〔公　開〕

ＴＲ－Ｃ－０１２６

階層的空間表現を用いた３次元物体間の実時間インタラクションに関する研究

北村　喜文
Yoshifumi KITAMURA

１９９６　２．２６

ＡＴＲ通信システム研究所

階層的空間表現を用いた3次元物体間の実時間インタラクションに関する研究

Real-time Interactions among 3-D Objects using Hierarchical Spatial Subdivision

北村 喜文
Yoshifumi Kitamura

1996年2月

もくじ

図一覧

表一覧

あらまし

本論文では，完全な仮想現実のシステムを構築するための要素技術として，３次元物体間の受動的相互作用，能動的相互作用，人為的相互作用の３種類の相互作用について述べる．まず，３次元物体間の受動的相互作用について，並進と回転を含む運動をする複数の３次元物体間の衝突面を，octree を用いて効率的に検出する方法を提案する．手法は，静止物体と運動物体を区別することなく，複数の複雑な形状の移動物体が存在する空間中で，連続な運動中に生じる衝突物体を離散時刻の検査によって見逃すことなく選びだし，さらに効率的に衝突面を特定することができる．物体の形状としては凸と凹両方の形状を許し，かつどちらも特別扱いしない．続いて，並列計算機を用いることによって実時間で，複雑な形状の運動物体間の衝突面を検出する方法について述べる．

次に，３次元物体間の能動的相互関係について，移動対象物体（ロボット），静止障害物，移動障害物の環境中の全ての物体を，その運動可能性を区別することなく同一の枠組 (octree) で表現する効率的な経路計画の方法を提案する．手法は，簡単なアルゴリズムでも効率的に３次元の動的環境での経路が発見されることが特徴であるが，それを支える技術の１つとして，並列計算機を用いて，回転と並進を含む任意の運動にともなって octree を実時間で更新する方法について述べる．

最後に３次元物体間の人為的相互作用について，実時間衝突面検出の方法によって検出された衝突面を用いることにより，複雑な形状の物体の接触面を動的に決定し，この面で物体の視覚的な運動を拘束する仮想物体操作を補助する方法について述べる．本手法は力フィードバック機構など特別なハードウエアを用いる必要がなく，自然なユーザインタフェースを提供する．これにより，操作者は仮想物体の配置操作を精度良く，また効率的に遂行することができる．

第 1 章

諸 論

1.1 仮想現実のシステム

仮想現実の技術は，人間の 3 次元空間知覚能力と日常生活を営む実世界での体験を利用して，直観的で洗練されたユーザインタフェースとして利用されることが多くなってきた[1]．これは，計算機の中に構築された世界に，視覚・聴覚・触覚などの感覚チャネルと自らの運動を通して，人が直接働きかける機能を持つ．その働きかけを実現する道具として，従来からあるキーボードやマウス，ダイヤルボックス，2 次元ディスプレイといった古典的な装置ではなく，人の身振りや手振りなどを直接入力するグローブ型のデバイスや，動きを検出するための 6 自由度のトラッカー，両眼視差などによって奥行き感を明確化する立体視ディスプレイなど，人間の 3 次元空間知覚・運動能力を最大限に生かすことができる空間型のインタフェース装置を用いてシステムが構成されることが特徴である．

こうした研究の初期のものに，Put-that-there [Bol80] がある．これは，3 次元空間内の位置と方向を検出できる磁気センサを指先に付け，スクリーンに映し出された絵を指でさすことによって，人が選択したい物体を計算機に入力することができた．また同時に音声認識をも利用することによって，「その物体をあそこに置け」といったように，選択した物体に対して操作をすることができた．同様のシステムはその後多く研究されてきたが [HMLS90, 野村 90, 平田 93]，その 1 つとして臨場感通信会議システム [岸野 89, 岸野 92, MKT95] がある．これは，遠隔地にいる人々にあたかも一堂に会するかのような視覚・聴覚的な相互作用を与え，現実の面談会議と同様の会議の場と同時に，現実には実現不可能な仮想的な協調作業の場を提供

[1]最近の仮想現実の技術を紹介した書籍は多いが，たとえば，[服部 91] [舘 92] [廣瀬 93] [廣瀬 95] [PT93] [Wex93] [Ste94] [BF95] [Vin95] [EVJ95] などがある．また論文特集号 [特集ロ 92] [特集電 95] [特集テ 95] [CGA95] などもある．

しようとするものである. (**Fig. 1.1**に試作システムを示す.) そのため操作者には, 視点追従型立体表示装置によって生成された奥行き感と運動視差をもった歪みのない仮想環境の画像を提示し, 6自由度の位置・方向検出装置を付加したグローブ型デバイスなどを操作することにより, 仮想物体を把持, 運搬, 回転, 解放などの直接操作をすることができる [TK92]. これは3次元環境における人の空間知覚能力を活かして仮想物体の配置操作を実行することができる一般的な構成の1つである.

Fig. 1.1: The Virtual Space Teleconferencing System.

以上のような仮想現実のシステムを分類する方法として有名なものに, AIP キューブ [Zel92] がある. これは **Fig. 1.2**のように, Autonomy, Interaction, Presence を3つの軸にしてシステムの特性を整理しようとするものである. Autonomy はシステムの自立性を示す軸であり, 計算機の中の世界の要素 (物体など) に能動的な挙動が可能かどうかの指標である. Interaction は, 計算機中の世界の要素の挙動を決定するための働きかけを表す軸であり, その仕組みとしてシステムと操作者の間の相互作用と, 要素間の相互作用がある. Presence は, 視覚・聴覚・触覚などの情報チャネルを通じて, 人が計算機が作り出す仮想の世界を自然なものとして受け入れ, また操作できるかどうかを表す軸である. 各軸上でその項目がなければ 0, 十分にあれば 1 として, さまざまなシステムをこの AIP キューブ内で分類することができ, 各軸の値が $(1, 1, 1)$ となるシステムを, 完全な仮想現実システムと呼んでいる.

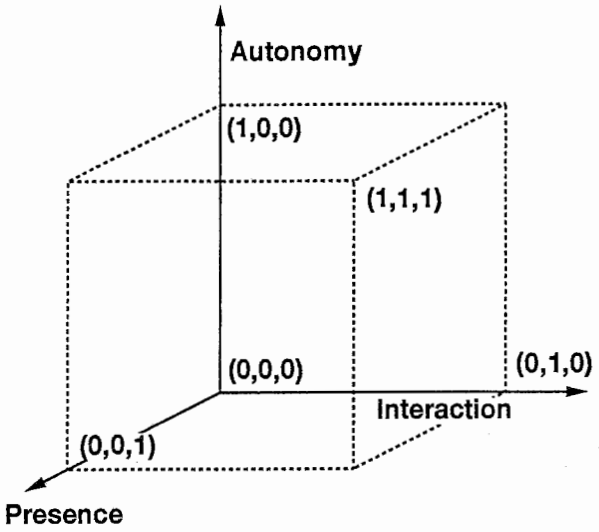


Fig. 1.2: The AIP cube.

AIP キューブで (1, 1, 1) となる完全な仮想現実のシステムを用いれば，仮想空間内の箱を手で直接つかんで机の上に置くというような仮想世界の作業を，実世界の作業と同じように簡単かつ効率的に遂行することができる．ただしこのためには，物体間の干渉の検出と回避，仮想物体に働く重力，物体と机の間の摩擦，反力の提示，接触音の生成など，物体間の相互作用を計算し，正確に模擬することが必要である．しかし計算機能力の限界から，通常は一部の限定された物理法則などのみを模擬することが多く，そのため実世界と全く違わない仮想環境を生成することは困難である．すなわち，Autonomy，Interaction，Presence のいずれかが不十分なシステムとなり，したがって実世界では簡単な作業も，仮想環境ではしばしば熟練を要する特殊な技能が必要な作業となっていた．

この問題を解決し，完全な仮想現実のシステムを構築するための 1 つの方法は，仮想空間内の個々の要素（物体）間の相互作用の計算を効率化・高速化し，実時間でこれらを処理することである．上のような仮想空間を利用した物体配置操作を行うシステムの場合，3 次元物体間の相互作用は，次の 3 種類に分類できる．

1. 物体の間に接触や衝突が発生した場合など，外力が働いた場合の受動的相互作用
2. 受動的要因の発生に依らず，3 次元物体が能動的に自らの挙動を決定する能動的相互作用
3. 3 次元物体の運動に人が介在した場合の，人為的相互作用

これら3種類の3次元物体間の相互作用それぞれについて，次節で詳しく述べる．

1.2 3次元物体間の相互作用

1.2.1 物体間の受動的相互作用

物体間の受動的相互作用は，仮想空間内の物体の挙動を決定するための仕組みの1つである要素間の相互作用として，AIP キューブの Interaction の軸に関する項目である．この受動的相互作用を考える上で最も重要な問題は，物体間の接触・衝突を検出する問題である．複雑な3次元物体間の衝突を正確に実時間で検出することができれば，3次元空間内で運動物体が接触した場合の挙動について，様々なシミュレーションを行なうことが可能となる．3次元物体間の衝突検出の問題は，ロボティックス，コンピュータグラフィックスなど多くの分野で中心的な問題の1つとして扱われてきたが，特に複雑な形状の運動物体を多く含むような3次元環境で，物体間の衝突を正確に検出する問題は，その計算量の多さから実時間で処理することが困難であった [Pen90, Hah88]．しかしながら，接触や衝突が発生した場合など，外力が働いた場合の物体間の受動的相互作用を正確に模擬するためには，解決しなければならない問題である．

物体の多面体表現を用いる方法は最も一般的な形状表現法の1つであり，比較的正確に面間の衝突を検出することができる [Boy79]．しかしこの方法では，衝突面検出の計算量は物体の数が多くなるにつれ，また物体形状が複雑になるにつれて増加してしまう．そこで，効率的な衝突面検出の方法についてもいくつか提案されている．[LC91, LMC94, CLMP95] は，物体の形状を凸剛体に限定して，サンプリングされる時間内の物体の移動量が微小で連続的であり，最近接の辺 - 面ペアは変化しないと仮定し，これらの間の衝突検出に利用した．物体間の距離に基づく方法は他にもいくつか見られる [GJK88, Qui94]．一方，サンプリングされる時間内の物体の移動量が微小であると仮定して，2物体を分離する平面を用いて衝突が生じてるか否かを素早く判定する方法 [Bar90] が提案されている．また全体の計算量を軽減するため，疎密的な方法を採用したものとして，階層的な外接図形を用た例 [Hah88] や，voxel，octree などの体積表現を用いた例 [MW88, Tur89, ZPOM93, SH92, Hay86, GSF94] がある．更に，仮想空間における物体操作シミュレーションといったアプリケーションのためには，発生する物体間の衝突を見逃すことなく，また実際に衝突が起きる直前に正確にこれを検出することが必要であり，さまざまな試みがなされてきた [Boy79, Can86, 岡野 88].

1.2.2 物体間の能動的相互作用

物体間の能動的相互作用は，仮想空間内の物体の自立的・能動的挙動を実現する項目として，AIP キューブの Autonomy に関わる．つまり完全な仮想現実システムの構築へ向けて，衝突など外部から与えられる受動的要因の発生に依らず，個々の3次元物体が能動的に自らの挙動を決定することが求められる．そのために必要なことの1つとして，各3次元物体に対して，その初期位置・方向と目的位置・方向が与えられた時，この間をつなぎ，空間中に漂う他の物体（障害物）に衝突しない経路を発見する能力があげられる．この問題は，自律移動ロボットの経路計画として知られてきた [Lat91, HA92a, 藤村 93]．しかしこれらの多くは，主に2次元の環境を対象として良い結果を出すことが報告されているが，3次元またはそれ以上の次元をもつ環境に対しては多くの計算時間がかかることが指摘されてきた．しかし，3次元空間内の物体の能動的相互作用を正確に模擬するためには，効率的な経路計画の手法を確立する必要がある．

代表的な方法として，ポテンシャル場を用いたアプローチ [MA85, Kha86, HA92b] や，quadtree や octree など階層的な空間分割法 [Sam90] によって静的な環境を表現するアプローチ [KD86, Her86, NNA89] などがある．前者は，障害物からの斥力と目的地への引力をスカラー量であるポテンシャルを用いて表現し，ポテンシャルの谷間を縫うことによって障害物に接触しない初期位置から目的地までの経路を探索する．簡単な原理から多くの場合に良い結果を導くので，ポテンシャルの発生方法に関しても多くの研究例が見られる．たとえば楕円ポテンシャル [奥富 83] や，ラプラスのポテンシャル [CB90, 佐藤 93] などがある．後者の場合，ロボットの初期位置と目的地を結ぶ障害物に衝突しない経路は，隣り合う white node の並びとして効率的に見つけることができる．octree は3次元空間の階層的な表現法として一般的な方法であり，octree データの生成，変換などの操作に関しても多くの有効な手法が提案されている [CH88, Sam90]．octree 表現された物体の運動に伴って tree を更新する方法もいくつか提案されている [AN84, WA87].

1.2.3 物体間の人為的相互作用

物体間の人為的相互作用は，システムと操作者の間の相互作用を通して仮想空間内の物体の挙動を決定するための仕組みを表す Interaction の軸と，操作者が，視覚・聴覚・触覚などの情報チャネルを通じて，その3次元空間知覚能力と日常生活を営む実世界での体験を利用して，計算機が作り出す仮想の世界を自然なものとして受け入れ，また操作できるかどうかを表す Presence の軸に関係する．すなわち，こ

こでは3次元物体の運動に人が介在した場合の，人と物体の関係について検討する必要がある．上の3次元物体を直接手でつかんで操作するという仮想空間でのユーザインタフェースにおいて，第1，第2の項目で検討された物体間の相互作用を模擬することにより，ある程度は実世界と同じような感覚で，仮想作業を行なうことができる．これにより，AIP キューブの各軸の値を1に近づけることはできる．しかし，それでも全ての相互作用と物理法則を模擬する完全なシステムを構築することは困難である．そこで，限られた種類の物体間の相互作用を模擬して操作者に提供する場合に，実世界の作業と同じように自然で簡単かつ効率的に仮想物体の配置操作などが遂行することができるインタフェースを提供するために，模擬することが必要な物体間相互作用の種類，その提示方法などについて検討する必要がある．また，こうしたインタフェースの設計とともに，それを評価し，次の設計に役立てていくことも必要である．

仮想環境における物体配置操作に，自然なユーザインタフェースを提供するため，操作者の手の動きの自由度を，互いに接触した面間に反力を発生させるような力フィードバック機構などの装置を利用して制限する方法 [Iwa90, 広田 91, 平田 93, SP94] や，操作者の手の動きは制限せずに，物体の視覚的位置のみの自由度に制限を設けようという例 [吉村 91, 竹村 94, SP94, BV93, 木島 95, Sny95] がある．

1.3 本論文の構成

まず，3次元物体間の受動的相互作用について，2章と3章で述べる．2章では，ある離散時刻ごとの位置と方向のみが既知であると仮定し，並進と回転を含む運動をする複数の3次元物体間の衝突面を，octree と多面体表現を用いて効率的に検出する方法を提案する．手法は，静止物体と運動物体を区別することなく，複数の複雑な形状の移動物体が存在する空間中で，連続な運動中に生じる衝突物体を離散時刻の検査によって見逃すことなく選びだし，さらに効率的に衝突面を特定することができる．物体の形状としては凸と凹両方の形状を許し，かつどちらも特別扱いしない．手法は，まず octree による形状表現を用いて空間全体を対象として大まかに物体間の干渉を調べ，次に，多面体による形状表現を用いて，第一段階で絞られた物体の面の間の衝突可能性を局所的に正確に調べる．まず衝突検出の手順を説明し，実験を通して本提案手法の有効性を述べた後，octree の深さと物体の多面体表現の関係について検討を加える．3章では，並列計算機を用いることによって実時間で，複雑な形状の運動物体間の衝突面を検出する方法について述べる．まず上の基本アルゴリズムをさらに高速に動作させるため，手法の最適化を図る．次にこの逐次ア

ルゴリズムをさらに高速に動作させるため，プロセッサ間の負荷分散の考え方が異なる2種類の並列アルゴリズムを提案し，実験を通してその特性を議論する．

次に，3次元物体間の能動的相互作用について，4章と5章で述べる．4章では，移動対象物体（ロボット），静止障害物，移動障害物の環境中の全ての物体を，その運動可能性を区別することなくoctreeで表現する効率的な経路計画の方法を提案する．障害物を表すoctreeの各black nodeを基準としてポテンシャル場を生成し，これを利用して3次元移動ロボットの障害物に衝突しない経路と向きを発見する．いくつかの実験結果を通して，簡単なアルゴリズムでも効率的に3次元の動的環境での経路が発見されることを示す．5章では，回転と並進を含む任意の運動に対して，効率的なアルゴリズムと並列計算機を用いて，octreeを実時間で更新する方法について述べる．

そして，3次元物体間の人為的相互作用について，6章で述べる．この章では，限られた物体間の相互作用を模擬して操作者に提供する仮想物体操作のインタフェースを設計し，それを評価する．つまり，3章で述べる実時間衝突面検出の方法によって検出された衝突面を用いることにより，複雑な形状の物体の接触面を動的に決定し，この面で物体の視覚的な運動を拘束する仮想物体操作を補助する方法について述べる．本手法は力フィードバック機構など特別なハードウエアを用いる必要がなく，自然なユーザインタフェースを提供する．これにより，操作者は仮想物体の配置操作を精度良く，また効率的に遂行することができる．

第 2 章

octree と多面体表現を用いた 3 次元物体間の効率的な衝突面検出

ある離散時刻ごとの位置と方向のみが既知であると仮定し，並進と回転を含む運動をする複数の 3 次元物体間の衝突“面”を，octree と多面体表現を用いて効率的に検出する方法を提案する．手法は，静止物体と運動物体を区別することなく，複数の複雑な形状の移動物体が存在する空間中で，連続な運動中に生じる衝突物体を離散時刻の検査によって見逃すことなく選びだし，さらに効率的に衝突面を特定することができる．物体の形状としては凸と凹両方の形状を許し，かつどちらも特別扱いしない．提案する手法は次の 2 つの段階により衝突を検出する．まず第 1 の段階では，物体の octree 表現を用いて，全作業環境中で干渉している物体を大まかに見つけ出す．第 2 の段階では，物体の多面体表現を用いて，物体の干渉が生じている部分を正確に特定する．第 1 の段階で見つかった干渉物体に含まれる限定された面の組合せのみの衝突を調べるため，限定されたデータに対して詳細な計算を集中的に行なうことができる．実験を通して，提案手法の有効性を述べる．最後に，octree の深さと物体の多面体表現の関係について検討を加える．

2.1 背景

3次元物体間の衝突や干渉を検出する問題は，主にマニピュレータや移動ロボットの経路探索や実時間監視などを目的として長く研究されてきた [HA92a] が，「精度」と「計算の効率」のトレードオフの問題があった．3次元移動物体の動きの正確なシミュレーション（たとえば簡単に，衝突面の法線ベクトルや速度ベクトルを考慮した反射シミュレーション）のためには，衝突が発生した部分を特定することに加えて，その部分の面の方向など，さらに詳細な情報が必要となる．しかし，複雑な形状の運動物体を多く含むような3次元環境で，物体間の衝突を正確に検出する問題は，その計算量の多さから実時間で処理することが困難であった [Pen90, Hah88]．たとえば，物体の多面体表現を用いる方法は最も一般的な形状表現法の一つであり，比較的正確に面間の衝突を検出することができる．正確に運動物体間の衝突を検出する最も単純な方法として，物体の運動をある時間間隔ごとにサンプリングし，各離散時刻ごとに作業空間中の物体のすべての面と稜線の組合せに対して，交差の有無を調べるという手法がとられることが多かった [Boy79]．しかしこの方法では，衝突面検出の計算量は物体の数が多くなるにつれ，また物体形状が複雑になるにつれて増加してしまう．今までにも，効率的な衝突面検出の方法はいくつか提案されているが，物体の形状は凸剛体に限定したり，サンプリングされる離散時間内の物体の移動量が微小で連続的であることを仮定するなど，動作環境に制約を加えたものが多い [LC91, LMC94, GJK88, Qui94, CLMP95]．（詳しくは2.2.4節で述べる．）

更に，これらの離散時刻ごとに干渉を検査する方法ではサンプリングされた離散時刻の間の衝突を見逃す可能性がある．例えば位置と方向のセンサのみをもつ移動ロボットの動作シミュレーションや，仮想現実環境における物体操作シミュレーション [竹村 94] といったアプリケーションのためには，発生する物体間の衝突を見逃すことなく，また実際に衝突が起きる直前に正確にこれを検出することが必要となる．そのためには，サンプリングされたある時間ごとの静的な検査だけではなく，離散時間内の物体の運動を補間または予測することが必要であるが，これには多くの計算量が必要なため，複雑な多面体表現された物体間の衝突面を効率的に検出することは困難であった．この問題に対してもさまざまな試みがなされてきたが [Boy79, Can86, 岡野 88]，物体形状や物体の運動，環境等に制限を加えたものが多く，一般的な環境で物体間の正確な衝突を効率的に検出できるものはない．

一方，octree は3次元空間の階層的な表現法として一般的な方法であり，特定の部分空間が効率的にアクセスできるという特徴がある [Sam90]．形状表現として octree を用いた場合，物体間の干渉は個々の物体を表す octree を並列にたどることに

よって検出することができるので，その計算量は実際にたどったノードの数に比例する [ACB80]. octree は物体の詳細な形状を木を深くすることによって表現できるが，領域計算量が増加してしまう．しかし限られた深さの octree でも，干渉が起きている大まかな場所を特定することができる．ところが従来，octree は静的な物体表現として利用されることが多く，運動物体の表現方法として利用される例は少ない．たとえば [登尾 87] では，octree 表現された静止物体と，多面体表現された運動物体の間の干渉を見つけるアルゴリズムが述べられている．しかしあらかじめ物体を移動物体と静止物体に分類 / 区別し，それぞれを異なった形状表現方法で与えるなど，制約が多く，しかも衝突した両物体の衝突“面”を特定することができない．

本章では，ある離散時刻ごとの位置と方向のみが既知であると仮定し，並進と回転を含む運動をする複数の 3 次元物体間の衝突面を，octree と多面体表現を用いて効率的に検出する方法を提案する．手法は，静止物体と運動物体を区別することなく，複数の複雑な形状の移動物体が存在する空間中で，連続な運動中に生じる衝突物体を離散時刻の検査によって見逃すことなく選びだし，さらに効率的に衝突面を特定することができる．物体の形状としては凸と凹両方の形状を許し，かつどちらも特別扱いしない．手法は，まず octree による形状表現を用いて空間全体を対象として大まかに物体間の干渉を調べ，次に，多面体による形状表現を用いて，第一段階で絞られた物体の面の間の衝突可能性を局所的に正確に調べる．以下，まず衝突検出の手順を説明し，実験を通して本提案手法の有効性を述べる．最後に，octree の深さと物体の多面体表現の関係について検討を加える．

2.2　衝突検出のための形状表現

2.2.1　多面体表現

多面体による物体の形状表現は，最も一般的な形状表現法の一つである．物体間の衝突または干渉は，作業空間中の物体のすべての面と稜線の組合せを調べることにより検出できる．例えば静止物体間の干渉は，複数の物体のすべての面と稜線 (edge) の組み合わせに対して，これらの交わり方が次の 3 つのどの状態にあるか，すなわち，(1) 稜線の両端点が面に対して同じ側にある，(2) 稜線と面を含む平面との交点が面の外部にある，(3) 稜線と面を含む平面との交点が面の内部にある，を調べることにより検出できる [Boy79]. しかしこの方法では，衝突面検出の計算量は物体の数が多くなるにつれ，また物体形状が複雑になるにつれて増加してしまう．

一般に，物体を多面体で表現した場合，空間中の n 個の物体間の衝突検出の平均

的な計算量は，$O(n^2 \cdot EF)$ となることがわかる．ただし，F，E は各々平均的な物体の面，稜線の数とする．つまり衝突検出の計算量は物体の数が多くなるにつれ，また物体形状が複雑になるにつれて計算量は2乗のオーダーで増加してしまうことがわかる．しかし比較的正確な衝突や干渉を調べることができるという特徴もあるので，他の方法で衝突しそうな面や稜線の候補を絞り込むことができれば有効な方法である．

すでに効率的な衝突検出へのアプローチは多くの研究例があり，これらは2.2.4節で述べる．しかしこれらを含めて多くの場合，サンプリングされた離散時刻の間の衝突を見逃す可能性がある．この問題に対してもさまざまな試みがなされてきた．たとえば移動物体間の衝突は，頂点が面上に接触する場合と稜線が面の縁(boundary)に接触する場合に分け，それぞれ端点の軌道やエッジの軌道面を数式で表し，これらと物体の面やエッジの交点を，すべての組み合わせについて解析的に解くことにより，衝突する時刻と場所を導くことができる．但し数式を解く手続きを簡単にするために，片方の物体の静止を仮定して物体の運動を平行移動とある軸回りの回転に制限したり [Boy79]，直線上一定速度またはある直線の周りに一定の角速度で運動する物体を仮定したり [Can86]，物体の運動を3次関数で表わす [岡野 88] などの簡略化が見られる．いずれの例も物体の運動，環境等に制限を加えたものが多く，一般的な環境で物体間の正確な衝突を効率的に検出できるものはない．

また物体とその運動によって掃かれる空間を考え，この掃引物体間の干渉を調べる方法もある．ところが，物体の移動速度に対して調べる時間間隔が長い場合，本当は衝突が起きていない場合でも，衝突していると判断される場合もある．これを回避するため，4次元空間（space and time）を導入した例 [Cam90, Hub93] もあるが，いずれも原理的には，物体と掃引物体を3次元または4次元の volume で表現しなければならないという問題もあり，物体形状や運動が複雑な場合には使いづらい．

2.2.2 octree 表現

octree によると物体は，全体空間を root node とし，順次それを8分割された木で表現される [Sam90]．各ノードは white node と black node にラベル付けされる．white node は完全に物体の外部の空間を示し，black node は完全に物体の内部の空間を示す．そうでない node（物体の内部と外部の両側にまたがる空間を示す node）は gray node とされ，あらかじめ決められた最小の大きさに達するまで8つの子 node に分割される．（octree による形状表現の例を 5.2.1節でも示す．）

octree 表現された 2 物体間の干渉は，それぞれの物体を表す 2 つの木を並列にたどることによって検出できる [ACB80]．いま，N_A，N_B を，それぞれの木の中で，ある対応する node であるとする．もし N_A，N_B のいずれもが black node の場合には干渉している．いずれか一方が gray node で，他方が black または gray であるとすると，この node が示す空間は干渉している可能性があるので，これらの子 node を調べる．もし，N_A，N_B のうちの少なくともどちらか一方が white node であるとすると，この node が示す空間は干渉している可能性がないと判断できるので，子 node を調べずに隣の node を調べる．n 個の物体間の干渉を調べる場合には，n 個の木をたどる必要がある．そして上と同じように，干渉を調べる．

