(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第4972642号
(P4972642)
(45)発行日　平成24年7月11日(2012.7.11)　(24)登録日　平成24年4月13日(2012.4.13)

(51)Int.Cl.　F I
G06T 15/06 (2011.01)　G06T 15/50 230

請求項の数 19 (全 24 頁)

(21)出願番号	特願2008-518466 (P2008-518466)
(86)(22)出願日	平成18年6月23日 (2006.6.23)
(65)公表番号	特表2009-514059 (P2009-514059A)
(43)公表日	平成21年4月2日 (2009.4.2)
(86)国際出願番号	PCT/US2006/024631
(87)国際公開番号	WO2007/002494
(87)国際公開日	平成19年1月4日 (2007.1.4)
審査請求日	平成20年12月2日 (2008.12.2)
(31)優先権主張番号	60/693,231
(32)優先日	平成17年6月23日 (2005.6.23)
(33)優先権主張国	米国(US)

(73)特許権者　503447933
メンタル　イメージス,　ゲーエムベーハー
ＭＥＮＴＡＬ　ＩＭＡＧＥＳ,　Ｇ．Ｍ．Ｂ．Ｈ．
ドイツ連邦共和国　ベルリン　１０６２３　ファサンエンストラッセ　８１
Ｆａｓａｎｅｎｓｔｒａｓｓｅ　８１,　１０６２３　Ｂｅｒｌｉｎ　(ＤＥ)．
(74)代理人　100136630
弁理士　水野　祐啓
(72)発明者　ケラー,　アレクサンダー
ドイツ連邦共和国　アルム　８９０７５　クレインクネッヘトベグ　３０

最終頁に続く

(54)【発明の名称】高精度の実時間レイトレーシング

(57)【特許請求の範囲】
【請求項１】
画像のピクセルに付いて、疑似カメラの像平面に記録されたシーンの一点を表すピクセル値を生成するコンピュータグラフィックス・システムであって、選択したレイトレーシング方法を用いて像の該ピクセル値を生成するよう構成されており、該選択したレイトレーシング方法は、前記ピクセルからシーン内へ選択した方向に沿って発せられた少なくとも1本の光線を含む光線ツリーの使用と、前記シーンにおける光線と物体の表面との交点の計算とを含む、コンピュータグラフィックス・システムにおいて、その改良方法が、
バウンディングボリューム階層を利用して前記シーン内の光線と物体との交点を計算する段階であって、該計算する段階が、
所与の光線に関して、該光線の起点に最も近い光線／物体交点の位置を特定するため、座標軸に平行なバウンディングボックスを定義し、利用し、再帰的且つ適応的にリファインする段階と、
所定の終了判定基準が満たされるまで、前記座標軸に平行なバウンディングボックスを反復的にリファインする段階と、
前記光線と物体との交点を計算した後、該計算した点の値を対応する光線の方向と共に使用して実際の交点により近い交点を反復的に再計算する段階とを含み、
定義し、利用し、再帰的且つ適応的にリファインする前記段階が、
（ａ）前記シーンの表面を分割して形成された三角形の座標に基づいて3次元ツリーを構築することによって、前記座標軸に平行な一つのバウンディングボックス内の仮想物体の

集合を左物体の集合と右物体の集合とに区分するための分割平面Mを決定する段階と、
（ｂ）前記分割平面を含むとともに前記左物体の集合が入る左バウンディングボックスを定義するためのＬ平面と、前記分割平面を含むとともに前記右物体の集合が入る右バウンディングボックスを定義するためのR平面とを特定する段階であって、前記分割平面Ｍと前記L平面間の重複部分の容積および前記分割平面Ｍと前記R平面間の重複部分の容積がそれぞれ最小となるように、互いに平行な前記Ｌ平面および前記Ｒ平面を特定する段階とを含む、
改良方法。

【請求項２】

3次元ツリー構築を実行する前記段階が、

区分すべき座標軸に平行なバウンディングボックス内の三角形の頂点座標に基づいて分割平面の候補を特定する段階と、

前記候補の集合から、現在の座標軸に平行なバウンディングボックスの最長辺の中心に最も近い平面を選択する段階とを含む、請求項１に記載の改良方法。

【請求項３】

選択する前記段階が、表面法線の最長成分が前記可能性のある分割平面の法線にマッチする三角形の座標のみを選択する段階を更に含む、請求項２に記載の改良方法。

【請求項４】

前記分割平面に交差する三角形が、座標軸に平行なバウンディングボックスの左右の区分のどちらに含まれているかを特定する段階を含み、特定する該段階が、まず該三角形の前記分割平面との交差を計算して「交差線」を生成する段階と、次に、該交差線が前記分割平面と前記バウンディングボックスとの交差により定義される長方形に対してどのように位置しているかを特定する段階とを含む、請求項２に記載の改良方法。

【請求項５】

前記バウンディングボリューム階層の走査の順序を光線方向に基づいて決定する段階を更に含む、請求項４に記載の改良方法。

【請求項６】

前記バウンディングボリュームの階層が、像を処理する必要に応じて実時間で計算される、請求項５に記載の改良方法。

【請求項７】

レイトレーシングのアクセラレーションデータ構造をバケットソートに基づいて構築する段階を更に含み、該構築が、

物体を含む前記座標軸に平行なバウンディングボックスを$n_x \times n_y \times n_z$個の座標軸に平行なボックスに区分する段階と、

各オブジェクトを選択した一点により前記ボックスの１つに厳密にソートする段階であって、該選択した一点は各三角形の重心又は第１頂点の何れかを含む、ソートする段階と、

各グリッドセルにおける前記物体を含む前記座標軸に平行なバウンディングボックスを特定する段階とを含む、請求項６に記載の改良方法。

【請求項８】

計算した交点が前記シーンの表面の下方にある場合、該計算点は前記表面の他方の側に移動されて修正計算点を与える、請求項1に記載の改良方法。

【請求項９】

前記移動は表面法線に沿って、又は該表面法線の最長成分により決定される軸に沿ったものである、請求項８に記載の改良方法。

【請求項１０】

前記移動が、交点を表す浮動小数点の仮数の最終ビットを整数イプシロンで移動することで実行され、2次光線は前記修正計算点から始まるよう定義される、請求項８に記載の改良方法。

【請求項１１】

光線／三角形交差テストを用いる段階を更に含み、該テストでは、前記シーンにおける光線と表面の三角形細分化の平面との交点が特定され、該光線における所定有効区間の外の交点は除外される、請求項1に記載の改良方法。

【請求項１２】

画像のピクセルに付いて、疑似カメラの像平面に記録されたシーンの一点を表すピクセル値を生成するコンピュータグラフィックス・システムであって、選択したレイトレーシング方法を用いて像の該ピクセル値を生成するよう構成されており、該選択したレイトレーシング方法は、前記ピクセルからシーン内へ選択した方向に沿って発せられた少なくとも1本の光線を含む光線ツリーの使用と、前記シーンにおける光線と物体との交点の計算とを含む、コンピュータグラフィックス・システムにおいて、その改良方法が、

バウンディングボリューム階層を利用することにより前記シーン内の光線と物体との交点の計算を可能とするよう動作可能なコンピュータ実行可能ソフトウェアコードを含み、該計算が、

所与の光線に関して、該光線の起点に最も近い光線／物体交点の位置を特定するため、座標軸に平行なバウンディングボックスを定義し、利用し、再帰的且つ適応的にリファインする段階と、

所定の終了判定基準が満たされるまで、前記座標軸に平行なバウンディングボックスの前記リファインを継続する段階と、

前記光線と物体との交点を計算した後、該計算した点の値を対応する光線の方向と共に使用して実際の交点により近い交点を反復的に再計算する段階とを含み、

定義し、利用し、再帰的且つ適応的にリファインする前記段階が、

（ａ）前記シーンの表面を分割して形成された三角形の座標に基づいて3次元ツリーを構築することによって、前記座標軸に平行な一つのバウンディングボックス内の仮想物体の集合を左物体の集合と右物体の集合とに区分するための分割平面Mを決定する段階と、

（ｂ）前記分割平面を含むとともに前記左物体の集合が入る左バウンディングボックスを定義するためのＬ平面と、前記分割平面を含むとともに前記右物体の集合が入る右バウンディングボックスを定義するためのR平面とを特定する段階であって、前記分割平面Ｍと前記L平面間の重複部分の容積および前記分割平面Ｍと前記R平面間の重複部分の容積がそれぞれ最小となるように、互いに平行な前記Ｌ平面および前記Ｒ平面を特定する段階とを含む、

