修　士　論　文

モデル非依存 BVH トラバースアルゴリズムの考案

北陸先端科学技術大学院大学
知識科学研究科知識システム基礎学専攻

瀬井 大志

2008 年 3 月

修　士　論　文

モデル非依存 BVH トラバースアルゴリズムの考案

指導教官　　宮田　一乗　教授

北陸先端科学技術大学院大学
知識科学研究科知識システム基礎学専攻

650035　瀬井 大志

審査委員：　宮田　一乗　教授（主査）
杉山　公造　教授
西本　一志　教授
由井薗　隆也　准教授

2008 年 2 月

A BVH traversal algorithm which is independent from models

Taishi Sei

School of Knowledge Science,
Japan Advanced Institute of Science and Technology
March 2008

Keywords: rendering, ray tracing, packet traversal, bounding volume hierarchy

For many years, photorealistic rendering of computer graphics has been an important issue. Ray tracing is a basic technique of photorealistic rendering. Ray tracing has long been a method to use for off-line rendering, however it is often too slow for interactive systems. In recent years, ray tracing has reached the level of interactive use due to the development of computer technology, and the result of research into ray tracing algorithms.

This thesis aims for the improvement of the existing algorithm. BVH-based packet traversal is the method to speed up performance of ray tracing. However, the effect of the BVH-based packet traversal depends on size of the packet and 3D scenes. Therefore existing BVH-based packet traversal is not effective in every case, and ineffective in some case.

This thesis describes a method that works equally well in every case. Our method is concentrated on the hierarchy of rays. It enables culling of rays from a packet, while the packet traversing a node of BVH. That can get rid of dependence on 3D scenes.

As a result of this thesis, BVH-based packet traversal is possible to use effectively in every situation. Our BVH-based packet traversal provides even faster performance of ray tracing.

目　次

図　目　次

表　目　次

第　1　章
序論

本章では，本論文で扱う分野について概説し，本論文の位置付けを示す．

1.1 はじめに

レンダリングはコンピュータグラフィックスにおける基幹要素であり，コンピュータによって 3 次元シーンから 2 次元の画像を生成する処理を指す．コンピュータグラフィックスが誕生して数十年が経過した今なお，レンダリングの写実性は重要な課題であり，盛んに研究開発が行われている．その背景には，コンピュータグラフィックスが応用される様々な産業分野からの需要がある．

工業デザイン分野においては，その多くの工程でコンピュータグラフィックスが応用されている．デザイン対象を写実的にレンダリングすることで，製作コストの高い実物を事前評価することができ，大幅なコスト削減に繋がる．特に，建築物や都市計画など施行後での修正が困難な分野への寄与は多大である．

映画やゲームなどのエンターテインメント産業における，映像製作での応用も盛んである．例えば，2007 年に公開された映画『トランスフォーマー』の劇中では，現実世界に存在しない金属生命体をコンピュータグラフィックスによって写実的にレンダリングし，その映像と実写とを合成することで迫力のある映像を実現し大きな話題となった．ゲームにおける映像品質も年を追うごとに向上しており，写実的な映像とは未だ隔たりがあるものの高品質なレンダリングが可能になっている．

レンダリング手法という観点から，今日のコンピュータグラフィックスを利用するアプリケーションを大別すると，以下の二つに分類することができる．一つは，レンダリング画像の品質を重視するオフラインレンダリング，もう一つはその速度を重視

するインタラクティブレンダリングである．

ゲームやバーチャルリアリティなどのインタラクティブ性が重視される分野では，DirectX や OpenGL で用いられるラスタライゼーションと呼ばれる手法をベースとしたレンダリングが用いられる．ラスタライゼーションでは，物体を多角形で構成し，各々の多角形を 2 次元画像へ変換することで最終的な画像を生成する．これは比較的計算量の少ない手法であり，インタラクティブなレンダリングを実現することができる．更に，GPU（Graphics Processing Unit）と呼ばれる専用ハードウェアが安価に市販されており，一般的な PC 上で高速なレンダリングを実現している．しかし，ラスタライゼーションは近似的な手法であり，そのレンダリング結果は物理的な正確さに欠ける．そのため，多くのラスタライゼーションを利用するアプリケーションでは，レンダリング品質を上げるため様々な擬似的効果をアーティストやデザイナによる手付けによって実現しており，その実装工程は複雑なものとなってしまっている．

一方で，映画などレンダリング画像の品質が求められる分野ではオフラインレンダリングが用いられる．昨今の映画作品で見られるように，実写と見紛うほどの高品質なレンダリングを実現しているオフラインレンダリングの技術基盤として，レイトレーシングがある．レイトレーシングは，現実世界の光の伝播を光学理論に基づいてシミュレートする手法である．そのため，大域照明など光学特性を考慮したレンダリングを実現するのに適した手法である．

コンピュータ処理性能の向上やアルゴリズムの発展に伴い，21 世紀初頭からレイトレーシングによるインタラクティブなフレームレートでのレンダリングが（未だ制約は多いものの）実現され始めている．2006 年，2007 年にはインタラクティブなフレームレートでのレイトレーシングを取り上げたシンポジウム“Symposium on Interactive Ray Tracing”[1]が開催され，多くの研究者の注目を集めている．

[1] http://www.uni-ulm.de/rt07/RT07.html

1.2 本論文の位置付け

本論文では，レイトレーシング高速化の一手法であるバウンディングボリューム階層（BVH: Bounding Volume Hierarchy）に対するパケットトラバースを改善したアルゴリズムを提案する．既存手法のパケットトラバース手法では，BVH トラバース過程が深い階層に進むほどレイのコヒーレンスが低下する．本論文では，このコヒーレンスの低下を抑え，3 次元シーン及びモデルに依存しないパケットトラバースの実現を目的とする．

1.3 本論文の構成

本論文の構成は以下の通りである．

第 2 章「レイトレーシングとその高速化」では，レイトレーシングの基本的な理論を示し，その計算量が膨大となる理由について述べる．そして，レイトレーシングを高速化するための空間データ構造及びハードウェアへの適応について述べる．第 3 章「バウンディングボリューム階層を用いたレイトレーシング」では，空間データ構造の一つである BVH，及びそれによるレイトレーシングの高速化について述べる．そして，BVH に対するパケットトラバースについて触れ，その効果及び問題点を実験に基づいて提示する．第 4 章「モデル非依存パケットトラバース」では，第 3 章で示したパケットトラバースの問題点を解決するためのアルゴリズムの提案を行う．そして，その効果について実験を通じて検証する．第 5 章「結論」では，本論文のまとめ，及び今後の課題，展望について述べる．

第　2　章
レイトレーシングとその高速化

本章では，レイトレーシングの基本的な理論を解説し，その計算量が膨大となる理由について述べる．そして，レイトレーシングを高速化するための空間データ構造及びハードウェアへの適応について述べる．

2.1 レンダリング

2.1.1 3 次元シーンのモデリング

3 次元シーンをレンダリングするためには，まず，レンダリング対象である 3 次元シーンをモデリングする必要がある．モデリングでは，3 次元シーンに含まれるオブジェクトの形状や材質，視点，光源等を設定する．図 2.1 にモデリングされた 3 次元シーンの例を示す．

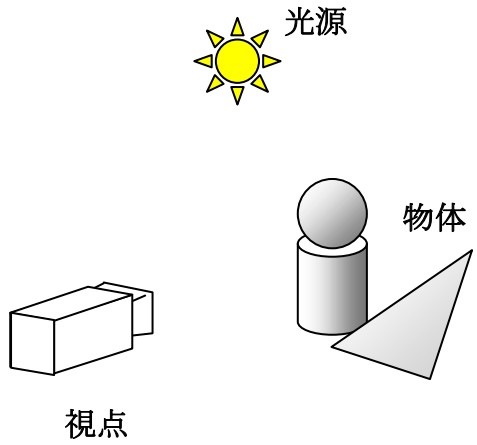


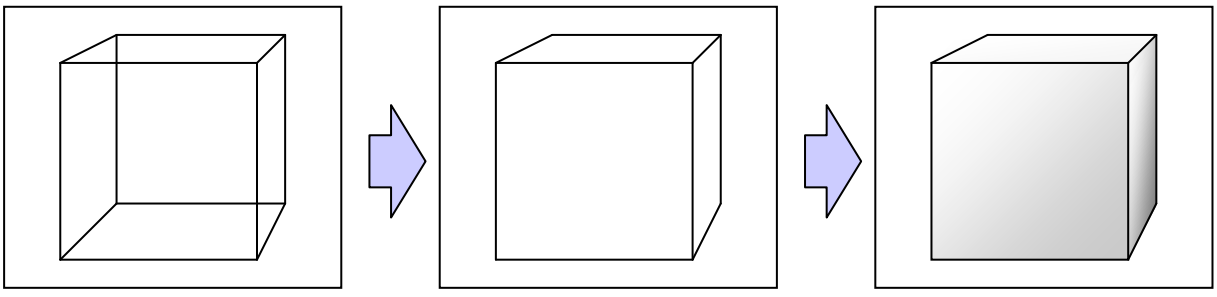
図 2.1　モデリングされた 3 次元シーン

一般的に物体（モデル）の形状は，多角形，球，直方体，曲面などの基本となるプリミティブ(primitive: 基本形状)と呼ばれる形状の変形・移動等による組み合わせで表現される．この中で，多角形は最も単純な形状であり，他のプリミティブは複数の多角形の集合として近似的に表すことができる．このような多角形の集合で表される形状はメッシュと呼ばれる．メッシュには任意の多角形を使うことができるが，中でも三角形を用いることが多い．三角形は面を表す最も単純な形状であり，同一面上に頂点が存在することが保証されるからである．また，レンダリングなどでの処理も他のプリミティブに比べ処理時間が少なくて済む．本論文では形状の表現として三角形ベースのメッシュを用いる．

2.1.2 レンダリングの流れ

モデリングによって構築された 3 次元シーンは，レンダリング処理を経て，2 次元画像として生成される．2 次元画像は複数のピクセルの集合で構成され，レンダリングは 2 次元画像を構成する全ピクセルに対して色を求める処理である．

現在の一般的なレンダリングにおける処理の流れを図 2.2 に示す．まず，画像を構成する各ピクセルに対応する面を求めるため, 透視投影という処理を行う．このとき，視点に対して最も手前の面をレンダリング対象とする隠面消去を行う．そして，各画素における色をシェーディングによる輝度計算で求める．



(a) 透視投影　　(b) 隠面消去　　(c) シェーディング

図 2.2　レンダリングの流れ

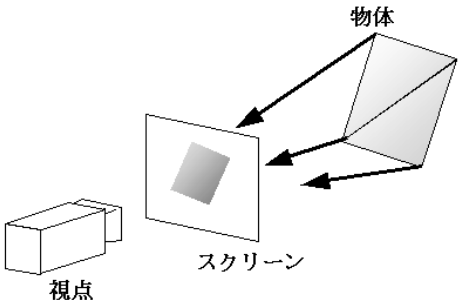


図 2.3　ラスタライゼーション

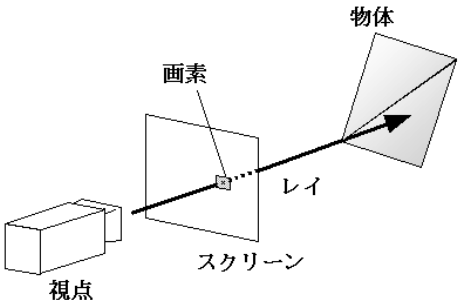


図 2.4　レイトレーシング

第 1 章で述べた通り，3 次元シーンのレンダリング手法はラスタライゼーションとレイトレーシングの 2 つのアプローチに大別することができる．これらの手法の大きな違いは，投影と隠面消去のアルゴリズムである．図 2.3 及び図 2.4 に，それぞれラスタライゼーション及びレイトレーシングにおけるレンダリングの概念図を示す．ここで，各図にあるスクリーンとは，レンダリング結果を出力する 2 次元画像に対応する 2 次元平面であり，その位置や解像度等は視点の持つ情報によって与えられる．

ラスタライゼーションは，図 2.3 に示すように，各三角形をスクリーンへ投影することで，三角形に対応する複数のピクセルを求め，Z バッファなどのアルゴリズムを用いて隠面消去を行う．

レイトレーシングは，図 2.4 に示すように，視点からスクリーン上の各ピクセルを通るレイと呼ばれる光線を飛ばし，その光線が交差する三角形のうち最も視点に近いものを求める．

各ピクセルにおいて対応するプリミティブを求めたら，シェーディングによってピクセルの色を計算する．シェーディングでは，光源の位置や性質，三角形面に設定された材質（色，反射特性など）やその方向，視点などの相互関係から対象の輝度（ピクセルの RGB 値）を求める．

ラスタライゼーションは，その処理を三角形単位で行うため，1 つの三角形の処理中に対象以外の三角形の情報を参照することはできない．そのため，影や反射・屈折のような他の物体の情報が必要になる処理を行うには，複雑な手法を組み合わせる必要がある．一方，レイトレーシングでは処理時に全ての三角形や光源情報を参照することができるため，影や反射・屈折といった効果を比較的簡単に実現することができ，写実性の高いレンダリングが可能となる．

2.2 レイトレーシングによるレンダリング

2.2.1 処理の流れ

レイトレーシングにおける処理の流れを図 2.5 に示す.