上で述べた方法による干渉検出の計算量は，実際にたどった node の数に比例するので，n 個の物体の干渉を octree を使って調べた場合の平均的な時間計算量は，平均的な octree の node 数を K とすると，$O(Kn)$ である．すなわち，物体の数に関しては計算量の増加は線形であり，多面体表現の場合よりも少ない時間計算量であることがわかる．一方，階層の深い octree を利用することによって物体の詳細な形状を表現することができるが，これは領域計算量，時間計算量ともに増加させてしまう．しかし限られた適当な深さの octree で，干渉が起きている大まかな場所を特定することができる．

octree は 3 次元空間の階層的な表現法として一般的な方法であり，特定の部分空間へのアクセスが効率的に行なえるという特徴がある．octree データの生成，変換などの操作に関しても多くの有効な手法が提案されており [CH88, Sam90]，中でも任意の並進と回転により octree を更新するアルゴリズム [WA87] や，実時間でこれを実行するアルゴリズムも提案されている（本論文 5章）．同様の 3 次元空間の表現方法の特殊なものとして，階層的な球を用いた例 [劉 89] は物体の回転に適しているとして提案されているが，隣接する球領域に重複があり，計算機上に多くの無駄な記憶容量が必要となるなど，複雑な物体が多数あるような環境に対しては，実用性を欠く．

2.2.3　移動物体と静止物体

3 次元環境中の物体を，ロボットなどの移動物体と，障害物などの静止物体に予め分類し，各々を異なった形状表現法で表して物体間の衝突や干渉を検出する研究例も多い．特に従来，octree は静的な物体表現として利用されることが多く，運動物体の表現方法として利用される例は少ない．たとえば静止物体を octree で表現し，動物体を多面体を用いて表現した例 [登尾 87] や球や円柱などの単純な形状で表現し

た例 [SH92] もあるが，作業空間内の移動物体と静止物体とを予め区別し，それぞれを異なった形状表現方法で与えるなど制約が多く，一般的ではない．しかも衝突した両物体の衝突“面”を特定することができない．複数のマニピュレータや移動ロボットなどが動作し，また障害物も移動するような動的な環境を表すためには，ロボットや障害物を含めたすべての物体を同一の方法で表現しておく必要がある．

2.2.4 効率的な衝突面検出の方法

多面体表現された物体間の衝突や干渉を効率良く検出するための第1のアプローチは，交差を調べる必要がある面の組合せの数を減らすことであり，この点からいくつかの研究がなされている．通常，衝突検出の初期段階では，物体形状を外接直方体や外接球などで近似し，これらの間の干渉を調べることにより，衝突しそうな（または，しそうにない）物体を素早く見つけ出す（または，除去する）という方法がよく使われる．この方法によれば2物体間の干渉は簡単に調べることができるが，物体の形状が複雑であったり凹であったりすると，形状の近似誤差が増大し，また物体の衝突する部分（面）を直接特定することが困難であるという欠点がある．この方法を発展させたものとして，階層的な外接図形を用いた例 [Hah88] もあるが，世界座標系の中で物体の衝突場所を特定するのに手間がかかるという問題もある．衝突箇所を大まかに特定するために物体のボリュームデータ (voxel) を用いた例 [GSF94] もあるが，付加的なデータとして特定の部分空間へのアクセスの効率が十分でなく，データにも無駄が多い．

作業環境内の物体の各組合せ毎にそれらの間の最近接の辺とペア面の間の距離を観察し，それがあるしきい値よりも小さくなった場合に衝突とみなす，という方法もある．この方法は非常に高速なアルゴリズムであるが，動作環境に制約が多い．例えば [LC91, LMC94, CLMP95] は，物体の形状を凸剛体に限定して，サンプリングされる時間内の物体の移動量が微小で連続的であり，最近接の辺 - 面ペアは変化しないと仮定し，これらの間の衝突検出に利用した．他にも，物体間の距離に基づく方法はいくつかあるが [GJK88, Qui94]，いずれも同様の仮定が必要である．他にも，サンプリングされる時間内の物体の移動量が微小であると仮定して，2物体を分離する平面を用いて衝突が生じてるか否かを素早く判定する方法 [Bar90] が提案されているが，これもやはり物体の形状が凸であることを仮定している．これらのアルゴリズムでは，凸でない物体を扱うためには，その物体を凸物体の集合に分割するという方法がよくとられる．しかしこれは不必要な頂点や辺を生じさせ，かつ物体の数も増加させる結果となり，複雑な物体を多数含むような環境に対しては，実用性

を低下させる.

他に主な衝突検出のアプローチをいくつか挙げる. [BV91] は BRep-Index と呼ばれる, BSP 木を拡張したデータ構造を用いて, 2 物体間の接触する領域を特定し, 面を高速にアクセスする方法を提案した. [SF91] はラスタ走査を通じた干渉検出を実行するためにzバッファを用いた可視面判定のアルゴリズムを利用している. [FHA90] が提案した方法は, 物体の組合せ毎にその間の距離でソートし, その結果常に距離が近い物体の衝突を調べることになる. [Hub93] はまず各物体の組合せが衝突する時刻を決定するために 4 次元の時空間を考え, 最も早く衝突する組のみを調べる. また, パラメトリックな形状表現を用いれば, 高速な物体の内外判定が可能となる. [Pen90] は物体を超 2 次関数で表し, 効率的に衝突を検出したが, 通常用いられることの多い多面体表現とは簡単に対応づけをとることができない.

2.3 octree と多面体表現を用いた衝突面検出

本章ではまず octree を用いて作業空間全体を対象として大まかに干渉を調べ, 次に限定された物体の面に対して, 多面体による形状表現を用いて詳しく面間の衝突の可能性を調べる 2 段階の干渉・衝突検出法を述べる.

作業空間中には n 個の物体（剛体）が存在するとし, その各々の多面体と octree の 2 種類の形状表現は予め与えられているものとする. 物体の形状は凸でも凹でもよいものとする. また物体の運動に関しては, ある十分に短い時間間隔 Δt 毎の時刻 $(\cdots, t_{i-1}, t_i, t_{i+1}, \cdots)$ における各物体の 3 次元空間内での位置と方向が既知であるとする. ここで, 時間 Δt 内に物体が並進移動する距離と回転角度は, それぞれ十分に小さいとする. なお, 与えられた並進と回転運動のパラメータによって, 多面体と octree の両形状表現は毎時刻に各々変換される.

Fig. 2.1に, 衝突面検出の流れを示す. 以下, 各処理毎にその内容を詳しく述べる.

2.3.1 形状表現の更新

多面体と octree の両形状表現は, 観察された並進と回転運動のパラメータによってある時間間隔毎に各々変換される. ここで多面体表現は簡単な座標変換により更新することができる. ここでは, 5章で述べる任意の並進と回転により octree を実時間で更新する方法を用いた. このアルゴリズムは, 各 black node 毎に変換してこれの部分木を作成し, 最後にこれらを組み合わせて全体の octree 表現を作成する. 変換後に生成される octree の cube (upright cube) は, その中心点が変換された cube

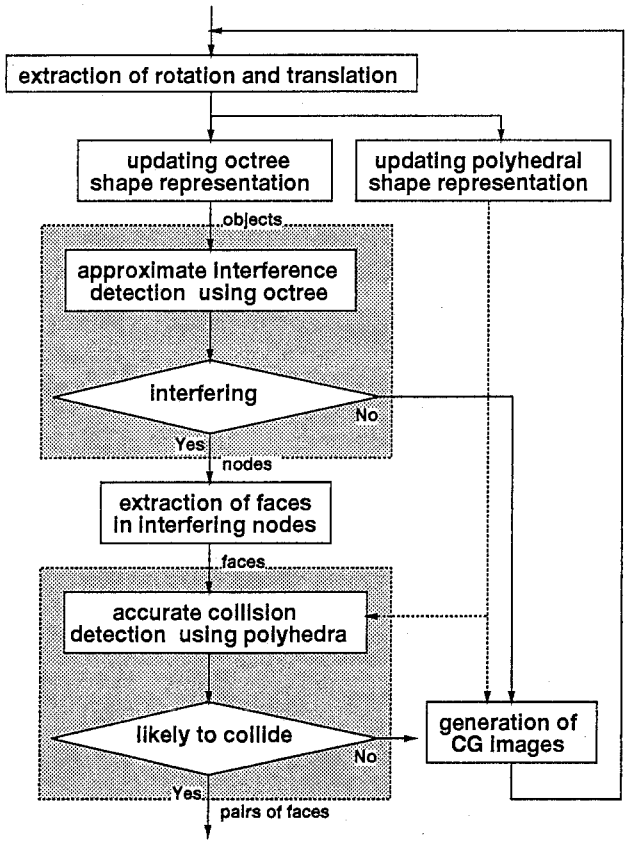


Fig. 2.1: Proposed method for colliding face detection.

(tilted cube) の内部にある場合， black となる．本研究では octree と多面体表現を各々独立に更新するので，両者の間に微妙な位置ずれが生じないよう，次のような検討に基づいて node の色を決定することにした．2 次元の平行移動の例を **Fig. 2.2**に示す．図で (a) に示された位置に置かれた物体は，右方向に移動するとする．(a) の初期状態において， node a と c は black node である．今，例えば 1/4 voxel の右方向への移動が生じた時，多角形（3 次元の場合は多面体）の物体は (b) 図のように移動する．しかしながら， b と d の octree の node は，これらの中心点が 1/4 voxel 移動した a と c の cube (a'とc') の外なので， white のままである． b と d の node は，物体の運動が 1/2 voxel を越えた時，初めて black になる．そこで我々は， octree の node の境界面の少なくとも一部が tilted cube の内部または表面上にある場合に，この node を black とすることにした．これは black node の数を増加させるので計算の効率を損なう要因となるが，検出するべき衝突を見逃さないために必要である．この問題は，各処理時刻に octree 表現を移動後の多面体表現から実時間で生成することができれば解決される．

Fig. 2.3に 3 次元の octree の場合の例を示す. (a) は多面体によって表現された物体 (スペースシャトル) であり, (b) は, その octree 表現である. 同 (c) は, (b) にある回転を与えた場合に更新された octree 表現である.

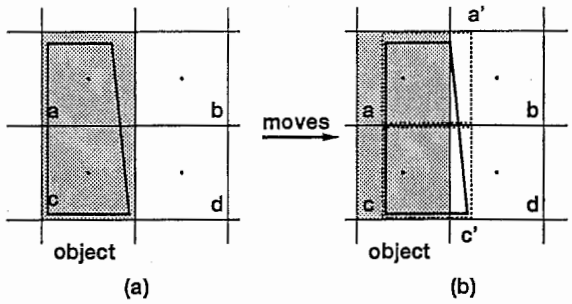
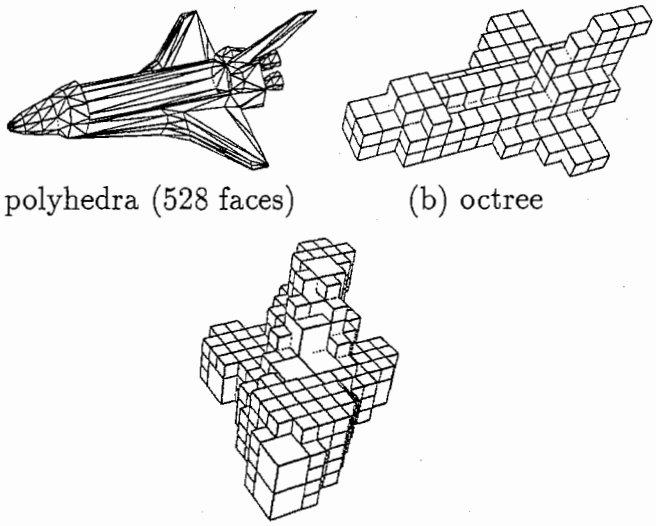


Fig. 2.2: An example of white nodes occupied partially by an object.



(a) polyhedra (528 faces)　　(b) octree

(c) a rotated octree shape representation of (b)

Fig. 2.3: The experimental space shuttle

2.3.2　octree を用いた大まかな干渉検出

作業空間全体の中での物体間の干渉を, 各々の octree を root node から順に並列に辿ることによって検出する. 複数の octree の対応する node が black の時, これらの node は干渉していると考えられる. 空間中に n 個の物体が存在する場合, n 個の octree を並列に辿り, 以下の要領で干渉を調べる.

- n 個の対応する node のうち n-1 個が white のとき，この node は干渉していない
- n 個の対応する node のうち少なくとも 2 つが black のとき，または 1 つが gray で残りの 1 つ以上が black のとき，この node は干渉している
- n 個の対応する node のうち少なくとも 2 つが gray のとき，この node が示す空間は干渉している可能性があるので，これらの子 node を調べる

すべての octree を辿り終えると，干渉している node とその物体が特定される．これは衝突が起こりそうな物体とその部分を大まかに示している．

2.3.3 干渉 cube からの面の抽出

上の手順によって干渉が見つかった node と交差している物体の面を検出する（**Fig. 2.4**参照）．干渉が見つかっている物体の各面の方程式に干渉 cube C の 8 頂点の座標を与え，その符号を調べる．もし少なくとも一つが他と異なれば，この面 F は C と交差している可能性がある．そこで次に，F を含む平面 T によって切られる C の断面多角形 S と F の交差を調べる．こうして干渉 cube から多面体の面を順次抽出する．

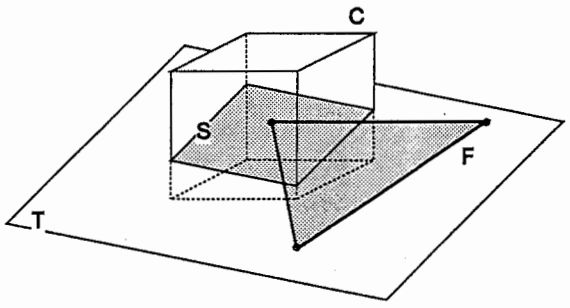


Fig. 2.4: Identification of object faces responsible for interference detected using octree representation.

2.3.4 多面体表現を用いた詳細な衝突検出

上で抽出された面間の衝突を調べる．そのため，物体の連続な運動中に生じる衝突を離散時刻の検査によって検出するために，ある特定の時刻のみの面間の干渉を調べるのではなく，任意の時刻 t_i において，時間 $[t_i,\ t_{i+1}]$ に各面が掃く掃引物体が

干渉していれば，これらの面は時刻 t_i と t_{i+1} の間に衝突すると判断することとした．(**Fig. 2.5**参照)．

具体的な手順について述べる．運動している各面 A に対して，この面が時間 $[t_i,\ t_{i+1}]$ に各面が掃く空間として，A^{t_i} の頂点 ($a_0^{t_i}$, $a_1^{t_i}$, $a_2^{t_i}$,...) と $A^{t_{i+1}}$ の頂点 ($a_0^{t_{i+1}}$, $a_1^{t_{i+1}}$, $a_2^{t_{i+1}}$,...) の凸包 $V_A^{t_i}$ を作成する (文献 [Prep92] 3 章)．同様に，A との衝突を調べようとする面 B についても，B^{t_i} の頂点と $B^{t_{i+1}}$ の頂点の凸包 $V_B^{t_i}$ を作成する．ここで，時刻 t_i における面 A，B を各々，A^{t_i}，B^{t_i} と表すことにする．

次に $V_A^{t_i}$ と $V_B^{t_i}$ の間の干渉を調べる．各々の面と稜線の組み合わせに対して，これらの交わり方を調べ，次の 3 つのうち 3 番目の組を見つけることにより干渉を検出する．

1. 稜線の両端点が面に対して同じ側にある
2. 稜線と面を含む平面との交点が面の外部にある
3. 稜線と面を含む平面との交点が面の内部にある

このように，干渉が見つかった octree の node から取り出された面間の衝突を調べることにより，時間 $[t_i,\ t_{i+1}]$ に衝突するすべての面の組が検出される．この方法は，各面の頂点数が多くなった時には時間がかかる処理であるが，凸多面体間の干渉検出として Muller-Preparata の方法 (参考文献 [Prep92] 7 章) などを用いればさらに効率的に実行することができる．**Fig. 2.5**では，最も簡単な例として，3 角形の場合を示している．

以上述べた手順は，衝突の発生に関しては必要条件ではあるが十分条件ではない．ここでは，ある時間間隔 Δt 毎の離散時刻 $(\cdots,\ t_{i-1},\ t_i,\ t_{i+1},\ \cdots)$ に，各物体の位置と方向が得られるという仮定のもとで衝突の必要条件を近似的に調べる．そのため，時間 Δt 内に物体が並進移動する距離と回転角度は，それぞれ十分に小さいと仮定した．この Δt の間をさらに詳しく調べるためには，この間の物体の動きを（たとえば直進とか，またはある点を中心にした回転 などと）仮定して解析的に解く，といった方法などが考えられる．

2.4　実 験

本章では，提案した衝突検出法を用いたシミュレーション結果について述べる．まず標準物体として球を用いて基本的な性能を評価し，続いてより一般的な形状の対象に対しての実験例を述べる．

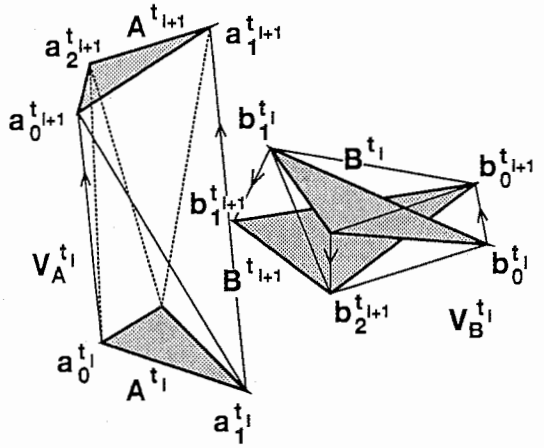


Fig. 2.5: Collision detection between moving faces identified by octree representation as potentially colliding.

2.4.1 標準物体を用いた評価実験

3 角形パッチによって表現された球状の物体を用いて，提案手法を評価した．球は全方向に対してほぼ均等に面が張られているので，衝突の評価実験には好都合である．実験に用いた球の octree 表現を **Fig. 2.6**に示す．図で各球は，root node を全空間とする，4, 5 level の階層を持つ octree によって表現されている．

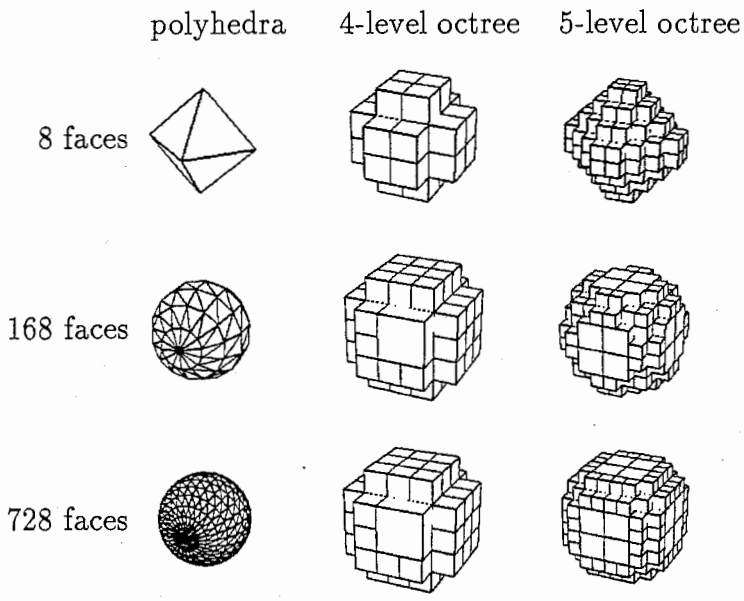


Fig. 2.6: Examples of experimental objects represented using polyhedra and octrees

面数が異なる運動 2 物体に対する実験

物体表面を構成する面数が異なる複数種の球を用いて，運動 2 物体に対する干渉と衝突を検出した．各球は **Fig. 2.6**のように，対応する多面体表現と octree 表現により記述されている．2 物体（A，B とする）の初期位置と方向，およびこれを衝突させるような運動パラメータは **Table 2.1**のように与えた．ここでは実験条件を明確にするため，表に示す初速度により，一定の並進と回転の運動を各物体に与えた．なお単位は voxel，すなわち octree の最小 cube (level 0) の辺の長さであり，座標系は **Fig. 2.12**と同様である．本実験では level 4 の octree を用い，作業空間は最下位の level で M^3 $(M = 2^4)$ voxel に分割される．また，球の直径は 3.8 (voxels) である．

Table 2.1: Initial positions, directions and motions given to the objects (unit: 4-level octree)

object	initial positions (*voxels*)	translation (*voxels/cycle*)	rotation (*degrees/cycle*)
A	(2.5, 2.5, 2.5)	(0.015, 0.0075, 0.01)	(0.0, 0.25 0.0)
B	(8.5, 6.5, 6.5)	(-0.015, -0.0075, -0.01)	(0.0, 0.25, 0.0)
C	(13.5, 13.5, 6.0)	(-0.0005, -0.0005, -0.0005)	(0.05, 0.1, 0.15)
D	(6.0, 2.5, 13.5)	(-0.0005, -0.0005, -0.0005)	(0.1 0.15, 0.05)
E	(2.5, 13.5, 2.5)	(0.0005, -0.0005, -0.0005)	(0.15, 0.05, 0.1)
F	(13.5, 8.0, 13.5)	(0.0005, -0.0005, -0.0005)	(-0.05, -0.1, -0.15)
G	(13.5, 2.5, 2.5)	(0.0005, -0.0005, -0.0005)	(-0.1, -0.15, -0.05)
H	(6.0, 13.5, 13.5)	(0.0005, -0.0005, -0.0005)	(-0.15, -0.05, -0.1)

各処理時刻において，各物体の octree と多面体表現は各々更新された後，全作業空間内で octree 表現を用いて干渉している物体を大まかに発見する．もし干渉している node が見つかると，多面体表現を用いて干渉や衝突が起きている部分を詳細に特定する．以下の実験では，各時刻における処理に要する計算時間（CPU 時間）をワークステーション (Silicon Graphics Onyx) を用いて計測した．

24，48，80，168，528，728 の面を持つ運動 2 物体に対する結果を **Fig. 2.7**に示す．面数 168 の 2 つの球間の衝突の場合，時刻 $t = 5$ (cycle) までは octree node 間の干渉が発見されていない．時刻 $t = 6$ から $t = 49$ の間は，octree の干渉 node は見つかっているが，それらから物体の面は抽出されないので，ここまでの計算時

間は比較的短い．時刻 $t = 50$ において干渉 node 内に面がひとたび発見されると，これらの衝突を調べるためにより長い計算時間がかかるようになる．時刻 $t = 116$ までは，干渉 node 内に面が発見されるものの，これらは衝突しない．最後に時刻 $t = 116$ において，これらの面間の衝突が検出され実験が終了する．この衝突検出の最終段階では，合計 32 の干渉 octree node から 15 組の衝突する面を検出するのに，251(ms) の時間がかかっている．ここで，octree で見つかっている干渉 node 毎に，その中に含まれる面間の衝突を調べた．

また，上と同じく面数 168 の 2 つの球間の衝突について，干渉が見つかった octree node 数，および干渉 node から抽出された 面の数の各時刻に対する変化の様子をそれぞれ，**Fig. 2.8**，**Fig. 2.9**に示す．なおここでは，異なった干渉 node から抽出された同一の面を重複を許して数えあげている．本手法の演算時間はほぼ，これらの項目に比例していることがわかる．

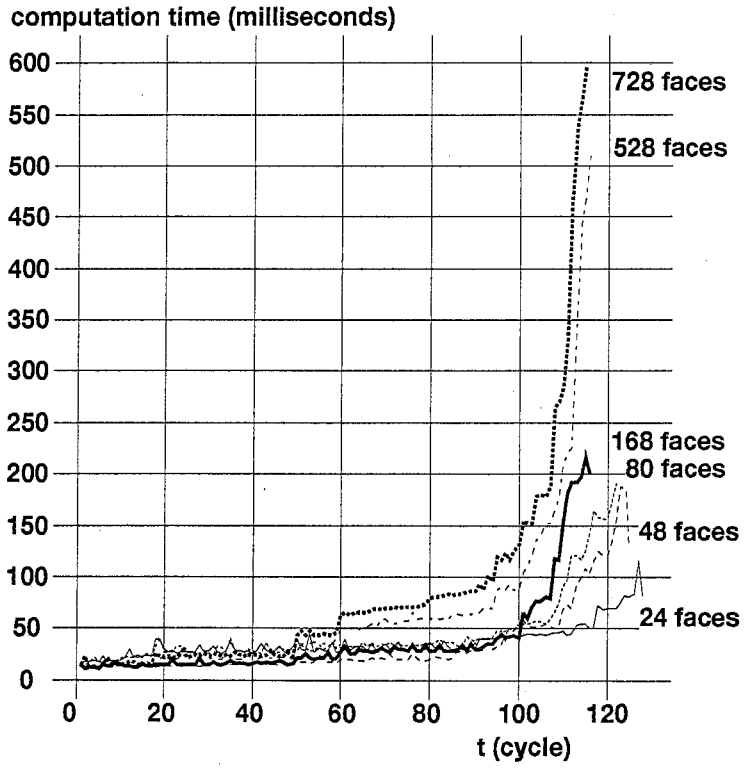


Fig. 2.7: Computation time for each processing cycle of collision detection for two identical objects having 24, 48, 80, 168, 528 and 728 faces.

