点の値を格納するための領域を確保し、3 次元目で各頂点の(x,y,z)を格納するための領域を確保している。

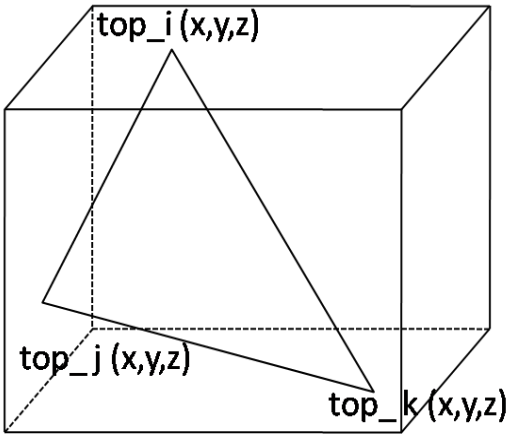


図 22　三角形の構造

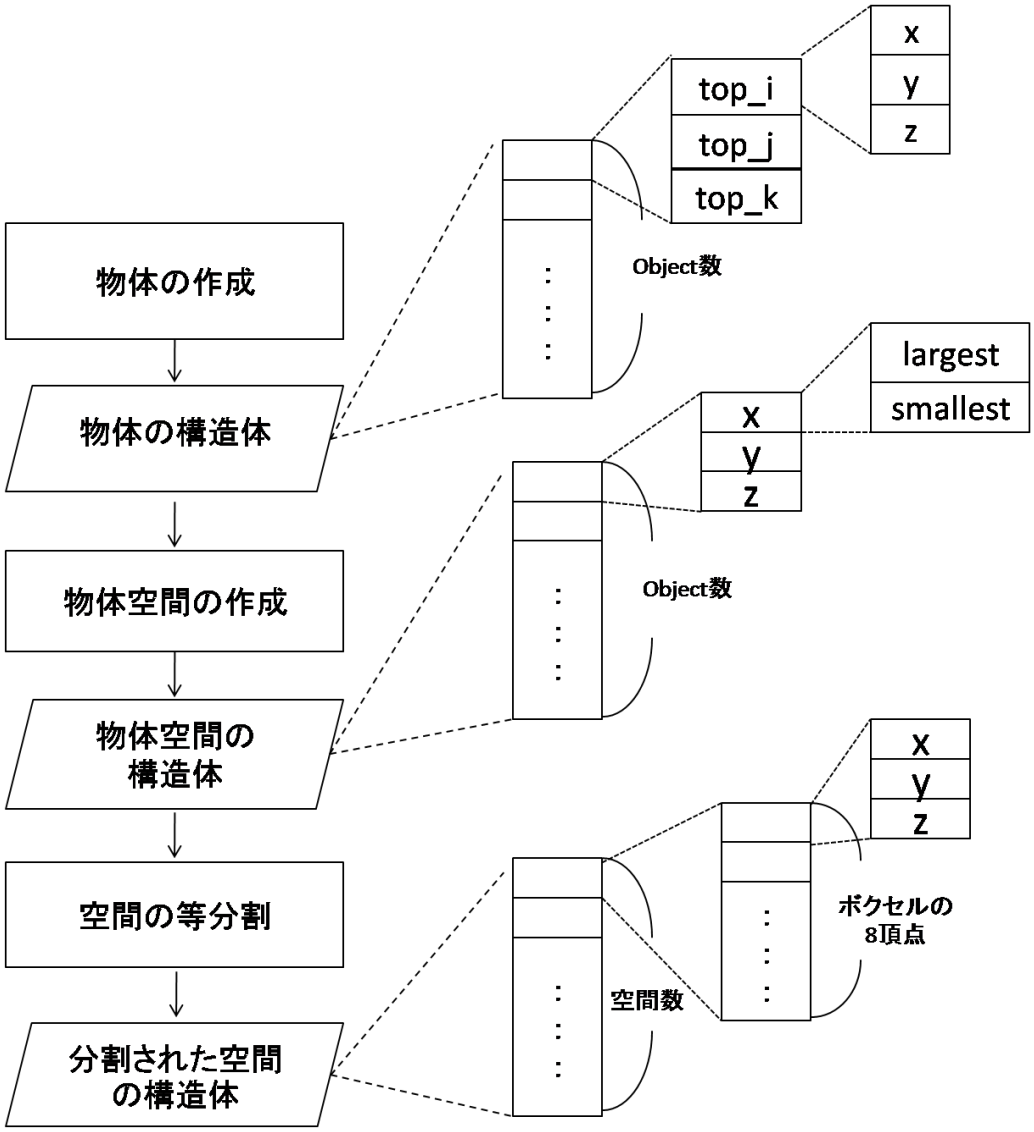


図 23　空間と三角形のデータ構造

図 25 の球の場合、１次元目で物体数個分の領域を確保し、２次元目で球の中心の座標、半径を格納する領域を確保し、球の中心の座行は(x,y,z)は構成されているので、３次元目に球の中心座標(x,y,z)を格納する領域を確保している。また、球における、物体空間の最大値、最小値は中心の(x,y,z)をそれぞれ+ ,－方向に半径分に拡張を行う。よって、最大値(x,y,z) = (x +r, y+r , z+r)となり、最小値(x,y,z) = (x－r, y－r , z－r)となる。