改良方法。

【請求項１３】

画像のピクセルに付いて、疑似カメラの像平面に記録されたシーンの一点を表すピクセル値を生成するコンピュータグラフィックス・システムであって、該コンピュータグラフィックス・システムは、選択したレイトレーシング方法を用いて像の該ピクセル値を生成するよう構成されており、該選択したレイトレーシング方法は、前記ピクセルからシーン内へ選択した方向に沿って発せられた少なくとも1本の光線を含む光線ツリーの使用と、前記シーンにおける光線と物体の表面との交点の計算とを含み、該コンピュータグラフィックス・システムが、

バウンディングボリューム階層を利用することによって前記シーン内の光線と物体との交点を計算する手段であって、該計算する手段が、

所与の光線に関して、該光線の起点に最も近い光線／物体交点の位置を特定するため、座標軸に平行なバウンディングボックスを定義し、利用し、再帰的且つ適応的にリファインする手段と、

所定の終了判定基準が満たされるまで、前記座標軸に平行なバウンディングボックスの前記リファインを継続する手段と、

前記光線と物体との交点を計算した後、該計算した点の値を対応する光線の方向と共に使用して実際の交点により近い交点を反復的に再計算する手段とを含み、

定義し、利用し、再帰的且つ適応的にリファインする前記手段が、

（ａ）前記シーンの表面を分割して形成された三角形の座標に基づいて3次元ツリーを構

築することによって、前記座標軸に平行な一つのバウンディングボックス内の仮想物体の集合を左物体の集合と右物体の集合とに区分するための分割平面Mを決定する処理と、
（ｂ）前記分割平面を含むとともに前記左物体の集合が入る左バウンディングボックスを定義するためのＬ平面と、前記分割平面を含むとともに前記右物体の集合が入る右バウンディングボックスを定義するためのR平面とを特定する処理であって、前記分割平面Ｍと前記L平面間の重複部分の容積および前記分割平面Ｍと前記R平面間の重複部分の容積がそれぞれ最小となるように、互いに平行な前記Ｌ平面および前記Ｒ平面を特定する処理とを実行する、
コンピュータグラフィックス・システム。

【請求項１４】

画像のピクセルに付いて、疑似カメラの像平面に記録されたシーンの一点を表すピクセル値を生成するコンピュータグラフィックス・システムであって、選択したレイトレーシング方法を用いて像の該ピクセル値を生成するよう構成されており、該選択したレイトレーシング方法は、前記ピクセルからシーン内へ選択した方向に沿って発せられた少なくとも1本の光線を含む光線ツリーの使用と、前記シーンにおける光線と物体の表面との交点の計算とを含む、コンピュータグラフィックス・システムにおいて、前記シーンにおいて光線と物体との交点を計算する方法であって、

バウンディングボリューム階層を構築する段階を含み、構築する該段階が、

所与の光線に関して、該光線の起点に最も近い光線／物体交点の位置を特定するため、座標軸に平行なバウンディングボックスを定義し、利用し、再帰的且つ適応的にリファインする段階と、

所定の終了判定基準が満たされるまで、前記座標軸に平行なバウンディングボックスの前記リファインを継続する段階と、

前記光線と物体との交点を計算した後、該計算した点の値を対応する光線の方向と共に使用して実際の交点により近い交点を反復的に再計算する段階とを含み、

定義し、利用し、再帰的且つ適応的にリファインする前記段階が、
（ａ）前記シーンの表面を分割して形成された三角形の座標に基づいて3次元ツリーを構築することによって、前記座標軸に平行な一つのバウンディングボックス内の仮想物体の集合を左物体の集合と右物体の集合とに区分するための分割平面Mを決定する段階と、
（ｂ）前記分割平面を含むとともに前記左物体の集合が入る左バウンディングボックスを定義するためのＬ平面と、前記分割平面を含むとともに前記右物体の集合が入る右バウンディングボックスを定義するためのR平面とを特定する段階であって、前記分割平面Ｍと前記L平面間の重複部分の容積および前記分割平面Ｍと前記R平面間の重複部分の容積がそれぞれ最小となるように、互いに平行な前記Ｌ平面および前記Ｒ平面を特定する段階とを含む、
交点を計算する方法。

【請求項１５】

バウンディングボックスを細分化する段階を更に含む、請求項２に記載の改良方法。

【請求項１６】

2次光線を開始する点を選択する段階を更に含み、選択する該段階が、バウンディングボックスの隅であって該ボックスの中央において表面法線に最も近い隅を選択し、且つ前記選択した隅に対応する前記点を用いて前記2次光線を開始する段階とを含む、請求項1に記載の改良方法。

【請求項１７】

3次元ツリーを構築する前記段階が、選択した深さから左平衡することでツリーの枝刈りを実行する段階を更に含み、該ツリーの枝刈りが、前記選択した深さから二分空間区分を近似的に左平衡することでツリー深さを枝刈りする段階を含む、請求項２に記載の改良方法。

【請求項１８】

前記バウンディングボリューム階層を実時間で且つ要求時に構築する段階を更に含む、

請求項1に記載の改良方法。
【請求項１９】
画像のピクセルに付いて、疑似カメラの像平面に記録されたシーンの一点を表すピクセル値を生成するコンピュータグラフィックス・システムであって、選択したレイトレーシング方法を用いて像の該ピクセル値を生成するよう構成されており、該選択したレイトレーシング方法は、前記ピクセルからシーン内へ選択した方向に沿って発せられた少なくとも1本の光線を含む光線ツリーの使用と、前記シーンにおける光線と物体の表面との交点の計算とを含む、コンピュータグラフィックス・システムにおいて、その改良方法が、
バウンディングボリューム階層を利用して前記シーン内の光線と物体との交点を計算する段階であって、該計算する段階が、
所与の光線に関して、該光線の起点に最も近い光線／物体交点の位置を特定するため、座標軸に平行なバウンディングボックスを定義し、利用し、再帰的且つ適応的にリファインする段階と、
所定の終了判定基準が満たされるまで、前記座標軸に平行なバウンディングボックスの前記リファインを継続する段階とを含み、
定義し、利用し、再帰的且つ適応的にリファインする前記段階が、
バウンディングボックスの隅であって該ボックスの中央において表面法線に最も近い隅を選択し、且つ前記選択した隅に対応する前記点を用いて前記2次光線を開始する段階と、
3次元ツリー構築を利用する段階とを含み、前記構築が、選択した深さから左平衡することでツリーの枝刈りを実行する段階を含み、該ツリーの枝刈りが、前記選択した深さから二分空間区分を近似的に左平衡することでツリー深さを枝刈りする段階を含み、
座標軸に平行なバウンディングボックスを定義する前記段階が、
（ａ）前記シーンの表面を分割して形成された三角形の座標に基づいて3次元ツリーを構築することによって、前記座標軸に平行な一つのバウンディングボックス内の仮想物体の集合を左物体の集合と右物体の集合とに区分するための分割平面Mを決定する段階と、
（ｂ）前記分割平面を含むとともに前記左物体の集合が入る左バウンディングボックスを定義するためのＬ平面と、前記分割平面を含むとともに前記右物体の集合が入る右バウンディングボックスを定義するためのR平面とを特定する段階であって、前記分割平面Ｍと前記L平面間の重複部分の容積および前記分割平面Ｍと前記R平面間の重複部分の容積がそれぞれ最小となるように、互いに平行な前記Ｌ平面および前記Ｒ平面を特定する段階とを含み、
前記終了判定基準が満たされるまで前記左右の物体を反復的に処理する段階を更に含み、
段階（ａ）において、3次元ツリーを構築する前記段階が、
（ａ１）区分すべき座標軸に平行なバウンディングボックス内の三角形の頂点座標に基づいて分割平面の候補を特定する段階と、
（ａ２）前記候補の集合から、現在の座標軸に平行なバウンディングボックスの最長辺の中心に最も近い平面を選択する段階であって、表面法線の最長成分が可能性のある分割平面の法線にマッチする三角形の座標のみを選択する段階を更に含み、
前記バウンディングボリューム階層が実時間で且つ要求時に構築され、
前記光線と物体との交点を計算した後、該計算した点の値を対応する光線の方向と共に使用して実際の交点により近い交点を反復的に再計算する、改良方法。
【発明の詳細な説明】
【技術分野】
【０００１】
関連出願の引用
本出願は、2005年6月23日付けで提出した米国特許仮出願第60/693,231号の優先権を主張し、その全体は参照して本明細書に援用する。
【０００２】