一方比較のため，octree による形状表現による拘束を加えずに，多面体表現のみを用いて面数 168 の 2 つの球間の衝突を調べたところ，衝突面を検出するのに 30 秒の時間がかかった．多面体表現のみを用いる方法では，処理の時刻毎に面の全組合

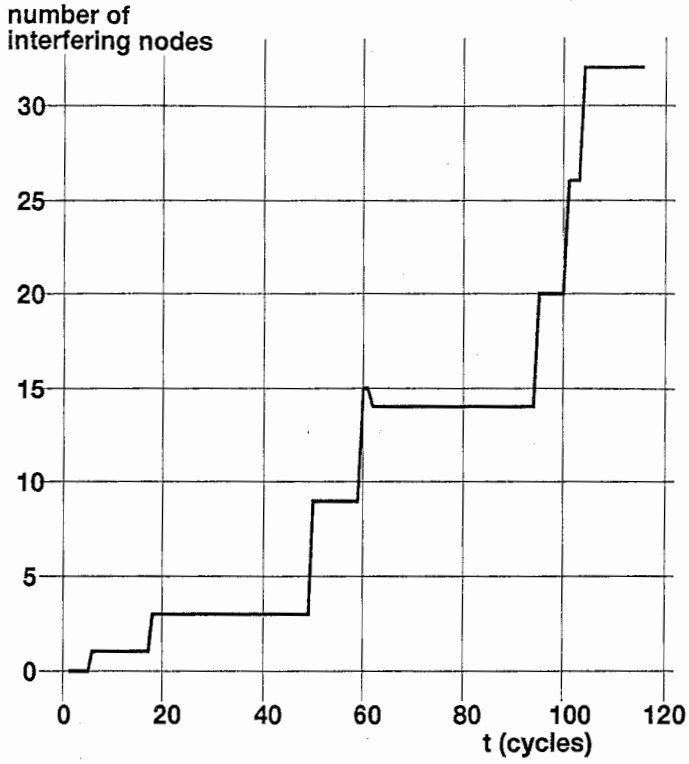


Fig. 2.8: The number of interfering nodes for each processing cycle of collision detection for two identical objects having 168 faces.

せに渡って衝突を調べるため，長い計算時間が必要となる．球という対象物体に限れば，2 球の中心間距離と半径の和を比較する方法や，他のパラメトリックな方法が衝突を検出する方法として考えられるが，このような方法は他の凹や複雑な形状の物体に対して一般的に用いられるべき方法ではないので，ここではそのような方法は用いていない．

Fig. 2.7は，24，48，80，168，528，728 の面を持つ 6 種類の運動 2 物体に対する結果を示しているが，処理時間のスケールは異なるものの処理時間の推移を表すグラフの形は似通っている．グラフの細部は演算時間計測の誤差や，面数が異なることによる物体形状の微妙な違いのため，多少の変動が見られるが，面数 728 の物体で，589(ms) の時間で衝突面を特定できている．そこで**Fig. 2.10**では，提案した衝突検出の最終段階で最も計算時間がかかる場面での 1 サイクルに要する計算時間を，対象物体の面数を変化させた場合に計測した結果を示す．この結果は，面数が増加して物体の形状が複雑になっても，提案手法は，全運動物体の octree 表現をその運動に伴って毎時更新するというオーバヘッドを持ちながらも，十分に効率的であることを示している．

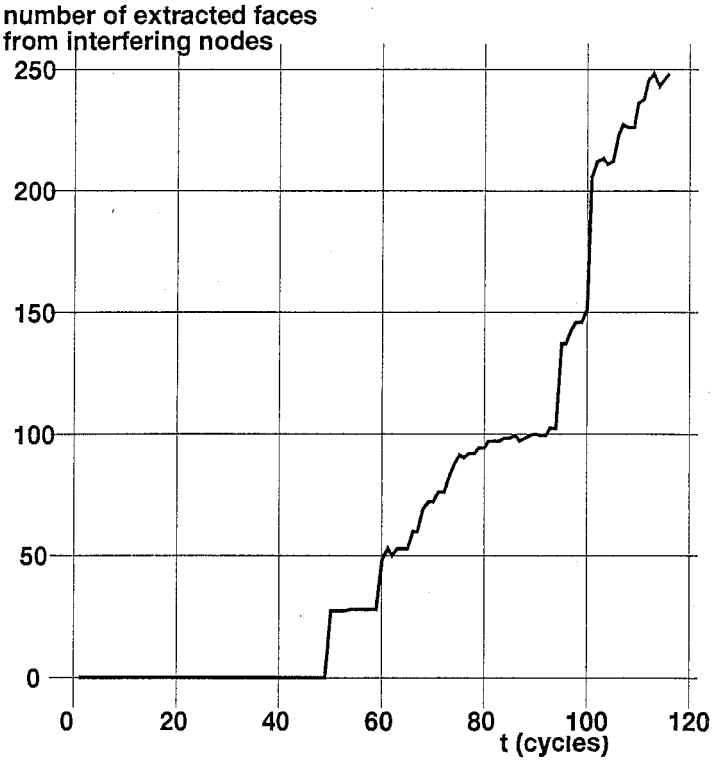


Fig. 2.9: The number of extracted faces for each processing cycle of collision detection for two identical objects having 168 faces.

複数の運動物体に対する実験

複数の運動物体が存在する空間で，物体間の衝突を検出した．実験に用いた物体は前述の 168 の面を持つ球で，level 4 の octree によっても表現されている．物体（A, B, C,...）の運動は各々一定の並進と回転からなり，そのうち物体 A と B のみが衝突するよう，**Table 2.1**のように運動パラメータを与えた．そして前節のように，各処理時刻の計算時間を計測した．

様々な物体数に対して実験した結果を**Fig. 2.11**に示す．この図では，提案した衝突検出の最終段階で最も計算時間がかかる場面での処理に要する計算時間を，対象物体の数を変化させた場合に計測した結果を示す．

本実験では，実験条件を明らかにするため，複数の物体の内の 2 つの物体のみが衝突するように運動パラメータを与えている．しかし本手法では，多面体表現に加えて物体毎に octree 表現を持っているので，運動に伴ってこれを毎時更新する計算量と，更新された octree を各々たどって干渉を発見する計算量は，物体の数に比例して増加する．（これらの過程は $O(n)$） 実際の計算時間はこの他に，干渉 node から面を抽出してさらにこれらの衝突を調べる計算が加わるが，この過程は上記のよ

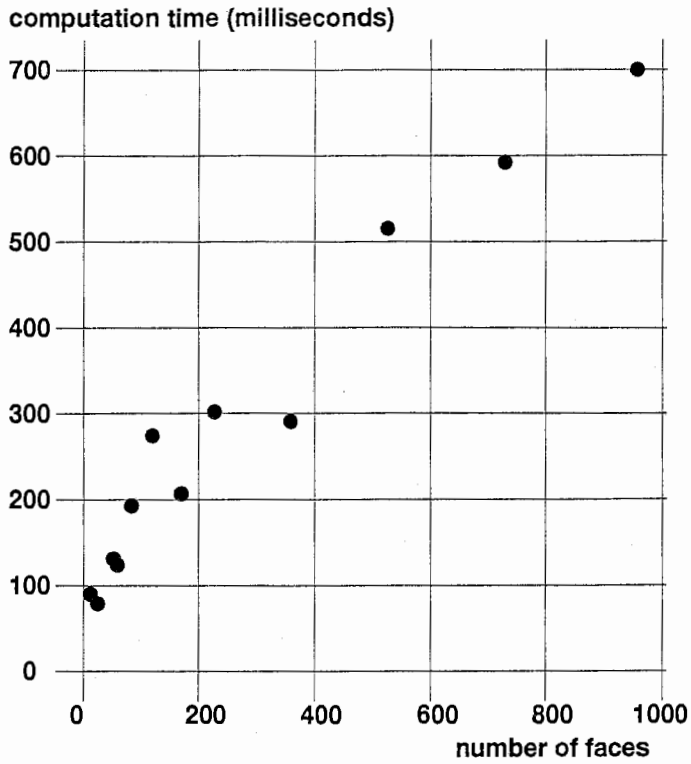


Fig. 2.10: Computation time of collision detection between two identical objects against the number of object faces.

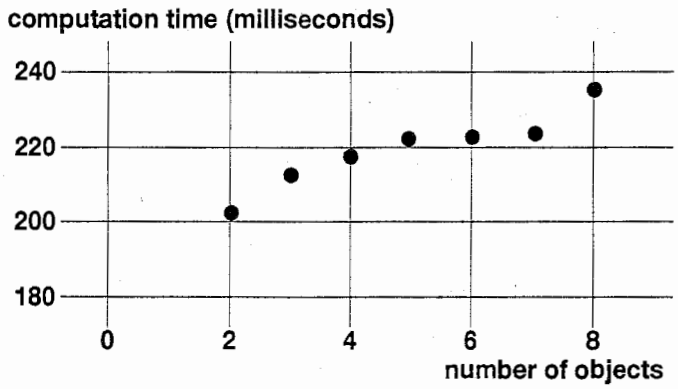


Fig. 2.11: Computation time of collision detection among multiple moving objects having 168 faces against the number of objects.

うな運動パラメータを与えた場合，ほぼ物体の数には依存せずに一定となる．Fig. 2.11に示す結果はこの様子を示している．

2.4.2 一般物体を含む環境に対する実験

提案した衝突検出の手法を，複雑な一般物体を含む実際的な環境に応用した．**Fig. 2.3**のように，多面体（3 角形パッチ）によって表現された面数 528 のスペースシャトル（図 (a)）と，文献 [北村 94] による方法で作成された octree （図 (b)）を用いて，n $(n = 2, 3, 4, 5)$ 個の物体間の干渉と衝突を検出した．同図 (c) は，(b) に世界座標系の y 軸回りに 45(*degree*) の回転を与えた場合の octree 表現である．実験に利用した環境中の物体（A から E）の初期状態を **Fig. 2.12**に示す．実験条件を明らかにするため，これらのうち A と B のみが衝突するよう，各々物体毎に一定の並進と回転のパラメータを与えた．ここでは，level 5 の octree を用いた．

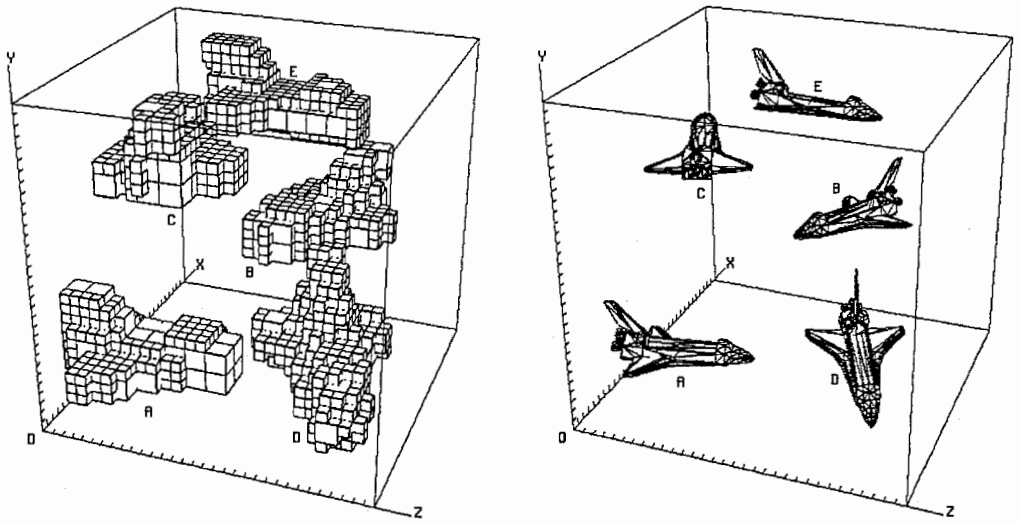
Fig. 2.13に実験結果を示す．図中，a, b, c, d は各々物体数が 2, 3, 4, 5 の場合である．また矢印は各グラフの終端を示す．時刻 $t = 70$ (cycles) まではどの物体間にも干渉が見つからないので，計算コストは比較的小さい．しかし，時刻 $t = 71$ に干渉している octree node が見つかると，この node から面を抽出し，さらにこれらの間の衝突を調べるのにより多くの計算が必要となる．時刻 $t = 123$ までは，干渉 node は見つかるが，これらに含まれる面間の衝突は検出されない．最終的に時刻 $t = 123$ において，面間の衝突が検出され，**Fig. 2.14**のように衝突の寸前（もう 1 単位時間進む間に衝突する）で実験を終了する．この図は，**Fig. 2.12**と同じ視点から眺めたものである．衝突検出の最終段階では，62 の干渉 octree node から 14 組の衝突する面を特定するのに，461(ms) が必要であった．

一方参考のため，octree による形状表現による拘束を加えずに，多面体表現のみを用いて 2 物体間の衝突を調べたところ，衝突面を検出するのに 269 秒の時間がかかった．また同じ方法で 5 つの物体間の衝突を検出するためには，2,684 秒が必要であった．これに対し，我々の提案手法では 577(ms) で衝突面を検出できた．

2.5 検 討

octree では物体形状の詳細な情報を，深い level の木を用いて表現することができるが，これは領域計算量を増加させる原因になる．逆に浅い level の octree を用いれば，少ないデータ量で表現することができるがその形状は抽象的であり，効率良く面を抽出しまたこれらの衝突を調べることができない．つまり効率的に面間の衝突を調べるためには，多面体表現に対して適切な深さの octree を用いることが重要であることがわかる．この関係をもう少し詳しく調べてみる．

深い level の octree を用いた場合，最下層の node (voxel) は小さくなるが，も



(a) octree　　(b) polyhedra

Fig. 2.12: Experimental space including five objects (space shuttles) at initial positions.

しこれが多面体表現の面の大きさよりも小さければ，近傍の複数の node で同一の面が発見される可能性が高くなる．逆に浅い level の octree を用いた場合，最下層の node (voxel) は大きくなるが，もしこれが多面体表現の面の大きさよりも大きければ，1 つの node に含まれる面の数が多くなる．いずれの場合も衝突を調べなくてはならない面の数が増加するので，計算の効率性を損なう原因になる．したがって， octree の最下層の node (voxel) が多面体表現の面の大きさにほぼ等しいような，適切な level の octree を用いる必要があることがわかる．

たとえば， L^3 の大きさの実験空間があるとし， octree の level を l ，多面体表現の面の大きさを w とすると，これらの関係は， $w = L/2^l$ となる．具体的には， L = 1,024 (mm)， w = 32 (mm) の時の適切な octree の level (l) はこの関係から 5 となる．

この関係を実験によって調べてみた． **Fig. 2.6**に例をあげたような各物体（同一半径の球）に対して level 3, 4, 5, 6 の 4 種類の octree 表現を用意し， 2.4.1節の実験と同じ要領で， 2 つの同一種の球を異なった level の octree を用いて衝突させた．以前の実験と同様，衝突面を検出する最終段階で最も計算時間がかかる場面での処理に要する計算時間をワークステーション (Silicon Graphics Crimson) を用いて計測した．ただし， 2.3.1節で述べたように octree と多面体表現間の位置ずれが生じ，またそれに起因して衝突を詳細に調べる面の数も多少変化する．ここでは，こうし

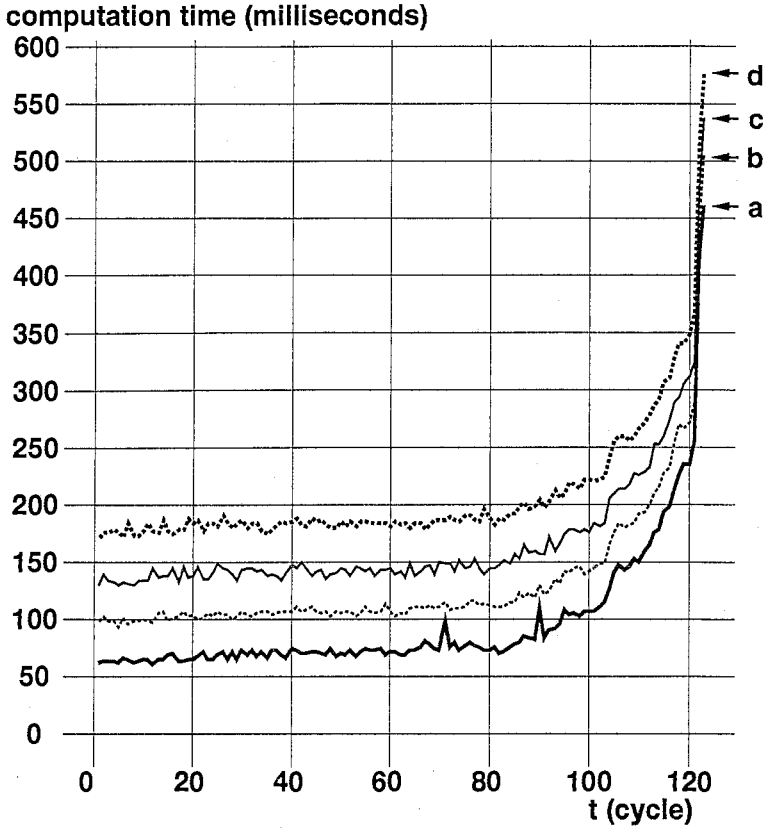


Fig. 2.13: Computation time for each processing cycle of collision detection among objects (space shuttle)

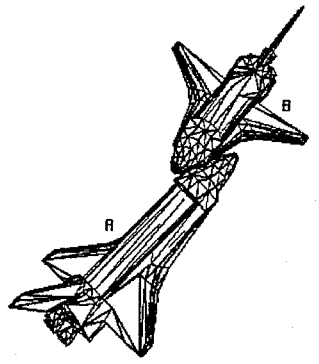


Fig. 2.14: A snapshot of experiment when colliding faces are detected.

た衝突条件の変動を考慮して，同一の物体の組合せによる衝突を各々 60 の異なった運動パラメータを与えて衝突させ，その平均をとった．**Fig. 2.15**では，面数 48, 360, 960 の同一種 2 球間の衝突に対して，この平均値を標準偏差とともに示してい

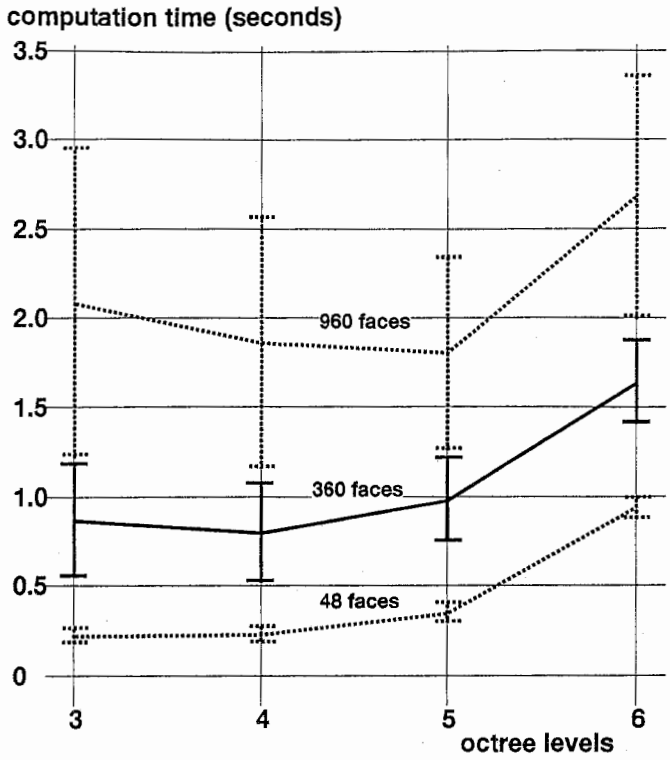


Fig. 2.15: Computation time (average of 60 collisions with standard deviation) of collision detection between two identical objects against the level of octree shape representations.

る．実験結果によれば，960 面の物体は level 5 の octree を用いた場合に，最も効率的であり，360 面の物体は level 4 の octree を用いた場合に最も効率的に衝突面を特定していることがわかる．

octree の level （深さ）が 1 段増加すると，最下層の node (voxel) の 1 辺の長さは 1/2 になる．一方，実験に用いた球はすべて等しい半径であるので，多面体表現では面数が増えるにつれて 3 角形は小さくなる （これらの面はほぼ辺の長さが等しいが，必ずしも正 3 角形ではない）．面数 48, 360, 960 の球で，各面の最長辺の平均をとってみると，各々 0.88 (level 3 octree voxel の 1 辺の長さ = 1)， 0.73 (level 4 octree voxel の 1 辺の長さ = 1)， 0.91 (level 5 octree voxel の 1 辺の長さ = 1) であった．ここでは物体を構成する面がほぼ正 3 角形に近く，均一に分布させた球を用いて実験を行なったが，物体の大きさを変えずに面の大きさを連続的に変化させることは困難である．**Fig. 2.15**は離散的な条件での結果であるが，多面体表現に対して適切な深さの octree を用いた場合に，最も効率的に面間の衝突を調べることができていることがわかる．

2.6 2章のまとめ

以上本章では，並進と回転を含む運動をする複数の3次元物体間の衝突面を，octree と多面体表現を用いて効率的に検出する方法を提案した．手法は，静止物体と運動物体を区別することなく，複数の複雑な一般形状の移動物体が存在する空間中で，連続な運動中に生じる衝突物体を離散時刻の検査によって見逃すことなく選びだし，さらに効率的に衝突“面”を特定することができた．ここで，物体の形状としては凸と凹両方の形状を許し，かついずれの形状も特別な前処理などを必要としないで同様に扱うことができた．まず octree による形状表現を用いて空間全体を対象として大まかに物体間の干渉を調べ，次に，多面体による形状表現を用いて，第1段階で絞られた物体の面の間の衝突可能性を局所的に正確に調べた．本章では，物体が並進と回転の運動をする一般的な環境で有効なことを，実験によって示した．また効率的に衝突面を特定するためには，比較的浅い階層の octree を用いれば十分である，つまり効率的に面間の衝突を調べるためには，多面体表現に対して適切な深さの octree を用いることが重要であることを確認した．

物体の運動に伴って octree を更新する際に，多面体表現との間で多少位置のずれが生じるという問題が生じるが，これは各処理サイクル内に，物体の octree 表現をその多面体表現から高速に作成できれば，解決することができる．またこれにより，運動中に変形する非剛体の衝突面を検出することが可能となる．本手法を並列計算機上にインプリメントし，さらに高速に動作させることにより，3次元環境における実時間衝突面検出を実現することができる．実際，2.3節に述べた一連の衝突面検出の手順の中で，2.3.4の手順に要する計算量は他の手順に比べると大きく，これらの処理の高速化が全体の処理時間を左右していると考えられる．しかしこの手順は比較的単純な処理の繰返しであるので，データ並列アルゴリズムとして比較的容易に実現可能である点に着目し，衝突面検出の並列処理による高速化を図ることができる．この点については，3章で述べる．また将来，本手法を実世界で稼働する移動ロボットなどの行動計画などに応用するためには，誤差などについて検討をする必要がある．これらは今後の研究課題である．

第 3 章

並列計算機による 3 次元物体の実時間衝突面検出

本章では，並列計算機を用いることによって実時間で，複雑な形状の運動物体間の衝突面を検出する方法について述べる．まず，離散時間の衝突検査で複雑な軌道をもつ物体間の衝突面を見逃すことなく，実際に衝突が起きる直前に効率的に特定する基本アルゴリズムについて説明する．手法は，物体の形状に制限を設けることなく多面体表現のみを利用し，他の余分な形状表現を用いない．本論文ではこの逐次アルゴリズムを更に高速に動作させるため，2 種類の並列アルゴリズムを提案する．1 つはプロセッサ間の負荷分散を静的に行うものであり，他方は，負荷が少ないプロセッサに動的に負荷を割り当てるものである．共有メモリをもつ並列度 10 程度の並列計算機を用いた実験により，4,000 面程度の物体間の衝突面を 80 (ms) で検出できた．

3.1 背景

物体の運動をある時間間隔毎にサンプリングし，各離散時刻毎に多面体表現された物体間の衝突や干渉を効率良く検出するための第1のアプローチは，交差を調べる必要がある辺と面の組合せの数を減らすことである．通常，衝突検出の初期段階では，物体形状を外接直方体や外接球などで近似し，これらの間の干渉を調べることにより，衝突しそうな（または，しそうにない）物体を素早く見つけ出す（または，除去する）という方法が良く使われる．この方法を発展させたものとして，階層的な外接図形を用た例[Hah88]や，octree や voxel などの体積表現を用いた例もある．[GSF94]は各物体の多面体表現とともにvoxelデータを蓄え，各voxelとその対応する3次元位置にある面とをポインタで結んでおく．そしてまず干渉しているvoxelを見つけた後に，そのvoxelから面に張られたポインタを辿って衝突する可能性のある面を選び出す．これは，octreeを用いれば高速に特定の空間にアクセスすることが容易になるため，さらに効率的に処理できる．2章では，各物体のoctree表現を予め蓄え，物体の運動によって更新されたoctree表現を用いてまず干渉しているoctreeのnodeを見つけ出し，次にそのnodeに占有される空間に含まれる面間の衝突を調べることにより，並進と回転を含む運動をする複数の3次元物体間の衝突面を，効率的に検出する方法を提案した．他にも octree や voxel などを用いた例[MW88, Tur89, ZPOM93, SH92, Hay86]は多いが，付加的なデータ構造を用いているため，領域計算量を増大させるといった問題がある．また物体は剛体に限られ，時間とともに形状が変化するような物体には適用できなかった．さらに2章の方法も，効率的に複雑な物体間の衝突面を検出できたが，実時間で検出するには至らなかった．

最近は並列計算機が安価で手に入るようになり，その利用に関心が高まってきた[特集情92]．特に汎用のワークステーションでも，数十個程度の高速のプロセッサを搭載し，並列処理が簡単に行える処理系と共に提供され始めているが[Bar92]，衝突面検出について並列アルゴリズムが報告された例はない．

本章では，並列計算機を用いることによって実時間で，複雑な形状の非剛体の運動物体間の衝突面を検出する方法について述べる．まず，離散時間の衝突検査で複雑な軌道をもつ物体間の衝突面を見逃すことなく，実際に衝突が起きる直前に効率的に特定する基本アルゴリズムについて説明する．次にこの逐次アルゴリズムをさらに高速に動作させるため，プロセッサ間の負荷分散の考え方が異なる2種類の並列アルゴリズムを提案し，実験を通してその特性を議論する．

3.2 衝突面の検出

多面体による物体の形状表現は，特にコンピュータグラフィックスを用いたアプリケーションなどで，最も一般的な形状表現法の一つである．まずこの形状表現法を用いて，衝突または干渉している面を検出する基本的な方法について述べる．続いてこれを効率的に行なう方法について述べる．