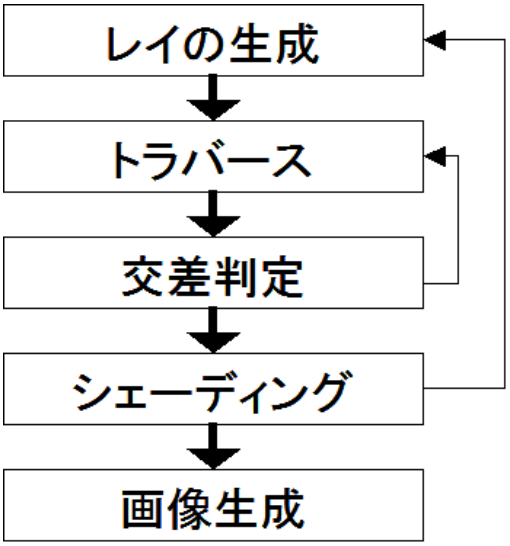


図 2.5　レイトレーシング処理の流れ

まず, 視点からスクリーン上の全ピクセル方向へ放射状にレイを生成する. そして, レイ毎に空間データ構造をトラバースし，交差する三角形を絞り込む．この処理については第 3 章にて詳述する．トラバースと交差判定を繰り返し，レイが最短距離で交差する三角形を導出する．そして，交差した三角形にある材質や光源からシェーディング処理を行う．シェーディングにおいて，反射や屈折が発生した場合，必要に応じて新たに 2 次レイを生成する. 全ピクセルに対応したレイにこれらの処理を行うことで，最終的な画像を生成する.

2.2.2　レイシューティング

視点からスクリーン上の各ピクセルへレイを生成する．この 3 次元シーンに対して視点からレイを飛ばす処理を，レイシューティングと呼ぶ.

レイは式(2.1)で表される線分，または半直線である．

$$R(t) = \mathbf{O} + t\mathbf{D} \quad (2.1)$$

ここで，$\mathbf{O}$ はレイの始点を表す位置ベクトル，$\mathbf{D}$ はレイの方向を表す方向ベクトルである．t はレイ上の点を示すパラメータである．1 次レイにおいては，$\mathbf{O}$ は始点位置，$\mathbf{D}$ はスクリーン上の各ピクセル内の点となる．

三角形とレイとの交差判定には，重心座標系(Barycentric Coordinates)と呼ばれる座標系への変換を用いた手法が広く知られている[MT97]．

三角形の頂点をそれぞれ $\mathbf{V}_0,\mathbf{V}_1,\mathbf{V}_2$ としたとき，重心座標系における点 $T(u,v)$は式(2.2)で表すことができる．

$$\mathbf{T}(u,v) = (1-u-v)\mathbf{V}_0 + u\mathbf{V}_1 + v\mathbf{V}_2 \quad (2.2)$$

$T(u,v)$で表される点は，三角形の存在する面上の点となる．レイとこの面との交点の t は式(2.3)で表される．

$$\mathbf{O} + t\mathbf{D} = (1-u-v)\mathbf{V}_0 + u\mathbf{V}_1 + v\mathbf{V}_2 \quad (2.3)$$

この式を変形させると，式(2.4)が求まる．

$$\begin{bmatrix} -D & V_1 - V_0 & V_2 - V_0 \end{bmatrix} \begin{bmatrix} t \\ u \\ v \end{bmatrix} = \begin{bmatrix} O - V_0 \end{bmatrix} \quad (2.4)$$

この線型方程式を解く事で，レイと三角形面の交点を求めることができる．また，交点が三角形内に存在するのは $u \le 0,\ v \le 0,\ u + v \le 1$ を満たすときとなる．

視点に最も近い三角形は，上記を満たす内で t が最も小さくなる三角形である．

2.2.3 シェーディング

2.2.2 項で求められたレイと三角形との交点に対してシェーディングを行うことで，レイに対応したピクセルの色を求めることができる．この色は，三角形を表す物体に設定された特性と光源によって決まる．

更に, レイシューティングと同様の方法で, 他の物体による影を得ることができる. シェーディングで用いる交点から光源方向へのレイが他の物体と交差するとき, つまり光源から交点への光が他の物体によって遮られるとき, その交点には影が発生する. この, 光源方向へのレイはシャドウレイと呼ばれる.

2.2.4　2 次レイの発生

2 次元画像レンダリングのためのレイシューティングアルゴリズムによる手法は, 1968 年 Appel によって, 元々は隠面消去アルゴリズムとして提唱された[Appel86]. 現在の, 反射・屈折による他の面からの間接光を考慮した写実的なレンダリングを得意とするレイトレーシングのレンダリングアルゴリズムは, 1980 年に Whitted によって確立された[Whitted80].

シェーディングにおいて, その面で反射・屈折が発生する場合, 図 2.6 に示すように, 2 次レイと呼ばれるそれぞれに対応した反射レイ・屈折レイを発生させる. そして, その先におけるシェーディングで求められた色を加算することで, 反射・屈折といった効果を得ることができる.

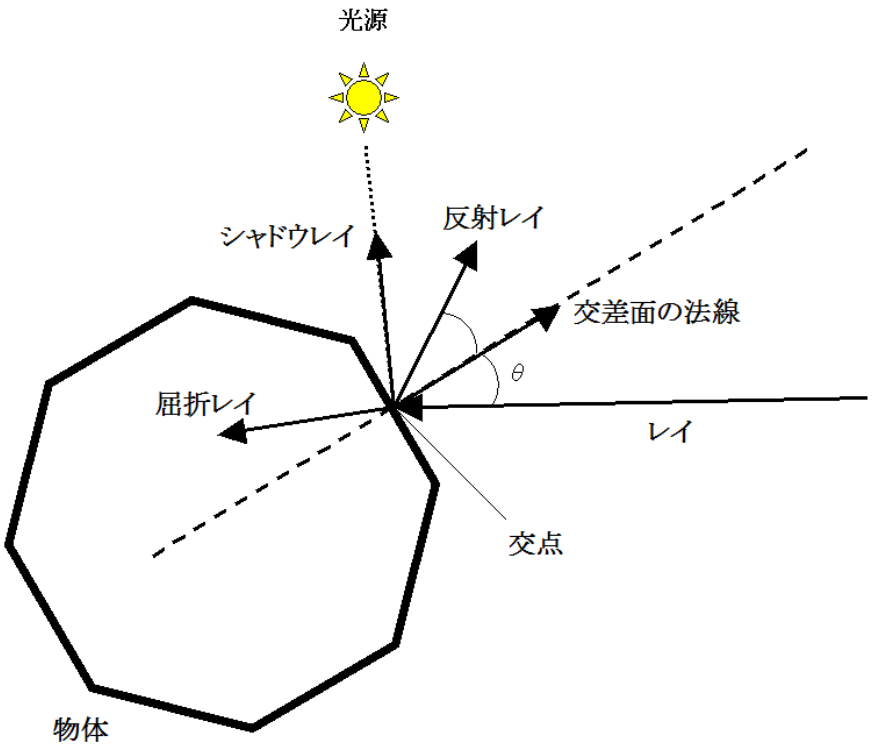


図 2.6　2 次レイの発生

Whitted のレンダリングアルゴリズムにより，より写実的なレンダリングを行うことができるようになった．しかし，2 次レイは指数的に増えていくため計算量が増えてしまう．1984 年には，Cook らによって分散レイトレーシングが提案され，被写界深度やモーションブラー，大域照明の表現が可能となった[CPC84]．しかし，ここでも 2 次レイの数が増加することとなった．

基本的にレイの数を増やせばレンダリング品質は上がるが，一方でその計算量は膨大になる．そのため，レイトレーシングの処理量を削減するために様々な手法が考案されてきた．

2.3 レイトレーシングの高速化

2.3.1 空間データ構造による高速化

レイトレーシングの高速化アプローチには様々なものがある．Wald の博士論文[Wald04]によれば，特に空間データ構造の改善による高速化が最も重要であると述べられている．

空間データ構造とは，3 次元シーンの存在する空間を分割し，階層的に表したデータ構造で，幾何計算における計算量の効率化手法として広く知られている．

シーンに存在する N 個の三角形に対して，総当りで交差判定を行おうとすると，その計算量は O(N)となる．そこで，空間データ構造を用いると，計算量をおよそ O(NlogN)にまで抑えることができる．

空間データ構造の構築には数多くのアルゴリズムが提案されている．Havran は各種空間データ構造を用いたレイとプリミティブとの交差判定を”ray shooting algorithms: RSAs” と呼び，彼の博士論文で kd-tree, octree, BSP, grid, BVH などの 12 個のアルゴリズムを比較している[Havran01]．結論としては，kd-tree が平均的に効果が高いと述べられている．しかし同時に，どのようなシーンに対しても最適なアルゴリズムは存在しないということも述べられている．このように，空間データ構造によるレイトレーシングの高速化の研究は，現在注目を集める領域である．近年では，BVH が動的シーンに対して（制約条件はあるが）有効であるという論文も発表されている[WBS07][Wald07]．

空間データ構造のトラバースには，レイのコヒーレンスを利用したパケットトラバースが有効である．ここで，コヒーレンスとはレイの類似性を表したものである．また，パケットとはコヒーレンスの高い複数のレイの集合である．コヒーレンスの高いレイ同士は，同じ AABB 或いは三角形と交差する確率が高い．この性質を利用してトラバースにおける計算量を減らすのがパケットトラバースである．パケットトラバースの詳細については第 3 章で解説する．

2.3.2 ハードウェアへの対応

近年では，ハードウェアへの対応によるレイトレーシングの高速化に関する研究も盛んに行われている．並列計算機能への適応や，レイトレーシング専用ハードウェアの製作[WSS05]といった取り組みがこの研究分野の成果である．

1990 年代後半から，パーソナルコンピュータにおける CPU にマルチメディア演算用の演算命令が搭載されるようになってきた．例として，Intel の CPU における，MMX, SSE, SSE2, SSE3, AMD の CPU における 3DNow!, IBM/Motorola の AltiVec などといった拡張命令が挙げられる．このような拡張命令セットでは，3D コンピュータグラフィックスを含む，複数の浮動小数点に同様の演算を行うようなアプリケーション向けの並列演算機能が提供されている．例えば，SSE では 4 つの 32 ビット浮動小数点数に対して，同時に演算を行うことができる．近年のレイトレーシングの高速化を目的とした研究では，こうした SIMD 演算機能を利用するものが多い．

GPU（Graphics Processing Unit）は，3DCG のレンダリング専用に設計されたハードウェアである．GPU では頂点やピクセルに対する演算を並列かつ高速に実現する．そのため，例えばデータ間の依存性が無い大量のデータを並列処理するような場面では CPU よりも高速に処理を行うことが可能である．近年では，このような GPU の特徴を利用して GPU を汎用計算機として用いる，GPGPU（General Purpose computation on GPU）と呼ばれる分野の研究が盛んである[宮田 04][新庄 07]. また，近年では gird, kd-tree 及び BVH といった各種空間データ構造を用いたレイトレーシングが GPU 上で実装されている[GPSS07][PGSS07].

また，近年の CPU の動向としてマルチコア CPU の登場が注目されている．従来までは，動作周波数を引き上げることが CPU の性能指針とされ，CPU ベンダは競ってその数値を高めていった．しかし，それに伴う消費電力の増大や集積限界のため，動

作周波数における進化は頭打ちとなってきた．そこで，CPU 性能を向上させる新たな技術として，複数の CPU コアを 1 つの CPU に集積したマルチコア CPU が登場した．CPU コアとは，従来の CPU に相当するもので，CPU 内における演算カイロ単体をさすこともあれば，それを含めたキャッシュも含む場合もある．CPU に内包されるコア数が 2 つならばデュアルコア，4 つならばクアッドコアと呼ばれることもあり，それらを総称してマルチコアと呼ばれる．2005 年に初めてパーソナルユースのマルチコア CPU として発売された Athlon64 X2 を筆頭に，CPU ベンダ最大手の Intel からは Core2Duo，Core2Quad 等，現在のハイエンド PC の多くにマルチコア CPU が搭載されている．また，ソニー・東芝・IBM の三社が共同で開発している Cell など，PC 向け以外での用途のマルチコア CPU も登場している．Benthin らは Cell 上でのインタラクティブなレイトレーシングを実装している[BWSF06]．また，Ize らは動的シーンに対応した BVH のインタラクティブな非同期再構築を，マルチコア環境で実装している[IWP07].