コンピュータ・プログラム付属文書への言及
本明細書と共にソースコードリスティングを提出し、その全体は参照して本明細書に援用する。このソースコードリスティングは本明細書では「付属文書」と呼び、「1.1.1.」形式で3桁の参照番号で識別される段落に構成されている。
【背景技術】
【０００３】
本発明は、一般にディジタル計算機システム内で該システムにより実行される、映画やその他の応用分野の像を描画するための方法及びシステムに関し、詳細には実時間精密レイトレーシングのための方法、システム、装置、及びコンピュータソフトウェアに関する。
【０００４】
従来技術の説明
「レイトレーシング」という用語は、光源とカメラとを結ぶ全ての光路を識別し且つこれらの寄与を合計することによって写実的な像を合成する技術に関する。このシミュレーションは、視線に沿った光線を追跡して可視性を判断し、且つ照度を測定するために光源からの光線を追跡する。
【０００５】
レイトレーシングは映画やその他の応用分野で主流となっている。しかし、現在のレイトレーシング技術には、数値的問題、動的シーンを処理する能力的限界、アクセラレーションデータ構造の設定が遅いこと、及び大量のメモリを必要とすることなど、多くの周知の限界及び弱点がある。従って、現在のレイトレーシング技術は、森林や人の頭髪を吹き抜ける風などの完全に動画表現されたシーンを効率的に扱う能力に欠けている。現在のレイトレーシングシステムの限界を克服すれば、例えば映画作品においてより高品位の動きのぼかしも描画できるようになろう。
【０００６】
レイトレーシングシステムの性能を向上させるため幾つかの試みが現在までになされているが、これらは幾つかの理由で十分な結果を残していない。例えば、現在のレイトレーシングシステムは、一般に、座標軸に平行な二分空間区分(原語：binary space partition)に基づくアクセラレーション構造として3次元ツリーを使用する。これらシステムは静的シーンの描画に主に焦点を当てているため、一般に、完全に動画表現されたシーンに関連して必要なデータ構造を構築するのに要するセットアップ時間がかなり長くなるという問題に対処できていない。これらを考慮して、ある製造者は、効率的な3次元ツリーを作成し且つこのツリーを走査するのに必要な時間を短縮できるアルゴリズムを開発することによって実時間レイトレーシングを改善している。しかし、このシステムでの予想メモリ要件は、レイトレーシングの対象となる物体の数の増加につれて二次式的に増大する。
【０００７】
別の製造者は、バウンディングボリューム階層を用いてシステム性能を向上させるレイトレーシング集積回路を設計している。しかし、追跡される非干渉性の2次光線が多くなりすぎると、このアーキテクチャの性能は低下する。
【０００８】
更に、書替え可能ゲートアレイ(FPGA)を用いた3次元ツリー走査アルゴリズムを実装することでシステム性能を向上させようとする試みもなされている。これらのシステムにおいて処理速度が増加するのは、主として、コヒーレント光線束を追跡し且つFPGAの能力を利用して高速のハードワイヤード計算を実行することによる。アクセラレーション構造の構築はハードウェアではまだ実装されていない。こうしたFPGA実装は、典型的には精度を落とした浮動小数点技法を用いる。
【０００９】
発明の概要
本発明は、その幾つかの側面で、上述の問題に対処し且つ現在のレイトレーシング装置

に容易に適合可能な高精度で高性能な技法、並びにこうした技法を実現する方法、システム、及びコンピュータソフトウェアを提供する。本明細書に記載する技法では、レイトレーシングの対象となる物体の数の増加に対してメモリ使用量は線形に増加する。ならし解析では、記載した技法は、現在の実時間レイトレーシング技術を性能で上回っている。

【００１０】

本発明の第1の側面は、実時間レイトレーシングに高度に適応した様態でバウンディングボリューム階層を利用する技術に関する。

【００１１】

本発明の別の側面は、レイトレーシングにおける自己交差の問題に対処する。光線と表面との交点を計算した後、計算した点を光線の方向と共に使用して光線と表面との交点を再計算することにより、精度を向上させる反復を実現する。

【００１２】

本発明の別の側面は、分割平面選択における最適化による高性能3次元ツリー構築、最小記憶構成、及び近似左平衡によるツリーの枝刈りを可能とする。

【００１３】

本発明の別の側面は高性能バウンディングボリューム階層の使用を含み、ここでは、座標軸に平行なバウンディングボックスを明示的に表現する代わりに、システムは区間の階層によって座標軸に平行なバウンディングボックスを暗黙的に表現する。ある実装では、物体のリスト及び座標軸に平行なバウンディングボックスが与えられれば、システムは、Ｌ及びR平面を決定し、物体の集合をそれに従って区分する。すると、システムは、なんらかの終了判定基準が満たされるまで左右の物体を反復的に処理する。内部ノードの数は拘束されているので、1つの物体しか残っていないときに終了に依存しても安全である。

【００１４】

本発明の別の側面は、本明細書に記載された3次元ツリー構築技術を介して分割平面Mを効率的に決定し、次に、座標軸に平行な新たなバウンディングボックスの得られたＬ及びR平面の重複部分と提案分割平面Mとの重複が最小となるように物体を区分することが含まれる。

【発明を実施するための最良の形態】

【００１５】

本発明は、レイトレーシングを行うための改良技法及び、高速レイトレーシングに必要なアクセラレーションデータ構造の効率的構築のための改良技法を提供する。以下にこれら技法に従った方法、構造、及びシステムを説明する。

【００１６】

当業者であれば、本明細書に記載した方法及びシステムは、マイクロソフト社のWindows（登録商標）、Linux、又はUnix（登録商標）のような従来のオペレーティングシステムに従って動作する或いはエミュレートするパーソナルコンピュータ(PC)のような従来のコンピュータ装置又は同等装置をスタンドアローン構成で或いはネットワークを介して用いて、ソフトウェア、ハードウェア、又はソフトウェアとハードウェアの組合せとして実装できることは理解できるはずである。従って、後述し且つ特許請求の範囲に記載したこれらの様々な処理手段又は計算手段は、適切に構成したディジタル処理装置又は装置のネットワークのソフトウェア及び／又はハードウェア要素で実装できる。処理は順次又は並列で実行でき、更に、特定目的ハードウェア又は再構成可能ハードウェアを用いて実装できる。

【００１７】

本発明による方法、装置、又はソフトウェア製品は、図1に例示したもののような広範囲な従来の計算機及びシステムの何れか（例えば、ネットワークシステム100）で動作でき、これらには、スタンドアローン、ネットワーク、携帯用、又は固定式の何れでもよく、又はインターネット又は他のネットワーク108を介した従来のパーソナルコンピュータ102、ラップトップ104、手持ち型又はモバイルコンピュータ106が含まれる。又、インターネット又は他のネットワーク108はサーバ110及び記憶装置112を含むことができる。

【００１８】
従来のコンピュータソフトウェア及びハードウェアで行われているように、本発明に従って構成されたソフトウェアアプリケーションは、例えば図2に示したようなパーソナルコンピュータ102内部で動作できる。コンピュータ102では、プログラム命令はCD-ROM116、磁気ディスク又は他の記憶装置120から読み出して、CPU118が実行できるようにRAM114にロードできる。データは、従来のキーボード、スキャナ、マウス、又は他の要素103を含む既知の任意装置又は手段を介してシステムに入力できる。
【００１９】
図3は、本発明の一様態による包括的方法200を示した流れ図である。この方法は、画像の各ピクセルについてピクセル値が生成されるコンピュータグラフィックス・システムの文脈で実施される。生成された各ピクセル値は、疑似カメラの像平面に記録されたシーンの一点を表す。このコンピュータグラフィックス・システムは、選択したレイトレーシング方法を用いて像のピクセル値を生成するよう構成されている。選択したレイトレーシング方法は、このピクセルからシーン内へ選択した方向に沿って発せられた少なくとも1本の光線を含む光線ツリーの使用を含み、シーンにおける光線と物体（及び／又は物体の表面）との交点の計算も含む。
【００２０】
図3の方法200では、バウンディングボリューム階層を用いてシーン内の光線と表面との交点が計算される。ステップ201で、シーンのバウンディングボックスが計算される。ステップ202では、所定の終了判定基準が満たされているか否かが判断される。満たされていなければ、ステップ203で座標軸に平行なこのバウンディングボックスをリファインするこの処理は終了判定基準が満たされるまで反復的に継続する。本発明の一様態によれば、この終了判定基準は、バウンディングボックス座標が、光線と表面との交点の浮動小数点表現から一単位の分解能でのみ異なる状態であると定義される。しかし、本発明の範囲はこれ以外の終了判定基準まで拡張される。
【００２１】
バウンディングボリューム階層をアクセラレーション構造として使用することは多くの理由で有利である。バウンディングボリューム階層のメモリ要件は、レイトレーシング対象となる物体の数において線形拘束可能である。また、後述するように、バウンディングボリューム階層は、完全に動画表現されたシーンで必要とされるように、3次元ツリーに比べ遙かに効率的に構築でき、そのためならし解析に非常に適している。
【００２２】
次に、レイトレーシング技術における幾つかの問題と、これら問題に対処する本発明の諸様態とについてより詳細に説明する。図4は「自己交差」問題を示す図である。図4は、像表面302と、観察点304と、光源306とを含むレイトレーシング手順300を示す。表面の像を合成するため、一連の計算を実行して観察点304と表面302との間に延びる光線の位置を特定する。図4ではそうした1本の光線308を示した。理想的には、次に光線308と表面302との正確な交点310が計算される。
【００２３】
しかし、コンピュータにおける浮動小数点演算により、計算した光線と表面との交点312が実際の交点310と異なってしまうことがある。更に、図4に示したように、計算した点312が表面302の「間違った」側に位置してしまうこともある。その場合、計算した光線と表面との交点312から光源306へ延びる第2光線314の位置を特定する計算を行うと、これら計算は、第2光線314が直接光源306まで延びるのでなく第2交点316で表面に当たることを示し、従って結像エラーとなってしまう。
【００２４】
この自己交差問題に対する1つの既知の解決法は、それぞれの光線308を表面302からの安全距離から開始することである。この安全距離は、典型的には汎用浮動小数点εとして表現される。しかし、この汎用浮動小数点εの決定は、当該シーンと、像を合成しているシーン内の位置とに大きく依存する。