3.2.1 衝突面検出の基本的な方法と計算量

多面体表現された運動物体間の衝突または干渉している面を正確に検出する最も単純な方法は，2.2.1節でも述べたとおり，物体の運動をある時間間隔ごとにサンプリングし，各離散時刻ごとに作業空間中の物体のすべての面と稜線の組合せに対して，交差の有無を調べるというものである [Boy79]．しかしアプリケーションによっては，物体間の干渉を検出するのではなく，実際に衝突が起きる前に正確にこれを検出することが必要となる．例えば，2 次元または 3 次元の移動ロボットの動作シミュレーションを行う場合，ロボットのある離散時刻ごとの位置と方向のみが既知であると仮定すれば，現在の地点で現在の運動を続けた場合，次に自己位置を認識するまでの間に他のロボットや障害物と衝突するかどうかを判断しなければならない．また仮想現実環境を利用して仮想物体を直接操作するといったシミュレーション [竹村 94] においても，同様のことが言える．これらのためには，サンプリングされたある時間間隔ごとの静的な検査だけではなく，離散時間内の物体の運動を補間または予測することが必要である．そこで物体の面各々の端点の軌道やエッジの軌道面を数式で表し，これらと物体の面やエッジの交点を，すべての組合せについて解析的に解くことにより，衝突する時刻と場所を導くことができる．但し数式を解く手続きを簡単にするために，片方の物体の静止を仮定して物体の運動を平行移動とある軸回りの回転に制限したり [Boy79]，等速直線運動またはある直線の周りの等角速度運動を仮定したり [Can86] ，物体の運動を 3 次関数で表す [岡野 88] などの簡略化が見られる．

また物体とその運動によって掃かれる空間を考え，この掃引物体間の干渉を調べる方法もある．更に時間軸を含めた 4 次元空間を導入した例 [Cam90] もあるが，いずれも原理的には物体と掃引物体を 3 次元または 4 次元の volume で表現しなければならないという問題もあり，計算量の増大から，物体形状や運動が複雑な場合には使いづらい．

3.2.2 実時間衝突面検出

多面体表現された物体間の衝突や干渉を効率良く検出するための方法として，階層的な外接図形を用いた例 [Hah88] や， octree や voxel などの付加的な形状表現を用いた例 [GSF94] もあるが，余分なデータ構造をもつ必要があるため領域計算量を増大させるといった問題がある．

物体に自由で複雑な運動を許し，その物体のある離散時刻ごとの位置と方向のみが既知であるとした条件で，離散時刻の検査で物体の連続的な動きの中で生じる面間の衝突を見逃すことなく検出する方法として，単位時間内に各面が掃く掃引物体が干渉していれば，これらの面は衝突すると判断する方法を考える．しかし，面ごとに掃引物体を作成したりこれらの間の干渉を検出するのに多くの計算量が必要なため，複雑な多面体表現された物体間の衝突面を実時間で検出することは困難であった．しかしこの過程は，比較的簡単な計算の繰返しであるため，この部分を効率的なアルゴリズムを採用するなどして，ある程度の高速化を図ることはできる．しかし，複雑な物体が数多く存在するような環境に対して実時間で処理をするには限界がある．このような複数のデータに対して同一の計算を行うような処理は，データ並列アルゴリズムとして比較的容易に高速化を図ることができる．

一般に移動ロボット等は，センサ情報の処理時間などに制約されたある離散時刻ごとに自己の現在位置と方向を確認する．また仮想現実環境を利用して仮想物体を直接操作するシミュレーション [竹村 94] などの場合も，同様に物体の位置や方向のデータはある離散時間間隔ごとに獲得される．例えば，画像処理を用いて計測する場合は，通常，処理に数フレーム要するとして毎秒数回〜十回程度の計測が可能であり，また後者のアプリケーションで位置，方向のセンサとして近年よく用いられることの多い磁気センサでは，およそ毎秒数十回の計測を行なうことができる．本研究では，運動物体の位置と方向を毎秒十回程度獲得すると仮定し，この離散時間 (100 ms) 内の衝突を安全に検出することを実時間処理の目安とする．この値は，人間の知覚特性に関する実験値（知覚プロセッサの周期時間） [CMN83] と等しく，衝突面検出結果をコンピュータグラフィックスを用いて視覚的フィードバックによるユーザインタフェースの一部として応用した場合に，人間に自然な運動印象を与えると言われる平均的な値である．

3.3 衝突面検出の最適化アルゴリズム

本節では，衝突面検出のための最適化された基本アルゴリズムについて述べる．これをもとにした並列アルゴリズムについては次節で述べる．

3.3.1 前提条件

本節で述べる衝突検出のアルゴリズムでは，いくつかの前提条件をおいた．作業環境中の物体はすべて多面体で表現されており，物体の形状には制限はなく，凸，凹どちらでもよいとする．物体は並進と回転を含む運動をしており，また形状の変形も許すものとする．そして，ある時間間隔 Δt ごとの離散時刻 $(\cdots, t_{i-1}, t_i, t_{i+1}, \cdots)$ の各物体の位置と方向，また変形がある場合には各面の頂点位置が既知であるとする．また離散時間内の物体（また頂点）の移動量は，3.3.2節で述べる手順（step 3）で使用する面 octree の最下層の node の立方体 (voxel) の大きさよりも十分に小さいとする．つまり，作業空間全体を1辺の大きさ L の立方体，面 octree の深さを l，空間中 j 番目の物体の時刻 t における位置ベクトルを $\vec{p}_{object_j}(t)$ としたとき，これらの関係は次式で与えられる．

$$|\vec{p}_{object_j}(t_i) - \vec{p}_{object_j}(t_{i-1})| \ll L/2^l$$

このような条件下で作業環境内のすべての物体を対象にして，衝突する「面」の組合せを特定する．

3.3.2 処理手順

概 要

Fig. 3.1に，衝突検出の流れを示す．実際に衝突が発生する直前にこれを検出するため，各離散時刻 t_i の運動パラメータを用いて1単位時間後の位置と方向を推定し，衝突する可能性のある面の組合せを検出する．この目的のため，手順の最終段階では面同士の衝突を直接調べる必要があるが，これは計算量の多い過程である．従って，ここで調べる衝突面候補の数を効率良く絞り込むため，疎密的な方法を用いる．まず物体の外接直方体を用いて大まかに干渉している物体を検出する．次に外接直方体の重なり領域と交差する面を抽出する．ここで抽出された面の衝突可能性を次の2段階の手順で調べる．まず，これらの面が共通の空間を共有するか否かを，各々の面 octree による空間分割により大まかに調べる．次に面 octree が干渉している面の組合せごとにその衝突可能性を詳しく調べる．こうして各段階の処理を

効率良く組み合わせることにより，最終段階の面間の衝突検査の回数（計算量）をできるだけ減らすようにした．以下，各手順を詳しく説明する．

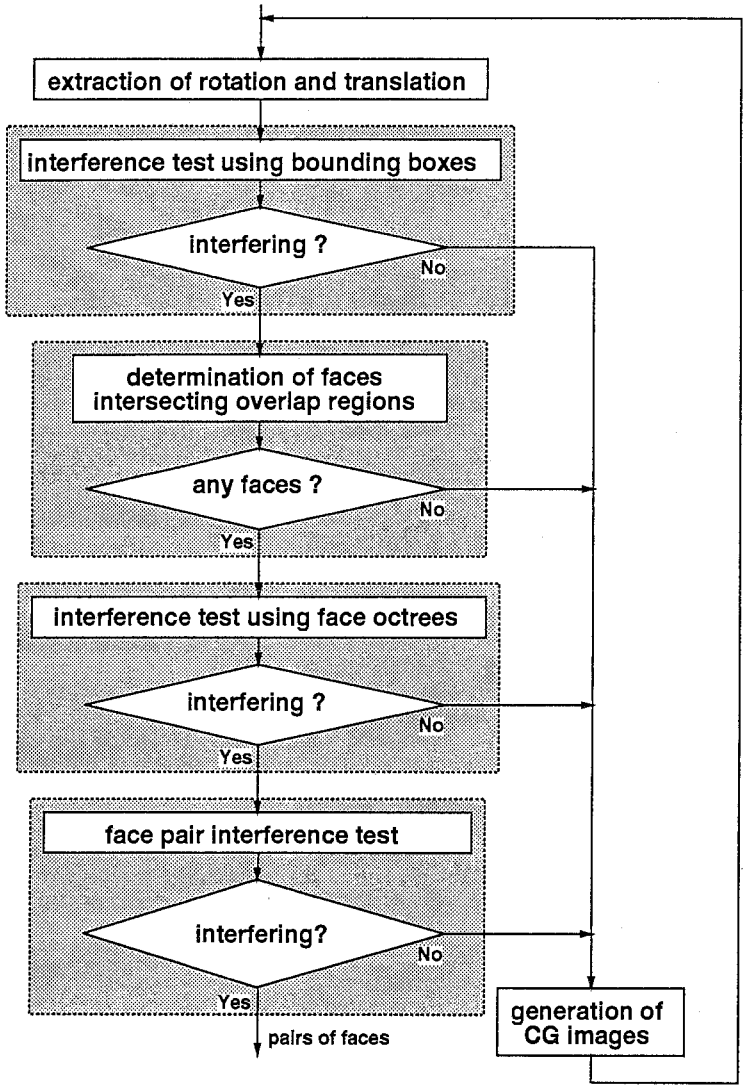


Fig. 3.1: Control flow of collision detection.

外接直方体を用いた大まかな干渉検出 (step 1)

各時刻において，作業環境中の各物体に対して世界座標系の座標軸に各辺が平行な外接直方体を生成し，これらの頂点座標値を全組合せにわたって比較することによって，これらに重なり領域(overlap region)があるかどうかを調べる（**Fig. 3.2**参照）．重なり領域が見つかればその物体をリストに蓄え，次のステップに進む．外接直方体に重なりが見つからなければ衝突検出処理を終了し，次の時刻の処理を行う．

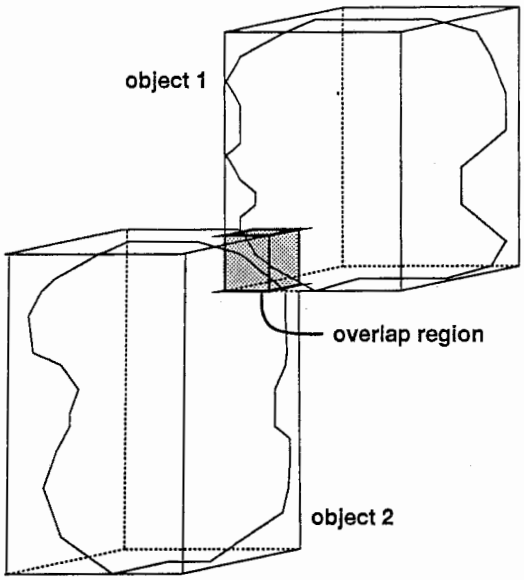


Fig. 3.2: The bounding boxes and overlap region

重なり領域と交差する面の抽出 (step 2)

上のステップで重なり領域が見つかった各物体に対して，その物体を構成するすべての面と重なり領域との交差を調べ，その重なり領域に含まれるかまたは交差する面を抽出する（**Fig. 3.3**参照）．ここで抽出された面（候補面と呼ぶことにする）を候補面リストに蓄える．候補面が 2 つ以上の物体から抽出された場合，次のステップでこれらの衝突を調べる．そうでなければ，処理を終了し，次の時刻の衝突を調べる．

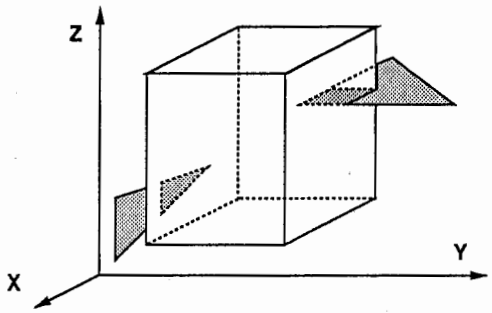


Fig. 3.3: Faces intersecting the overlap region

面 octree による空間分割 (step 3)

上で抽出された候補面間の衝突を正確に調べる前に，まずこれらの面が空間的に共通する部分をもっているかどうかを大まかに調べる．まず，候補面リストから候補面を 1 つ選び，これが世界座標系で占有する空間を，ある適当な深さの octree で表す．これを面 octree と呼ぶことにする．具体的には，作業空間全体を 1 辺の大きさ L の立方体とし，これを root node として，順次それを 8 分割した木で表現する．各 node は白 と黒 にラベル付けされる．全く面と交差しない空間を white node とし，そうでない node（面と交差する空間を示す node）はいったん gray node とする．そして 8 つの子 node に更に分割し，あらかじめ決められた深さ l に達すると，この node を black とする．ここで，octree の最下層の node (voxel) の大きさは，面が 1 離散時間内に移動する量よりも十分に大きいものとする．

原理的には，各候補面毎にこうした面 octree を作成し，これらの木を並列に root node から順にたどって，複数の木の同じ場所（同一の空間を示す node）が black node であることを見つければ，これらの面 octree は干渉しており．もとの面は衝突する可能性が高いと判断できる．しかし本研究では，この手順を次のように最適化して効率的な処理方法を実現することにした．

まず，面 octree 作成とこれを使った干渉検出の過程を分離せず，同時に行なう．更に，候補面を順に処理するのではなく，すべての候補を同時に同一の木を用いて処理する．つまり 2 つ以上の物体からの候補面が含まれている場合に限り，その node を 8 つの子 node に分割し，候補面との交差を調べる．そして，最下層のレベルにおいてもし 8 つの子 node がすべて候補面と交差していても，これらはそのままにしておく（通常の octree による形状表現ではこの場合，8 つの子 node を消去してその親 node を black node に変化させる）．そうすると，面 octree の最下層に node (voxel) が存在すれば，この voxel には複数の物体からの面が含まれることになる．このような各 voxel に対して，同一の voxel に含まれる，異なる物体からの面の組合せを重複がないように選び出し，面ペア検査リストに書き込む．

以上の処理において，実際に octree を作成する必要はなく，またこれを記憶するためにメモリを割り当てたり (allocate)，解放したり (deallocate) する必要はない．単に，世界座標系の座標軸に平行な辺をもつ 1 辺の長さ $L/2^{\lambda}$ $(\lambda = 0, 1, 2, \cdots, l)$ の立方体と，候補面との交差を繰り返し調べるだけである．従って計算資源を不必要に使用することなく，高速な処理が可能である．

候補面と世界座標系の座標軸に平行な辺をもつ立方体の交差は，実際の立方体と多面体の間で正確に判定している．多面体の外接直方体や，立方体の外接球などを

用いて近似的に交差を判定する例 [MT83] もあるが，実行しなければならない交差判定の回数が増加するためかえって計算量が増加するので，ここでは採用していない．

面ペア間の正確な衝突検出 (step 4)

上で作成された面ペア検査リストの各面ペア間の衝突を正確に調べる．そのため，物体の連続な運動中に生じる衝突を離散時刻の検査によって検出するために，2.3.4節で述べた方法により，ある特定の時刻のみの面間の干渉を調べるのではなく，任意の時刻 t_i において，時間 $[t_i,\ t_{i+1}]$ に各面が掃く掃引物体が干渉していれば，これらの面は時刻 t_i と t_{i+1} の間に衝突すると判断することとした．

3.4 並列処理による高速化

3.3.2節に述べた一連の衝突面検出の手順の中で，step 3 と step 4 に要する計算量は他の手順 (step 1, 2) に比べると大きく，これらの処理の高速化が全体の処理時間を左右すると考えられる．特に step 4 では，面ペア検査リストに含まれる複数の面ペア各々に同一の手順で処理する結果，計算量が多くなっている．一方この基本手順は，物体の多面体表現以外の余分なデータ構造を用いていないため，メモリ資源を浪費したりこれらをアクセスするために頻繁に通信するといったことがない．そこでこの手順が，データ並列アルゴリズムとして比較的容易に実現可能である点に着目し，衝突面検出の並列処理による高速化を図る．

3.4.1 共有メモリを用いた並列アルゴリズム

3.3.2節で述べた衝突面検出手順をデータ並列アルゴリズムとして実現する基本的な考え方は，step 4 を並列処理することである．すなわち，step 3 で作成された面ペア検査リストの要素である面ペアごとに，異なったプロセッサでこれらの衝突検出処理を行なうようにする．これを効率的に実行させるためには，使用するプロセッサの負荷分散が均一になるように，面ペアを各プロセッサに割り当てる必要がある．そのためには，適当なプロセッサ間で面ペアや衝突の結果などのデータを高速に送受しなくてはならない．一般に並列計算機では，プロセッサ間の通信手段として，すべてのプロセッサから平等にアクセスできる共有メモリを利用する方法と，プロセッサごとに局所的なメモリ（分散メモリ）をもち．必要なデータの送受を通信路を介して行なうメッセージパッシングの方法がある．前者の場合は，バスを経由して共有メモリにデータを読み書きすることによりプロセッサ間の通信を行うの

で，時間がかかると言われている．しかし，我々が用いた計算機は，Silicon Graphics Onyx で，24 個の R4400 RISC プロセッサ（150MHz）と 512 MByte の主記憶（共有メモリ）をもつ．各プロセッサは 16 Kbyte の命令 / データキャッシュと 1 Mbyte の 2 次命令 / データキャッシュをもち，共有メモリは四つのバンクに分かれている．このように，共有メモリを利用した並列処理に適したアーキテクチャとなっている．また使用した OS は IRIX 5.2 で，共有メモリを用いた並列処理を行うための並列ライブラリーを含んでおり，システムコール (m_fork) を用いることによって生成された複数のプロセスは，共有メモリ上の同一のアドレスをアクセスできる．

以下，MIMD（複数命令ストリーム，複数データストリーム方式）の共有メモリをもつ密結合の，並列度が 10 程度の並列計算機のアーキテクチャを想定し，前章で述べた手順を高速化するための 2 種類の並列アルゴリズムについて述べる．

3.4.2 静的負荷分散型並列アルゴリズム

まず，比較的計算量の少ない step 1，2，3 を逐次的に一つのプロセッサで処理し，step 4 の処理のみを並列化することを考える．step 1, 2, 3 の結果，3.3.2節で述べたように面ペア検査リストを作成するが，この完成された面ペア検査リストを全プロセッサからアクセス可能な共有メモリ上に書くようにする．そして step 4 では，面ペア検査リストの面ペアを複数のプロセッサに均等に分配して，各面ペア間の衝突を調べる．つまり，もし N 個のプロセッサがある場合，その中の i 番目のプロセッサは，面ペア検査リストの i, $i+N$, $i+2N$,... 番目の面ペアを取り出して処理することになる．この場合，面ペアのプロセッサへの割当ては固定的なものとなるため，各プロセッサの処理時間が等しい場合には，全体の負荷が均等に分配され良い結果をもたらすことが期待できる．

3.4.3 動的負荷分散型並列アルゴリズム

上で述べた静的負荷分散型のアルゴリズムでは，面ペアのプロセッサへの割当ては固定的なものとなるため，各プロセッサの処理時間が等しくない場合には，プロセッサ間に負荷のばらつきが生じ，結果として十分に高速化が図れない．実際 3.3.2節の step 4 の手順では，割り当てられた面ペアが衝突していないと比較的早期に判断できる場合もあるが，衝突している場合などはすべての検査を経なければならず，これがプロセッサ間の処理時間の不均一の原因となる．

そこで，面ペアを負荷が少ないプロセッサに動的に割り当てるようなアルゴリズムを考える．まず，step 1 と 2 を逐次的に一つのプロセッサで処理する．続いてあ

る一つのプロセッサがstep 3 の処理を行ない，面ペアを作成するが，ここでは面ペアを一つ発見するたびに直ちにこれを全プロセッサからアクセス可能な共有メモリ上に順次書いていく．他のプロセッサは，この面ペアを一つずつ読み込んでstep 4 の手順に基づいてこの面ペアの衝突可能性を判定する．各プロセッサは面ペア一つの処理を終えると，第1のプロセッサによって追加されたがまだ処理されていない面ペアを読み込み，同様の処理を行う．ここで複数のプロセスからの共有メモリへのアクセスに関しては，システムコール (*m_lock*) を用いて排他制御することができる．

本アルゴリズムは，第1のプロセッサが面ペア検査リストを生成する処理と各面ペアの衝突検査を同時に実行する．つまり，step 3 と 4 を並列処理しているという点，および動的にプロセッサを割り当てているので負荷のバランスがとれている点で，上で述べた静的負荷分散型のアルゴリズムよりも高速化が図れると予想される．

3.5 実験

本節では，提案した衝突面検出法を用いた実験結果について述べる．まず標準物体（球）を用いて，1つのプロセッサを用いた逐次処理，複数のプロセッサを用いた並列処理について，基本的な性能を評価する．

球は全方向に対してほぼ均等に面が張られているので，初期位置や運動パラメータの与え方の違いによる実験結果の変動が小さい．球という対象物体に限れば，2球の中心間距離と半径の和を比較する方法や他のパラメトリックな方法が，衝突を検出する方法として考えられるが，このような方法は他の凹や複雑な形状の物体に対して一般的に用いられるべき方法ではないので，ここでは用いていない．また，凹物体や一般形状の物体を実験に用いることは可能であるが，衝突箇所によって全く異なった結果が得られてしまうので，ここでは用いていない．

3.5.1 逐次処理による実験結果

3角形パッチによって表現された球状の物体を用いて，提案手法を評価した．**Fig. 3.4**に例を示すように，物体表面を構成する面数が異なる複数種の球を用いて，運動2物体に対する衝突面を検出した．実験では，同一の2物体を作業空間中の適当な位置に十分離して配置し，これらに互いに衝突させるような並進と回転の運動パラメータを与えた．以下の実験では，各時刻における処理に要する計算時間（CPU 時間）を計測した．なおいずれの実験も，深さ6の面octreeを用いている．

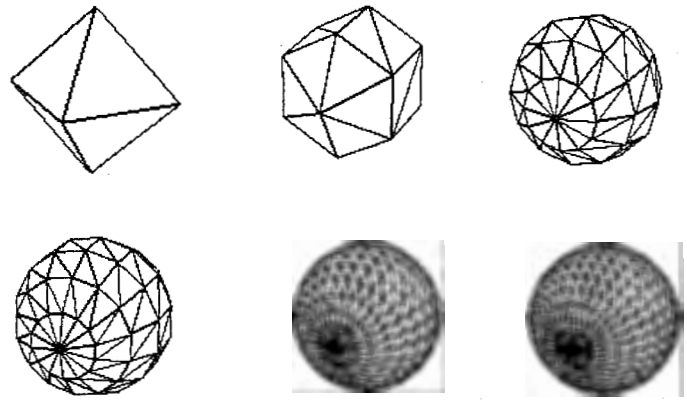


Fig. 3.4: Examples of experimental objects (standardized spheres with different numbers of faces)

Fig. 3.5に，面数 960 と 3968 の 2 種類の球に対して，3.3節で述べた手順をそのまま，1 つのプロセッサを用いて逐次的に処理させた場合の実験結果を示す．両物体が十分に離れている間は，包含直方体による処理 (step 1) だけであり，この過程は 2(ms) 程度で処理が完了している．物体が互いに近づくにつれ衝突判定に多くの時間を要するようになり，3968 面の球同士の場合，$t = 45$(cycle) には step 3 の面 octree による空間分割の処理が加わって，12(ms) 程度の処理時間がかかっている．そして最終的に時刻 $t = 72$(cycle) で，1,160 組の面ペアの検査をして 121 組の衝突する面を検出するのに，434(ms) の時間が必要であった．ここで，面間の衝突を詳しく調べた面ペアは，可能な面の全組合せの 0.007(%) であり，step 1 〜 3 の処理で十分に候補が絞り込まれていることがわかる．また，960 面の球同士の場合も同じようなグラフの形状であるが，衝突面検出の最終段階で，186 組の面ペアの検査をして 21 組の衝突する面を検出するのに，126(ms) の時間がかかった．この場合は，全組合せの約 0.02(%) の面ペアに対して詳しく衝突の検査をしているにすぎない．

3.5.2 並列処理による実験結果

上の実験で，最も計算量が多いのは，衝突面検出の最終段階（3.3.2節の step 4）である．実際，一つのプロセッサを用いた逐次処理の場合に，衝突面検出の最終時刻 $t = 72$(cycle) に step 1 から 4 の各処理に要する時間を計測すると，**Table 3.1** のような割合であった．この表からも，全体の 83% と最も処理時間を要する step 4 の過程を並列化し，高速化することが全体の処理時間を左右することがわかる．

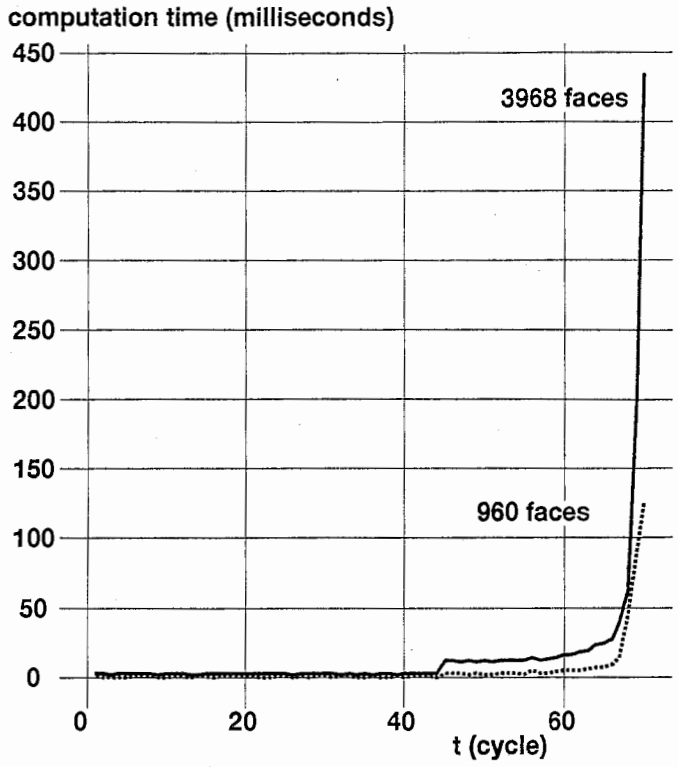


Fig. 3.5: Computation time for each processing cycle for two standarized objects by sequential algorithm

Table 3.1: Computation time for each processing step

Processing step	computation time (%)
step 1	0.5
step 2	2.4
step 3	13.2
step 4	83.9