第　3　章
バウンディングボリューム階層構造を用いたレイトレーシング

本章では，空間データ構造の一つである BVH，及びそれによるレイトレーシングの高速化について解説する．そして，レイのコヒーレンスを用いてトラバースを高速化するパケットトラバースについて解説し，その効果及び問題点を実験に基づいて提示する．そして，問題点の理由を考察する．

3.1 バウンディングボリューム階層（BVH）

BVH とは，プリミティブを完全に内包するバウンディングボリュームの階層的な構造である．BVH は（主に 2 分木の）木構造で表現される．バウンディングボリュームの種類としては，球や直方体などがあるが，レイトレーシング用途でのバウンディングボリュームはレイとの交差判定処理負荷の低いものが適している．本論文では，一般的に利用されている Axis-Aligned Bounding Box（AABB: 軸並行境界ボックス）を用いる．図 3.1 に 2 次元での AABB を示す．3 次元空間においては，三角形及び AABB ともに奥行きを持つ．

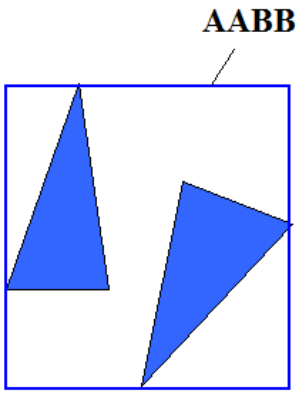


図 3.1　三角形を内包する AABB

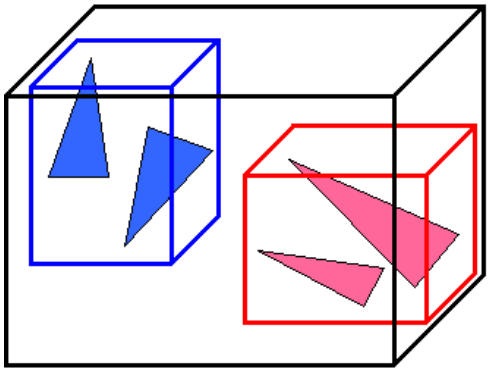


図 3.2　AABB の階層構造

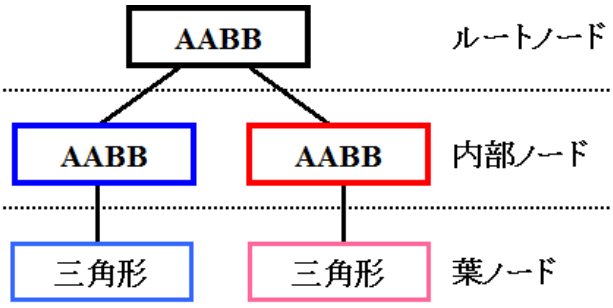


図 3.3　図 3.2 の BVH における木構造表現

図 3.2 に AABB の階層構造を示す．図 3.2 の 3 次元シーンには，4 つの三角形と，3 つの AABB が存在する．三角形は青と赤の三角形が 2 枚ずつ 2 グループに分けられ，それぞれ青と赤の AABB に囲まれる．更に，黒い AABB は，それら 2 つの AABB を取り囲む．このように，三角形を内包する AABB を複数取り囲む AABB，それを更に複数囲む AABB, といったバウンディングボリュームの階層構造を BVH という．

図 3.2 の BVH を木構造で表現すると図 3.3 のようになる．各ノードは同色の AABB に対応している．BVH においては，ルートノードは全ての AABB 及び三角形を完全に内包する．内部ノードは子の AABB を完全に内包する．そして，葉ノードは，1 つ以上の三角形のリストを持っている．

AABB は，レイと交差する三角形の刈り込みを効果的に実現するため，内包するプリミティブをできる限りタイトに囲み，その表面積を最小となるように構築する．そのため，三角形を内包する AABB の各面は三角形のいずれかの頂点に接している．AABB を囲む親ノードの AABB も，同様に子ノードの面に接する．

BVH 木構造の各ノードは 2 つ以上の子ノードを持つことができる．レイトレーシングにおいては，分岐数が増えると内部ノードが減るためメモリ使用量を抑えることができる反面，レンダリング効率は下がってしまう傾向がある [木村 07][CSE06]．

本論文では，最もレンダリング効率の良いとされる 2 分木の BVH を用いる．

3.2 BVH に対するトラバース

第 2 章で概説したとおり，BVH 等の空間データ構造によってレイトレーシングは高速化される．

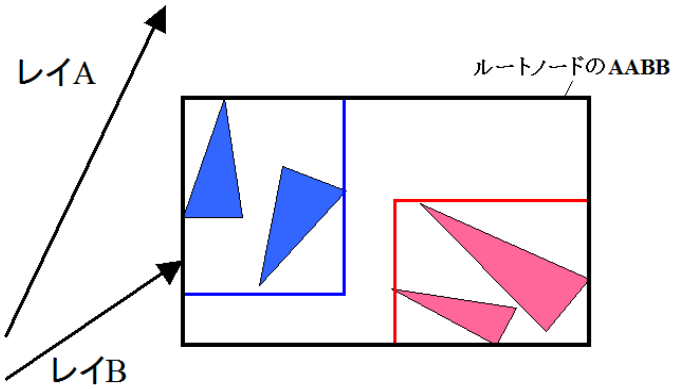


図 3.4　ルートノードの AABB とレイとの交差判定

BVH を用いてレイと交差する三角形を求める処理は，図 3.4 のようにルートノードの AABB との交差判定から始める．図 3.4 のレイ A のように，ルートノードの AABB とレイとが交差しない場合は，そのレイは 3 次元シーン内の三角形と交差することはない．図 3.4 のレイ B のように，ルートノードの AABB と交差する場合は三角形と交差する可能性があるため，子ノードの AABB との交差判定を行う．そして，レイと AABB との交差判定を行いながら葉ノードへ向かって交差する可能性のある三角形を刈り込み，葉ノードに達したときその葉ノードが持つ三角形リストにある三角形

との交差判定を行う．このように，ルートノードから葉ノードへと BVH を辿る一連の処理をトラバース，または探索という．図 3.5 に，図 3.4 のレイ B が葉ノードへ至るまでのトラバース過程のイメージを示す．

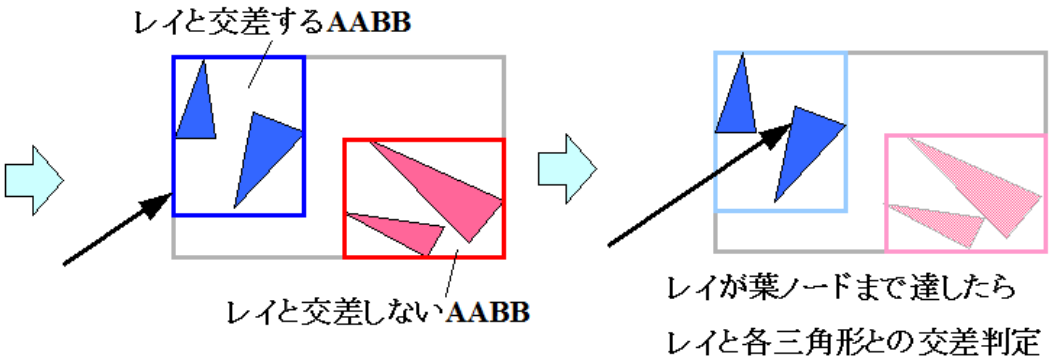


図 3.5　BVH のトラバース過程

BVH を用いたトラバースを行うことで，総当りでは O(N)であった処理量を O(NlogN)まで減らすことができる．

3.3 BVH の構築

本論文では，BVH における構築に，SAH（Surface Area Heuristics）と呼ばれるコスト関数に基づいた構築手法を採用する．

レイと AABB の交差する確率は，AABB の表面積にほぼ比例する．SAH はその原理に基づいて，構築される BVH における AABB の表面積の総和がなるべく小さくなるように BVH を構築する手法である．Wald らは，BVH の構築に式(3.1)で表される SAH を適用し，高速なレイトレーシングを可能とした[WBS07]．

$$T = 2T_{AABB} + \frac{A(S_1)}{A(S)} N(S_1) T_{tri} + \frac{A(S_2)}{A(S)} N(S_2) T_{tri} \qquad (3.1)$$

ここで，S は分割されるノードに含まれる三角形の集合，S_1 及び S_2 は分割後の 2 つの子ノードに含まれる三角形の集合である．A(S)及び N(S)は，それぞれ S を囲む AABB の表面積，及び S に含まれる三角形の数を表す．

また，T_{AABB} はレイと AABB との交差判定に掛かる時間，T_{tri} はレイと三角形との

交差判定に掛かる時間を表す．Reshtov らは T_{AABB}: T_{tri} = 2 : 1 のコストモデルを提案している[RSH05]．本論文では，これに習い T_{AABB} = 2.0, T_{tri} = 1.0 として SAH の構築を行う．

SAH による BVH の構築は，全ての三角形を内包するルートノードから，トップダウンで行われる．まず 3 次元の XYZ 軸各々でソートした三角形群を軸に垂直な面によって 2 つのグループに分ける．この全ての分割面において SAH を適用して求められたコストが最小となる分割面で分けられる 2 つの三角形グループをそれぞれ子ノードの三角形とし再帰的に構築を行う．もし，全てのコストが葉ノードを構築するコストである $T_{tri} \times N(S)$ より大きいならば，そのノードを葉ノードとして構築する．

図 3.6 に，SAH の構築を行う擬似コードを示す．

```
S を三角形リストに持つ葉ノードを作る場合のコストを計算し初期値とする
for each 軸 {
    軸に沿って三角形をソート
    for each 分割面 {
        S に含まれる三角形を分割面で S1 と S2 のグループに分ける
        SAH によってコストを求める
        コストが最小の時，その分割面とコストを記憶する
    }
}

if ( 葉ノードを作るより最適な分割が存在しない ) {
    S を三角形リストに持つ葉ノードを作成
} else {
    分割面で三角形を S1 と S2 のグループに分け，
    それぞれを子ノードとして再帰的に構築
}
```

図 3.6. SAH を用いた BVH の構築

3.4 BVH に対するパケットトラバース

3.4.1 レイのコヒーレンス

パケットトラバースとは，複数のレイの集合をレイパケットとし，パケット単位で BVH のトラバース処理を行う手法である．

レイ同士にはコヒーレンスと呼ばれる相関関係がある．コヒーレンスとは，直訳では可干渉性と訳される．光学の分野においては光波の干渉性を表すものとして用いられている．レイトレーシングにおいては，レイ同士の類似性を表すものとして用いられる．また，コヒーレンスの高いレイ同士をまとめてコヒーレントなレイ，コヒーレンスの低いレイ同士をインコヒーレントなレイと呼ぶ．

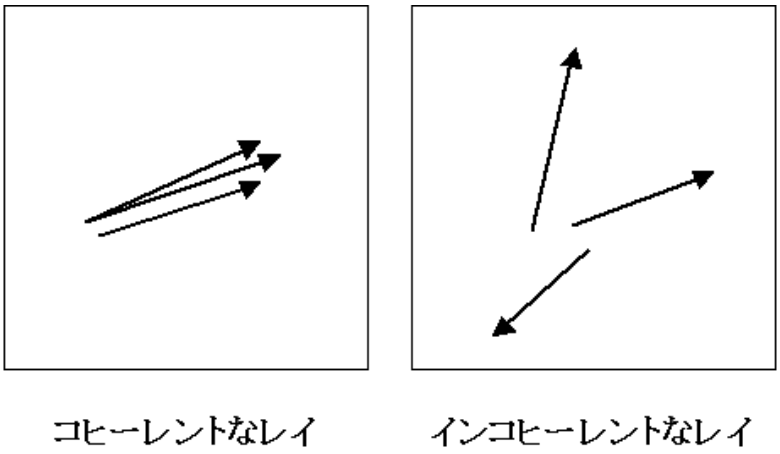


図 3.7　コヒーレント及びインコヒーレントなレイ

コヒーレントなレイは，同じ AABB 及び三角形に交差する可能性が高いといえる．パケットトラバースは，コヒーレントなレイをレイパケットとしてまとめてトラバースすることで，処理負荷を減らすという手法である．

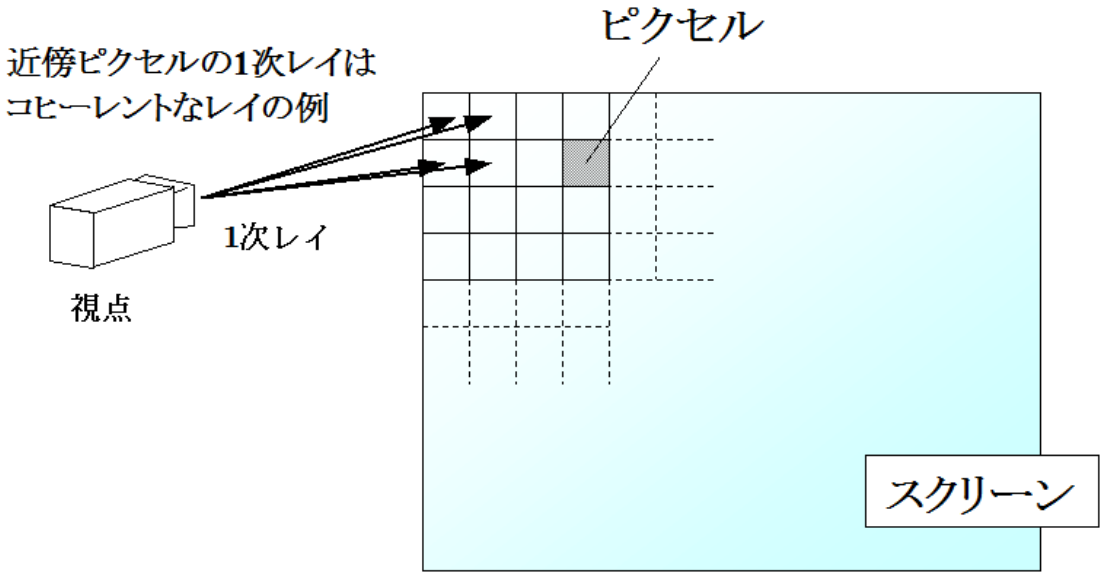


図 3.8　コヒーレントな 1 次レイ

1 次レイにおいては，図 3.8 に示すように近傍ピクセルを通るレイ同士はコヒーレンスが高い. パケットトラバースではこのような近傍ピクセルのコヒーレントな 1 次レイを一定ピクセルごとに区切りそれを 1 つのレイパケットとする. この区切るピクセルの縦横の幅をパケットサイズと呼ぶ. パケットサイズには，2x2，4x4，8x8 など，2 のべき乗数がよく用いられる.

シェーディングによって発生する 2 次レイ以降については，コヒーレンスが乱れることが多く, その扱いが困難であるため, 本論文では考慮しないこととする. よって，以降本論文で扱うレイパケットは，全て 1 次レイのレイパケットであるとする.

パケットトラバースは，Wald らによって BVH に適用された[WBS07]. 次項から，Wald らのパケットトラバースにおけるアルゴリズムの要点を示す.