【００２５】
本発明の一様態はより高精度の代替的な方法を提供する。計算した光線と表面との交点312が得られると、計算した点312及び光線308の方向を用いて実際の交点310により近い交点を再計算する。この交点の再計算は、精度を向上させる反復としてこのレイトレーシング技法に組み込まれる。この反復計算された交点が表面302の「間違った」側に位置してしまった場合、これは表面302の「正しい」側に移動される。この反復計算された交点は表面法線に沿って、又は法線の最長成分により決定される軸に沿って移動できる。汎用浮動小数点εを使う代わりに、点を浮動小数点の仮数の最終ビットまで整数εで移動させる。

【００２６】
上述の手順は2倍精度計算を避け、指数により決定される浮動小数点数のスケールに暗黙的に適応するという利点がある。よって、この実装では、全ての2次光線はこれら修正点から直接始まり、εオフセットを不要とする。従って、交点の計算時に、妥当性のある光線区間はオフセット値でなく0で始まると推定できる。

【００２７】
仮数の整数表現を修正することで、どの点がどの側にあるかを特定するために三角形と平面とを交差させる際の数値問題も回避する。

【００２８】
凸結合の凸閉包特性を利用して、光線と自由曲面との交点を発見するには、光線の起点に最も近い交点を含む座標軸に平行なバウンディングボックスをリファインすればよい。このリファインは浮動小数点数の分解能に到達するまで、すなわちバウンディングボックス座標が、浮動小数点表現から一単位の分解能でのみ異なるまで継続できる。すると、自己交差問題は、バウンディングボックスの隅であってバウンディングボックスの中央において表面法線に最も近い隅を選択することで回避できる。すると、この隅の点を用いて第2光線をスタートする。この「光線と物体との交点テスト」は非常に効率的であり、自己交差問題を回避できるという利点がある。

【００２９】
アクセラレーションデータ構造を構築した後、三角形は所定の位置で変換される。新たな表現は縮退三角形を符号化して、交点テストが余分な手間を掛けずにそれらを扱うことができるようにする。単に縮退三角形がグラフィックスパイプラインに入るのを防ぐこともちろん可能である。付属文書の段落2.2.1及び2.2.2には、本発明のこの側面に従ったソースコードのリスティングが記載されている。

【００３０】
このテストは、まず光線と三角形の平面との交点を特定し、続いて光線に沿った妥当な区間}0,
result.tfar}の外にある交点を除外する。これはたった一回の整数テストで達成される。+0が妥当な区間から除外されることに注目されたい。非正規化した浮動小数点数がサポートされていない場合、これは重要である。この第1判定が成功すれば、このテストは交点の重心座標を計算しながら進行する。完全な包含テストを行うには、整数テストのみ、すなわちより具体的には2ビットのみのテストが必要となるだけである。従って、分岐の数は最小である。この効率的なテストを可能にするには、三角形の辺と法線を変換段階で適切に基準化する(原語：scale)。

【００３１】
このテストの精度は十分に高く、誤った又は外れた光線交点を避けることができる。しかし、走査時には、ロバストな交差テストのために三角形を拡張するのが適切な状況が発生することがある。これは三角形を変換する前に実行できる。三角形はその法線の最長の成分によって識別される軸に沿って投影されるので、この投影ケースを記憶しておく必要がある。これは、アクセラレーションデータ構造の葉ノードのカウンターで実現できる。すなわち、三角形参照は投影ケースにより記憶され、1つの葉は各クラスの三角形の数を表すバイトを含む。

【００３２】
本発明の更なる側面は、レイトレーシングのアクセラレーションデータ構造を構築する改良アプローチを提供する。幾つかの異なる最適化に従う従来のソフトウェア実装と比べて、本明細書に記載したアプローチは、より優れたレイトレーシング性能を備えたかなり平坦なツリーを形成する。
【００３３】
分割平面の候補は、区分対象となる座標軸に平行なバウンディングボックス内の三角形の頂点座標により与えられる。これは、実際にはバウンディングボックスの外部に位置する頂点だが、バウンディングボックスが定義する3つの区間の1つに位置する少なくとも1つの座標を備えた頂点を含む。これら候補の中から、現在の座標軸に平行なバウンディングボックスの最長辺の中心に最も近い平面が選択される。更なる最適化により、表面法線の最長成分が可能性のある分割平面の法線にマッチする三角形の座標のみが選択される。三角形の頂点を通るよう分割平面を配置すると、分割平面により分割される三角形の数が暗黙的に減少するので、この手順は非常に平坦なツリーを構成する。更に、表面は厳密に近似され、空の空間が最大化される。三角形の数が指定した閾値より高く、分割平面の候補がそれ以上ない場合は、ボックスはその最長辺に沿った中心で分割される。これにより、例えば長い斜め物体の使用を含む他のアプローチによる非能率を回避できる。
【００３４】
どの三角形が階層のノードの右及び左の子に属するかを決定するための再帰的手続は、典型的には大規模なブックキーピング及び記憶割当てを必要としていた。しかし、例外的な場合だけしか失敗しない遙かに単純なアプローチがある。レイトレーシングされる物体への参照の2配列のみが割り当てられる。第1配列は物体参照で初期化される。再帰的空間区分の間に、左側の要素のスタックが配列の始めから成長する一方、右に分類される要素は、配列の終わりから中心に向かって成長するスタックに維持される。分割平面に交差する(すなわち、左右両方である)要素を高速で記憶可能とするためには、第2配列がそれらのスタックを維持する。従って、バックトラッキングは効率的且つ単純である。
【００３５】
表面積ヒューリスティックを用いてツリーの枝刈りを行う代わりに、固定深さから二分空間区分を近似的に左平衡することでツリー深さを枝刈りする。徹底的な実験により観察されるように、グローバル固定深さパラメータは極めて多種多様なシーンで指定できる。これは、一定程度の二分空間区分の後で、通常は空間的に比較的平坦な連結成分が残っていることが観察されることから理解できる。付属文書の段落2.3.1には、本発明のこの側面に従ったソースコードのリスティングが記載されている。
【００３６】
バウンディングボリューム階層を用いることで、レイトレーシング対象の各物体は一度だけ参照される。結果として、又、3次元ツリーとは対照的に、階層を走査する間に1本の光線による1つの物体の多交差を防止するためのメールボックス機能を必要としない。これはシステム性能の観点からみて大きな利点であり、共用記憶域システムにおける実装を非常に単純にする。2番目の重要な結果としては、バウンディングボリューム階層のツリーにおいて、レイトレーシング対象となる物体総数よりも内部ノードが多くなることがない。従って、アクセラレーションデータ構造のメモリ使用量は、構築前に物体の数において線形拘束可能である。こうしたアプリオリな拘束は、メモリの複雑性がレイトレーシング対象となる物体の数に対して二次式的に増大すると考えられる3次元ツリーの構築に関しては利用できない。
【００３７】
従って、本明細書では、現在の3次元ツリーレイトレーシング技術よりかなり高速なバウンディングボリューム階層の新規な概念であって、レイトレーシング対象となる物体の数の増加につれ、メモリ要件が予想通り二次式的に増大するのでなく線形に増大する新規な概念を記載する。バウンディングボリューム階層が3次元ツリーより性能が優れている概念の核心は、バウンディングボリュームそのものに注目するのでなく、どのように空間

を区分できるかに注目した点である。

【００３８】

3次元ツリーでは、バウンディングボックスは単一の平面で区分される。本発明のこの様態によれば、2つの平行な平面を用いて2つの座標軸に平行なバウンディングボックスを定義する。図5は基本データ構造400を示す図である。