これに対し，処理を単純化して高速化を図るという考え方もある．たとえば計算量が多くなる掃引物体間の干渉を調べるという方法ではなく，step 4 の処理を簡素化して，単純に時刻 t_i における 2 つの面間の交差を調べる，という方法も考えられる．step 4 をこの手法に置き換えた場合の処理時間の推移を **Fig. 3.6**に示す．図では，上と同じ実験条件で 3,968 面の標準物体（球）間の衝突をさせた場合，3.3節で述べた方法による処理時間を Method 2，その step 4 のみを単純な面間の交差判定に置き換えた方法による処理時間を Method 1 として表示している．単純な面間の交差

判定に置き換えた場合，約 70 (ms) で衝突面を検出できているが，離散時間内の物体間の衝突を見落とす恐れがある．

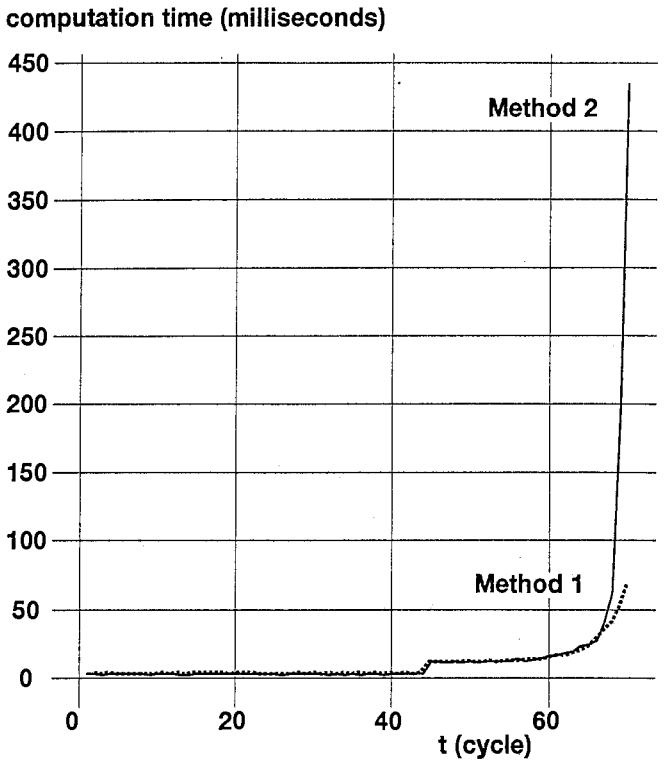


Fig. 3.6: Comparison between two face pair tests: computation time for each processing cycle for two standarized objects

静的負荷分散型並列処理による実験結果

まず，3.4.2節で述べた静的負荷分散型並列アルゴリズムの実験結果について述べる．上と同じく，同一の二物体を作業空間中の適当な位置に十分離して配置し，これらに互いに衝突させるような並進と回転の運動パラメータを与えた．衝突面検出の最終時刻における処理に要する計算時間（CPU 時間）を，使用するプロセッサの数を 1 から 12 まで変化させて計測した．3,968 面の球同士，960 面の球同士の場合の結果を各々 **Fig. 3.7**，**Fig. 3.8**に示す（図中，static と付してある）．3,968 面の球同士の場合，一つのプロセッサを使用した場合には約 430(ms) 要していたが，プロセッサ数を増加させるとともに処理時間は減少し，12 個のプロセッサを用いた場合 120(ms) で衝突面を検出できている．しかしプロセッサの数を 9 個以上に増やしてもそれほど処理時間が短縮できていない．

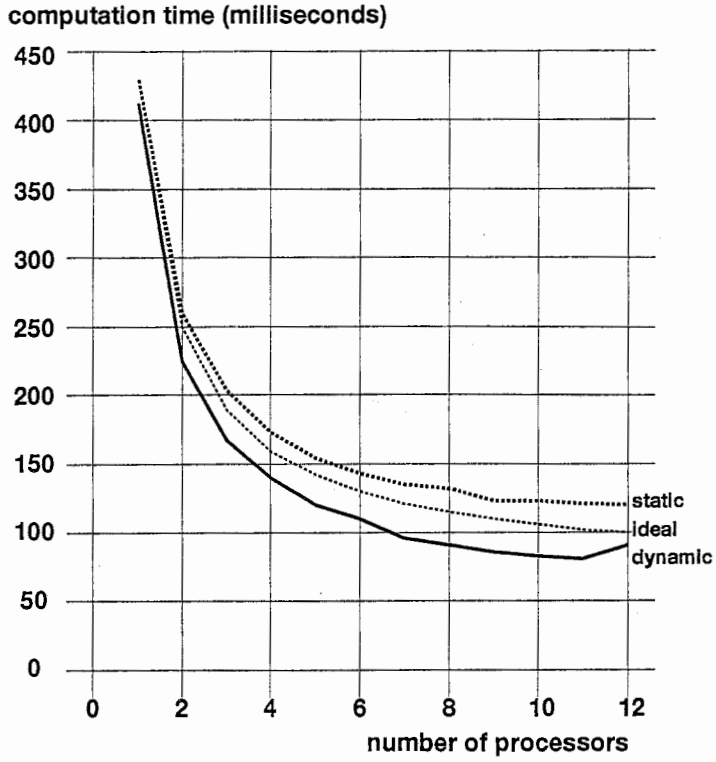


Fig. 3.7: Computation time at the last cycle of collision detection for two standarized objects (3968 faces) by parallel algorithm against the number of processors

動的負荷分散型並列処理による実験結果

続いて，3.4.3節で述べた動的負荷分散型並列アルゴリズムを，上と同じ条件で実験した結果を **Fig. 3.7**，**Fig. 3.8**に示す（図中，dynamic と付してある）．3,968 面，960 面いずれの場合も，今回実験した 12 個までのプロセッサ数に対しては，上の静的負荷分散型並列アルゴリズムよりも動的負荷分散型の方が高速に衝突面を検出できている．3,968 面の球同士の場合，11 個のプロセッサを使用した場合に 80(ms) で衝突面を検出している．また 960 面の球同士の場合，8 個のプロセッサまでは使用するプロセッサの増加に対して処理時間は単調に減少し，8 個のプロセッサで 27(ms) で衝突面を検出しているが，8 個以上プロセッサ数を増加させると逆に処理時間が増加している．これは，静的負荷分散型の並列アルゴリズムに比べて各プロセッサからの共有メモリへのアクセス回数が増えるため，これらの排他処理とバスを介したデータ転送そのものに時間が要しているためと考えられる．

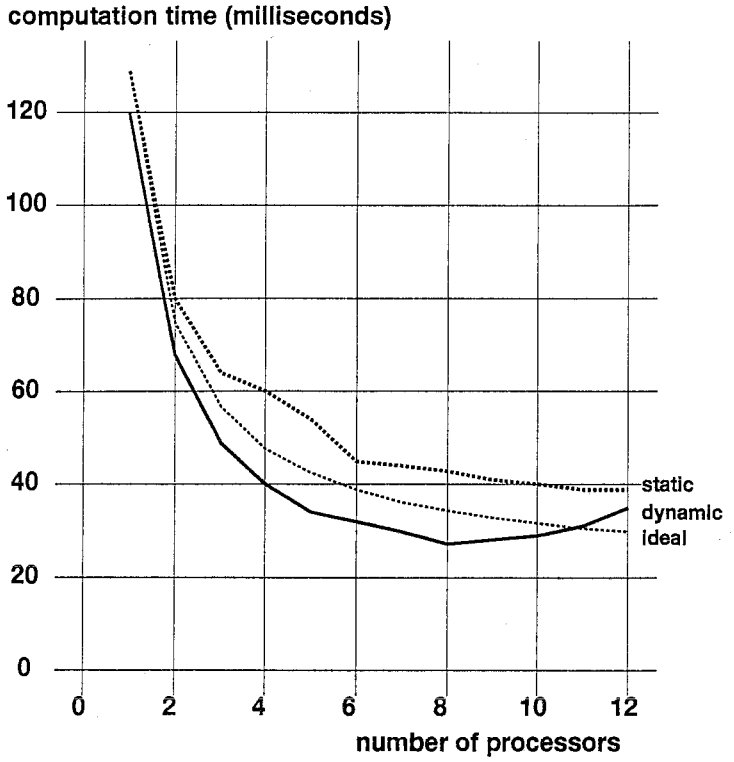


Fig. 3.8: Computation time at the last cycle of collision detection for two standarized objects (960 faces) by parallel algorithm against the number of processors

3.6 考察

本節では，2種類の衝突面検出の並列アルゴリズムの実験結果について考察する．

3.6.1 通信量に関する考察

3.4節で述べた並列アルゴリズムでは，いずれも共有メモリを利用してプロセッサ間の通信を行った．しかし通常は，複数のプロセッサが同時に共有メモリにアクセスすることはできないため，一つが共有メモリ上のデータを読み / 書きしている間はロックをかけ，他のプロセッサのアクセスを許さないように排他処理を行う．従って，共有メモリをアクセスする回数が多いと並列化の効果を妨げる原因となる可能性がある．今，3.3.2節の step 3 の処理の結果，P 個の面ペアが要素として格納されている面ペア検査リストが生成される場合を考える．すると静的負荷分散型の並列アルゴリズムの場合，その並列処理の段階では面ペア検査リストは既に完成しているので，プロセッサからは合計 P 回共有メモリ上の面ペア検査リストを参照すればよい．もし P の面ペアのうち p 組が衝突していれば，その結果をやはり共有メモリ

上に書くので，合計 $P+p$ 回共有メモリをアクセスすることになる．一方，動的負荷分散型の並列アルゴリズムでは，面ペア検査リストを作成しながらその要素を取り出して検査するので，同じ状況では合計 $2P+p$ 回共有メモリをアクセスすることになる．従って動的負荷分散型の並列アルゴリズムは，静的負荷分散型の並列アルゴリズムに比べて各プロセッサからの共有メモリへのアクセス回数が増えるため，これらの排他処理に時間がかかると考えられる．実際前章の 3,968 面の物体間の実験では，最も計算量が多い衝突の最終時刻では，1,160 組の面ペアの検査をして 121 組の衝突面が検出されていた．この場合 1 回の処理の間に，動的負荷分散型の並列アルゴリズムは 2,441 回，静的負荷分散型の場合は 1,281 回，共有メモリにアクセスしていることになる．

また，1 回にかかる通信の時間も問題になる可能性がある．本並列アルゴリズムのように，共有メモリを利用してプロセッサ間の通信を行う場合は，バスを経由して共有メモリにデータを読み書きするので，この通信にかかる時間がボトルネックになると言われている．本アルゴリズムのプロセッサ間の通信には，共有メモリを介したデータ（面ペアなど）の読み書きと，排他制御を行なうためのシステムコールなどがある．特に本実験で用いた計算機は高速の RISC プロセッサを使用しており，プロセッサの処理速度に比べてバス上のデータ転送速度が遅く，結果として，使用するプロセッサ数を増加させても処理速度が向上しない上限が存在すると思われる．本実験の場合，**Fig. 3.7**，**Fig. 3.8**からは，8–10 個のプロセッサを用いた場合にこの上限を迎えていると思われる．

3.6.2 並列化の効果に関する考察

3.4節で述べたいずれの衝突面検出の並列アルゴリズムも，基本的には 3.3.2節の step 4 の処理を並列化している．ここでは特に静的負荷分散型のアルゴリズムについて，プロセッサ間の通信などのオーバヘッドが全くない理想的な場合を想定し，期待できる並列化の効果を考えてみる．もともと，3.5.1節の逐次的な処理の実験結果で，各過程に要する処理時間の割合は **Table 3.1**のようであった．今，仮に N 個のプロセッサを用いて 3.4.2節の静的負荷分散の方法で step 4 の処理を並列化した場合を考える．プロセッサ間の通信などのオーバヘッドは無視するので，step 4 に要する処理時間のみが $1/N$ となり，他の過程 (step 1–3) に要する処理時間（t_1, t_2, t_3）は変化しない．従ってこの場合，期待できる処理時間 T_N は，一つのプロセッサのみを使用した場合の step 4 の処理時間を t_4 として，

$$T_N = t_1 + t_2 + t_3 + \frac{t_4}{N}$$

で与えられる. **Fig. 3.7**, **Fig. 3.8**では, プロセッサ数 N を変化させたときの, 面数3,968と960の物体間の衝突各々について, 上の式で与えられる値を併せて示している（図中, idealと付してある）. この値は, 3.4.2節の静的負荷分散型の並列化の方法で期待できる, 理論上の限界値であると考えられる. 実際には, 共有メモリをアクセスしたりする際にオーバヘッドが生じるので, この値よりも多くの時間がかかっている. また, プロセッサ数が増すにつれてこのオーバヘッドも増加することが予想されるが, **Fig. 3.7**, **Fig. 3.8**は, その考察の結果をよく表している.

3.7 3章のまとめ

以上, 3次元空間内の複雑な形状の運動物体間の衝突面を, 並列計算機を用いることによって実時間で検出する方法について述べた. まず, 離散時間の衝突検査で複雑な軌道をもつ物体間の衝突面を見逃すことなく, 実際に衝突が起きる直前に効率的に特定する基本アルゴリズムについて説明した. 本章では, これを更に高速に動作させるため, プロセッサ間の負荷分散の考え方が異なる2種類の並列アルゴリズムを提案した. 標準物体を用いた実験では, まず一つのプロセッサを用いてこれを逐次的に処理させた結果, 4,000面程度の物体間の衝突面を, 434(ms) で検出できた. 更に並列アルゴリズムを共有メモリをもつMIMD型の並列計算機を用いて実験を行い, 4,000面程度の物体間の衝突面を, 最高 80 (ms) で検出できることを確認した. しかし本並列アルゴリズムではプロセッサ数を増加させても処理速度が向上しない上限があった. これは, 共有メモリを介したプロセッサ間の通信がボトルネックになっているものと考えられる. プロセッサ間の通信手段としてメッセージパッシングを利用した場合や, 分散メモリをもつ場合など, 異なったアーキテクチャのための並列アルゴリズムを検討していくことが今後の課題である.

第 4 章

octree とポテンシャル場を用いた 3 次元動的環境での経路探索

無人のヘリコプターや潜水艇など，3 次元の動的環境を移動する自律ロボットのための効率的かつ簡単な経路計画の手法を提案する．手法は，ロボット，静止障害物，移動障害物の環境中の全ての物体を，その運動可能性を区別することなく octree で表現し，障害物を表す octree の各 black node を基準としてポテンシャル場を生成してこれを利用することにより 3 次元移動ロボットの障害物に衝突しない経路と向きを発見する．本章で述べる手順は簡単な計算の繰り返しとして実現できるため，データ並列アルゴリズムとして容易に並列処理による高速化を図ることができる．いくつかの実験結果を通して，簡単なアルゴリズムでも効率的に（環境にも依存するが，市販の計算機を用いて数秒程度で）3 次元の動的環境での経路が発見されることを示す．また，環境の複雑さと経路探索の計算時間について考察し，本手法の効率性と問題点を議論する．

4.1 背景

無人のヘリコプターや潜水艇など，3次元を移動する自律ロボットの研究が盛んになってきている．これらのロボットの動作環境は2次元世界のそれよりも苛酷であるため，高度な運動能力が求められる．空中や水中に漂う障害物に衝突しない初期位置から目的地までの経路を発見しようとする経路計画の問題にも，2次元の移動ロボット以上に効率的な方法が求められる．経路計画は自律移動ロボットの基本的な問題として，多くの研究がなされてきた [Lat91, HA92a, 藤村 93]．これらの多くは，主に2次元の環境を対象として良い結果を出すことが報告されているが，3次元またはそれ以上の次元をもつ環境に対しては多くの計算時間がかかることが指摘されている．

ポテンシャル場を用いたアプローチ [MA85, Kha86] は代表的な方法の1つである．障害物からの斥力と目的地への引力をスカラー量であるポテンシャルを用いて表現し，ポテンシャルの谷間を縫うことによって障害物に接触しない初期位置から目的地までの経路を探索する．障害物からの斥力を表すポテンシャルとして最も単純なものは，障害物が凸形状であればその輪郭線からの距離に応じて単調に減少するような関数を用いるものである（たとえば [HA92b]）．しかし障害物の形状が凸形状でなければ，これを複数の異なった形状の凸物体の集合に分割してからポテンシャルを計算するという手段がとられることが多いが，一般に分割された個々の物体の辺の数など形状が一定ではなく，これが効率的なポテンシャル場生成への妨げとなっていた．

quadtree や octree など階層的な空間分割法によって環境を表現するアプローチも代表的な方法であり，効率的に経路を探索する手法として用いられることが多い．しかしながら，一般に移動する物体を quadtree や octree を用いて表現するのが困難であるため，移動障害物やロボットは他の静止障害物と異なった形状表現方法が用いられることが多い．たとえばロボットの形状として，点や円に限定して用いた例 [KD86]，他の単純な形状 [Her86] や簡単な多角形を用いた例 [NNA89] などがある．いずれの場合も，ロボットや移動障害物は他の静止障害物と異なった形状表現方法が用いられているため，経路計画のアルゴリズムは複雑になり，また一般的な動的環境での利用が困難となる．

本章では，ロボット，静止障害物，移動障害物の環境中の全ての物体を，その運動可能性を区別することなく octree で表現する経路計画の方法を提案する．障害物を表す octree の各 black node を基準としてポテンシャル場を生成し，これを利用して3次元移動ロボットの障害物に衝突しない経路と向きを発見する．いくつかの

実験結果を通して，簡単なアルゴリズムでも効率的に3次元の動的環境での経路が発見されることを示す.

4.2 環境の表現

4.2.1 octreeによる動環境の表現

octreeによると物体は，全体空間をroot nodeとし，順次それを8分割された木で表現される[Sam90]. 各ノードはwhite nodeとblack nodeにラベル付けされる. white nodeは完全に物体の外部の空間を示し，black nodeは完全に物体の内部の空間を示す. そうでないnode（物体の内部と外部の両側にまたがる空間を示すnode）はgray nodeとされ，あらかじめ決められた最小の大きさに達するまで8つの子nodeに分割される. octreeによって障害物を表現すれば，ロボットの初期位置と目的地を結ぶ障害物に衝突しない経路は，隣り合うwhite nodeの並びとして効率的に見つけることができる.

環境中の静止障害物を表現する手段としてoctree（または2次元環境でのquadtree）を用いた例は多い. quadtreeを用いた階層的な経路探索法[KD86]はquadtreeで表現された静止環境下での手法を提案しているが，ロボットの断面形状は円に限定され，また移動障害物を含む動的な環境には利用できない. また，同じくquadtreeを静的な環境表現に用いた例[NNA89]もあるが，多角形で表現されたロボットを用いている. octreeに基づいた方法[Her86]も提案されているが，やはり静的な環境に限定され，ロボットの形状は球や円柱など基本形状を組み合わせたものとして与えている. 障害物の運動が完全に既知な動的環境でoctreeを用いて経路探索を行なった例[FS89]もあるが，やはりロボットの形状は点として扱われる. これらの例が示すように，quadtreeやoctreeなど階層的な空間分割法によって環境を表現する方法では，ロボットや移動障害物の形状が単純なものかまたは他の方法を用いて表現されるため制約が多く，動的な環境への応用が困難である. ロボット，静止障害物，移動障害物の全てをoctreeという同じ形状表現方法で与えている例は見当たらない.

octreeは3次元空間の階層的な表現法として一般的な方法であり，octreeデータの生成，変換などの操作に関しても多くの有効な手法が提案されている[CH88, Sam90]. octree表現された物体の運動に伴ってtreeを更新する方法もいくつか提案されている[AN84, WA87]が，中でも本論文5章で述べる，任意の並進と回転によりoctreeを実時間で更新する手法では，市販の計算機（本章4.4節で使用したものと同じ）を用いて1回当たり平均36(ms)の計算時間で，860余りのblack nodeから構成される

物体の octree の更新を完了することが報告されている．本章ではこの octree 更新の方法を利用することにより，複雑な一般形状のロボット，静止障害物，移動障害物の環境中の全ての物体を，その運動可能性を区別することなく octree で表現し，運動が観測されればその octree 表現を更新する．

4.2.2 ポテンシャル場

ポテンシャル場を用いたアプローチも，移動ロボットの経路探索の方法として多くの研究がなされてきた [MA85, Kha86]．障害物からの斥力と目的地への引力をスカラー量であるポテンシャルを用いて表現し，ポテンシャルの谷間を縫うことによって障害物に接触しない初期位置から目的地までの経路を探索する．簡単な原理から多くの場合に良い結果を導くので，ポテンシャルの発生方法に関しても多くの研究例が見られる．たとえば楕円ポテンシャル [奥富 83] や，ラプラスのポテンシャル [CB90, 佐藤 93] などがあるが，特に後者の場合，ラプラスの方程式を解きポテンシャルを計算するのに莫大な計算時間がかかるという問題があった．

本研究では，障害物からの斥力を表すポテンシャルのうち，最も単純なものの 1 つとして論文 [HA92b] で用いられている，障害物の輪郭線からの距離に応じて単調に減少する関数を用いる．3 次元環境中の点のポテンシャルは，次式の関数 p_{HA} を用いて表される．

$$p_{HA} = \frac{1}{\delta + \sum_{i=1}^{s}(g_i + |g_i|)} \tag{4.1}$$

ただし，g_i は，凸領域の輪郭を表す線形関数（$\bigcap_{i=1}^{s} g_i \leq 0$ は凸領域を表す），s は輪郭セグメントの数，δ はある正の定数である．p_{HA} は領域の内部で最大値 $1/\delta$ をとり，領域外部では距離に反比例して単調に減少する．この関数 (4.1) では領域は凸形状である必要があるため，非凸の領域の場合にはこれを複数の凸物体の集合に分割してからポテンシャルを計算する必要がある．しかし，一般に分割された個々の物体の辺の数など形状が一定ではなく，これが効率的なポテンシャル場生成への妨げとなっていた．

そこで我々は，障害物を表す octree の各 black node をポテンシャル計算の基本要素とし，関数 (4.1) を用いて環境全体のポテンシャル場を生成することとした．したがって上式中，いつも $s = 6$ である．octree の各 node の形状は立方体であるため，同一形状で大きさだけが 2^n 倍ずつ異なる凸の領域であることが保証されている．したがってポテンシャル場は非常に簡単な方法で計算できる．またこの過程は簡単な計算の繰り返しとして実現できるため，データ並列アルゴリズムとして容易に並列処理による高速化を図ることができる．

4.3 経路計画の方法

ロボット，静止障害物，移動障害物の環境中の全ての物体を，octree で表現し，障害物を表す octree の各 black node を基準として計算されたポテンシャル場を評価関数として経路を探索する方法について述べる．

4.3.1 octree 表現された環境の更新

以下では，環境中の各物体の octree 表現とそれぞれの位置と方向は既知であるものとする．物体の octree 表現を作成する方法としては，カメラで撮像された画像や距離画像などセンサ情報などから作成する方法や，多面体表現など他の形状表現方式から変換する方法などが提案されている [CH88].

octree 表現された環境中の物体（ロボット，障害物）に運動（並進と回転）のパラメータが観測されると，その物体を表現する octree を更新する必要がある．ここでは，任意の並進と回転により octree を実時間で更新する手法として，本論文 5章で述べる方法を用いた．このアルゴリズムは基本的には，各 black node 毎に並進と回転のマトリクスを施してこれの部分木を作成し，最後にこれらを組み合わせて全体の octree 表現を作成するものである．この計算量は元の octree の black node の数に比例するため，本論文 5.3.2節で述べる，与えられた物体を前もってより少ない数の cube で表現するように圧縮変換する方法が利用できる．さらに効率的な立方体の交差判定の手法を導入し，また 5.4.4節の，複数のプロセッサを用いた並列アルゴリズムも利用できる．

4.3.2 ポテンシャル場の生成

障害物からの斥力と目的地への引力をスカラー量であるポテンシャルを用いて表現する．障害物からの斥力を表すポテンシャルとしては，最も単純なものの 1 つとして論文 [HA92b] で用いられている関数を用いる．障害物を表す octree の各 black node をポテンシャル計算の基本要素とし，環境全体のポテンシャル場を生成する．

番号 j の物体 O_j の i 番目の black node $B_{i,j}$ が 3 次元環境中の点 $A(x,y,z)$ に生成するポテンシャルを，次式 (4.2) を用いて表す．

$$p_{i,j} = \frac{p_{max}}{1+g} \tag{4.2}$$

ここで,

$$\begin{aligned} g(x,y,z) &= (x_0 - l/2 - x) + |x_0 - l/2 - x| \\ &+(x - x_0 - l/2) + |x - x_0 - l/2| \\ &+(y_0 - l/2 - y) + |y_0 - l/2 - y| \\ &+(y - y_0 - l/2) + |y - y_0 - l/2| \\ &+(z_0 - l/2 - z) + |z_0 - l/2 - z| \\ &+(z - z_0 - l/2) + |z - z_0 - l/2| \end{aligned} \tag{4.3}$$

ただし, p_{max} は, 最大のポテンシャルを表す定数, (x_0, y_0, z_0) は, $B_{i,j}$ の中心座標, l は, $B_{i,j}$ の1辺の長さとする. この関数は, $B_{i,j}$ の内部で最大値をとり, 外部では $B_{i,j}$ からの距離が増大するにつれて単調に減少する.

点 A におけるポテンシャルは, 自由空間でポテンシャルの局所最大が生じないようにするため, 全ての black node から計算されるポテンシャルの最大値として与えることとする. つまり, m を障害物 O_j を表す octree の black node の数, n を環境中の障害物の数として,

$$\begin{aligned} P_o = \ & Max\{p_{i,j}\}. \\ & (1 \leq i \leq m,\ 1 \leq j \leq n) \end{aligned} \tag{4.4}$$

最後に, 目的地への引力を表すポテンシャルとして, 目的地からの距離に応じて値が単調に増加する (4.5) 式で与えられる値を用いた.

$$P_g = C\sqrt{|x - x_g|^2 + |y - y_g|^2 + |z - z_g|^2} \tag{4.5}$$

ただし, $G(x_g, y_g, z_g)$ は目的地の座標, C は定数である. 本論文では, 環境内のポテンシャルとして, P_o と P_g の合計で表される値 P を用いることとする.

$$P = P_o + P_g. \tag{4.6}$$

以上述べたポテンシャル生成の過程は, 簡単な計算の繰り返しとして実現できるため, データ並列アルゴリズムとして容易に並列処理による高速化を図ることができる. また, この計算量は元の octree の black node の数に比例するため, 4.3.1節と同様, 与えられた物体を前もってより少ない数の立方体で表現するように圧縮変換することによっても, さらに高速化を図ることができる.