3.4.2 Early hit test

パケット内に含まれるいずれかのレイが AABB とヒットした場合，残りのレイと AABB との交差判定をスキップして，子ノードのトラバースへ進む．これは，コヒーレンスの高いレイパケットに含まれるレイ同士は，同じ AABB と交差する確率が高いという性質を利用したものである．これにより，パケット内の残りのレイと AABB との交差判定処理を省くことができる．

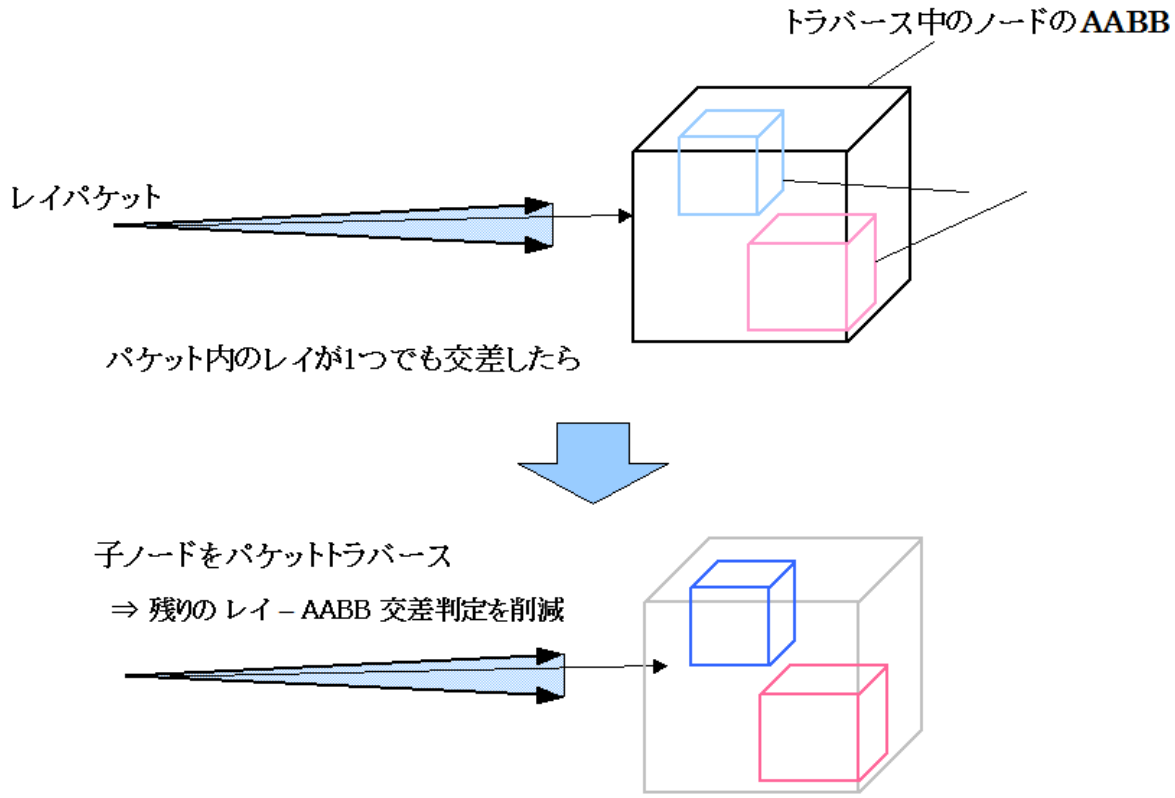


図 3.9　Early hit test

3.4.3 Early miss exit

early hit test はトラバースの効率を非常に高める手法である．更に，効率を高めるために，パケット内の全てのレイが交差しないケースを，Interval Arithmetic による錐台-AABB 交差判定[BWS06]を用いて判定する．ここで，錐台は完全にパケット内のレイを全て内包している．この判定では，AABB とパケット内のレイが交差しないほとんどのケースを除外することができ，パケット内のレイを 1 つずつ交差判定する処理を省くことができる．

3.4.4 Packet test of last resort

Early miss exit でノードの AABB と錐台が交差したときは，いずれかのレイが AABB と交差するまで全てのレイについて AABB との交差判定を行う．もし，Early miss exit で錐台と AABB とが交差していても，全てのレイが AABB と交差しないケースも存在するため，最終的には 1 本ずつ判定を行わなければならない．

3.4.5 First hit active ray tracking

トラバースが内部ノードに達したとき，そのノードの AABB に交差しなかったレイは，その子ノードから葉ノードまでの AABB にも交差することは無い．よって，子ノードのトラバース中にそのレイは交差判定をする必要が無い．そのため，パケットに含まれるレイを，交差しなかったレイ，トラバース中のレイ，未テストのレイと分けることで，同じレイを交差判定する不要な交差判定を除くことができる．

パケット内のレイのトラバースは，順番に行うため，この分別はトラバース中のレイを識別するインデックスを記憶しておくだけでよい．

3.4.6 パケットトラバース

以上の要素の他に，Ordered Traversal，SIMD 実装等の最適化があるが，パケットトラバースのコンセプトに深く関与しないと判断し，ここでは説明を省く．

図 3.10 にパケットトラバースの処理を表す擬似コードを示す．

```
if ( トラバース中のレイと AABB とが交差する ) {    // active ray tracking
    交差したレイで子ノードをトラバース
} else if ( レイ錐体と AABB が交差しない ) {    // early miss exit
    ノードのトラバース終了
} else {    // last resort
    for each パケット中の未テストのレイ {
        if ( レイと AABB が交差 ) {
                交差したレイで子ノードをトラバース
        }
    }
    // 交差するレイが見つからなかった
    ノードのトラバース終了
}
```

図 3.10　パケットトラバースアルゴリズム

3.5 パケットトラバースによる効果の検証

3.5.1 実験

ここでは，3.4 節で解説したパケットトラバースを実装し，その効果について検証する．実験環境は，Windows Vista Business, Intel(R) Core(TM)2 Duo 6600 @ 2.4GHz x2 (2 コア), システムメモリ 2GB の PC にて, コンパイラには Visual C++ 2005 を用いて行った．レンダリングの出力画像は横 1024 ピクセル，縦 768 ピクセルとした．シェーディングについては，面の法線ベクトルの XYZ 値をそれぞれ RGB の輝度とする．

図 3.11～3.16，及び表 3.1 に実験で用いたレンダリング対象の一覧を示す．これらレンダリング対象モデルのうち，tree10, balls4A, balls4B, balls4C はレイトレーシングの性能評価に用いられる SPD（Standard Procedural Database）[Haines87]によって生成したものを用いた．各メッシュ名の後の数字は，物体の生成時に使用したサイズファクタである．また，balls4A, balls4B, balls4C は共通のメッシュを用い，視点位置のみを変化させている．それぞれ対象モデルから視点までの距離が，中距離（balls4A），近距離（balls4B），遠距離（balls4C） の 3 次元シーンを表す．

表 3.1　レンダリング対象一覧

レンダリング対象	図番号	頂点数	三角形数	BVH 平均深さ
bunny	図 3.11	556025	69451	14.9
castle	図 3.12	27927	33276	12.4
tree10	図 3.13	810616	270206	9.9
balls4A	図 3.14	2391448	797150	10.7
balls4B	図 3.15	2391448	797150	10.7
balls4C	図 3.16	2391448	797150	10.7

図 3.11　bunny

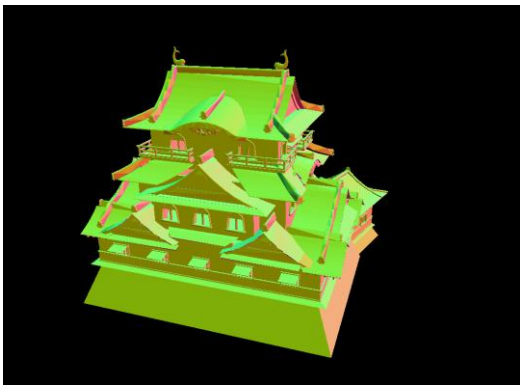


図 3.12　castle

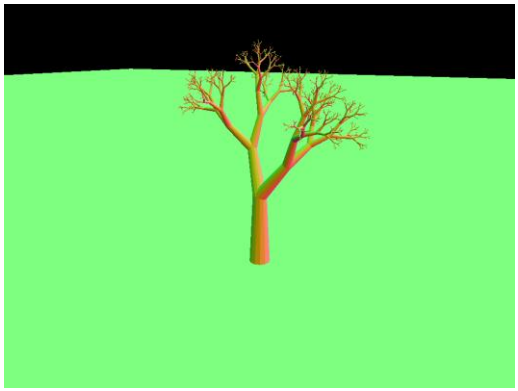


図 3.13　tree10

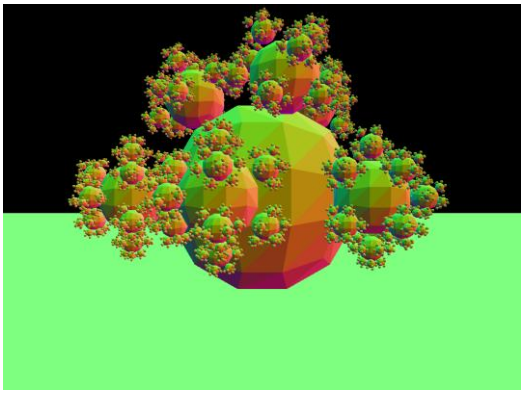


図 3.14　balls4A

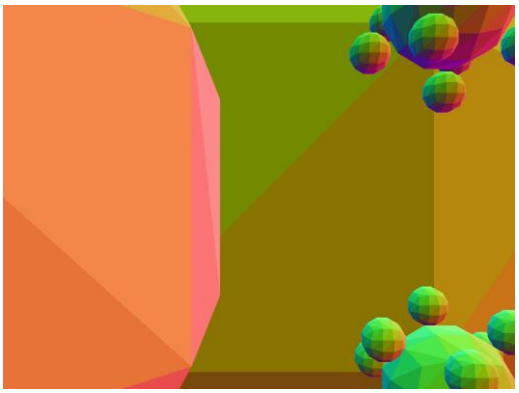


図 3.15　balls4B

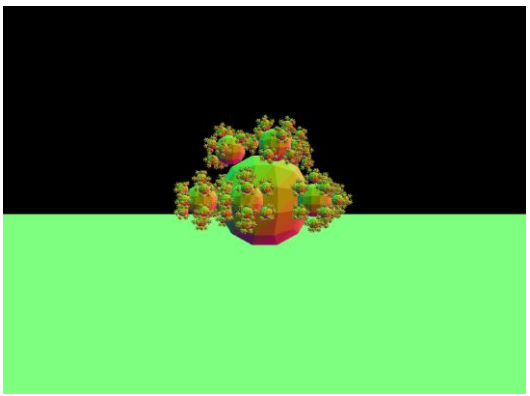


図 3.16　balls4C

3.5.2 実験結果

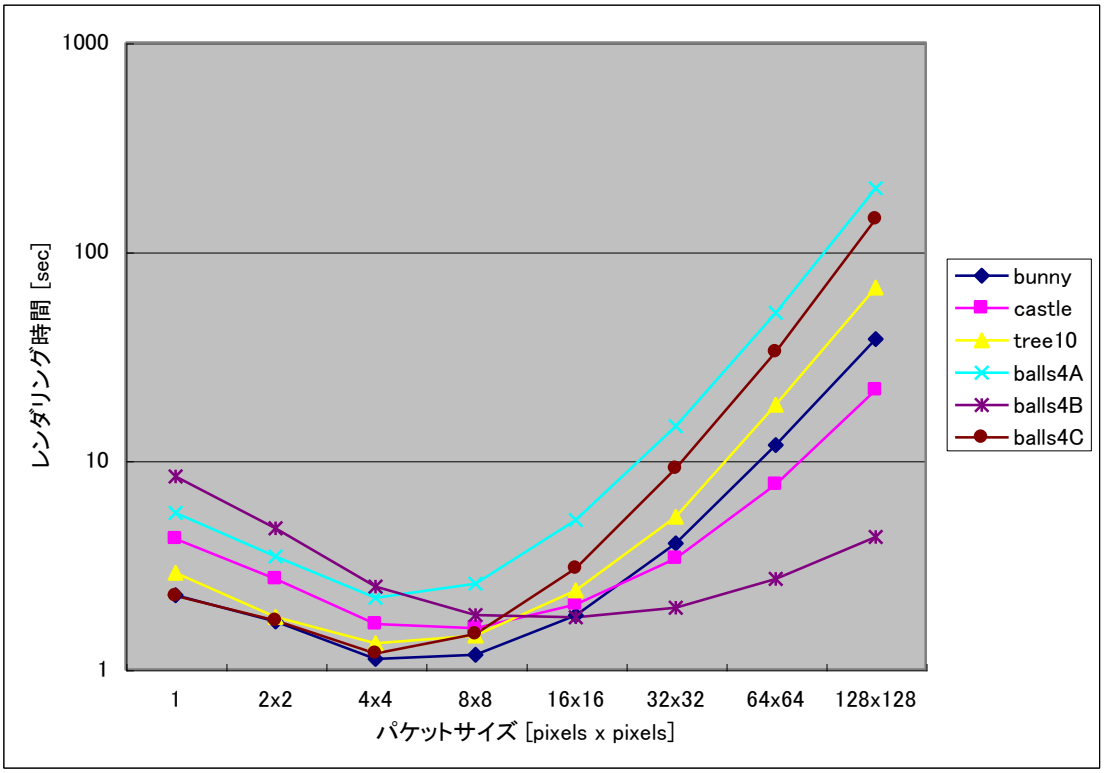


図 3.17　パケットサイズによるレンダリング時間の変化

図 3.17 にパケットサイズによるレンダリング時間の変化を示す. パケットサイズが 1 の数値は，通常のレイ毎のトラバースを行った.

この結果より，パケットサイズが大きくなる程レンダリング時間が指数的に増えていることが分かる．また，レンダリング対象によってパケットトラバースの効果の高いパケットサイズは異なっている．例えば，bunny においては 4x4 のパケットサイズにおいて最も効果的であるのに対し，balls4B では 16x16 のパケットサイズが最も効果が高くなっている.