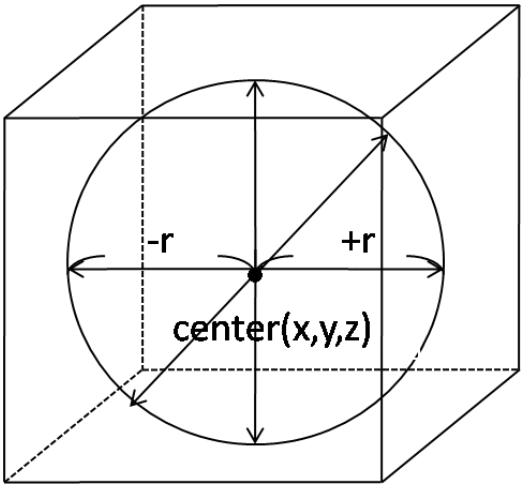


図 24　球の構造

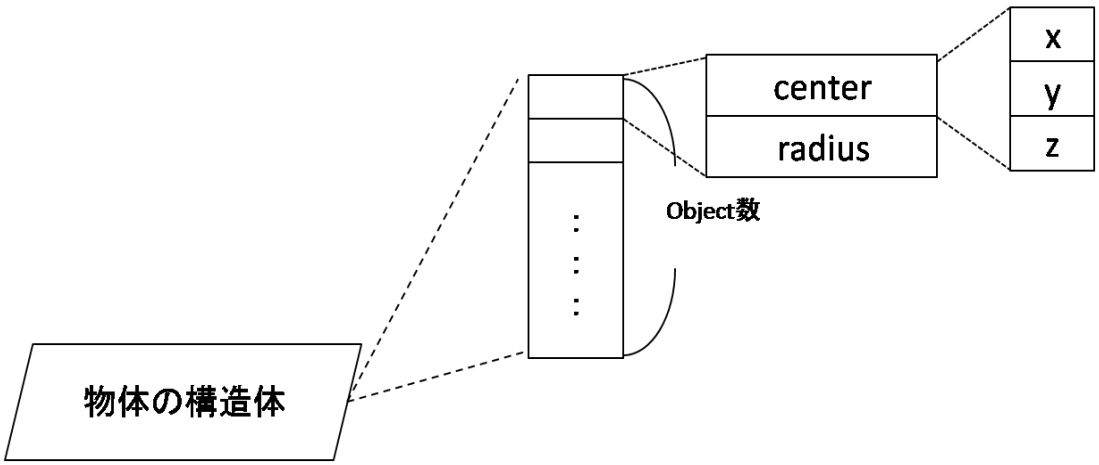


図 25　球の物体の構造体

図 27 の円柱の場合、上底の中心点を top_center、上底の半径を top_radius、下底の中心点を base_center、下底の半径を base_radius としている。１次元目で物体数個分の領域を確保し、２次元目で円柱の、top_center、top_radius、base_center、base_radius を格納する領域を確保し、３次元目に上底、下底の(x,y,z)を格納するための領域を確保している。また、円柱における、物体空間の最大値、最小値は、上底、下底の半径を比較し、半径の大きいものを中心から拡張する半径とし、上底と下底の(x,y,z)を比較して、+,－方向に半径分の拡張を行う。図 26 では、top_r と base_r では base_r の方が大きい。