【００３９】

図5は座標軸に平行なバウンディングボックス402を正面図で示す。L平面402及びR平面404は座標軸に平行且つ互いにも平行であり、バウンディングボックス402を座標軸に平行な左右のバウンディングボックスに区分するのに用いられる。左バウンディングボックスは、元々のバウンディングボックス402の左側壁406からＬ平面402まで延伸する。右バウンディングボックスは、元々のバウンディングボックス402の左側壁408からR平面404まで延伸する。従って、左右のバウンディングボックスは互いに重なり合っていてもよい。光線412の走査を決定するには、光線410の妥当な区間[N,
F]412についてのL及びR平面402及び404との交差位置を用いる。

【００４０】

図5のデータ構造400では、L及びR平面402及び404の互いに対する位置は、元々のバウンディングボックス402の空間というよりは、バウンディングボックス402内に含まれる物体の集合を区分するように決定される。3次元ツリー区分とは対照的に、2つの平面を備えることで、これら2平面間の空の空間を最大化するという可能性が与えられる。結果的に、シーンの境界を近似する速度がかなり向上する。

【００４１】

図6乃至8は、データ構造400を更に示す一連の3次元図である。図6はバウンディングボックス402の図を示す。例示目的で、バウンディングボックス402内の仮想物体は抽象円416として示した。図7及び8に示したように、Ｌ平面404及びR平面406を用いてバウンディングボックス402を左バウンディングボックス402a及び右バウンディングボックス402bに区分する。Ｌ及びR平面は、その間の空の空間が最大化されるように選択する。それぞれの仮想物体416は左バウンディングボックス402aか右バウンディングボックス402bの何れかに入る。図8の底部に示したように、仮想物体416は「左」物体416aと「右」物体416bとに区分される。得られるバウンディングボックス402a及び402bそれぞれも区分され、これは終了判定基準が満たされるまで継続される。

【００４２】

図9は上述の方法500のフローチャートである。ステップ501で、シーンのバウンディングボックスが計算される。ステップ502では、平行なＬ及びR平面を用いて座標軸に平行なバウンディングボックスを、一部重複可能な座標軸に平行な左右のバウンディングボックスに区分する。ステップ503では、左右のバウンディングボックスを用いて、元々の座標軸に平行なバウンディングボックス内に含まれた仮想物体の集合を、左仮想物体の集合と右仮想物体の集合とに区分する。ステップ504では、左右の物体の処理は、終了判定基準が満たされるまで反復的に継続する。

【００４３】

従来の実装で用いられていたように1つの分割パラメータではなく、2つの分割パラメータが1つのノード内に記憶される。ノードの数はレイトレーシング対象となる物体の数により線形拘束されるので、全ノードのアレイは一度に割り当て可能である。従って、構築時における3次元ツリーの高額なメモリ管理は不要となる。

【００４４】

この構築技術は3次元ツリー構築のアナログより遙かに単純であり、再帰的な方法で或いは反復的バージョン及びスタックを用いて容易に実装できる。物体のリスト及び座標軸に平行なバウンディングボックスが与えられれば、Ｌ及びR平面が決定され、物体の集合がそれに従って決定される。すると、左右の物体は、終了判定基準が満たされるまで反復的に処理される。内部ノードの数は拘束されているので、1つの物体しか残っていないときに終了に依存しても安全である。

【００４５】
この区分は、これは非常に効率的且つ数値的に絶対安定である、x、y、及びz軸に垂直な平面に沿った物体のソートのみに依存することに注目されたい。3次元ツリーとは対照的に、ここでは、コスト高で且つ数値的にロバストな方法で実行するのが困難である、物体の分割平面との正確な交差を計算する必要がない。頂点及び辺において外れた三角形(原語：missed
triangle)などの3次元ツリーの数値問題は、バウンディングボリューム階層の構築に先立って三角形を拡張することで回避できる。又、3次元ツリーでは、一部重複した物体は座標軸に平行な左右のバウンディングボックス両方にソートしなければならず、ツリーの予想される二次曲線的な増大を引き起こす。

【００４６】
Ｌ及びR平面と従って実際のツリーレイアウトとを決定するには多くの技法を利用できる。図6乃至8を再度参照すると、1つの技法としては、上述した3次元ツリー構築技法を用いて平面M418を求め、座標軸に平行な新たなバウンディングボックスの得られたＬ及びR平面の重複部分と提案分割平面M418との重複が最小となるように物体を区分するものがある。得られるツリーは対応する3次元ツリーに非常に似ているが、空間でなく物体の集合が区分されるので、得られるツリーはずっと平坦になる。別のアプローチとしては、可能なら子ボックスの重複が最小となり且つ空の空間が最大化されるようにR及びＬ平面を選択するものがある。物体によっては座標軸に平行なバウンディングボックスが不効率となることは注目すべきである。こうした状況の例としては、座標軸に平行なバウンディングボックスの対角線上に位置した、半径が短い長尺円筒がある。

【００４７】
図10は、本発明のこの様態による方法600のフローチャートである。ステップ601で、シーンのバウンディングボックスが計算される。ステップ602で、3次元ツリー構築を実行して分割平面Mを決定する。ステップ603では、平行なＬ及びR平面を用いて座標軸に平行なバウンディングボックスを、分割平面Mとの重複が最小となる座標軸に平行な左右のバウンディングボックスに区分する。ステップ604では、左右のバウンディングボックスを用いて、元々の座標軸に平行なバウンディングボックス内に含まれた仮想物体の集合を、左仮想物体の集合と右仮想物体の集合とに区分する。ステップ605では、左右の物体の処理は終了判定基準が満たされるまで反復的に継続する。図3に示した方法200と同様に、図10に示した方法600を、3次元ツリー構築、実時間処理、バケットソート、自己交差などに関する技術を含んだ本明細書に記載した他の方法と組み合わせてよいことに注目されたい。

【００４８】
3次元ツリーの場合は、空間的細分化を継続して物体周辺の空間の空部分を切除する。上述したバウンディングボリューム階層の場合、こうした物体をより小さいものに区分すると類似の振る舞いとなる。メモリ要件の予測可能性を維持するため、バウンディングボックスの最大サイズが定義される。バウンディングボックス最大サイズを超える範囲を備えた全物体はこの要件を満たすようにより小さな部分に分割される。最大許容サイズは、データセットを走査して全ての物体における最小範囲を求めることにより発見できる。

【００４９】
本明細書で記載したデータ構造により、高速3次元ツリー走査の原理をバウンディングボリューム階層に移行できる。走査の場合分け(原語：case)は似ており、次の通り。(1)左の子のみ、(2)右の子のみ、(3)左の子の後で右の子、(4)右の子の後で左の子、又は(5)光線が分割平面の間にある(すなわち空の空間)。上述した技術の1つのノードは2つの平行平面で分割されるので、複数ボックスをどのように走査するかという順序は光線の方向により決定される。図6は、上述の技法を組み込んだソースコードリスティングを記載したものである。

【００５０】
以前のバウンディングボリューム階層技術は、子ノードをどようにに走査するかという順序やヒープデータ構造を更新するなど必要な付加的努力を効率的に決定できなかった。更

に、本アプローチでは2平面距離を必要とするにすぎないが、以前の技術ではバウンディングボリューム全体をロードして、光線に対してテストする必要があった。しかし、ソフトウェアにおいて光線を2つの平面に対してチェックすれば、高価になると思われる。この走査が3次元ツリーにおけるボトルネックとなっており、ここで更なる計算をよりうまく実行することがメモリアクセスの待ち時間を隠す。更に、バウンディングボリューム階層ツリーは、同一性能の対応する3次元ツリーよりもかなり小さくなる傾向にある。

【００５１】

新たなバウンディングボリューム階層について本明細書で記載したが、3次元ツリーの走査への強いリンクがある。すなわち、L = Rと設定すると、古典的な二分空間区分が得られ、走査アルゴリズムは3次元ツリーの走査アルゴリズムに圧縮される。

【００５２】

上述したバウンディングボリューム階層も適用して、自由曲面を細分化することで光線と自由曲面との交点を効率的に見つけることができる。そうすることにより、実際の浮動小数点演算によるが、自由曲面と凸閉包特性との交点及び細分化アルゴリズムが浮動小数点精度まで計算できる。1つの細分化ステップが、例えば、多項式表面、有利表面、及び近似細分化表面に関して実行される。空間内の各軸に関して、重複の可能性があるバウンディングボックスが上述のように決定される。二分細分化の場合は、新たなメッシュの新たなバウンディングボックスに関するLボックスの交点及びRボックスの交点。ボックスの空間的順序が知られているので、ここで上述の走査を効率的に実行できる。バウンディングボリュームの階層を予備計算する代わりに、これを処理途中で計算できる。この手法は自由曲面に関して効率的であり、アクセラレーションデータ構造のメモリ節約になり、これはバックトラッキングにより走査しなければならない小さなスタックのバウンディングボリュームに置き換えられる。この細分化は継続され、光線と表面との交点が、浮動小数点精度内の点又は小サイズの区間に圧縮された(原語：collapsed)バウンディングボリューム内に位置するまで終了しない。付属文書の段落2.2.1には、本発明のこの側面に従ったコードのリスティングが記載されている。