3.6 パケットトラバースの考察

パケットトラバースによりトラバースの効率は大きく向上する．しかし，その効果はパケット内に含まれているレイ数に大きく依存する．Wald らの結果では，8x8 もしくは 16x16 のパケットサイズが幅広いシーンにおいて効果的であると結論付け，しばしば 8x8 を用いるとした．

パケットトラバースの効果はシーンに依存している．厳密に言えばシーンから構築された BVH の AABB の大きさに依存する．これは，レイのコヒーレンスの持つ相対的な性質による．図 3.18 に示すように，同じサイズのレイパケットであっても，トラバース対象の AABB のサイズや，視点と AABB との距離よってそのコヒーレンスは相対的に変化する．

AABB に対してレイパケットが大きすぎる場合，つまりパケットのコヒーレンスが比較的低い場合は，3 次元シーン中の多くの AABB とレイパケットが交差してしまい，レイと三角形との交差判定が失敗するケースが多く発生してしまう．逆に，AABB に対してレイパケットが小さすぎる場合は，パケットのコヒーレンスは比較的高いが，1 次レイの数は一定であるためレイパケットの総数が増大してしまい，最適なパケットサイズによるパケットトラバースよりも，レイと AABB との交差判定が多く発生してしまう．

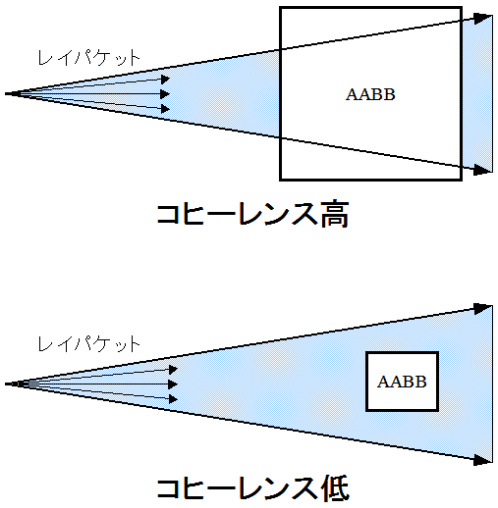


図 3.18　AABB のサイズによるレイパケットのコヒーレンスの違い

BVH の各ノードが保持する AABB は，図 3.19 に示すようにルートノードに近いほど大きく，葉ノードに近づくほど小さくなる．そのため，トラバースする階層が深くなる度に，レイのコヒーレンスは低くなっていく．また，1 次レイにおいては，視点からの距離が大きくなる程レイパケットのコヒーレンスは低くなる．また，AABB の大きさは一様ではなく，モデル形状に依存する．

このような要因から，最適なパケットサイズを一意に決めることは難しく，平均的に効果的である 8x8 や 16x16 といったパケットサイズが用いられる．

本論文では，このようなパケットトラバースの効果が 3 次元シーン及びモデルに依存することを解決し，それらに依存しないパケットトラバースのアルゴリズムを提案する．

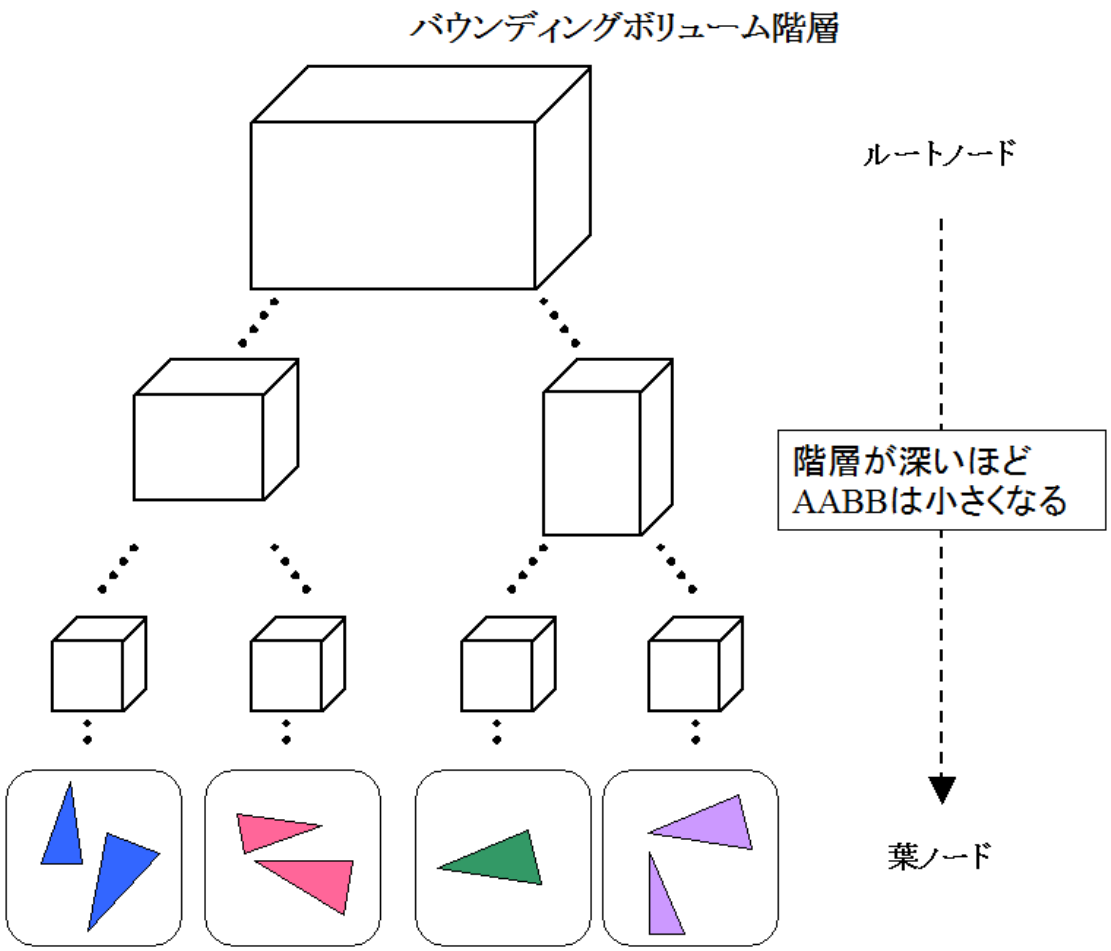


図 3.19　BVH 階層の深さと AABB サイズとの関係

第　4　章
モデル非依存パケットトラバース

本章では，第 3 章で示したパケットトラバースの問題点を解決するためのアルゴリズムの提案を行う．そして，その効果について実験を通じて検証し，考察する．

4.1 既存手法の問題点

第 3 章で述べたとおり，パケットトラバースの効果はレイパケットのサイズと 3 次元シーン及びモデルに大きく依存しており，パケットサイズによっては効果が下がってしまうことがある．本論文では，この問題を解決するための，パケットトラバースを改良した手法を提案する．

4.2 モデル非依存のパケットトラバース手法

4.2.1 レイパケットの階層構造

図 4.1 に示すように，4 本のレイを持つ 2x2 レイパケットは，パケットサイズが最も小さく，かつコヒーレンスの最も高いレイパケットである．そして，2 のべき乗をパケットサイズとするレイパケットは，この 2x2 レイパケットを最小単位として図 4.2 のような階層構造を成している．本論文では，パケットサイズが一回り大きいものを上位パケット，小さいものを下位パケットと呼ぶ．レイパケットは下位パケットを 4 つ内包していることになる．本論文では，このレイパケットの持つ階層構造を利用し，既存手法によるパケットトラバースで発生していた，深い階層を辿ることによるコヒーレンスの低下を抑えたパケットトラバースの改善手法を提案する．

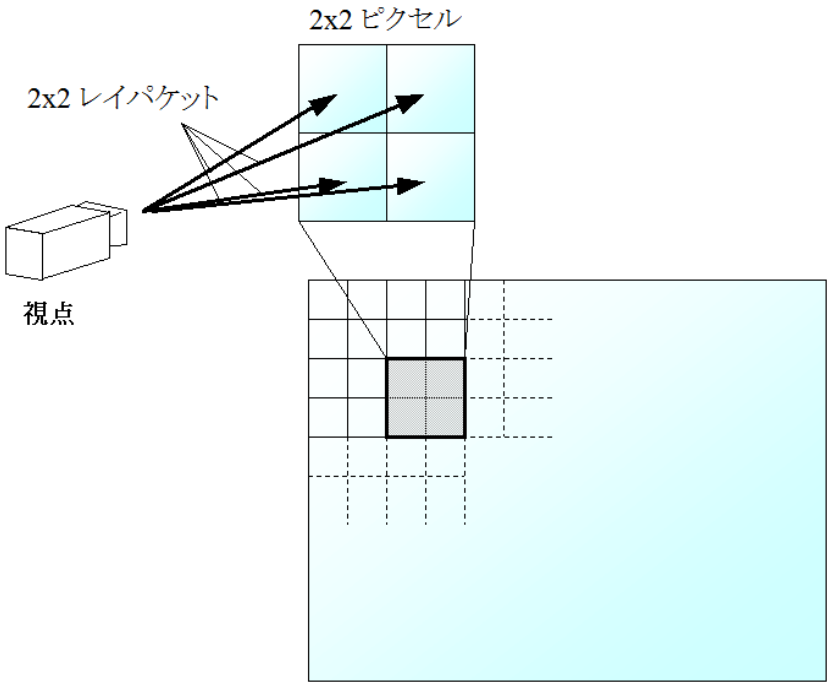


図 4.1　2x2 レイパケット

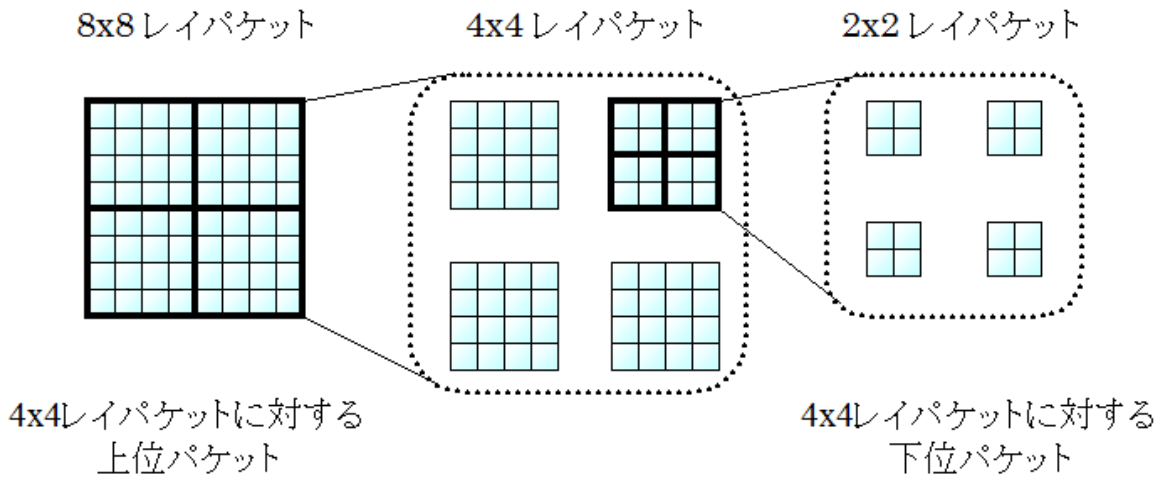


図 4.2　レイパケットの階層構造

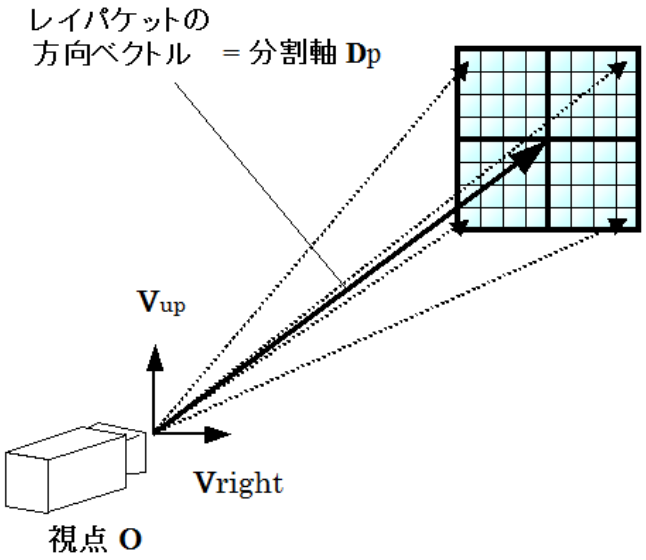


図 4.3　レイパケットの分割

図 4.3 に示すように，レイパケットは自身が含むレイの方向ベクトルの平均となる方向ベクトルと，視点に対して水平及び垂直方向のベクトルとで成る 2 つの面によって，4 つの下位パケットに分割される．この分割軸となるレイパケットの平均方向ベクトルを $\mathbf{Dp}$，視点の水平及び垂直方向を表す方向ベクトルを，それぞれ $\mathbf{V}_{up}$, 及び $\mathbf{V}_{right}$ とする．ここで $\mathbf{V}_{up}$ と $\mathbf{V}_{right}$ とは必ず直交している．また，視点の位置ベクトル（つまり 1 次レイ全ての始点）は $\mathbf{O}$ とする．

このとき，水平分割面は図 4.4 に示すように，位置ベクトル $\mathbf{O}$ と面の法線方向 $\mathbf{N}_v$ で定義される．そして，$\mathbf{N}_v$ はレイパケットの平均方向ベクトル $\mathbf{D}_p$ と視点の水平方向を表す $\mathbf{V}_{right}$ の外積 $\mathbf{N}_v = \mathbf{V}_{right} \times \mathbf{D}_p$ で求めることができる. 垂直分割面も同様に，位置ベクトル $\mathbf{O}$ と面の法線方向 $\mathbf{N}_h = \mathbf{V}_{up} \times \mathbf{D}_p$ で定義することができる.