4.3.3 経路の探索

3次元環境の中でロボットの初期位置から目的地への並進と回転を含む障害物に衝突しない経路は, 隣り合う領域 (white node) の並びとして効率的に探索することが

できる．ここで，ポテンシャルの 1 つの node 内の平均値を評価関数として利用し，障害物が移動しない静的な環境であることがわかっていれば，最良優先探索法で経路を探索する．障害物も移動するような動的な環境である可能性がある場合には，現地点から隣の white node のうちどの node に向かえばよいかを山登り法を用いて決定する．**Fig.4.1**に，動的環境での経路探索の手順を示す．

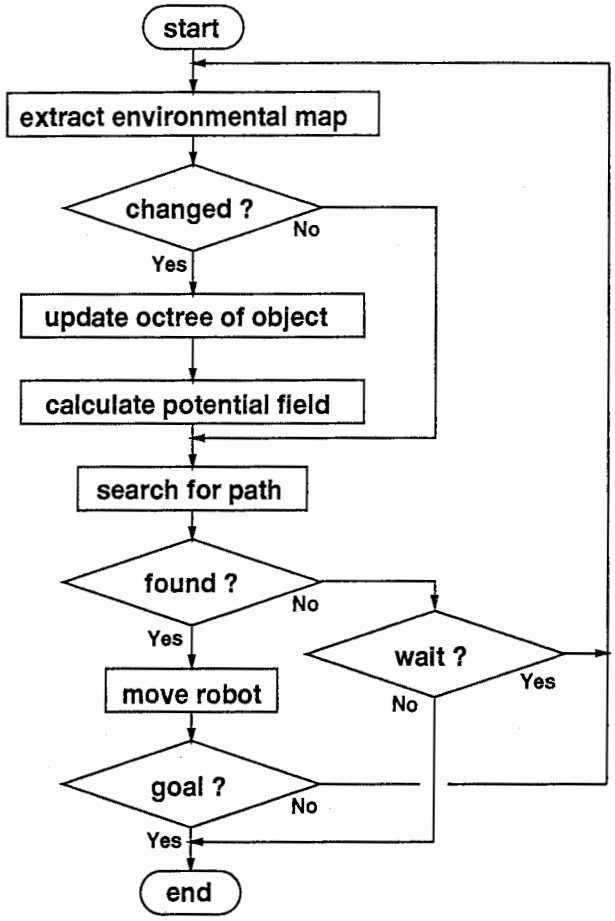


Fig. 4.1: Control flow of the proposed path planning in a dynamic environment.

4.3.4　一般形状ロボットの回転と並進

本研究では，点や球など単純化された形状のロボットではなく，octree 表現された一般形状のロボットを用いる．そこで，ロボットの octree 表現の全 black node が占有する空間のポテンシャル値の総合計を，ロボットの運動を決めるための評価関数として導入する．ここでは，ロボットの並進運動には回転よりも高い優先順位を与えている．したがって，ロボットは障害物に衝突しない経路を，まずその並進運動だけで発見しようとする．もしこれに失敗すれば，探索木の leaf node の 1 つにお

いて，回転を試みる．こうしてロボットが狭い回廊を通過する場合には，その向きを回転させて進むことができる．この時，上の評価関数を最小とする回転角度を見つけるが，計算時間短縮のため，各軸方向に α 度ごとの離散的な角度に対して評価値を計算し，この中からあるしきい値以下で最小となる角度を選択する．octree表現されたロボットの並進と回転運動のために，4.3.1節で述べたoctreeの更新方法と同じ手順を用いる．

4.4 実験

提案した手法を用いて，静的環境と動的環境に対する一般形状のロボットの経路探索を行なった実験結果を述べる．説明を簡単にするため，まず2次元の環境に対する実験結果を述べ，その後3次元の環境に対する結果を述べる．

4.4.1 2次元環境に対する実験

静的環境に対する実験

2次元の静的環境で実験を行なった．ここで用いたロボットは**Fig. 4.2**に示すような形状をしている．図ではquadtreeで与えられたロボットの形状（左上）と，これを30度ずつ順に回転させたものを示している．環境とロボットの初期位置(start)と目的地(goal)が与えられると，まずポテンシャル場を計算する．**Fig. 4.3**は，計算されたポテンシャル場を鳥瞰的に眺めたものである．続いて**Fig. 4.4**にその結果を示すように，障害物に衝突しない経路が探索される．図では，各quadtree nodeのポテンシャルを濃淡で表している．すなわち，薄い色は低いポテンシャル，濃い色は高いポテンシャルの値を表し，初期位置と目的地以外の最も濃い色（黒い部分）は障害物の内部であることを示している．ロボットの経路は障害物に衝突しないwhite nodeの並びとして，ポテンシャルを評価関数とした最良優先探索法で探索された．中央部に • 印のついたnodeは探索されたnodeを示し，これらを接続した折線は，経路を表すnodeの中央と隣接するnodeの共有辺の中点を接続し，最終的に発見された経路を示す．環境中には一部幅が狭い部分があり，その部分を通過できるようにロボットの向きを変えていることがわかる．

動的環境に対する実験

Fig. 4.5に示すように，上の実験とは異なった形状のロボットをquadtreeで与え，動的環境に対する実験を行なった．**Fig. 4.6**(a)のように障害物の配置と，ロ

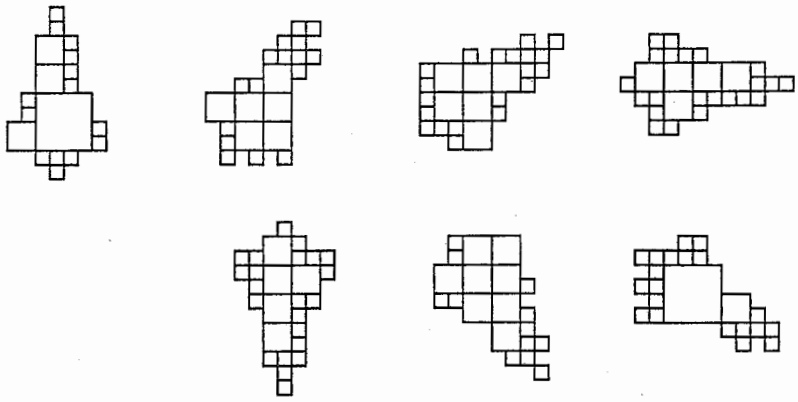


Fig. 4.2: An experimental robot represented by a quadtree for Fig. 4.4. The upper left one is the original shape, and the others are rotated shapes of the robot.

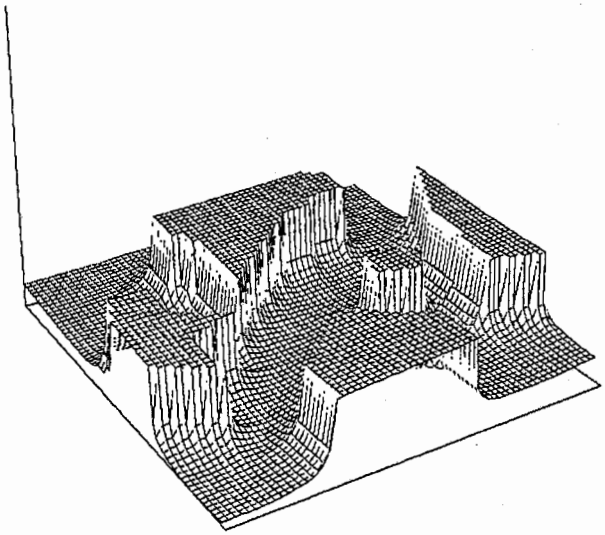


Fig. 4.3: Generated potential field for the given environment in 4.4.1.

ボットの初期位置と目的地を与え，経路探索の実験を開始した．ここで，環境中の全ての障害物もやはり quadtree で表現されている．開始後しばらくして，中央部の大きな障害物が移動したことが観測されたとする．この観測結果をもとにして **Fig. 4.6**(b) のように環境を更新し，経路探索を続けた．障害物も quadtree で表現されているので，その更新方法はロボットに対する方法と同一の手順である．上の実験と同様，各 quadtree node のポテンシャルを濃淡で表し，ロボットの経路はポテンシャルを評価関数とした山登り法で探索された．中央部に • 印のついた node は探

Fig. 4.4: Experimental result of an arbitrarily shaped robot in the static environment of Fig. 4.3.

索された node を示し，これらを接続した折線はそのつど決定された経路を示す．ロボットは，**Fig. 4.6**(a) の中央部の大きな障害物によって道がふがれているため，目的地に向かえなかったが，障害物が移動した結果同図 (b) のように目的地へ向かう経路が発見された．しかし一部幅が狭い部分があり，その部分を通過できるようにロボットの向きを変えていることがわかる．

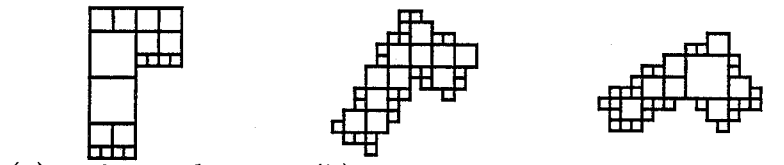
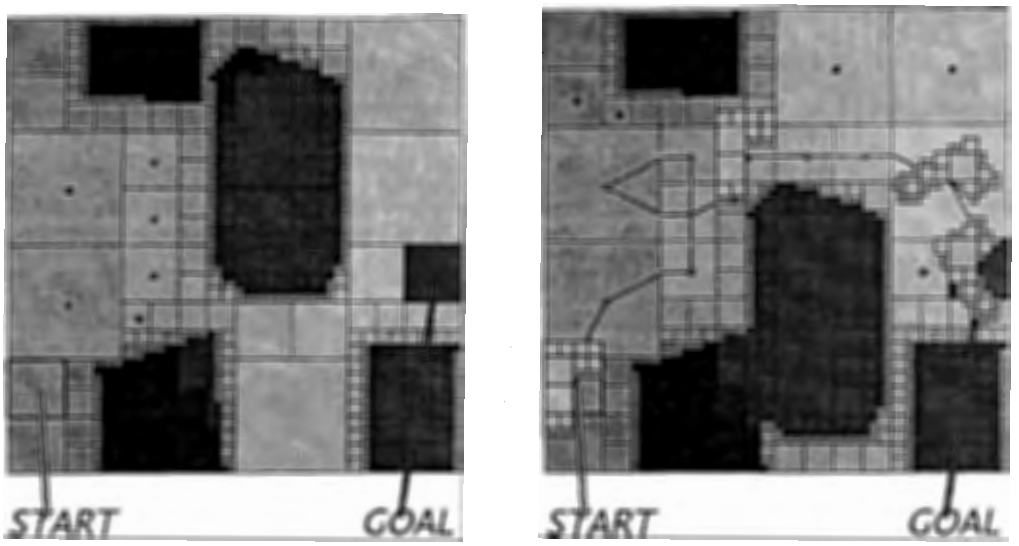


Fig. 4.5: An experimental robot represented by a quadtree for 4.4.1 with rotated shapes.



(a) before movement of an obstacle (b) after movement of an obstacle

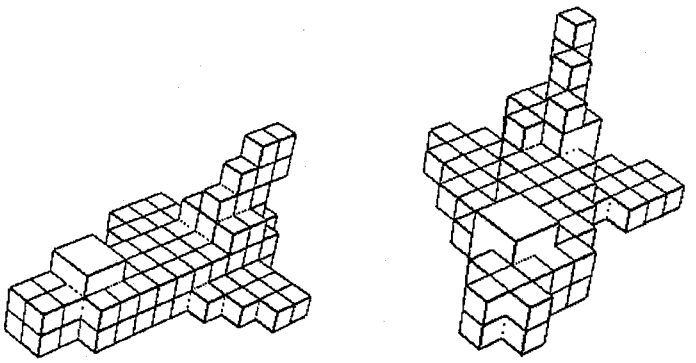
Fig. 4.6: Experimental result of an arbitrarily shaped robot in a dynamic environment.

4.4.2 3次元環境に対する実験

3次元環境に対する実験を行なった．まず，**Fig. 4.7**に示すようにoctreeで表現されたロボットを用いて経路探索の実験を行なった．2次元の場合と同様，環境が与えられるとまずポテンシャルを計算し，その後経路を探索する．**Fig. 4.8**は，ある与えられた静的環境に対する実験結果である．ここでは，全ての物体は深さ5のoctreeを用いて表現されている．この環境中の障害物のoctree表現は合計981のblack nodeから構成されており，ロボットは初期状態で74のblack nodeから構成されている．ロボットの経路は障害物に衝突しないwhite nodeの並びとして，2次元の場合と同様，ポテンシャルを評価関数とした最良優先探索法で探索された．図中に表示している折線は，経路を表すnodeの中央と隣接するnodeの共有面の中心点を接続し，最終的に発見された経路を示す．図には表示できないが，環境中には一部幅が狭い部分があり，その部分を通過できるようにロボットの向きを変えている．本実験では市販のワークステーション(Sillicon Graphics Onyx)を用いて実験を行なったが，この**Fig. 4.8**の環境に対して，ポテンシャル発生に2.8秒，経路の探索に4.7秒の計算時間を要した．

次に，本手法の効率性と問題点を明らかにするため，9種類の静的環境を用意し，適当に初期位置と目的地を与えて上と同様に経路探索の実験を行い，ポテンシャル発生と経路の探索に要する計算時間を測定した．与えた環境のうち，典型的な3つを**Fig. 4.9**，**Fig. 4.10**，**Fig. 4.11**に示す．これらの環境はそれぞれ，2,689,

303, 1,777 の octree の black node から構成されている. 一部 (**Fig. 4.10**など) は, **Fig. 4.7**とは異なった形状の椅子型のロボット (初期状態で 115 の octree node から構成) を用いても経路探索の実験を行なった. 計測した結果を **Fig. 4.12**に示す. このグラフは, 様々な種類の環境に対して, その中の障害物を構成する octree node の総数を横軸にとり, ポテンシャル発生と経路の探索に要する計算時間を縦軸にとったものである. プログラムは最適化を図られたわけではなく, またこのような計算時間の計測は多少のノイズなどを含みがちであるため傾向を示すのみではあるが, 環境中の octree node の総数にほぼ比例してポテンシャルの発生に要する計算時間が増加していることがわかる. 一方, 経路の探索に要する計算時間は, ロボットの種類や, 同じ程度の octree node の数で構成された環境でも, 幅の狭い箇所などでロボットが回転する必要がある回数などによって大きく計算時間が変わるが, 環境中の octree node の総数が増加して環境が複雑になるにつれ, おおむね長い計算時間がかかるような傾向が見られる.



(a) given robot (b) a rotated robot

Fig. 4.7: Experimental robot (space shuttle) in 3-D environment

4.5 検討と今後の課題

実際に 3 次元動的環境を動き回る自律移動ロボットの経路探索に利用するためには, さらに高速化を図る必要がある. この点から本節では, 提案した経路探索法の計算の効率について検討する.

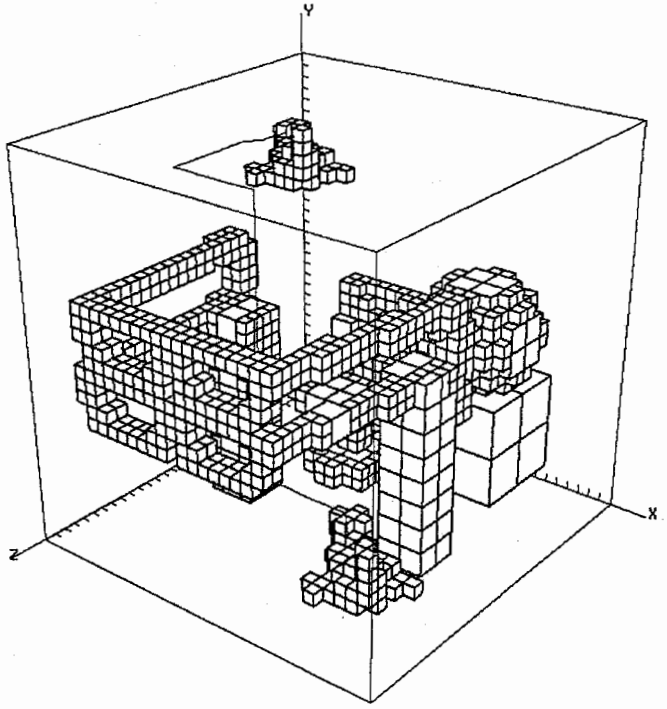


Fig. 4.8: Experimental result of a robot (space shuttle) in a 3-D environment having 981 octree black nodes

4.5.1 ポテンシャルの発生に要する計算時間

動的環境に対する経路探索全体に要する計算時間は，移動障害物の数や移動頻度などにも依存するので一概には議論できないので，ここでは一度環境が定まった場合の静的環境に対する経路探索について考える．経路探索全体に要する計算時間は大きく分けてポテンシャル発生と経路の探索に要する計算時間の和として考えることができる．このうちポテンシャルの発生は 4.4.2節の実験の通り，効率的な計算時間で処理を終了しているが，この計算時間をさらに短縮するためには，アルゴリズムの並列化が考えられる．本論文で提案した手法によるポテンシャル発生の計算時間は，**Fig. 4.12**でも示した通り，ほぼ環境中の octree node の数に比例して増加する．4.3.2節で述べたポテンシャル生成の過程は，簡単な計算の繰り返しとして実現できるため，データ並列アルゴリズムとして容易に並列処理による高速化を図ることができる．つまり，環境中の全ての障害物の各 black node のポテンシャル生成計算を，それぞれ異なったプロセッサで同時に計算しようというわけである．ここでは，各プロセッサに任される各プロセッサごとの計算負荷はほぼ同一であると考えられるので，最も簡単な実現方法として，各 black node を前もって各プロセッサ

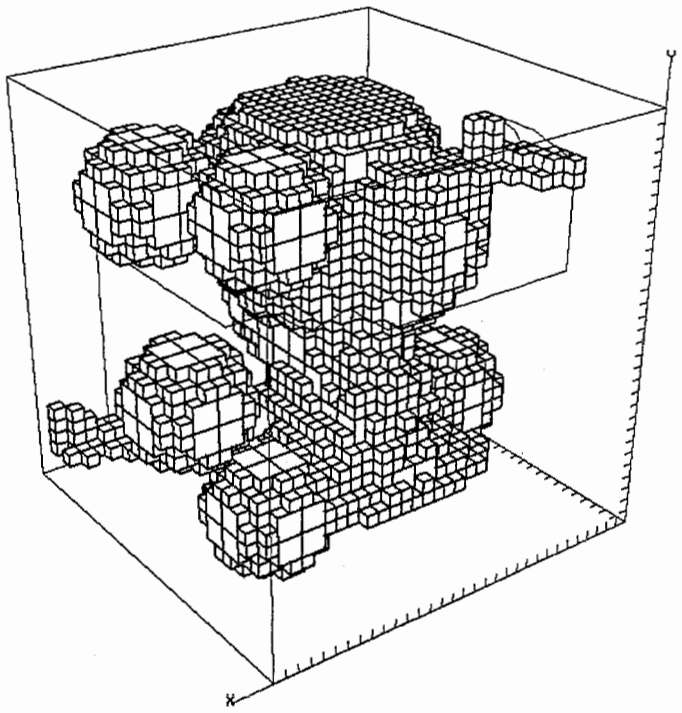


Fig. 4.9: Experimental result of a robot (space shuttle) in a 3-D environment having 2689 octree black nodes

に静的に割り振っておく方法が考えられる．今 N 個のプロセッサがあるとすると，この中の i 番目のプロセッサは，i, $i+N$, $i+2N$, ... 番目の black node を受けとり，これらの black node のポテンシャル計算を順次こなすことになる．これにより，並列度に応じてかなりの高速化が図れるものと考えられる．

また，この計算量は元の octree の black node の数に比例するため，本提案手法をさらに高速に動作させるためには，この node の数を減少させることも重要な項目となる．つまり，与えられた物体を前もってより少ない数の立方体の集合で表現するように圧縮変換することによっても，さらに高速化を図ることができる．このような立方体の集合は，もはや一辺の長さが 2^n 倍ずつ異なる octree ではないが，より少ない数の立方体で完全に octree の black 領域をカバーするような立方体の集合である．厳密な意味で最小の数の立方体の集合を，全ての場合について作成することは困難であり実用的でないが，本論文 5.3.2節で述べる平均的に約半分の数の立方体の集合に変換する方法を利用することも考えられる．

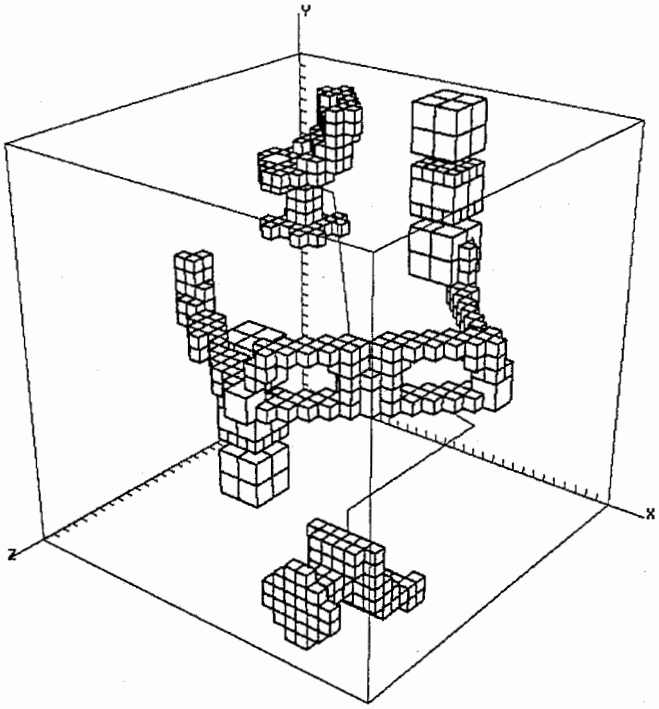


Fig. 4.10: Experimental result of a robot (chair) in a 3-D environment having 303 octree black nodes

4.5.2 経路の探索に要する計算時間

一方，経路探索全体に要する計算時間のうち，経路の探索に要する計算時間についても，**Fig. 4.12**で示した通り，ほぼ環境中の障害物の octree の black node の数に比例して増加する傾向がみられた．環境中の障害物の octree の black node の数が増加すると必ずしも環境が複雑になるとは言えないが，一般的に，環境中の octree node の総数が増加して環境が複雑になるにつれ，おおむねその量に比例して長い探索時間がかかるようになると言える．

前節では十分な種類のロボットを用意することが困難なため今回は実験を行なっていないが，経路の探索に要する計算時間はロボットを表現する octree の node の数にもほぼ比例すると予想される．その理由は，4.3.4節で述べた通り，ロボットの octree 表現の全 black node が占有する空間のポテンシャル値の総合計を，ロボットの運動を決めるための評価関数として利用しているためである．そのため上と同じようにロボットの形状を，前もってより少ない数の立方体の集合で表現するように圧縮変換することによっても，この過程をさらに高速化を図ることができる．この過程は，4.3.1節で述べた octree の更新方法と同じ手順を用いて octree 表現されたロ

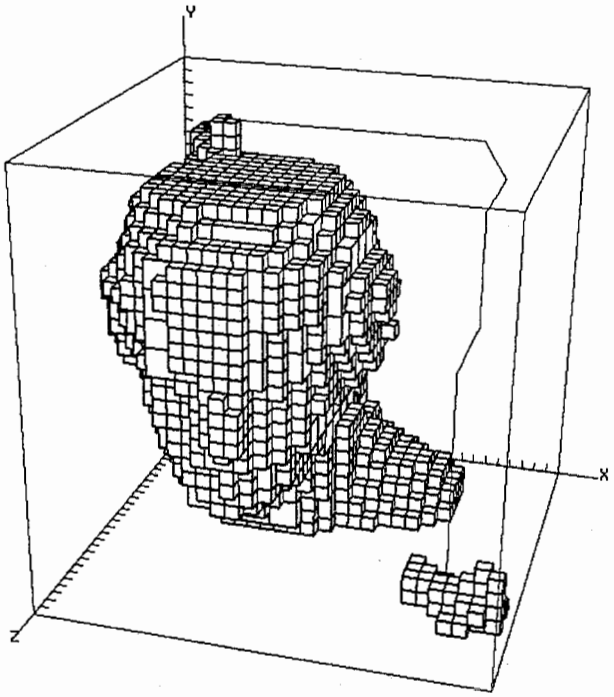


Fig. 4.11: Experimental result of a robot (space shuttle) in a 3-D environment having 1777 octree black nodes

ボットの並進と回転の運動を与えているが，複数のプロセッサを用いた並列処理の利用も，本提案手法をさらに高速に動作させるために考えられる．

4.4節の実験では，木の深さが 5 の octree を用いて実験を行なったが，異なった深さの octree を用いると探索空間の大きさが変化するため，この計算時間はそれにともなっても変化することは容易に想像できる．たとえばこの要因だけを考慮した場合，深さ 4 の octree を用いて環境を表現すれば，経路の探索に要する計算時間は約 1/8 に，逆に 深さ 6 の octree を用いれば 約 8 倍の計算時間が経路の探索に要すると思われる．

また，本研究では最良優先探索と山登り法を用いたが，動的な環境でリーズナブルな解を導くと言われる RTA*[Kor90] や，さらに効率的な発見的探索の方法を検討することも必要かもしれない [RK91].