【００５３】

標準グリッドをレイトレーシングのアクセラレーションデータ構造として使用することは単純だが、空間適応性の不足及び多数の空グリッドセルの走査により効率が犠牲となる。階層標準グリッドはこの状況を改善できるが、バウンディングボリューム階層及び3次元ツリーと比べてなお劣る。しかし、標準グリッドを用いてアクセラレーションデータ構造の構築速度を改善できる。アクセラレーションデータ構造を構築するための技術はクイックソートと似ており、O(n log n)で実行すると考えられる。線形時間で実行するバケットソートを適用することで改善が得られる。従って、物体の座標軸に平行なバウンディングボックスは$n_x \times n_y \times n_z$個の座標軸に平行なボックスに区分される。次に、各オブジェクトは、たとえば重心などの選択した一点によりこれらボックスの１つに厳密にソートされる。あるいは、各三角形の第１頂点を用いてもよい。すると、各グリッドセルにおける物体の実際の座標軸に平行なバウンディングボックスが特定される。これら座標軸に平行なバウンディングボックスが内包する物体でなく、これらバウンディングボックスを用いるが、それはバウンディングボックスが分割平面の何れかと交差しない限りにおいてである。そうなった場合は、ボックスはアンパックして、ボックス内の物体を直接使用することになる。この手順は多くの比較及びメモリアクセスを不要とし、構築技法の順序の定数を有意に向上し、更に、反復的に適用できる。上述の技法は物体のストリームを処理することで実現できるので、特にハードウェア実装に魅力的である。

【００５４】

アクセラレーションデータ構造は要求時に、すなわち光線が物体を備えた座標軸に平行なバウンディングボックスを走査する時に構築できる。そして、アクセラレーションデータ構造は、光線に対して透明な空間領域でリファインされることは決して無く、触れられ

ていないデータによってキャッシュが汚染することがない。他方で、リファイン後、光線と交差する可能性がある物質は既にキャッシュに入っている。

【００５５】

上述の説明から、本発明がレイトレーシングにおいて長らく知られていた問題に対処し、アクセラレーションデータ構造の精度、全体的速度、及びメモリ使用量が向上したレイトレーシングの技術を提供することが分かるはずである。数値精度の向上は他の数体系にも、例えばARTレイトレーシングチップのハードウェアで用いられる対数数体系にも伝達する。プロセッサ又は専用ハードウェアに関するIEEE浮動小数点標準の実装が性能に甚大な影響を与えることがある。例えば、ペンティアム（登録商標）４チップでは、非正規化数が性能を100倍以上低下させることがある上述のように、本発明の実装はこれら例外を回避する。本明細書で記載したバウンディングボリューム階層のビューにより、これら階層が実時間レイトレーシングに適したものとなる。ならし解析では、本明細書で記載した技法は従来の技術水準を能力面で凌いでおり、より精密な技術を、例えば製作現場などにおいて完全に動画表現されたシーンで動きのぼかし計算に使用できる。特にハードウェア実装において更に巨大なシーンに関し、ここで記載したバウンディングボリューム階層が、3次元ツリー及び他の技法に比べて、大きな利点を備えていることは上述の記載から明らかなはずである。ならし解析では、ここで記載したバウンディングボリューム階層は、現在の3次元ツリーを性能において少なくとも2倍上回っている。更に、メモリ使用量は前もって特定でき、物体の数に線形である。

【００５６】

上述の記載は当業者による本発明の実施を可能とする詳細情報を含んでいるが、この記載は例示的なものであり、本明細書の教示を知っている当業者には多くの修正及び変形が可能なことは明らかなはずである。従って、本発明は添付の請求の範囲によってのみ定義され、請求の範囲は従来技術から見て可能な限り広く解釈されるよう意図されている。

コンピュータ・プログラム付属文書

コードリスティング 2.2.1

```
void Triangle::Transform()
{
Point *p = (Point *)this;
Vector n3d;
Vector n_abs = n3d = (p[1]-p[0])|(p[2]-p[0]);
// 投影の最大成分を検索 (0=x,1=y,2=z)
uintCast(n_abs.dx) &= 0x7FFFFFFF;
uintCast(n_abs.dy) &= 0x7FFFFFFF;
uintCast(n_abs.dz) &= 0x7FFFFFFF;
// 退縮三角形を扱う必要がある(エッジサインを設定)
if(!((n_abs.dx+n_abs.dy+n_abs.dz) > DEGEN_TRI_EPSILON))
//(!(...) > EPS) NaN's を扱う
{
d = p[0].x;
p0.u = -p[0].y;
p0.v = -p[0].z;
n.u=n.v = 0.0f;
e[0].u = e[1].u = e[0].v = e[1].v = 1.0f;
return;
}
U32 axis = 2;
if(n_abs.dx > n_abs.dy)
{
if(n_abs.dx > n_abs.dz)
axis = 0;
}
else if(n_abs.dy > n_abs.dz)
axis = 1;
Point p03d = p[0];
Point p13d = p[1];
Point p23d = p[2];
float t_inv = 2.0f/n3d[axis];
e[0].u = (p23d[PlusOneMod3[axis]]-p03d[PlusOneMod3[axis]])*t_inv;
e[0].v = (p23d[PlusOneMod3[axis+1]]-p03d[PlusOneMod3[axis+1]])*t_inv;
e[1].u = (p13d[PlusOneMod3[axis]]-p03d[PlusOneMod3[axis]])*t_inv;
e[1].v = (p13d[PlusOneMod3[axis+1]]-p03d[PlusOneMod3[axis+1]])*t_inv;
t_inv *= 0.5f;
n.u = n3d[PlusOneMod3[axis]] *t_inv;
n.v = n3d[PlusOneMod3[axis+1]]*t_inv;
p0.u = -p03d[PlusOneMod3[axis]];
p0.v = -p03d[PlusOneMod3[axis+1]];
d = p03d[axis] + n.u*p03d[PlusOneMod3[axis]] + n.v*p03d[PlusOneMod3[axis+1]];
}
```

コードリスティング 2.2.2

```
U32 *idx = pointer_to_face_indices;
U32 ofs = projection_case;
for(U32 ii = num_triData; ii; ii--,idx++)
{
float t = (triData[*idx].d - ray.from[ofs]
- triData[*idx].n.u*ray.from[PlusOneMod3[ofs]]
- triData[*idx].n.v*ray.from[PlusOneMod3[ofs+1]])
/ (ray.d[ofs] + triData[*idx].n.u*ray.d[PlusOneMod3[ofs]]
+ triData[*idx].n.v*ray.d[PlusOneMod3[ofs+1]]);
if(uintCast(t)-1 > uintCast(result.tfar)) //+0.0f は-1
continue;
float h1 = t*ray.d[PlusOneMod3[ofs]] + ray.from[PlusOneMod3[ofs]]
+ triData[*idx].p0.u;
float h2 = t*ray.d[PlusOneMod3[ofs+1]] + ray.from[PlusOneMod3[ofs+1]]
+ triData[*idx].p0.v;
float u = h1*triData[*idx].e[0].v - h2*triData[*idx].e[0].u;
float v = h2*triData[*idx].e[1].u - h1*triData[*idx].e[1].v;
float uv = u+v;
if((uintCast(u) | uintCast(v) | uintCast(uv)) > 0x40000000)
continue;
result.tfar = t;
result.tri_index = *idx;
}
```