top_center(2,5,0)、base_center(1,0,1)とすると、最大値(x,y,z) = (2 + base_r, 5 + base_r , 1 + base_r)となり、最小値(x,y,z) = (1 - base_r, 0 - base_r , 0 - base_r)となる。

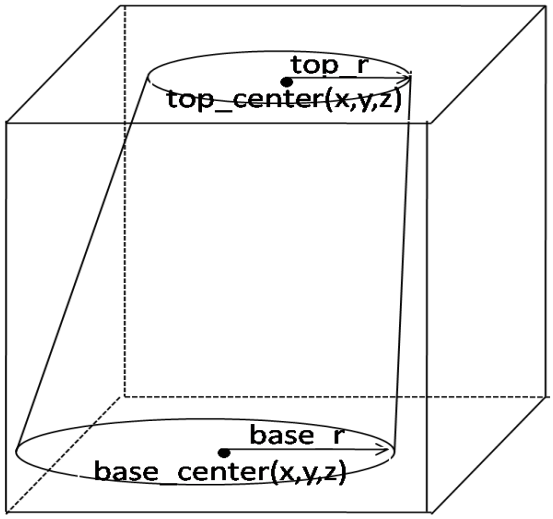


図 26　円柱の構造

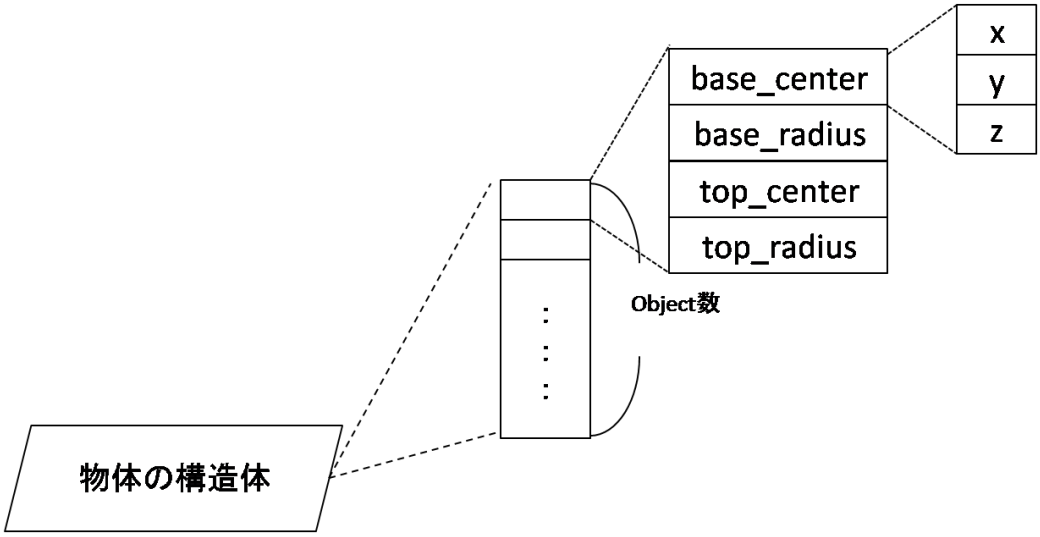


図 27　円柱の物体の構造体

4.4　空間分割の実験

4.3 で述べた、物体の作成、物体空間の作成、分割された空間の作成の実験を行う。実験環境は以下の環境で行う。

・OS：windows7 Ultimate 64 bit
・プロセッサ ：Intel(R)Core(TM)i7 CPU950 @3.97GHz
・メモリ ： 6.00GB