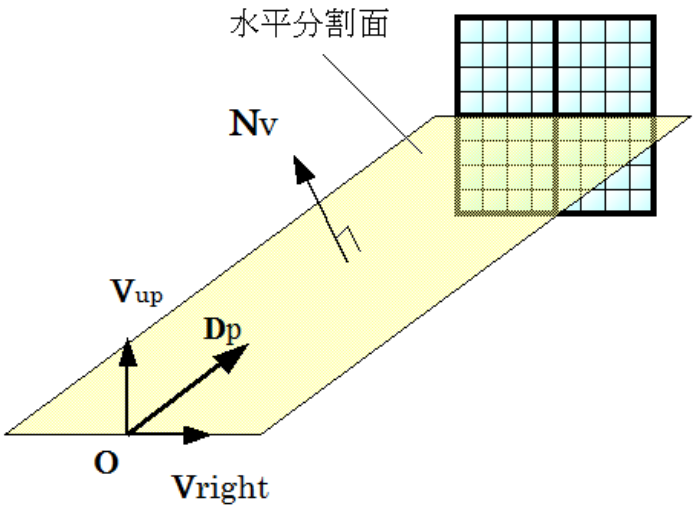


図 4.4　水平分割面の法線ベクトル

4.2.2　分割面と AABB との相対位置判定

レイパケットに定義された 2 つの分割面と AABB との位置判定は，それぞれの分割面において AABB を構成する 8 つの頂点のうち 2 頂点が分割面によって分離されるかを判定することで求めることができる．

ある頂点 **V** が与えられたとき，**V** が分割面の法線方向，またはその逆側にあるかの判定は，内積（**V**－**O**）・**N** の符号で判定することが出来る．この符号が正であるとき，**V** は分割面の法線方向側に位置し，負であるときその逆側に位置している．

この位置判定を，分割面の法線方向に沿って最も離れた AABB の頂点と，その対角線上にある頂点とで行い，符合が異なる場合，AABB は分割面と交差していることになる．符号が一致した場合は，AABB と分割面とは交差せず，分割面によって二分される空間のいずれかに位置している事になる．このように分割面と AABB とが取りうる位置関係は，それぞれの分割面において 3 通りずつある．直交する 2 つの分割面で見た場合は 9 通りあることになる．

4.2.3　パケットトラバース中のレイパケット分割

図 4.5 はトラバース過程のレイパケットとノードの AABB を，レイパケットの水平又は垂直方向から見た図である．このとき，ノードの AABB は分割面から見て完全に下側に位置している．よって，分割面より上側にある下位パケットと，このノードより下位ノードの持つ AABB とは交差することは無い．よって，この下位パケットはこの時点で切り捨てることができる．この下位パケットを切り捨てるパターンは，4.2.2 項で述べた 9 通りの分割面と AABB との位置関係から，切り捨てない 1 通りを差し引いた 8 通り存在する．図 4.6 に示すように，トラバース不要な下位パケットを切り捨て，AABB の存在する側の下位パケットのみで子ノードをトラバースすることで，効率的にトラバースすることが可能となる．この処理は大きなパケットサイズのレイパケットになるほど，効果が高いと考えられる．

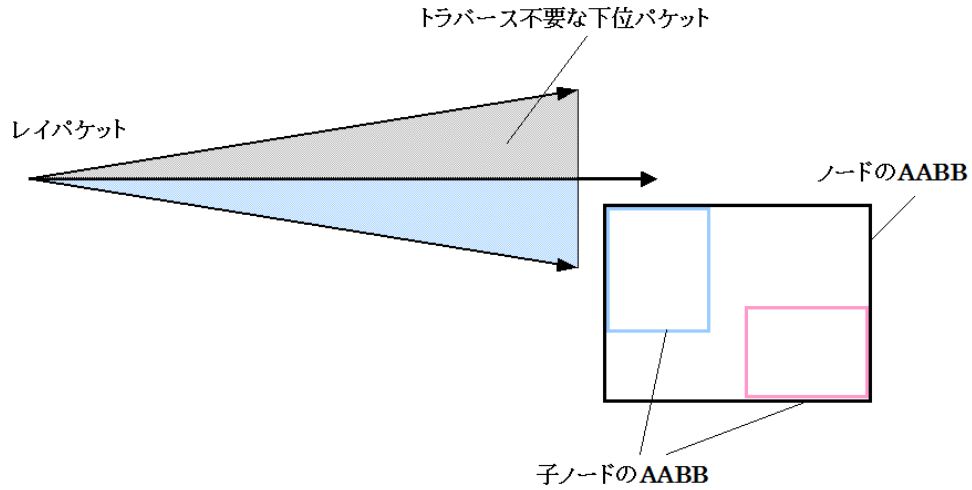


図 4.5　分割面判定によるレイパケットの切り捨て

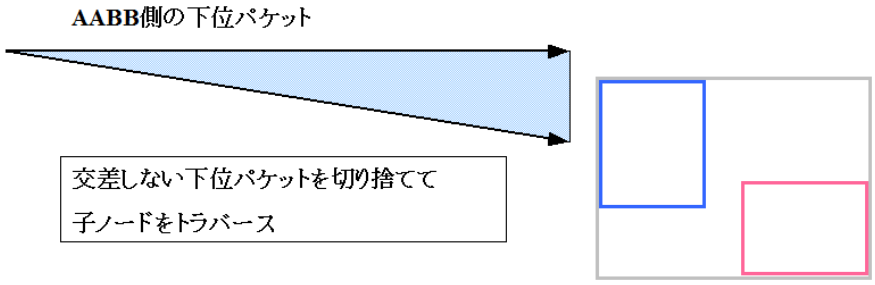


図 4.6 AABB 側の下位パケットによる子ノードのトラバース

4.2.4　既存手法への適応

図 4.7 にレイパケット分割を適応したパケットトラバースの擬似コードを示す.

```
if ( トラバース中のレイと AABB とが交差する ) {    // active ray tracking
    if (レイパケットが分割可能) {
        レイパケットを分割
    }
    交差したレイで子ノードをトラバース
} else if ( レイ錐体と AABB が交差しない ) {    // early miss exit
    ノードのトラバース終了
} else {    // last resort
    if (レイパケットが分割可能) {
        レイパケットを分割
    }
    for each パケット中の未テストのレイ {
        if ( レイと AABB が交差 ) {
            交差したレイで子ノードをトラバース
        }
    }
    // 交差するレイが見つからなかった
    ノードのトラバース終了
}
```

図 4.7　レイパケット分割を適応したパケットトラバースアルゴリズム

このアルゴリズムにより, ノードの AABB に対してパケットサイズが大きなレイパケットは分割され，コヒーレンスを保ったパケットトラバースが可能になる.

4.3 実験 I

4.3.1 準備

ここでは，4.2 節で解説したモデル非依存のパケットトラバースを実装し，その効果について検証する．実験に用いた環境やレンダリング対象，シェーディングなどの条件は，第 3 章と同様である．

4.3.2 実験結果

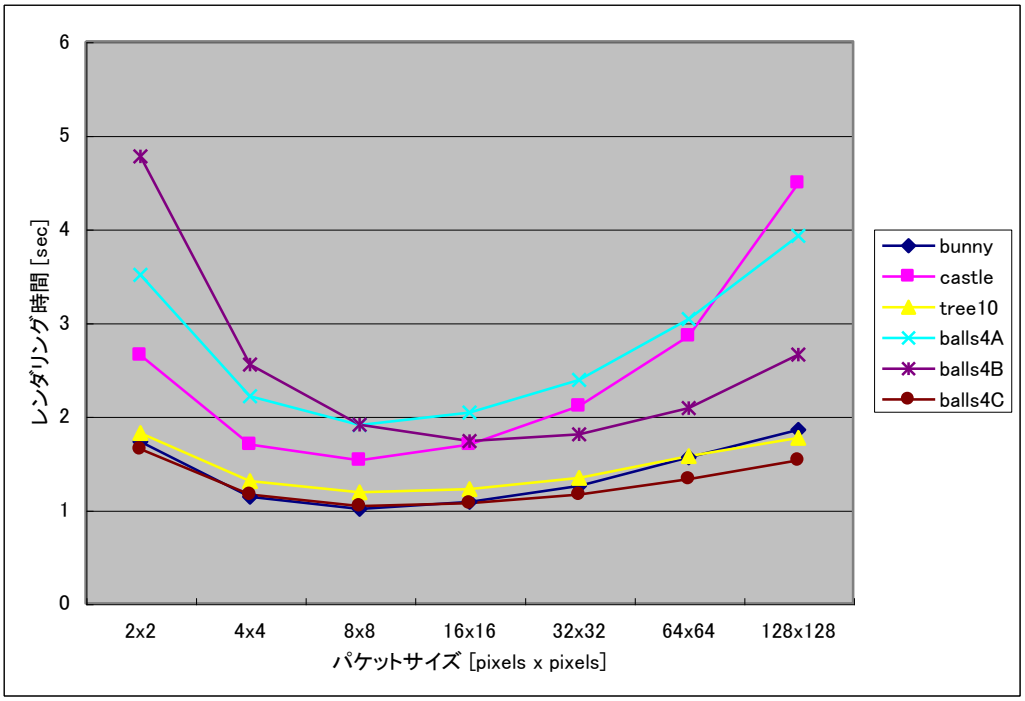


図 4.8　パケットサイズによる提案手法のレンダリング時間の変化

図 4.8 にパケットサイズによる提案手法のレンダリング時間の変化を示す．また，図 4.9 ～4.14 に，各レンダリング対象における第 3 章の既存手法における結果との比較を示す．