コードリスティング 2.3.1

```
Point *p = (Point *)&triData[tri_index];
int boxMinIdx, boxMaxIdx;
// boxMinIdx と boxMaxIdx は三角形の最小及び最大頂点にインデックスを付ける
// 分割平面の成分 dir[0]において
if(p[0][dir[0]] < p[1][dir[0]])
{
if(p[2][dir[0]] < p[0][dir[0]])
{
boxMinIdx = 2;
boxMaxIdx = 1;
}
else
{
boxMinIdx = 0;
boxMaxIdx = p[2][dir[0]] < p[1][dir[0]] ? 1 : 2;
}
}
else
{
if(p[2][dir[0]] < p[1][dir[0]])
{
boxMinIdx = 2;
boxMaxIdx = 0;
}
else
{
boxMinIdx = 1;
```

```
boxMaxIdx = p[2][dir[0]] < p[0][dir[0]] ? 0 : 2;
}
}
/* 三角形が分割平面内にあるか分割平面の一方の側に完全に位置していることが数値エラー無しで、
すなわち三角形が描画システムに入力される精度で決定できる。ここでイプシロンを使用するのは誤り
且つ不要。
*/
if((p[boxMinIdx][dir[0]] == split) && (p[boxMaxIdx][dir[0]] == split)) // 分割平
面内 ?
{
on_splitItems++;
if(split < middle_split) // より小さなボリュームへ
left_num_divItems++;
else
{
unsorted_border--;
U32 t = itemsList[unsorted_border];
right_num_divItems--;
itemsList[right_num_divItems] = itemsList[left_num_divItems];
itemsList[left_num_divItems] = t;
}
}
else if(p[boxMaxIdx][dir[0]] <= split) // 三角形は完全に左か ?
left_num_divItems++;
else if(p[boxMinIdx][dir[0]] >= split) // 三角形は完全に右か ?
{
unsorted_border--;
U32 t = itemsList[unsorted_border];
right_num_divItems--;
itemsList[right_num_divItems] = itemsList[left_num_divItems];
itemsList[left_num_divItems] = t;
}
else
// ここで詳細な決定、三角形は分割平面に交差していなければならない ...
{
/* その後、三角形が左及び／又は右へ行くかを判断する。三角形は線分で分割平面に交差しなければなら
ないことは既に分かっている。
全ての計算は、より高精度の計算が最初になされるように順序づけられている。スカラー積及び外積は最後に評価される。
状況によっては、バウンディングボックスをイプシロンで拡張する必要がありうる。しかし、これは必要メモリを大幅に増大させる。
こうした状況に遭遇した場合、イプシロンを使用しないため数値的に分析した方がいいかもしれない。...
この時点で、 p[boxMaxIdx][dir[0]] < split < p[boxMaxIdx][dir[0]] 且つ
p[boxMidIdx][dir[0]] \in [p[boxMaxIdx][dir[0]], p[boxMaxIdx][dir[0]]]であることが
分かっている。
三角形は現在のボクセルとの空でない交差を備えていることも分かっている。三角形は分割平面内に位置できないし、
その頂点は一方の側に位置することもできない。
*/
int boxMidIdx = 3 - boxMaxIdx - boxMinIdx; // ミッシングインデックス、3 = 0 + 1 + 2
により見つけられる
/* ここで分割平面の一方の側に単独で存在する頂点を特定する。
この孤立頂点が左右のどちら側に位置しているかによっては、三角形が右か左かどちらに行くかの決定を後にスワップする必
要がある。
*/
int Alone = (split < p[boxMidIdx][dir[0]]) ? boxMinIdx : boxMaxIdx;
```

```
int NotAlone = 3 - Alone - boxMidIdx;
// == (split < p[boxMidIdx][dir[0]]) ? boxMaxIdx : boxMinIdx;
// idx の和 = 3 = 0 + 1 + 2 なので。
float dist = split - p[Alone][dir[0]];
U32 swapLR = uintCast(dist)>>31; // == p[Alone][dir[0]] > split;
/* ここで、孤立頂点を残りの 2 つの頂点と連結する線分に分割平面が交差する。a1 及び a2 は交点。
特別なケース"if(p[boxMidIdx][dir[0]] == split)"[最適化可能な a x / x を与える] はメッシュの頂点の最高価と同じ頻度で起こりうるにすぎないので、何の助けともならない...
*/
float at = dist / (p[boxMidIdx][dir[0]] - p[Alone][dir[0]]);
float at2 = dist / (p[NotAlone][dir[0]] - p[Alone][dir[0]]);
float a1x = (p[boxMidIdx][dir[1]] - p[Alone][dir[1]]) * at;
float a1y = (p[boxMidIdx][dir[2]] - p[Alone][dir[2]]) * at;
float a2x = (p[NotAlone][dir[1]] - p[Alone][dir[1]]) * at2;
float a2y = (p[NotAlone][dir[2]] - p[Alone][dir[2]]) * at2;
// n は三角形の交差 a1a2 の線に対して垂直のベクトルである。
// 更に分割平面にも。
float nx = a2y - a1y;
float ny = a2x - a1x;
// 符号は、交差線に対して垂直のベクトルの四分円を示す
U32 nxs = uintCast(nx)>>31; // == (nx < 0.0f)
U32 nys = uintCast(ny)>>31; // == (ny < 0.0f)
/* 数値的精度: 消去により、桁が異なるものを加算する前に、概ね同じ指数の浮動を最初に減算すべきである。その後の全ての大括弧は数値的精度に不可欠である。
これらを変えると BPS でのエラーが増える...
pMin は原点を bBox.bMinMax[0]とした座標系における孤立点
pMax は原点を bBox.bMinMax[1]とした座標系における孤立点
*/
float pMinx = p[Alone][dir[1]] - bBox.bMinMax[0][dir[1]];
float pMiny = p[Alone][dir[2]] - bBox.bMinMax[0][dir[2]];
float pMaxx = p[Alone][dir[1]] - bBox.bMinMax[1][dir[1]];
float pMaxy = p[Alone][dir[2]] - bBox.bMinMax[1][dir[2]];
// バウンディングボックスの座標を求めるが、p + a1 を原点として計算。
float boxx[2];
float boxy[2];
boxx[0] = (pMinx + a1x) * nx;
boxy[0] = (pMiny + a1y) * ny;
boxx[1] = (pMaxx + a1x) * nx;
boxy[1] = (pMaxy + a1y) * ny;
/* 三角形と分割平面との交差線がバウンディングボックスの近くを通るかをテストする。これは、交差線に対して垂直のベクトルの四分円によってバウンディングボックスの座標にインデックスを付けることで実行する。これは「実時間描画(Real-Time Rendering)」という著書で Haines により紹介された 3 次元テストの巧みな実装である。
インデックス付けにより頂点が選択されるが、これらは交差線から最も遠い。
三角形は現在のボクセルと空でない交差を含んでいるはずなので、現在のボクセルを完全に通過できないことに注目されたい。
*/
U32 resultS;
if(pMinx + MAX(a1x,a2x) < 0.0f) // ボックスの左の交差 a1a2 の線分
resultS = uintCast(pMinx)>>31;
else if(pMiny + MAX(a1y,a2y) < 0.0f) // ボックスの下の交差 a1a2 の線分
resultS = uintCast(pMiny)>>31;
else if(pMaxx + MIN(a1x,a2x) > 0.0f) // ボックスの右の交差 a1a2 の線分
resultS = (pMaxx > 0.0f);
```

```
else if(pMaxy + MIN(a1y,a2y) > 0.0f) // ボックスの上の交差a1a2の線分
resultS = (pMaxy > 0.0f);
else if(boxx[1^nxs] > boxy[nys])
// 線はbboxを通過する ? => 三角形は一方の側のみに位置できる
resultS = (a1y*a2x > a1x*a2y);
// 外積a1 x a2 の符号をチェックしてどちらが側かを決定する
else if(boxx[nxs] < boxy[1^nys])
resultS = (a1y*a2x < a1x*a2y);
else
// ここで、三角形は左右両方にあるはず
{
stackList[currStackItems++] = itemsList[left_num_divItems];
unsorted_border--;
itemsList[left_num_divItems] = itemsList[unsorted_border];
continue;
}
if(swapLR != /*^*/ resultS)
{
unsorted_border--;
U32 t = itemsList[unsorted_border];
right_num_divItems--;
itemsList[right_num_divItems] = itemsList[left_num_divItems];
itemsList[left_num_divItems] = t;
}
else
left_num_divItems++;
}
```