図 28 に三角形を構成要素とするシーンデータ、teapot,tetra,mount,nurbs を示す。また、表 1 に各シーンデータにおけるポリゴン数と全体空間のサイズを示す。表 2 に三角形を構成要素とするシーンデータでの、空間における物体の占有率を示す。

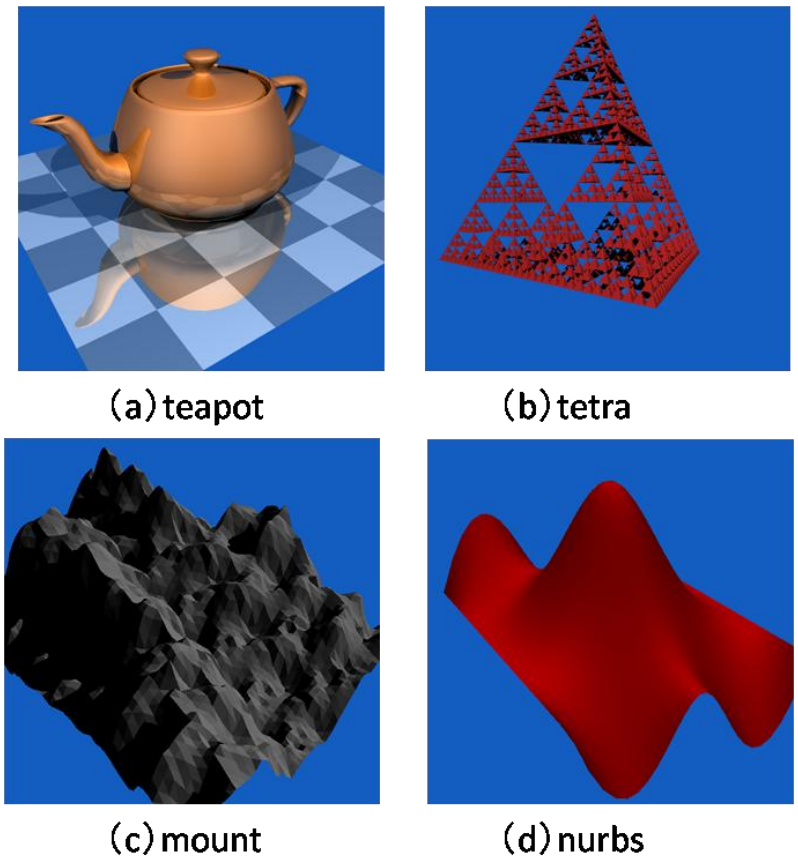


図 28　三角形を構成要素とするシーンデータ

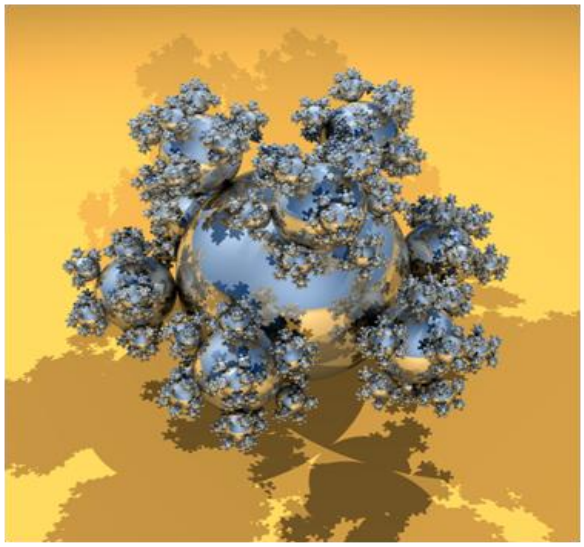
表 1　ボクセル数と空間のサイズ

シーンデータ	ポリゴン数	空間のサイズ	シーンデータ	ポリゴン数	空間のサイズ
teapot	2328	-4.00～4.00	mount	8192	-1.00～1.00
tetra	4096	-1.00～1.00	nurbs	1500	-3.16～3.16