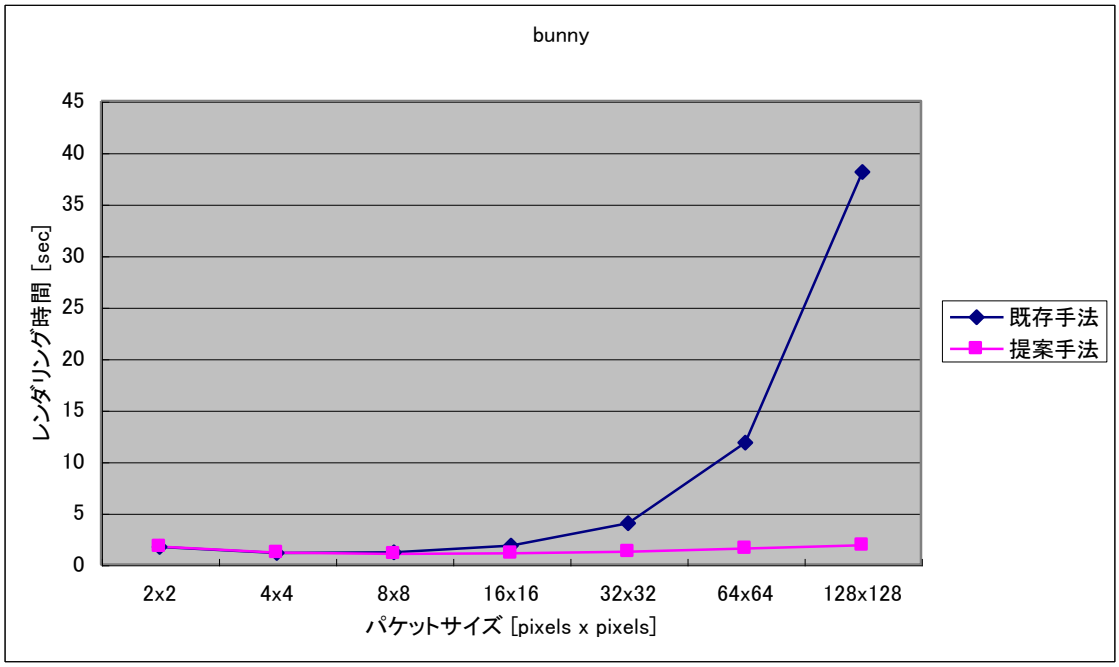


図 4.9　bunny におけるレンダリング時間の比較

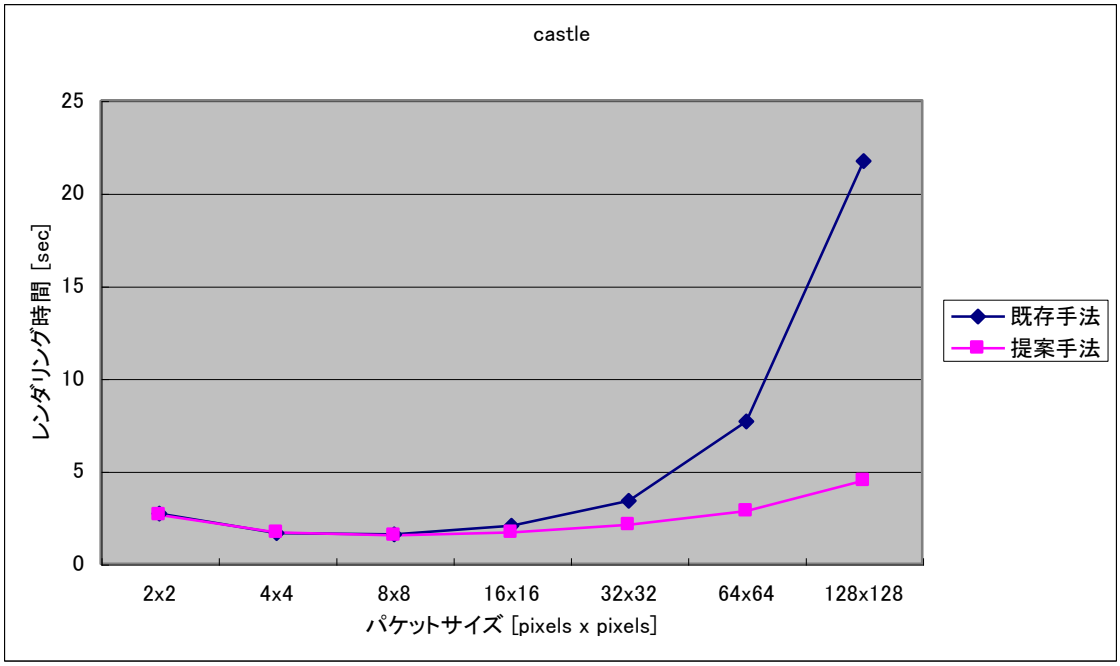


図 4.10　castle におけるレンダリング時間の比較

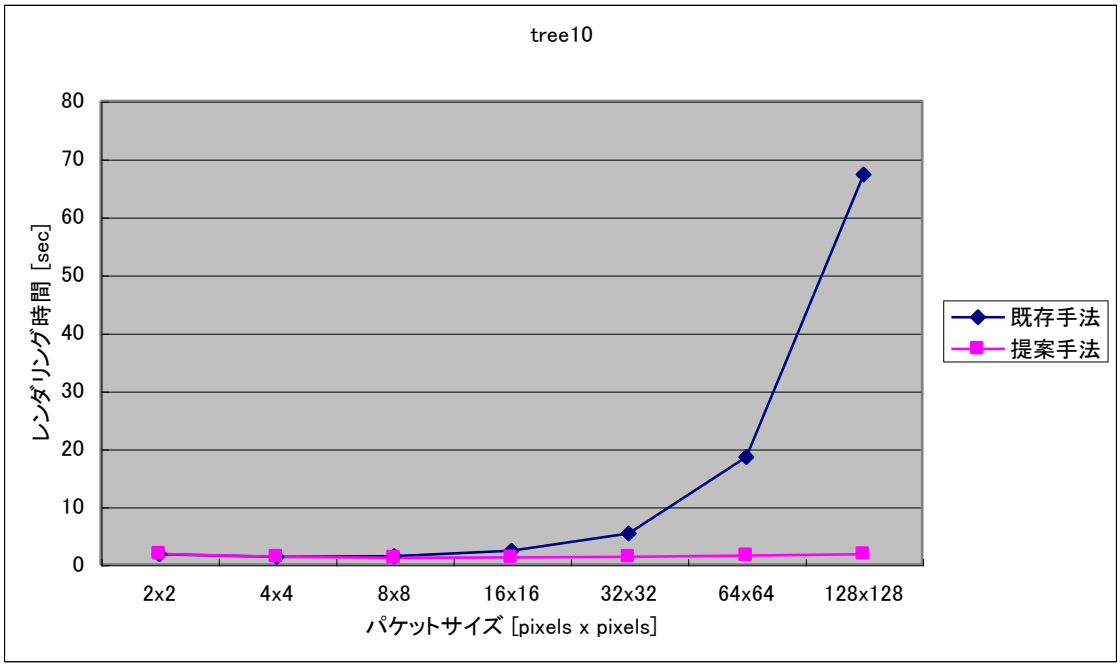


図 4.11　tree10 におけるレンダリング時間の比較

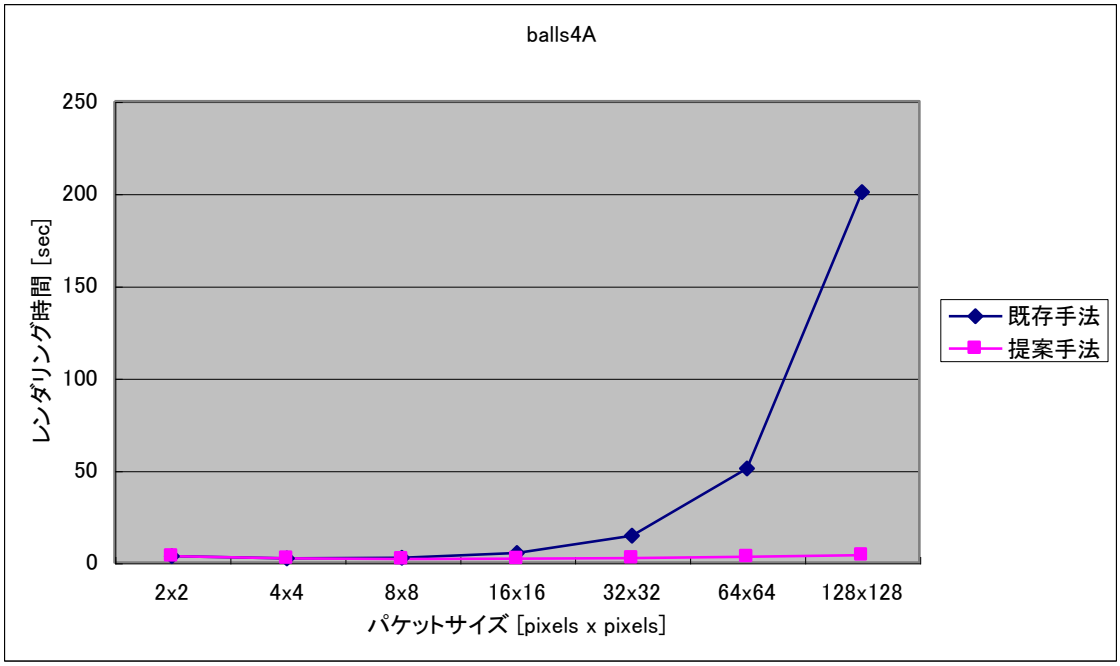


図 4.12　balls4A におけるレンダリング時間の比較

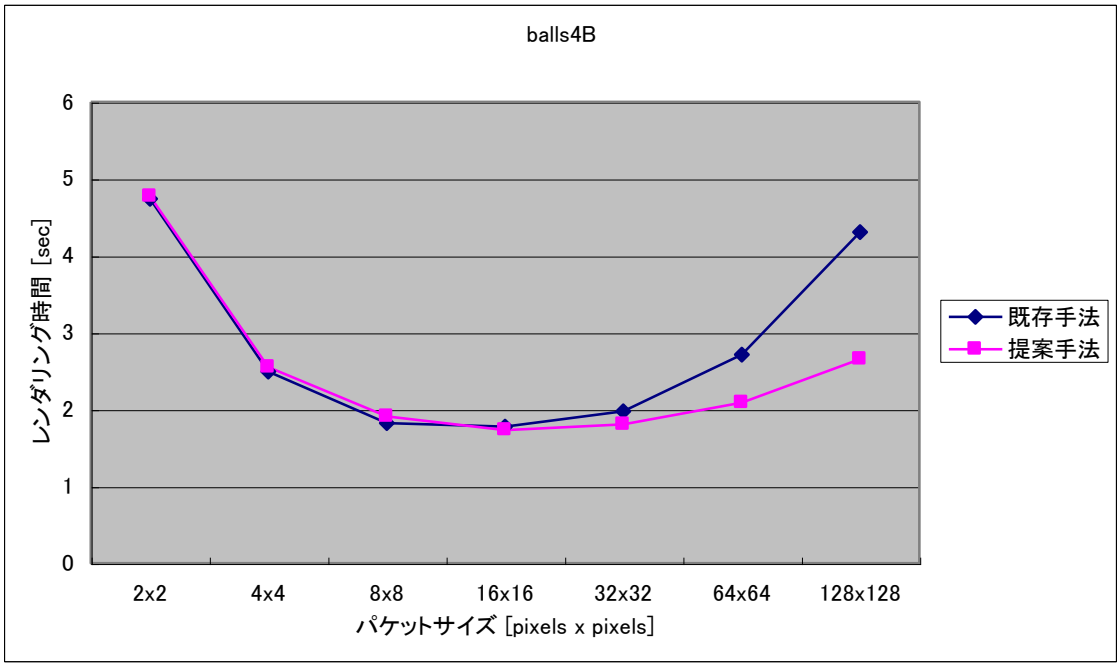


図 4.13　balls4B におけるレンダリング時間の比較

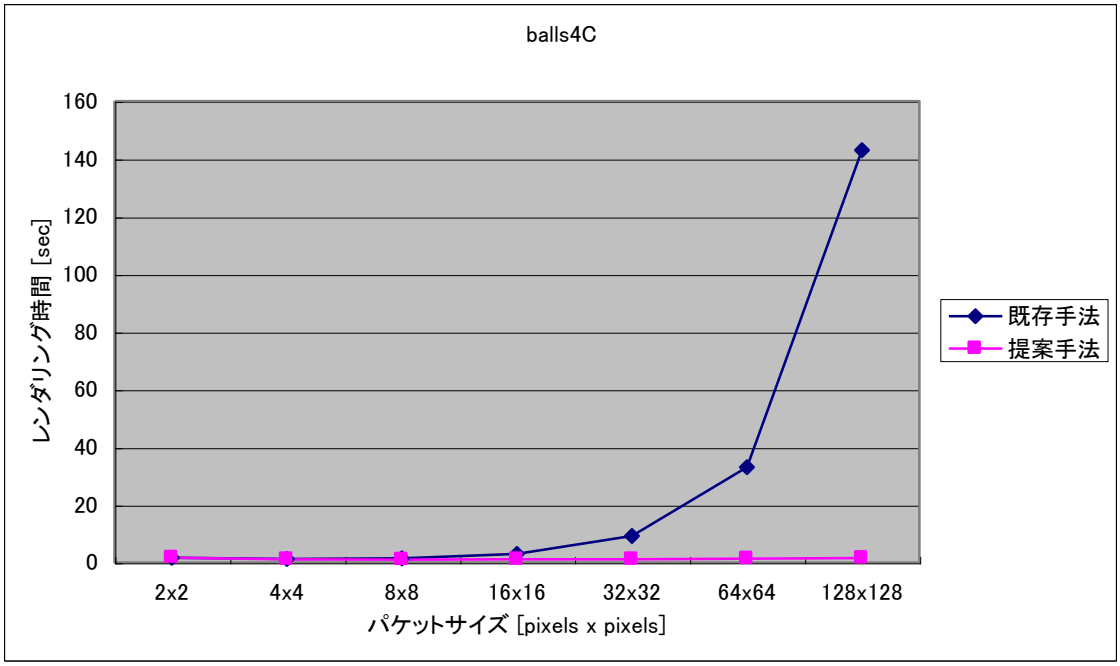


図 4.14　ball4C におけるレンダリング時間の比較

4.3.3 考察

図 4.9〜4.14 の結果より，パケットサイズによって指数的に増加していたレンダリング時間を大幅に抑える事ができたといえる．これにより，パケットサイズが最適なサイズより大きい場合でも，パケットトラバースの効果を損なうことのないトラバースが可能となった．

しかし，図 4.8 より，本手法の適用後もパケットサイズによってレンダリング時間が増加していることが伺える．これはおそらく，トラバース毎に 1 度だけ，つまり 4 つの下位パケットへとしか分割していない事が一因として挙げられるだろう．このため，トラバース毎にできる限りレイパケットを分割するように改善することで，よりコヒーレンスの低下を抑えたパケットトラバースが可能となると予測される．次節では，この仮定を実験によって検証する．

4.4 実験 II

4.4.1 準備

4.3.3 節で仮定したトラバース毎の分割を増やすことによる効率化を検証するために追加実験を行った．実験 I では，ノードのトラバース毎にレイパケットを 4 つの下位パケットへ分割していた. これを図 4.15 のように再帰的に 2 回行うことによって，レイパケットを 2 階層下の 16 の下位パケットへ分割する．

実験に用いた環境やシェーディングなどの条件は，レンダリング対象を除いて先行の実験 I と同様である．レンダリング対象は，図 4.8 の結果から，パケットサイズによるレンダリング時間の増加が顕著であった castle 及び balls4A を選択した．

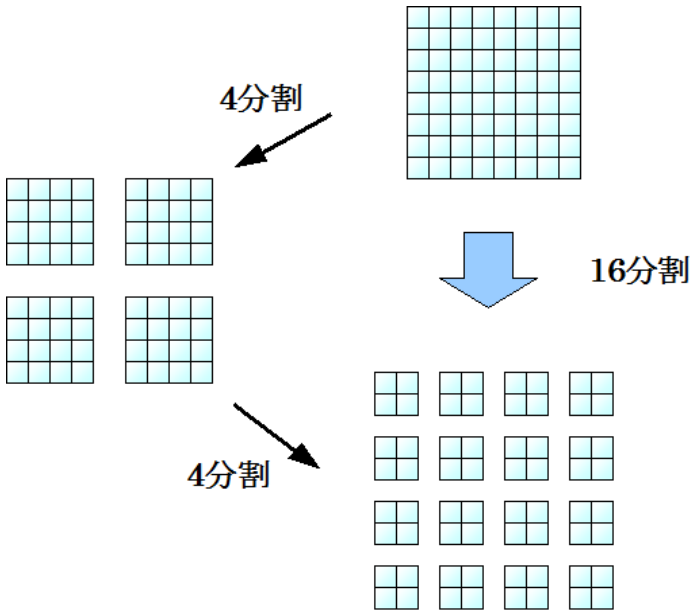


図 4.15　レイパケットの 16 分割

4.4.2 実験結果

トラバース過程におけるレイパケットの分割数によるレンダリング時間の変化を，図 4.16 及び図 4.17 に示す.