コードリスティング 2.4.1

```
Intersection Boundary::Intersect(Ray &ray) //ray.tfar は変更される!
{
// 最適化された逆計算(3つの除算のうち2つを省く)
float inv_tmp = (ray.d.dx*ray.d.dy)*ray.d.dz;
if((uintCast(inv_tmp)&0x7FFFFFFF) > uintCast(DIV_EPSILON))
{
inv_tmp = 1.0f / inv_tmp;
ray.inv_d.dx = (ray.d.dy*ray.d.dz)*inv_tmp;
ray.inv_d.dy = (ray.d.dx*ray.d.dz)*inv_tmp;
ray.inv_d.dz = (ray.d.dx*ray.d.dy)*inv_tmp;
}
else
{
ray.inv_d.dx = ((uintCast(ray.d.dx)&0x7FFFFFFF) > uintCast(DIV_EPSILON)) ?
(1.0f / ray.d.dx) : INVDIR_LUT[uintCast(ray.d.dx) >> 31];
ray.inv_d.dy = ((uintCast(ray.d.dy)&0x7FFFFFFF) > uintCast(DIV_EPSILON)) ?
(1.0f / ray.d.dy) : INVDIR_LUT[uintCast(ray.d.dy) >> 31];
ray.inv_d.dz = ((uintCast(ray.d.dz)&0x7FFFFFFF) > uintCast(DIV_EPSILON)) ?
(1.0f / ray.d.dz) : INVDIR_LUT[uintCast(ray.d.dz) >> 31];
}
Intersection result;
result.tfar = ray.tfar;
result.tri_index = -1;
//
```

```
//BBox-チェック
//
float tnear = 0.0f;
worldBBox.Clip(ray,tnear);
if(uintCast(ray.tfar) == 0x7F7843B0) //ray.tfar==3.3e38f //!!
return(result);
//
U32 current_bspStack = 1; //空のスタックケース == 0 なので
U32 node = 0;
//
//BSP-走査
//
const U32 whatnode[3] = {(uintCast(ray.inv_d.dx)>>27) & sizeof(BSPNODELEAF),
(uintCast(ray.inv_d.dy)>>27) & sizeof(BSPNODELEAF),
(uintCast(ray.inv_d.dz)>>27) & sizeof(BSPNODELEAF)};
U32 bspStackNode[128];
float bspStackFar[128];
float bspStackNear[128];
bspStackNear[0] = -3.4e38f; // 標識
do
{
//ノードは葉か (type<0) 又は分岐か(type>=0)
while (((BSPNODELEAF&)bspNodes[node]).type >= 0)
{
//分割-次元(x|y|z)
U32 proj = ((BSPNODELEAF&)bspNodes[node]).type & 3;
float distl = (((BSPNODELEAF&)bspNodes[node]).splitlr[whatnode[proj]>>4]
- ray.from[proj])*ray.inv_d[proj];
float distr = (((BSPNODELEAF&)bspNodes[node]).splitlr[(whatnode[proj]>>4)^1]
- ray.from[proj])*ray.inv_d[proj];
node = (((BSPNODELEAF&)bspNodes[node]).type - proj) | whatnode[proj];
//タイプ & 0xFFFFFFF0
if(tnear <= distl)
{
if(ray.tfar >= distr)
{
bspStackNear[current_bspStack] = MAX(tnear,distr);
bspStackNode[current_bspStack] = node^sizeof(BSPNODELEAF);
bspStackFar[current_bspStack] = ray.tfar;
current_bspStack++;
}
ray.tfar = MIN(ray.tfar,distl);
}
else
if(ray.tfar >= distr)
{
tnear = MAX(tnear,distr);
node ^= sizeof(BSPNODELEAF);
}
else
goto stackPop;
}
//
//面-交差
... code omitted ...
```

```
//
//
//ヒットは見つかったか?
//
do //!! NEEDS bspStackNear[0] = -3.4e38f; !!!!!
{
stackPop:
current_bspStack--;
tnear = bspStackNear[current_bspStack];
}while(result.tfar < tnear);
if(current_bspStack == 0)
return(result);
node = bspStackNode[current_bspStack];
ray.tfar = bspStackFar[current_bspStack];
} while (true);
}
```

【図面の簡単な説明】

【００５７】

【図１】本発明を導入可能な従来のディジタル処理システムの概略図である。

【図２】本発明を導入可能な従来のパーソナルコンピュータ又は類似の計算装置の概略図である。

【図３】本発明の第1の様態による包括的方法を示した流れ図である。

【図４】「自己交差」問題を示したレイトレーシング手順の図である。

【図５】本発明の別の様態によるアクセラレーションデータ構造として用いられる座標軸に平行な区分バウンディングボックスの正面図である。

【図６】L及びR平面におけるバウンディングボックスの区分を示した、図５の座標軸に平行なバウンディングボックスの一連の等角図である。

【図７】L及びR平面におけるバウンディングボックスの区分を示した、図５の座標軸に平行なバウンディングボックスの一連の等角図である。

【図８】L及びR平面におけるバウンディングボックスの区分を示した、図５の座標軸に平行なバウンディングボックスの一連の等角図である。

【図９】本発明の更なる様態によるレイトレーシング方法のフローチャートである。

【図１０】本発明の更なる様態によるレイトレーシング方法のフローチャートである。

【図１】

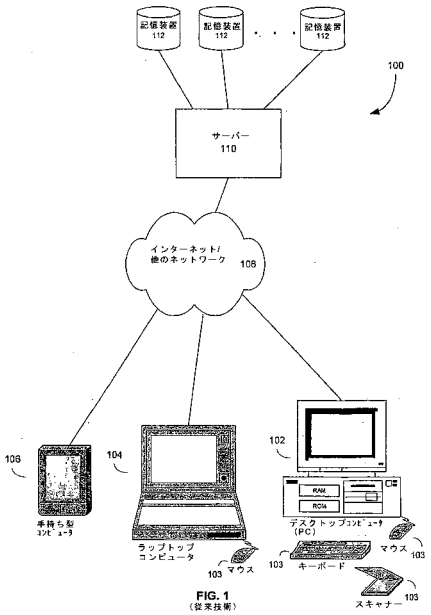

FIG. 1
（従来技術）

【図２】

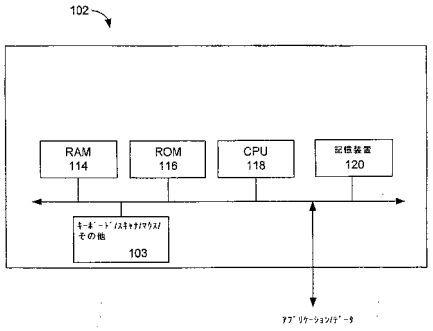

FIG. 2
（従来技術）

【図３】

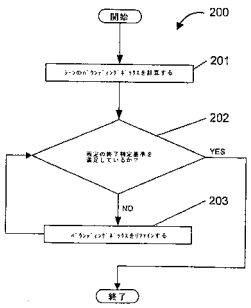

FIG. 3

【図４】

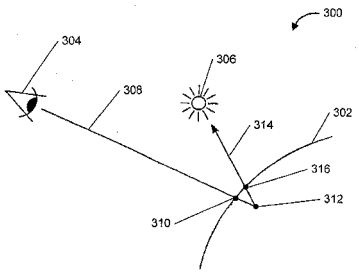

FIG. 4

【図５】

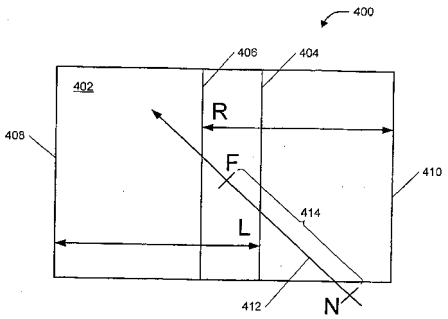

FIG. 5

【図６】

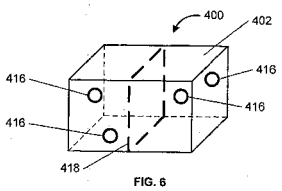

FIG. 6

【図７】

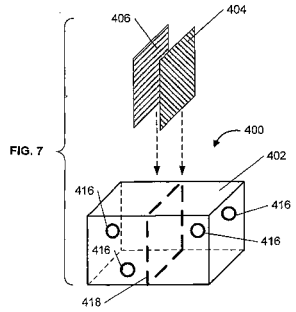

FIG. 7

【図８】

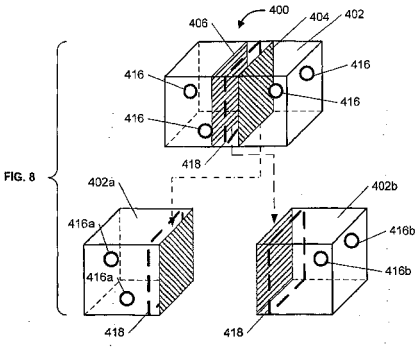

FIG. 8

【図９】

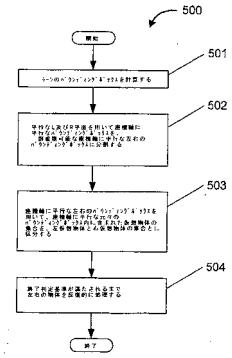

FIG. 9

【図１０】

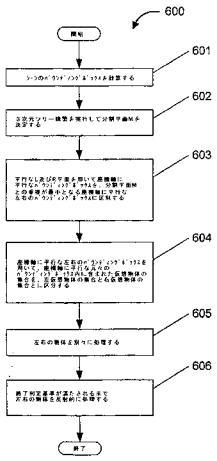

FIG. 10

フロントページの続き

(72)発明者　バエヒター,　カーステン
ドイツ連邦共和国　ベルリン　１０６２３　ファサンエンストラッセ　８１

審査官　岡本　俊威

(56)参考文献　特開２００３－２７１９８８（ＪＰ，Ａ）
特表２００２－５０４７２９（ＪＰ，Ａ）

(58)調査した分野(Int.Cl.，ＤＢ名)
G06T　15/00-15/87