表 2　物体の占有率(%)

空間数	2^3	4^3	8^3	16^3	32^3
teapot	50	35	21	13	12
tetra	50	31	12	12	13
mount	100	60	34	18	13
nurbs	75	39	20	13	13

図 29 に球を構成要素とするシーンデータ、balls を示す。また、表 3 に balls におけるポリゴン数と全体空間のサイズを示す。表 4 に球を構成要素とするシーンデータでの、空間における物体の占有率を示す。



(a) balls

図 29　球を構成要素とするシーンデータ

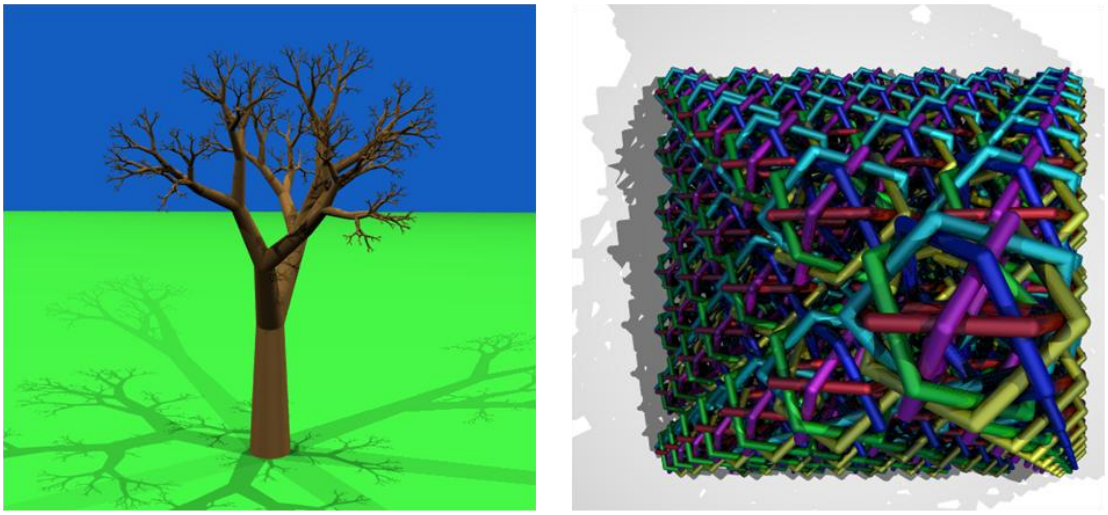
表 3　ボクセル数と空間のサイズ

シーンデータ	ポリゴン数	空間のサイズ
balls	7380	-0.95～0.95

表 4　物体の占有率(%)

空間数	2^3	4^3	8^3	16^3	32^3
balls	100	23	14	13	13

図 30 に円柱を構成要素とするシーンデータ、tree,rings を示す。また、表 5 に tree,rings におけるポリゴン数と全体空間のサイズを示す。表 6 に三角形を構成要素とするシーンデータでの、空間における物体の占有率を示す。



(a)tree　　(b)rings

図 30　円柱を構成要素とするシーンデータ

表 5　ボクセル数と空間のサイズ

シーンデータ	ポリゴン数	空間のサイズ
tree	8190	-3.13～3.13
rings	8400	-19.25～19.25

表 6　物体の占有率(%)

空間数	2^3	4^3	8^3	16^3	32^3
tree	50	17	15	13	12
rings	50	21	17	15	14

それぞれのシーンデータにおいて、空間数が2^3個の場合、空間の占有率は mount,balls で最大の 100%、teapot,nurbs で最少の 50%であった。分割数を細分化していくことにより、空間の占有率は減少している。また、空間数が16^3個から空間の占有率は収束傾向にある。これらのことから、空間数が16^3個で空間分割を行うことにより、2^3,4^3,8^3の場合より、大きな効果が得られると考える。また、空間数を32^3より細分化しても、空間の占有率は変化しないので、更に細分化する必要はないと考える。しかし、mount では16^3で 18%であり、32^3で 13%となっている。このことから、実行するシーンデータ次第で、更なる細分化を行うことにより、交差判定の回数を効率よく減少させることが可能だと考える。