4.6 4章のまとめ

本章では，簡単なアルゴリズムで効率的に 3 次元動的環境での移動ロボットの経路を探索する方法を提案した．本手法では，ロボット，静止障害物，移動障害物の

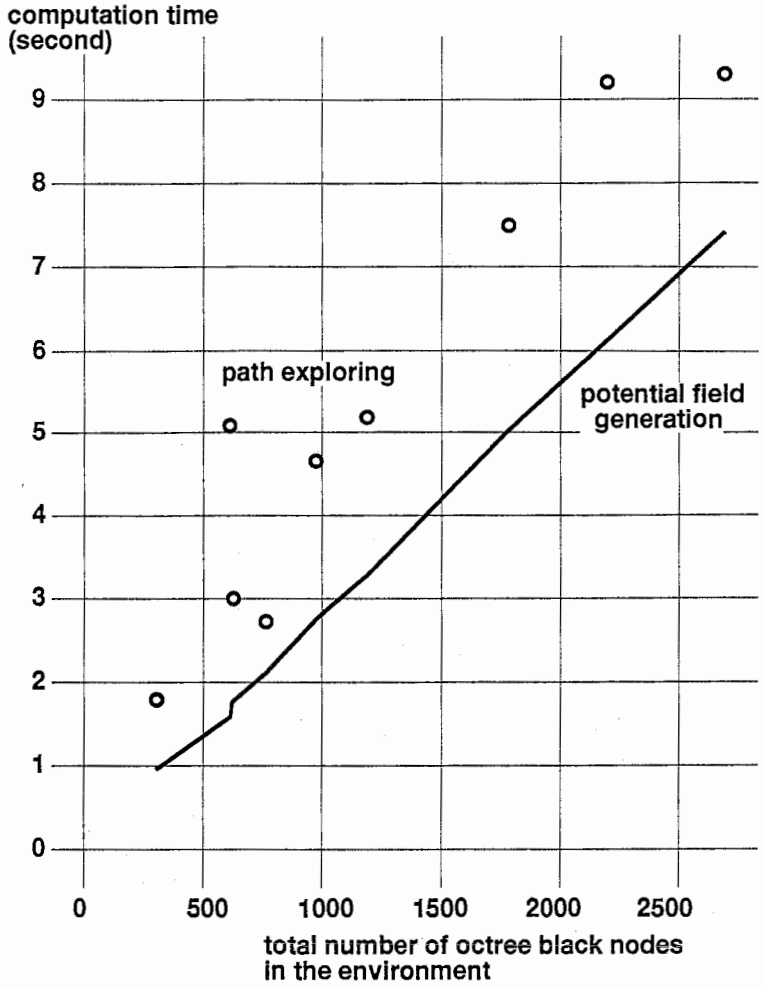


Fig. 4.12: Computation time for potential field generation and path exploring against the total number of octree black nodes in the environment.

環境中の全ての物体を，その運動可能性を区別することなく octree で表現した．そして障害物を表す octree の各 black node を基準としてポテンシャル場を生成し，これを利用して 3 次元移動ロボットの障害物に衝突しない経路と向きを発見した．いくつかの実験結果を通して，簡単なアルゴリズムでも効率的に（環境にも依存するが，市販の計算機を用いて数秒程度で）3 次元の動的環境での経路が発見されることを示した．また，環境の複雑さと経路探索の計算時間について考察し，本手法の効率性と問題点を議論した．

本章で述べた手順は簡単な計算の繰り返しとして実現できるため，データ並列アルゴリズムとして容易に並列処理による高速化を図ることができる．また，障害物を表す octree を，前もってより少ない数の立方体の集合で表現するように圧縮変換

することによっても，この過程をさらに高速化を図ることができる．さらに，経路の探索に要する計算時間についても，ロボットを表現するoctreeを圧縮変換し，また並列計算機を用いることによりさらに高速化を図ることができる．これは今後の研究課題である．

本手法は，障害物とロボットの全ては同じ方法であるoctree（2次元の場合quadtree）で表現されているため，そのいずれかに移動が観測された場合は同じ枠組で簡単に処理することができる．実験では，障害物がある時刻に移動するという条件設定のみで実験を行なったが，複数の移動ロボットが存在するという条件でも同じようにして経路を探索することができる．今後，こういった問題にも取り組んでいきたい．

第 5 章

運動にともなう octree の実時間更新

本章では，回転と並進を含む一般の運動に対して，物体の octree 表現を更新する効率的な方法を提案する．まず物体の octree 表現 (source octree) が与えられると，全体の計算量を減らすため，これをより少ない数の立方体の集合に圧縮変換する．次に立方体 (source cube) を順次取りだし，それぞれに並行と回転の変換行列をかける．そして，移動変換後の（傾いた）立方体と，世界座標系座標軸に平行な立方体との交差判定を再帰的に繰り返すことにより，この立方体の octree 表現 (target octree) を，作成するが，ここでは，両者の交差判定を近似的な方法ではなく，正確な方法を用いて判定する．その結果，交差判定の回数を減少させることができ，計算を効率化することができる．実験により，改善点が効率的なことと，提案手法が従来手法よりも高速に処理できることを確認する．本章では，提案手法をさらに高速に動作させるため，複数のプロセッサを用いて octree 表現を更新する並列アルゴリズムも提案する．最後に，物体の並進運動のみの場合に限って，特に高速に octree を更新する方法について述べる．

5.1 背景

octree は一般的な 3 次元形状表現法の 1 つとして，コンピュータビジョン，コンピュータグラフィクス，ロボットの動作計画など多くの分野で利用されている．それは，階層的に空間を分割することによって物体の形状表現の特定の部分空間へ非常に高速にアクセスができるという特徴を有するためである [CH88, Sam90]．この特徴を生かすため，通常は複数の物体を同時に扱う場合には，空間全体を表す世界座標系を基準として用いて，個々の物体をそれぞれ octree で表現することが多い．しかし，この octree で表現された物体に並進や回転などの移動が生じた場合，新規の位置と方向を反映するように世界座標系の中で octree 表現を更新しなければならない．ところがこの更新は他の形状表現法（たとえば多面体表現など）のように単純な方法では実現できず，計算量が非常に多くなるので，実時間で octree を更新することが困難であるという問題があった．回転と並進を含む任意の運動に対して octree を更新する研究は少なく [WA87, HO87]，実時間のアルゴリズムや並列アルゴリズムが報告された例はない．

octree 表現を更新する一連の手順の中で，立方体同士の交差の有無を判定する段階は単純ではあるが，繰り返しの回数が多いため，全体の処理速度を決定する律速段階であると言える．Weng と Ahuja は，正確な交差判定は計算時間がかかるとし，その外接球を用いて立方体の間の交差を調べるという近似的な方法が効率的であると主張している [WA87]．しかしながら，論文 [WA87] でとられている方法は，幾何学的には，上記外接球のさらに外接直方体を用いた交差判定であるため，2 重に近似をしていることになる．また，もともと近似的な方法では，結果として交差判定をすべき立方体の組合せの数が増加してしまい，全体として処理速度が向上しない．

本章では，回転と並進を含む一般の運動に対して，物体の octree 表現を更新する効率的な方法を提案する．まず一般的な問題点について検討し，特に論文 [WA87] の方法の問題点を明らかにする．続いてこの問題点を解決する効率的な octree 更新の方法を述べる．まず物体の octree 表現 (source octree) が与えられると，全体の計算量を減らすため，これをより少ない数の立方体の集合に圧縮変換する．次に立方体 (source cube) を順次取りだし，それぞれに並行と回転の変換行列をかける．そして，移動変換後の（傾いた）立方体の octree 表現 (target octree) を，この立方体と世界座標系座標軸に平行な立方体との交差判定を再帰的に繰り返すことにより作成するが，ここでは，両者の間の交差判定は，近似的な方法ではなく，正確な方法を用いる．その結果，交差判定の回数を減少させることができ，計算を効率化することが

できる.

実験により，改善点が効率的なことと，提案手法が従来手法よりも高速に処理できることを確認する．本章では，提案手法をさらに高速に動作させるため，複数のプロセッサを用いてoctree表現を更新する並列アルゴリズムも提案する．最後に，物体の並進運動のみの場合に限って特に高速にoctreeを更新する方法を述べる.

5.2 octreeによる形状表現とその運動

ここでは，octreeによる3次元形状表現とその動きにともなう更新について，基本的な方法を述べる.

5.2.1 octreeによる形状表現

octreeによると物体は，全体空間をroot nodeとし，順次それを8分割された木で表現される [Sam90]．各ノードはwhite node とblack node にラベル付けされる．white nodeは完全に物体の外部の空間を示し，black nodeは完全に物体の内部の空間を示す．そうでないnode（物体の内部と外部の両側にまたがる空間を示すnode）はgray nodeとされ，あらかじめ決められた最小の大きさに達するまで8つの子nodeに分割される．この最下層のnodeをvoxelと呼ぶことにする．octreeによる形状表現の例を**Fig. 5.1**に示す．(a)は8分割する際の子nodeの番号，(b)は例として与えられた物体，(c)はそのoctree表現である.

octreeによって表現された形状を記憶するために明示的なポインタを用いた場合，非常に多くのメモリ容量が必要となる．そのため，よりコンパクトな線型の表現方法が提案されている．その1つがoctreeのDF表現と呼ばれるもので [Man88]，octreeを決められた順にたどって出会ったnodeの種類を順にlistに格納するという方法である．たとえば3つの記号を用いてそれぞれ，“(” (gray node)，“B” (black node)，“W” (white node) を表すことにすると，octreeを記憶するのに各nodeにつき2 bitで十分である．例として，**Fig. 5.1** (c)に示すポインタベースのoctreeのDF表現は，次のようになる.

“(B(BWBBBWWWBWBWB(B(BWBBBWBWBBBWBW”

5.2.2 運動にともなうoctree更新の基本的方法

並進と回転を含む任意の運動にともなって物体のoctree表現を更新する基本的な方法を述べる [WA87]．**Fig. 5.2**にその主な手順を示す．まず物体のoctree表現が

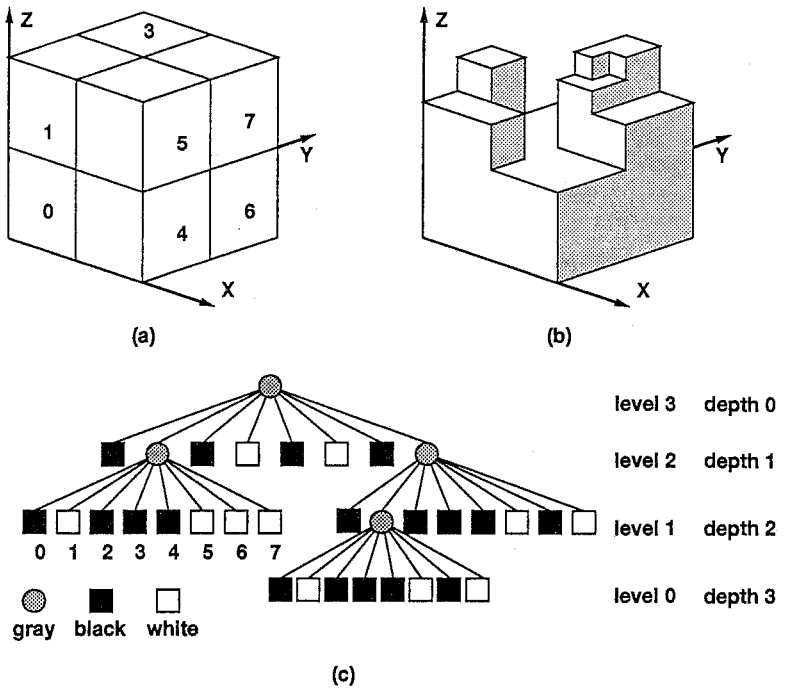


Fig. 5.1: The octree shape representation. (a) is the ordering of octants, (b) is an example octree, and (c) is the pointer-based representation of the example octree.

与えられると（これを source octree と呼ぶことにする），この木をたどって black node を順次見つける．そしてこの各 black node （形状は立方体）ごとに並行移動と回転移動の変換行列をかける．例えば，ある位置ベクトル X の node が，原点を通る単位ベクトル $\vec{n} = (n_x, n_y, n_z)$ 回りの反時計方向の角度 ϕ の回転 R と，平行移動 T により X' に移動したとすると，

$$X' = RX + T \tag{5.1}$$

ただし，

$$\mathrm{R} = \begin{pmatrix} (n_x^2-1)(1-cos\phi)+1 & n_xn_y(1-cos\phi)-n_zsin\phi & n_xn_z(1-cos\phi)+n_ysin\phi \\ n_yn_x(1-cos\phi)+n_zsin\phi & (n_y^2-1)(1-cos\phi)+1 & n_yn_z(1-cos\phi)-n_xsin\phi \\ n_zn_x(1-cos\phi)-n_ysin\phi & n_zn_y(1-cos\phi)+n_xsin\phi & (n_z^2-1)(1-cos\phi)+1 \end{pmatrix}$$

さらに，他の回転 R' と平行移動 T' によって X' から X'' に移動する時，すなわち，$X' = RX + T$ に続いて $X'' = R'X' + T'$ を実行する場合には，

$$X'' = (R'R)X + (R'T + T') \tag{5.2}$$

のように，元のベクトル X に対して回転 $R'R$ と平行移動 $R'T + T'$ を考える．ここで物体の運動は，常に source tree と呼ばれる元のリファレンス octree に対して施す

ため，連続する運動に対する変換の量子化誤差などが生じにくい．（たとえば角度ϕ回転させた後に$-\phi$回転させたときのoctree表現が一致する．）

次に，移動変換後の（傾いた）black nodeのoctree表現を，改めて世界座標系に対して作成する．新しく作成されたoctreeをtarget octreeと呼ぶことにする．ここでは傾いた立方体と，世界座標系座標軸に平行な立方体との交差判定を，空間全体を表す立方体から始めて順に編の長さが$1/2^n$倍の立方体へ再帰的に繰り返す．変換され傾いた立方体と世界座標系座標軸に平行な立方体それぞれとの交差判定の結果，target octreeの各nodeは，双方の交差の状態に応じてblack，white，grayに分類される．ただしこの最下層のnode (voxel)をblackにするかwhiteにするかは，アプリケーションに応じて決定する．この基本的なアルゴリズムを2次元に単純化し，quadtreeのblack nodeが更新される場合の例を**Fig. 5.3**に示す．立方体同士の交差判定の手順は，単純ではあるが繰り返しの回数が多いため，効率的なアルゴリズムを採用すると全体の処理速度を向上させることができる．交差判定の具体的な方法については本章付録で説明し，また5.4.1節でも議論する．

最後に元のoctreeの各black nodeから作成した部分octreeを合成して，物体全体の移動変換後のoctree表現を得る．つまり，target octreeの各gray nodeについて，これらの子nodeが全て同じ色かどうかを調べる．もし全て同じ色なら，全子nodeを削除し，これらと同じ色をこのnodeに与える．

5.3 octreeの圧縮変換

5.2.2節で述べたように，運動にともなうoctree表現の更新のための計算量は，そのoctreeのblack nodeの数に比例するので，この数を少なくすることが効率的なoctree更新への第一歩である．そこでここでは，与えられたoctreeを前もってより少ない数の立方体の集合で表現するように圧縮変換する方法について述べる．

5.3.1 立方体の集合

octree表現された物体の運動は5.2.2節で述べたように，常にsource treeと呼ばれる元のoctreeに対して施すため，多少時間がかかっても前処理として，このsource treeのblack nodeの数を減少させておくことが，運動にともなうoctree表現の効率的な更新のためには必要であると考えられる．そこで5.2.2節で述べた基本的な手順を最適化する方法の1つとして，与えられている物体のsource octree（すなわち，辺の長さが2^n倍ずつ異なる立方体の集合）を，辺の長さの階層性にとらわれずに物体の領域を完全に覆うできるだけ少ない数の立方体の集合として表し直すことを考

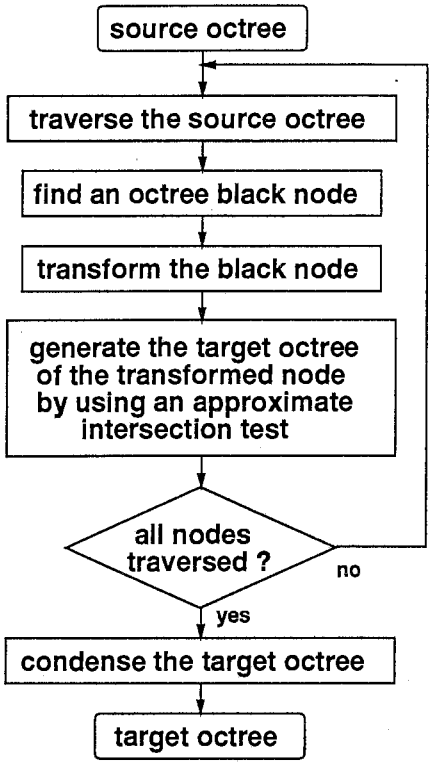


Fig. 5.2: Control flow of octree motion basic algorithm

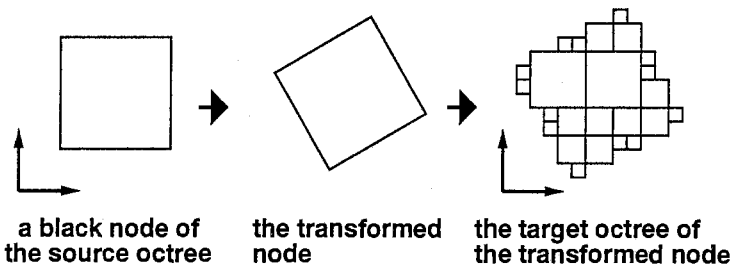


Fig. 5.3: The octree motion basic algorithm illustrated for the 2-D case (quadtree).

える.

与えられた物体の source octree の black node の領域を完全に覆う最小の数の立方体の集合を見つけるには，辺の長さが整数の立方体を順に作り，この全ての組合せについて調べるという方法が考えられる．しかし，最適の場合として最小の数の立方体の集合を見つける問題は，かなり困難であるため，ここでは octree の圧縮変換をすることが，運動にともなう octree 表現の更新のために有効であることを示すため，簡単に圧縮変換をする例を次に述べる.

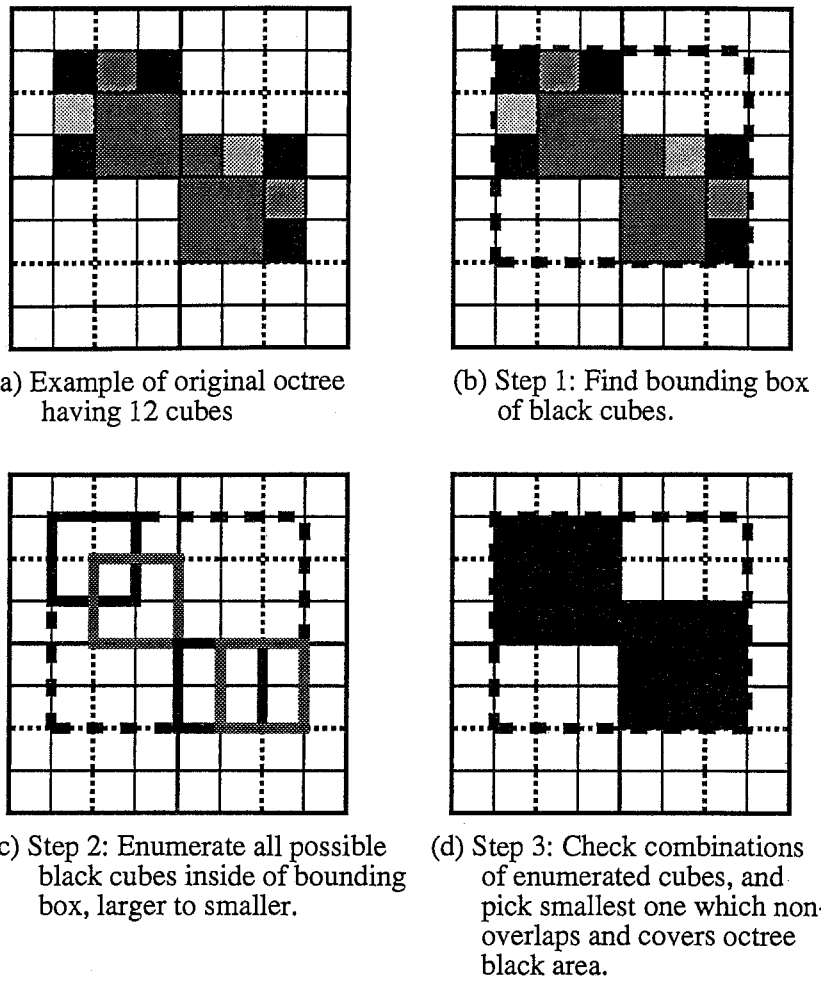
5.3.2 octree 圧縮変換のアルゴリズム

与えられた物体の source octree の black node の領域を完全に覆うより少ない数の立方体の集合を見つける方法を述べる．最初に，物体を表すoctreeの全 black nodeを含み，世界座標系の座標軸に平行な面からなる外接直方体を見つける (step 1)．次に，外接直方体の内部にあって辺の長さが整数の立方体を大きい方から順に作成し，これを外接直方体の内部でスキャンする (step 2)．そして，この立方体が元のsource octree の white node の領域と交差するかを調べる (step 3)．交差しない（すなわち，元のsource octree の black node の領域に完全に含まれる）立方体が見つかると，これをあるリスト L_1 に格納しておく．ここで同じ大きさの立方体は，世界座標系 (x, y, z) のいずれかによって順序付けされている．次にある別の空リスト L_2 を用意する．L_1 から立方体を大きい方から順に取り出し，L_2 の中の立方体と交差しない立方体が見つかるとこれを L_2 に書き込む．L_1 の立方体を全て処理すると，L_2 には与えられた物体の source octree の black node の領域を完全に覆い，互いに重なり合わない立方体の組が作成される．以上で述べた一連のoctree 圧縮変換手順の例を，2次元のquadtreeとして **Fig. 5.4**に示す．図(a)で与えられた物体は12個の black node から構成されているが，最終的に(d)のように2個の立方体（正方形）に圧縮変換されている．

この方法は，厳密な意味で必ずしも最小の数の立方体を発見するわけではないが，少ない数（平均的に約半分の数）の互いに重なり合わない立方体の集合を発見する．またある種の物体においては，立方体の一部で重なりを許す方がより少ない数の立方体の集合が見つかることがあるかもしれないが，ここでは考えないことにする．最適解，すなわち最小の数の立方体の集合を発見する方法は，今後の課題であるが，遺伝アルゴリズムなどを利用することにより，さらに圧縮率を高められると考えられる．

5.4 一般の運動にともなうoctreeの更新

本節では，まず従来手法の問題点を明らかにし，その解決策として立方体間の正確な交差判定を用いた効率的なアルゴリズムを提案する．さらに提案手法を，複数のプロセッサで高速に動作させるための並列アルゴリズムについても述べる．



(a) Example of original octree having 12 cubes

(b) Step 1: Find bounding box of black cubes.

(c) Step 2: Enumerate all possible black cubes inside of bounding box, larger to smaller.

(d) Step 3: Check combinations of enumerated cubes, and pick smallest one which non-overlaps and covers octree black area.

Fig. 5.4: An example of octree compaction for the 2-D case (quadtree).

5.4.1 従来手法の問題点

5.2.2節で述べた基本手順の中で，立方体間の交差判定は単純ではあるが繰り返しの回数が多いため，全体の処理速度を決定する律速段階であると言える．しかし，片方の立方体は世界座標系の座標系に対して平行ではないので，これらの交差の判定はそれほど単純な処理というわけではない．

Weng と Ahuja [WA87] は，世界座標系の座標軸に対して平行な立方体と平行でない（傾いた）立方体の間の正確な交差判定は計算時間がかかるとし，後者の立方体の代わりにその外接球を用意し，これと前者の立方体の間の交差を調べるという近似的な方法が効率的であると主張した．しかしながら，ここに 2 つの問題点がある．第 1 に，上記の主張にもかかわらず論文 [WA87] では，幾何学的には，上記外接球のさらに外接直方体を考え，これと世界座標系の座標軸に対して平行な立方体と

の交差を調べているため，2重に近似をしていることになる（本章付録参照）．第2に，**Fig. 5.5**に示すように，このような近似による交差判定よりも，近似をしない立方体間の正確な交差判定の方が早期に交差していないと判断できる場合が多いため，結果として繰り返す交差判定の回数を減らすことができ，全体として処理速度を向上させることができるという点である．

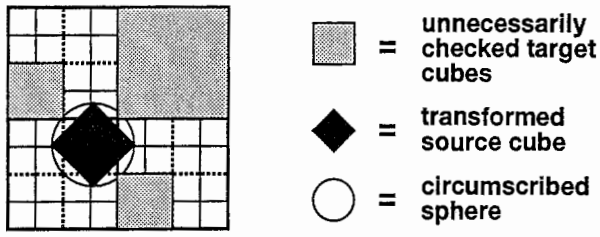


Fig. 5.5: An example of a problem with related work: approximate cube/cube intersection test for the 2-D case (quadtree).

5.4.2 立方体間の正確な交差判定

世界座標系の座標軸に対して平行な立方体と平行でない（傾いた）立方体の間の正確な交差判定は，**Fig. 5.6**に示すように疎密的に段階を追って進められる．ここで，両方の立方体の移動前の（世界座標系の座標軸に対して平行な状態の）8頂点の座標と中心の座標，および並進・回転の変換行列とその逆行列は与えられているものとする．そして結果としてつぎのいずれかを決定する，

- 2つの立方体は交差している
- 2つの立方体は交差していない
- 片方が他方を完全に包含している

外接球と内接球を用いた干渉判定

まず，ここで交差を調べる2つの立方体のうち小さい方を特定する．つまり，source octreeのblack nodeを表す立方体と，今交差を調べているtarget octreeのnodeを表す立方体の大きさを比べる．もし両者同じ大きさであれば，source octreeの立方体の方を小さいとみなすことにする．そして小さい方の立方体の外接球と内接球の直径も決める（それぞれをr_c，r_iとする）．次に，この立方体の中心座標を与えられている並進・回転の行列で変換し，この点を中心にした半径r_cの球（すなわち変

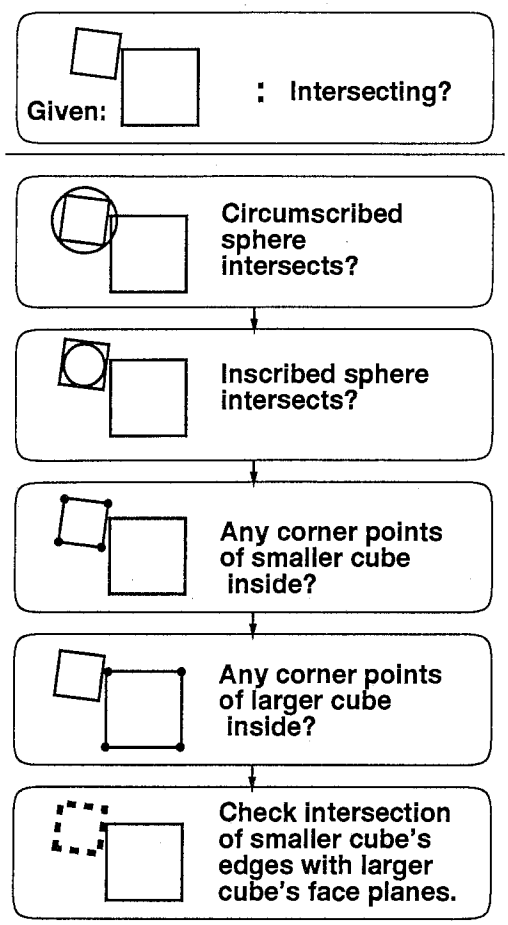


Fig. 5.6: Flow of an efficient, exact cube/cube intersection test.