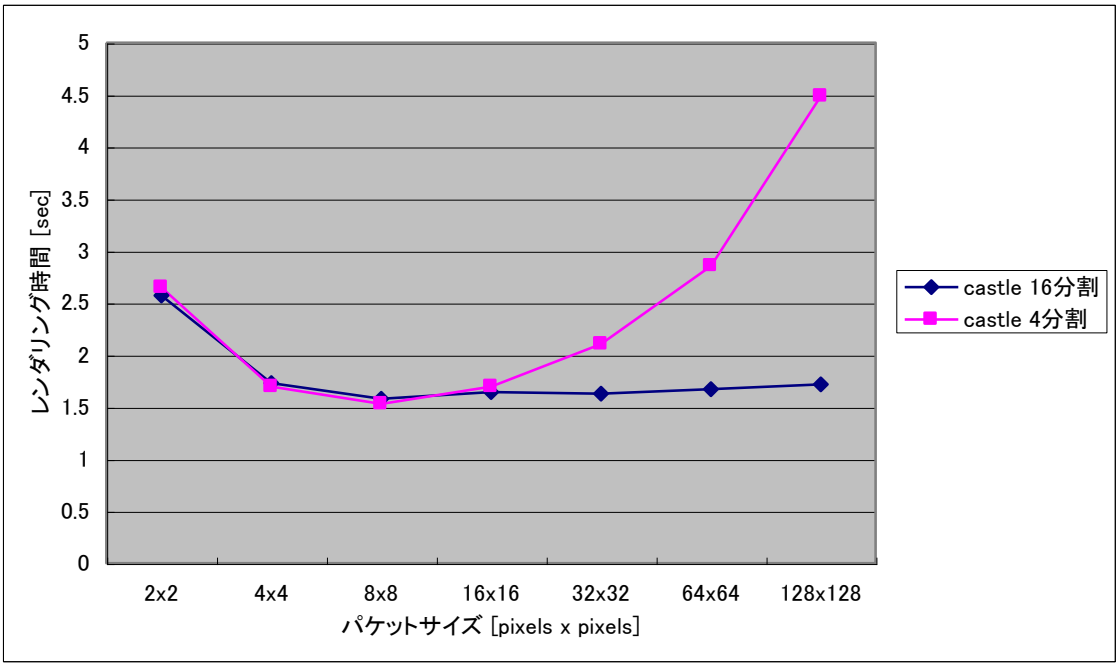


図 4.16　castle におけるレイパケット分割数によるレンダリング時間の変化

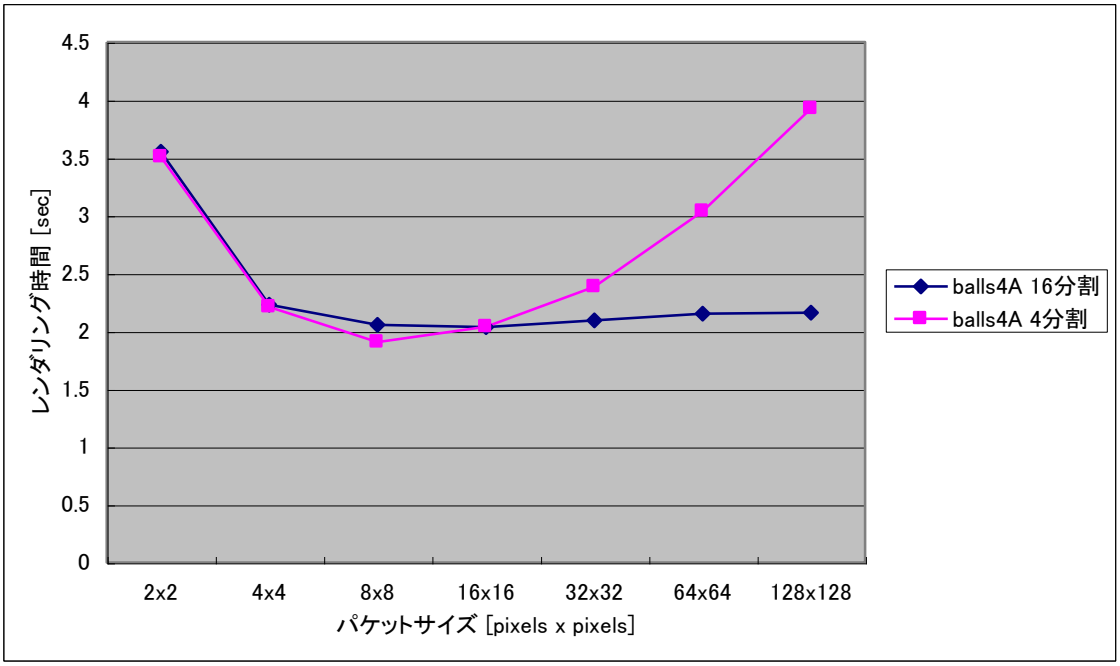


図 4.17　balls4A におけるレイパケット分割数によるレンダリング時間の変化

4.4.3 考察

図 4.16 及び図 4.17 の結果より，ノードのトラバース毎におけるレイパケットの分割数を 4 から 16 へと変化させたことによって，パケットサイズへの依存性を大幅に削減できていることが分かる．

このことから，ノードのトラバース毎におけるレイパケットの分割をできる限り行うことで，より効果的なパケットトラバースを実現できると考えられる．

4.5 第 4 章のまとめ

4.3 節の実験 I より，レイパケットを分割することによって，パケットサイズが最適でない場合でも効率を損ねることのないパケットトラバースを実現することができた．また，4.4 節の実験 II では，ノードのトラバース毎の分割数を増加させることによる最適化を行った．

以上の結果より，ノードのトラバース毎に AABB との位置関係を基にレイパケットを分割しトラバースが不要な下位レイパケットを切り捨てることで，トラバース過程で発生するレイパケットのコヒーレンス低下を抑えたトラバースが可能であるといえる．

本論文では，視点からのレイである 1 次レイのみを対象として扱ってきた．しかし，写実的なレンダリングを実現するには，反射や屈折，影といった表現をするため多くの 2 次レイを生成しなければならない．2 次レイ以降のレイは，コヒーレンスが乱れることが多く，1 次レイと比べてその扱いは非常に難しい．レイのコヒーレンスを用いたレイパケットベースのレンダリングは，今日でもまだまだ研究途上のテーマである．Boulos らの，BVH パケットトラバースを用いた Whitted のアルゴリズムによるレイトレーシング及び分散レイトレーシングに関する研究[BEL+07]によれば，シャドウレイや反射レイといった 2 次レイ以降においては，パケットサイズによる計算量の増大が 1 次レイ以上に顕著になるという結果が報告されている．つまり，2 次レイ以降におけるパケットトラバースでは，より最適なレイパケットを用いることが重要となってくる．本論文で提案した手法では，トラバース中にレイパケットを最適なパケットサイズへ近づけることができる．よって，本論文の提案する手法を適用したパ

ケットトラバースでは，2 次レイ以降におけるパケットトラバースにおいても高い効果を得ることができると予想される．

また，本論文では，レンダリング時に不必要な計算はなるべく行わないようにするため，レイパケット階層の生成，パケット分割面及び，分割面の法線の計算は全て事前に計算している．しかし，ウォークスルーと呼ばれる視点が時間的に移動するアプリケーションでは，視点の移動に伴って分割面とその法線は変化する．そのため，ウォークスルーアプリケーションへ本手法を適応する場合，分割軸，分割面の法線の計算，及び分割面と AABB との位置判定は，全て動的に行わなければならない．このため，レンダリング時における処理量は幾分増加してしまい，パフォーマンスに影響する可能性がある．このような，本論文の提案手法をウォークスルーへ対応させた場合の問題点については，今後検証していく必要がある．

第　5　章
結 論

本章では，本論文のまとめ及び今後の課題，展望について述べる．

5.1 まとめ

本論文では，写実的なレンダリングの基盤技術であるレイトレーシングの高速化手法について述べた．

第 1 章では，研究における背景と，その中での本研究の位置付けについて述べた．第 2 章では，　レンダリングアルゴリズムにおけるレイトレーシングの位置付けや，その基本的なアルゴリズムを示した．レイトレーシングによって写実的なレンダリングが可能となる一方で，レンダリングの品質向上に伴いその計算量は増大する．この問題を解決するため，近年様々な高速化手法が研究されている事を示し，その中でも空間データ構造によるアプローチが最も重要であることを述べた．また，近年のハードウェアへの対応による高速化について紹介した．第 3 章では，空間データ構造アルゴリズムの 1 つである BVH について紹介し, その構築手法や高速化の理論を示した．更に, 高速なトラバースを可能とするパケットトラバースについて, その要点を示し, 実験によってその効果を確認した．そして，実験結果より既存手法における問題点について考察した．第 4 章では，第 3 章での考察に基づきパケットトラバースの改善手法を提案した．本論文で提案する手法は，1 次レイパケットの持つ階層性を利用したものである．パケットトラバースを用いて子ノードをトラバースする度にレイパケットを分割することによる，コヒーレンスの低下を抑えたパケットトラバース手法を提案し，実験によってその効果を裏付けた．

5.2 今後の課題

本論文で提案した手法により，パケットトラバース過程におけるレイパケットのコヒーレンス低下を抑え，パケットトラバースの効果が 3 次元シーン及びモデルに依存するという問題を解決することができた．

本論文で行った実験では，レイパケットの階層構造や分割面情報は全てレンダリング前に計算している．このため，動的に視点が移動するウォークスルーへ適応可能かどうかは未知数である．この点については今後の課題とし，検証する必要がある．

写実的なレイトレーシングによるレンダリングでは，シェーディングの際に反射や屈折等を表現するため 2 次レイが発生する．これら 2 次レイは，物体の形状に依存して予測不可能な複雑な軌跡を辿っていくためコヒーレンスが乱れやすく，パケットトラバースでの扱いが困難である．そのため，最適なパケットサイズを用いることは，1 次レイ以上に重要となってくる．本論文での提案手法では，レイのコヒーレンス低下を抑えたトラバースが可能であるため，2 次レイ以降におけるパケットトラバースでは，よりその効果を得ることができると予想される．これを検証するためには，近年研究されている 2 次レイ以降のパケットトラバース手法に本手法を適用する必要がある．

参 考 文 献

[Appel86] Arthur Appel. Some Techniques of Shading Machine Renderings of Solids. Proceedings of the Spring Joint Computer Conference, 1986.

[BEL+07] Solomon Boulos, Dave Edwards, J.Dylan Lacewell, Joe Kniss, Jan Kautz, Peter Shirley and Ingo Wald. Packet-based Whitted and Distribution Ray Tracing. Proceedings of Graphics Interface, 2007.

[BWS06] Solomon Boulos, Ingo Wald and Peter Shirley. Geometric and Arithmetic Culling Methods for Entire Ray Packets. School of Computing, University of Utah, Technical Report#UUCS-06-010, 2006.

[BWSF06] Carsten Benthin, Ingo Wald, Michael Scherbaum and Heiko Friedrich. Ray Tracing on the CELL Processor. Proceedings of the 2006 IEEE Symposium on Interactive Ray Tracing, pp.25-23, 2006.

[CPC84] Robert L.Cook, Thomas Porter and Loren Carpenter. Distributed Ray Tracing. SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques, pp.137-145, 1984.

[CSE06] David Cline, Kevin Steele and Parris Egbert. Lightweight Bounding Volumes for Ray Tracing. to appear in Journal of Graphics Tools, 2006.

[GPSS07] Johannes Günther, Stefan Popov, Hans-Peter Seidel and Philipp Slusallek. Realtime Ray Tracing on GPU with BVH-based Packet Traversal. IEEE/Eurographics Symposium on Interactive Ray Tracing 2007, 2007.

[Havran01] Vlastimil Havran. Heuristic Ray Shooting Algorithms. PhD thesis, the Faculty of Electrical Engineering, Czech Technical University, 2001.

[Haines87] Eric Haines. A Proposal for Standard Graphics Environments. IEEE Computer Graphics and Applications, Vol.7, pp.3-5, 1987. http://tog.acm.org/resources/SPD/

[IWP07] Thiago Ize, Ingo Wald and Steven G.Parker. Asynchronous BVH Construction for Ray Tracing Dynamic Scenes on Parallel Multi-Core Architectures. Proceedings of the 2007 Eurographics Symposium on Parallel Graphics and Visualization, 2007.

[MT97] Tomas Möller and Ben Trumbore. Fast, Minimum Storage Ray/Triangle Intersection. Journal of Graphics Tools, Vol.2, No.1, pp.21-28, 1997.

[PGSS07] Stefan Popov, Johannes Günther, Hans-Peter Seidel and Philipp Slusallek. Proceedings of Eurographics 2007, pp.415-424, 2007.

[RSH05] Alexander Reshetov, Alexei Soupikov and Jim Hurley. Multi-Level Ray Tracing Algorithm. Proceedings of ACM SIGGRAPH 2005, pp.1176-1185, 2005.

[Wald04] Ingo Wald. Realtime Ray Tracing and Interactive Global Illumination. PhD thesis, Computer Graphics Group, Saarland University, 2004.

[Wald07] Ingo Wald. On Fast Construction of SAH based Bounding Volume Hierarchies. Proceeding of the 2007 Eurographics/IEEE Symposium on Interactive Ray Tracing, 2007.

[WBS07] Ingo Wald, Solomon Boulos and Peter Shirley. Ray Tracing Deformable Scenes using Dynamic Bounding Volume Hierarchies. ACM Transactions on Graphics, Vol.26, No.1, 2007.

[WSS05] Sven Woop, Jörg Schmittler and Philipp Slusallek. RPU: A Programmable Ray Processing Unit for Realtime Ray Tracing. Proceedings of ACM SIGGRAPH 2005, pp.434-444, 2005.

[Whitted80] Turner Whitted. An Improved Illumination Model for Shaded Display. Comm.ACM, Vol.23, No.6, pp.343-349, 1980.

[木村 07] 木村秀敬, 宮田一乗. SIMD 命令を用いた木探索によるレイトレーシングの高速化. CG アニメーションカンファレンス, 2007.

[新庄 07] 新庄貞昭, 高橋誠史, 木村秀敬, 白井暁彦, 宮田一乗. GPU の先駆的利用の研究動向と将来像. 芸術科学会論文誌, Vol.6, No.3, pp.168-178, 2007.

[宮田 04] 宮田一乗, 高橋誠史, 黒田篤. GPU コンピューティングの動向と将来像. 芸術科学会論文誌, Vol.4, No.7, pp.13-19, 2004.

謝辞

本研究を進めるにあたり，多くの方々の暖かいご支援を頂きました．この場を借りて深く御礼申し上げます．

特に，指導教官である知識科学教育センター 宮田一乘教授には終始熱心なご指導を頂き，また充実した研究環境を提供して頂きました．ここに厚く御礼申し上げます．

また，副テーマ指導教官としてご指導頂きました由井薗隆也准教授に，深く御礼申し上げます．

本研究の審査委員である，杉山公造教授，西本一志教授，由井薗隆也教授には，中間審査の段階から貴重なご助言を頂き，深く御礼申し上げます．

最後に，宮田研究室の皆様には公私に渡って大変お世話になりました．心より感謝を申し上げます．