4.5　型空間分割の検討

適応型空間分割では、空間の分割、物体と空間の判別を行い、あらかじめ物体の分布を調べ、物体を含まない空間の結合を行う。空間の分割では、全体の空間を x,y,z 軸をそれぞれ 128 等分など細かく分割し、各空間に属している物体を調べる。ここで物体数を登録し、物体の存在しない空間同士が隣接している場合、隣接している空間を結合し、新たに 1 つのボクセルとする。空間の結合は、物体の存在しない空間から、x 方向で、＋方向、－方向に結合する。図 31 に空間の結合を示す。(a)は空間を等空間分割したものであり、(b)は空間を x 軸方向に結合したものである。図 31 では、最もボクセル番号(0,0,0)からボクセル番号(3,0,0)までの空間に物体が存在しなかった場合である。

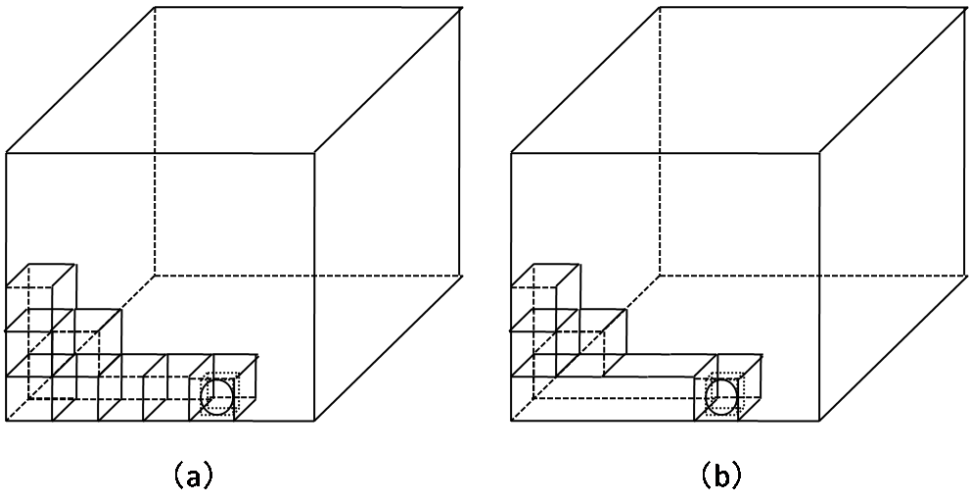


図 31　空間の結合

適応型に分割された空間はボクセル番号、ボクセルの座行(x,y,z)、物体情報で管理されている。空間分割をレイトレーシングに適用するために、光線の座標(x,y,z)とボクセルの座標(x,y,z)の交差判定を行う。最初に、各ボクセルに対し、物体を保有するボクセル、物体を保有しないボクセルをそれぞれ識別させておく。その後、物体を保有するボクセルの(x,y,z)と光線の座標(x,y,z)が交差関係にあるとき、ボクセル内に存在する物体とのみ交差判定を行う。交点が存在する場合は、反射等の計算を行い、交点を新たな光源とみなす。物体を保有しないボクセル(x,y,z)と光線の座標(x,y,z)が交差関係にあるとき、交差判定を行わずに、光線は進行方向へ進む。図 32 に空間分割の適用を示す。図 32 では、光線がボクセル番号が x,y がそれぞれ 0~1 と交差している。このとき、光線と物体の交差判定は物体 a とのみ行われる。同様に、光線がボクセル番号 x が 1~2、y が 0~1 と交差している。このとき、光線と物体の交差判定は物体 a,b と行われる。

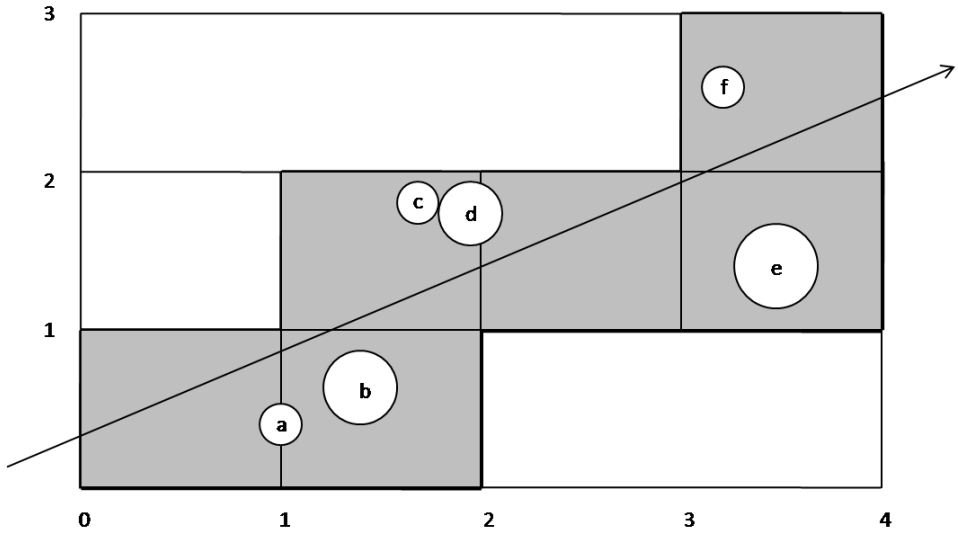


図 32　空間分割の適用

5. おわりに

本論文では、GPU を用いた画面分割の並列化、等空間分割、適応型空間分割について述べた。レイトレーシング法では、画面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ため、並列処理の効果を得やすい特徴を利用し、ストリーミングマルチプロセッサによる、サイクリック分割の検討を行った。サイクリック分割では、ストリーミングマルチプロセッサを任意の画素に割り当てる必要があるが、任意の画素に割り当てることで、速度向上を得ることができなかった。

空間分割では、等空間分割では空間の分割数を変更し、占有率の測定を行った。多くのシーンデータでは、分割数が2^3の場合、空間の占有率は 50%を上回り、分割数が16^3あたりから占有率が 15%前後で収束傾向となった。よって、分割数が16^3あたりから空間分割を適用することにより、大きな効果が得られると考えられる。

謝辞

本研究に機会を与えてくださり、ご指導を頂きました山崎勝弘教授に心より深く感謝いたします。さらに、本研究に助言をして頂いた孟林助手および、山崎研究室の皆さまに感謝の意を表します。

参考文献

[1] 上野謙二郎：GPU を用いたリアルタイムレイトレーシングの並列化の検討と実現, 立命館大学理工学研究科修士論文, 電子システムコース専攻, 2012.

[2] 孟林,上野謙二郎,山崎勝弘 : GPU を用いたリアルタイムレイトレーシングの並列化, 第 12 回情報科学技術フォーラム, FIT2012, 2C-1, 2012

[3] 吉谷崇史：空間分割による並列レイトレーシング法, 立命館大学理工学研究科修士論文情報システム学専攻, 1997.

[4] 原田博之：マルチプロセッサシステム上での画面/空間分割によるレイトレーシング法の高速化, 立命館大学理工学研究科修士論文情報システム学専攻, 1995.

[5]千葉則茂,土井章男： 3 次元 CG の基礎と応用, サイエンス社, 2009.

[6]岡田賢治：CUDA 高速 GPU プログラミング入門, 秀和システム, 2010

[7]岡崎大輔,孟林,山崎勝弘：GPU を用いた空間分割によるリアルタイムレイトレーシングの検討, 平成 24 年度情報処理学会関西支部大会 D-101

[8] 増田匠吾, 山田遼, 孟林, 山崎勝弘 : GPU を用いたレイトレーシングの高速化, 平成 24 年度情報処理学会関西支部大会, C-16

[9] 金森由博, 西田知是, : GPU を用いたメタボールの高速レンダリング, 第 127 回グラフィクスと CAD 研究会 , 2007