換後の傾いた立方体の外接球）と r_i の球（同じく内接球）が target octree の node を表す立方体（世界座標系の座標軸に平行）と交差するかどうかを調べる．ここで，実際の球と立方体間の交差判定は，文献 [Gla90]（335 ページ）に記述されている方法を用いる．外接球が交差していなければ，2 つの立方体は交差していないと断定でき，逆にもし内接球が交差していれば，これら 2 つの立方体は交差していると断定することができる．いずれの条件も満たさなければ，次の段階でさらに詳しく調べる．

頂点の内外判定

まず，小さい方の立方体の 8 頂点の座標を，それぞれ与えられている並進・回転の行列で変換する．もし 8 頂点すべてが target octree の node を表す立方体（世界座標系の座標軸に平行）の内部に来ることがわかれば，source octree の black node を表す立方体は target octree の立方体に完全に包含されるとして処理を終了する．

もし，8頂点のうちの少なくとも1つの座標がtarget octreeのnodeを表す立方体（世界座標系の座標軸に平行）の内部であることがわかれば，2つの立方体は交差しているとして処理を終了する．

いずれの条件も満たさなければ，次に大きい方の立方体の8頂点の座標を，それぞれ与えられている並進・回転の行列の逆行列で変換する．もしこの8頂点のうちの少なくとも1つの座標が変換前のsource octreeのnodeを表す立方体（世界座標系の座標軸に平行）の内部であることがわかれば，2つの立方体は交差しているとして処理を終了する．そうでなければ，次の段階へ進む．

稜線の交差判定

ここまでの処理を終えて，もし2つの立方体が交差しているとすれば，それは**Fig. 5.7**のように，双方のある稜線が他方の面を貫いている場合だけであることがわかる．この可能性を調べるため，まず小さい方の立方体の8頂点の座標を，それぞれ与えられている並進・回転の行列で変換する．そしてこの変換後の立方体と，target octreeのnodeを表す立方体（世界座標系の座標軸に平行）との交差の有無を次の2つの段階で調べる．第1の段階では，変換後の立方体の各稜線について，その両端点がtarget octreeのnodeを表す立方体の表面の各平面で区切られる近接領域の同一の領域に属しているかを調べる．**Fig. 5.8**に2次元の場合の例を示す．図で，稜線e_1とe_2は同一の領域に属し，稜線e_3，e_4，e_5は複数の領域に属するとみなす．第2の段階では，ここで見つかった複数の領域に属する稜線に対して，その稜線がまたぐ平面との交点を求め，これがtarget octreeのnodeを表す立方体の面の内部である稜線が1つでも見つかれば，2つの立方体は交差していると断定できる．**Fig. 5.8**の例の場合，稜線e_3と平面f_1との交点はtarget octreeのnodeを表す立方体の面F_1の外部であり，また稜線e_4と平面f_1，f_2との交点はいずれも面F_1，F_2の外部であるが，稜線e_5の場合は平面f_2との交点は面F_2の内部である．

以上のように，疎密的な方法で稜線と立方体の交差を判定する．

5.4.3 一般の運動にともなうoctreeの効率的な更新

5.4.2節で述べた立方体間の正確な交差判定を用いて，物体のoctree表現を，並進と回転とを含んだ一般の運動にともなって効率的に更新する手順を**Fig. 5.9**に示す．まず物体のoctree表現(source octree)が与えられると，5.3.2節で述べた方法によってこれをより少ない数の立方体の集合に圧縮変換する．次に立方体(source cube)を順次取りだし，それぞれに並行移動と回転移動の変換行列をかける．この要領は，

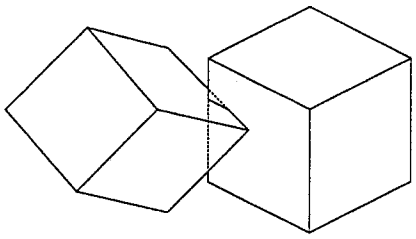


Fig. 5.7: An example case that each cube has edges intersecting a face of the other cube

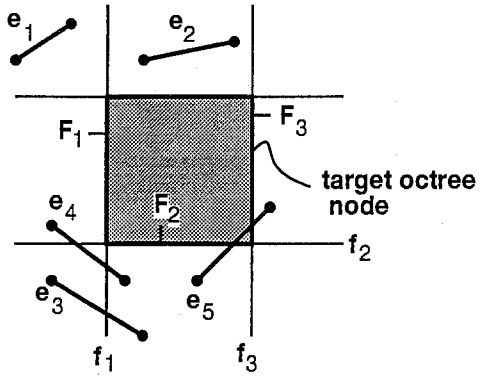


Fig. 5.8: An example of edge intersection test (quadtree)

5.2.2節で述べたのと同じ方法である.

次に，移動変換後の（傾いた）立方体の octree 表現 (target octree) を，改めて世界座標系に対して作成する．ここでは傾いた立方体と，世界座標系座標軸に平行な立方体との交差判定を，空間全体を表す立方体から始めて順に辺の長さが $1/2^n$ 倍の立方体へ再帰的に繰り返す．変換され傾いた立方体と世界座標系座標軸に平行な立方体それぞれとの交差判定は，5.4.2節で述べた正確な方法を用いる．その結果，target octree の各 node は，双方の交差の状態に応じて black, white, gray に分類される．ただしこの最下層の node (voxel) を black にするか white にするかは，アプリケーションに応じて決定する.

最後に圧縮された octree の各立方体 (source cube) から作成した部分 octree を合成して，物体全体の移動変換後の octree 表現を得る．つまり，target octree の各 gray node について，これらの子 node が全て同じ色かどうかを調べる．もし全て同じ色なら，全子 node を削除し，これらと同じ色をこの node に与える.

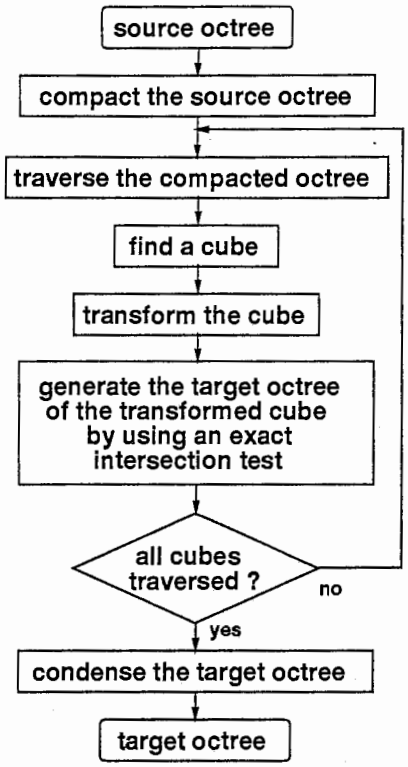


Fig. 5.9: Control flow of proposed octree motion algorithm

5.4.4 並列処理による高速化

物体のoctree表現を，並進と回転とを含んだ一般の運動にともなって効率的に更新する手順は，上で述べたように効率的なアルゴリズムであるが，この計算時間をさらに短縮するためには，アルゴリズムの並列化が考えられる．処理の大部分は独立した立方体間の交差の判定という，簡単な計算の繰り返しとして実現できるため，この過程をデータ並列アルゴリズムとして容易に並列処理による高速化を図ることができる．つまり，圧縮変換後の各立方体(source cube)ごとの処理を，それぞれ異なったプロセッサで同時に計算しようというわけである．ここでは，各プロセッサに任される計算負荷は確率的にほぼ同一であると考えられるので，最も簡単な実現方法として，各立方体を前もって各プロセッサに静的に割り振っておく方法が考えられる．今N個のプロセッサがあるとすると，この中のi番目のプロセッサは，i, $i+N$, $i+2N$, ... 番目の立方体を受けとり，これらの立方体の座標変換，target octree作成の計算を順次こなすことになる．そして全ての立方体の部分octreeが生成されたら，ある1つのプロセッサがこれらを合成して物体全体のtarget octreeを作成する．これにより，並列度に応じた高速化が図られる．

5.5 並進運動のみの場合の octree の更新

物体の運動が並進のみであることがわかっている場合には，さらに効率的な octree 更新の方法が考えられる．本節ではこの方法について述べる．

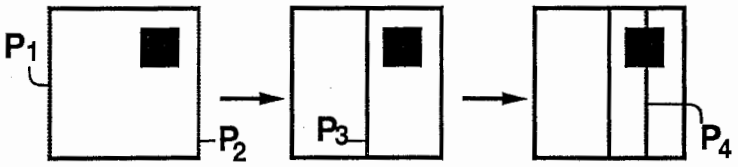
物体の運動が並進のみである時は，座標変換された立方体はいつも世界座標系の座標軸に平行であるため，5.4.2節で述べたような複雑な手順で立方体間の交差を調べる必要がない．立方体 (source cube) を座標変換した結果を，octree 表現 (target octree) として改めて世界座標系に対して表現し直す際には，単に両立方体の 6 平面の座標間の大小の比較を繰り返すだけでよい．ここで，target octree の各立方体の各平面は，複数の立方体で共有されているため，1 つの平面に対する大小比較を記憶しておけば，これを複数の立方体間の交差判定に利用することができるため，さらに効率的である．

そこで具体的には，各 source cube ごとに，x，y，z の各方向に対して，2 分木の 1 つである BSP-tree の考え方を用いる．つまり，各軸ごとの 1 次元の世界を考え，これをまず均等に 2 分割し，source cube の面のこの軸を横切る座標が含まれる区間をさらに 2 分割する．これを与えられた物体の全立方体 (source cube) に対して行なう．x，y，z の各方向の BSP-tree が完成したら，これらを合成し，target octree を作成する．**Fig. 5.10**に 2 次元の場合の例を示す．(a) のようにある 1 つの node が並進運動をした場合，この node の octree (quadtree) 表現を作成するため，単純な方法で立方体（正方形）間の交差を調べた場合，36 回の平面 - 立方体（線 - 正方形）間の座標値の大小比較を行なう必要がある．しかし，同図 (b)(c) のように各軸方向の BSP-tree を利用することで，重複した平面の大小比較を省略できるため，P_1 から P_8 までの 8 回の座標値の大小比較で十分である．

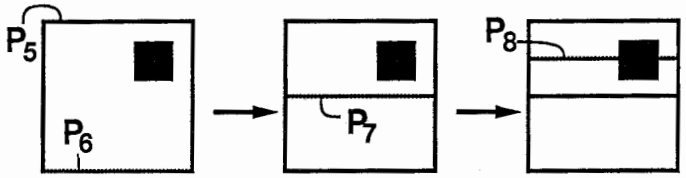
5.6 実 験

ここまでで述べてきた octree 更新手法の性能を評価するため，Silicon Graphics Onyx を用いて実験を行なった．この計算機は，4 つの RISC プロセッサ（おのおの 150 MHz MIPS R4400）と 128 Mbyte の主記憶をもつ．まず，並進と回転とを含む一般の運動にともなって物体の octree 表現を更新させてみた．ここでは target voxel に関するアプリケーションに依存したルールとして，逆変換された target voxel の中心点が変換前の source octree node の内部にあれば，この target voxel は source node と交差すると判断し，black の色を target voxel に与えることとした．これは Weng と Ahuja の方法 [WA87] でとられているのと同じルールであり，両者の結果

(a) An example of translation of a node (quadtree)



(b) Compare against X planes



(c) Compare against Y planes

Fig. 5.10: An example of octree translation (2-D wuadtree)

を直接比較することができる.

実験に用いた物体は, **Fig. 5.11**に示すようなスペースシャトルである. (a) は与えられた物体の octree であり, 863 の black node から構成される. なおここでは, 深さ 5 の octree を用いている. (b) は, (a) を 5.3.2節の方法で圧縮変換した結果であり, 458 の立方体の集合に変換された. この物体に, ある一定の回転と並進の運動を適当な回数 (39 回) 与え続け, 各回において, 物体の完全な target octree を生成するのに要する計算時間を計測した. 与えた運動パラメータは, 並進成分として (-0.1, 0, 0) (voxel/cycle), 回転成分として (1.5, -1.5, 1.5) (degree/cycle) である. **Fig. 5.11**(c)(d) はそれぞれ, 運動中のある時刻に生成された target octree である. **Fig. 5.12**に一般の運動に対する octree 更新時間の実験結果を示す. このグラフでは, 提案手法で 1 つのプロセッサのみを用いた場合の, 1 回の octree 更新に要する計算時間を, 従来手法として論文 [WA87] の手法による計算時間と比較している. また, この従来手法は立方体間の交差判定に近似的な方法を用いているが, これを 5.4.2節で述べた立方体間の正確な交差判定に置き換えた場合の結果も, あわせて表示している. これらの結果から, 立方体間の交差判定は近似的な方法よりも正確な判定の方が効率的であることがわかる. 計算時間で比較すると, 約 45% 処理

速度が向上していることがわかる．実際，ある1回のoctreeの更新において交差判定の回数をみてみると，従来手法 [WA87] による立方体間の近似的な交差判定では，54,007回の立方体間の交差を調べたが，これを立方体間の正確な交差判定に置き換えた場合は，43,223回で済んでいた．他の回 (cycle) もほぼ同様の回数であった．**Fig. 5.12**により，提案手法は効率的にoctreeを更新できることが確認できた．

次に，上と同じ物体と運動パラメータに対して，2，3，4個のプロセッサを用いて並列処理の実験を行なった．一般に計測データは，オペレーティングシステムによるプロセスのスケジューリングなどの影響で10-20 (ms) の誤差を含みがちであるが，連続して39回octreeの更新を繰り返した場合の，1回当たりの平均計算時間を**Table 5.1**に示す．複数のプロセッサを用いると，この間の通信などに時間がかかるようになるため，使用するプロセッサの数に比例して計算時間は早くなるというわけではないが，実験の範囲では使用するプロセッサ数の増加に対して処理時間は単調に減少し，4個のプロセッサを用いた場合に36 (ms) でtarget octreeを生成している．これは，従来手法の約4倍の計算速度である．

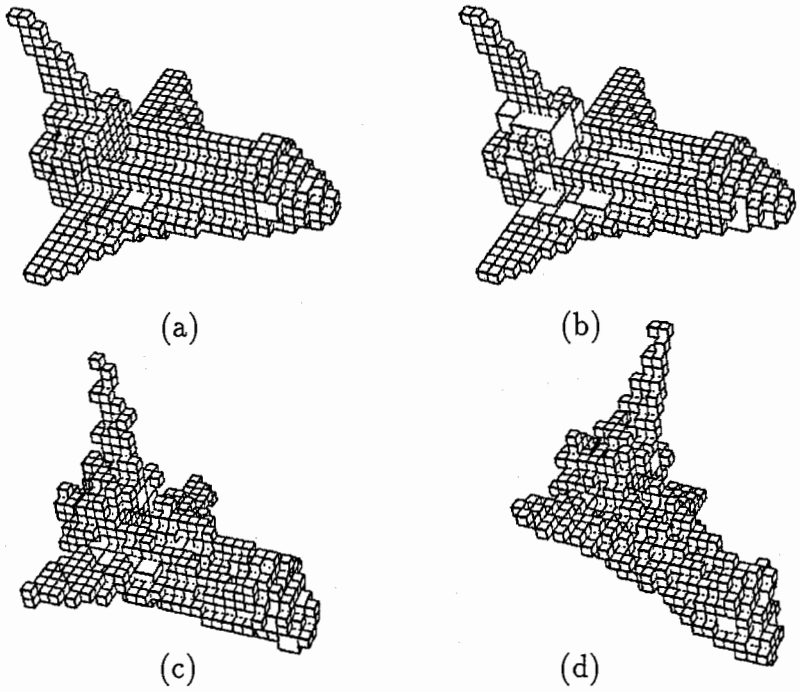


Fig. 5.11: The space shuttle experimental object. (a) is the original octree representation with 863 black nodes, (b) is the result of octree compaction with 458 black nodes, and (c) and (d) are snapshots of created target octrees during motion (certain translation and rotation).

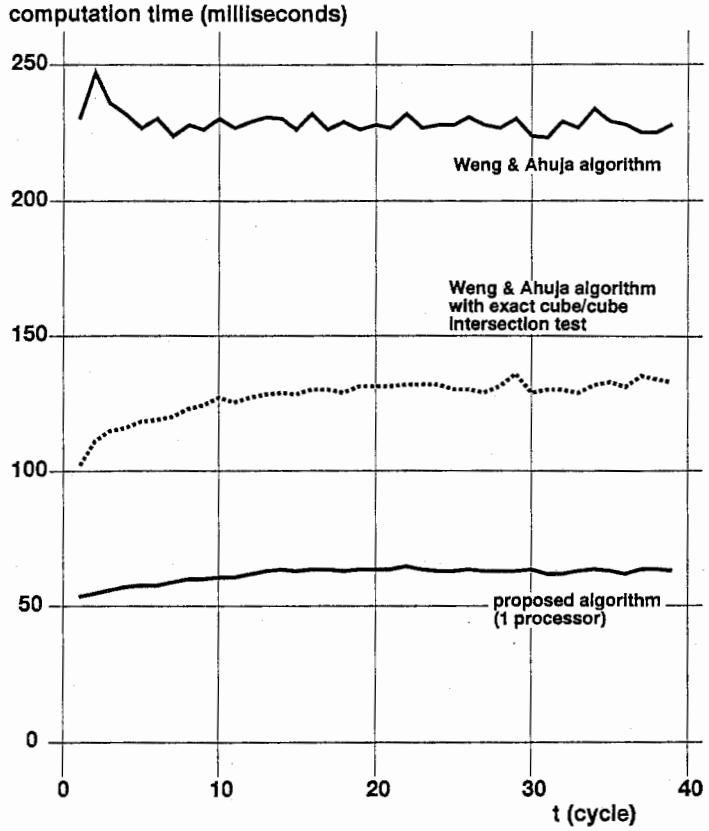


Fig. 5.12: Results from the tests done for the arbitrary motion algorithms.

Table 5.1: The average computation time for the arbitrary motion algorithms.

algorithm		computation time (milliseconds)
Weng & Ahuja's algorithm		229
Weng & Ahuja with exact cube/cube intersection test		127
proposed algorithm	1 processor	62
	2 processors	41
	3 processors	38
	4 processors	36

最後に，5.5節で述べた並進運動のみの場合の効率的なoctreeの更新に関する実験を行なった．ここではtarget voxelに関するアプリケーションに依存したルールとして，target voxelの少なくとも一部がsource octree nodeと交差すればこのtar-

get voxel は source node と交差すると判断し，black の色を target voxel に与えることとした．上の一般の運動に関する実験の運動パラメータのうち並進成分のみを39回，やはり上と同じ物体（スペースシャトル）に与え続け，各回において，物体の完全な target octree を生成するのに要する計算時間を計測した．本実験では，単純な方法で立方体（正方形）間の交差を調べる方法（従来手法）と提案手法の効率性を比較するのが目的であるが，同時に octree の圧縮変換の効果を見るため，並進運動のための提案手法に，octree の圧縮変換を利用したものと利用しないものの2種類の実験をした結果を **Fig. 5.13** に示す．結果より，圧縮変換しなくても提案手法は明らかに従来手法よりも効率的であるが，圧縮変換を利用するとさらに高速に octree の更新ができていることがわかる．そして平均して一回当たり 33 (ms) の計算時間で **Fig. 5.11**に示すスペースシャトルの更新を完了している．これは，従来手法の約 2.2 倍の計算速度である．以上より，物体の運動が並進のみであるとわかっていれば，回転を含む一般的な場合よりも高速に octree の更新ができることが確認できた．

5.7 5章のまとめ

以上，並進と回転を含む一般の運動に対して，物体の octree 表現を更新する効率的な方法を提案した．

まず物体の octree 表現 (source octree) が与えられると，全体の計算量を減らすため，5.3.2節で述べた方法によってこれをより少ない数の立方体の集合に圧縮変換した．次に立方体 (source cube) を順次取りだし，それぞれに並行と回転の変換行列をかける．そして，移動変換後の（傾いた）立方体の octree 表現 (target octree) を，この立方体と世界座標系座標軸に平行な立方体との交差判定を再帰的に繰り返すことにより作成するが，ここでは，両者の間の交差判定は，近似的な方法ではなく，5.4.2節で述べた正確な方法を用いた．その結果，交差判定の回数を減少させることができ，計算を効率化することができた．

実験により，立方体間の正確な交差判定は近似的な方法よりも結果として45%程度効率的なこと，提案手法は従来手法よりも約3.7倍高速に処理できることを確認した．また，本提案手法を4個のプロセッサを用いて並列化することにより，平均36(ms)（従来手法の約6.4倍）の計算時間で，863個の black node から構成される物体の octree 表現を更新できた．さらに，以上より，物体の運動が並進のみであるとわかっていれば，平均33(ms) と，回転を含む一般的な場合よりも高速に octree の更新ができることを確認した．

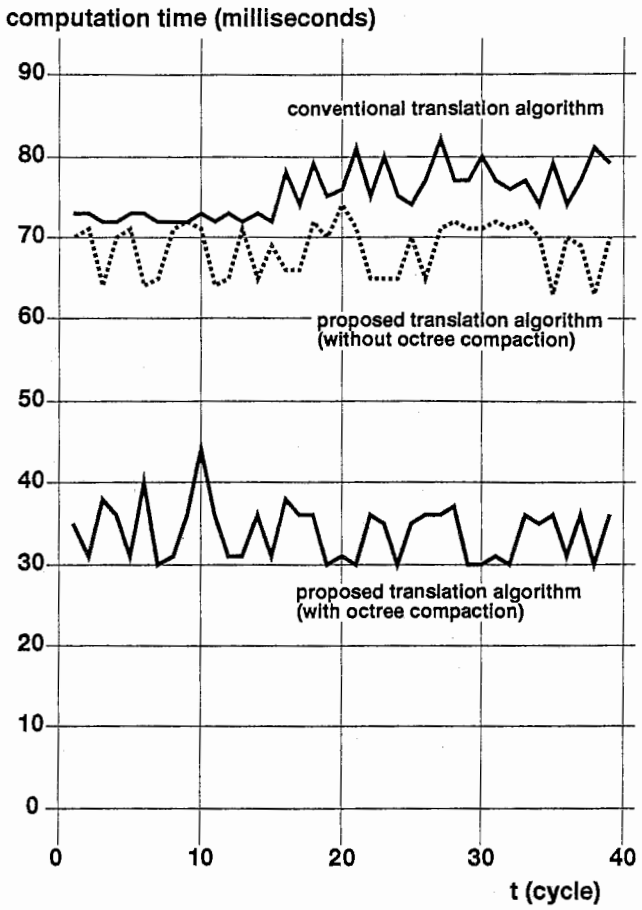


Fig. 5.13: Results from the tests done for the translation motion only algorithms.

一般の運動に対して，物体のoctree表現を更新する方法は，HongとOshmeierによっても提案されている[HO87]．この方法は，WengとAhujaの方法[WA87]と非常に類似しているが，今回はインプリメントしていない．ただ，論文[HO87]で掲載されているoctree更新のための計算時間に比べて，本章で提案した手法は約300倍以上も高速であるとの結果を得た．この差は，計算機ハードウエアなどの進歩を考慮したとしても，提案手法の方が高速であると確信させるに十分である．

付録：立方体間の近似的な交差判定

intersectionの検出がこのアルゴリズムで最も重要な部分である．source tree T_1 のあるleaf nodeをa_1，その移動後のcubeをa_1'とする．(**Fig. 5.14**参照) a_1'に相当するtarget tree T_2を生成するためには，傾いた(tilted) cube a_1'と，target treeの各nodeを表す直立した(upright) cube a_2'のintersectionを調べる必要がある．アルゴリズムのポイントは次の3点である．

- upright cube a_2'に含まれる各単位voxelの中心が，tilted cube a_1'の内部にあるのか外部にあるのかを判断するため，cube a_2'よりも単位長さ分だけ辺の長さが短い小cubeをとる．
- 2つのcubeのうち，傾いている方のcubeに外接する球を考え，これと，他方との重なりを調べる．
- 大きいcubeの外接球と小さいcubeの間ではintersectionの誤検出の可能性が高くなるので，この場合には両cubeを運動前に逆変換し，uprightとtiltedの関係を逆転させた2つのcube間で重なりを調べる．

いま，cube a_1のlevelをl_1，upright cube a_2'のlevelがl_2であるとする．また，intersectionを調べる2つのcubeのうち，小さい方のcubeをq，大きい方のcubeをQとする．qに外接する球をSとする．qとQのintersectionを調べるのに，球SとQのintersectionを調べる．

intersectionの検出は次のように行なう．ただし，

$$disa = \frac{\sqrt{3}\,2^{l_1} + 2^{l_2} - 1}{2} \tag{5.3}$$

$$disb = \frac{2^{l_1} + \sqrt{3}(2^{l_2} - 1)}{2} \tag{5.4}$$

また，a_1'の中心を$X_1' = (fcx_1', fcy_1', fcz_1')$，$a_2'$の中心を$X_2' = (cx_2', cy_2', cz_2')$，$a_2$の中心を$X_2 = (bcx_2, bcy_2, bcz_2)$，$a_1$の中心を$X_1 = (cx_1, cy_1, cz_1)$とする．

1. $l_1 < l_2$のとき (**Fig. 5.14**(a))

   (a) Inside (a_2'がa_1'の内側) ： あり得ない

   (b) Outside (QがSの外側)： if

$$|fcx_1' - cx_2'| > disa \;\; or \;\; |fcy_1' - cy_2'| > disa \;\; or \;\; |fcz_1' - cz_2'| > disa.$$

(c) Partial (部分的な重なり)： その他の場合
— 8 つの children に分割し，再帰的に intersection を調べる

2. $l_1 \geq l_2$ のとき (**Fig. 5.14**(b))

(a) Inside (a'_2 が a'_1 の内側) ： if

$$|x_2 - cx_1| \leq 2^{l_1-1} \ and \ |y_2 - cy_1| \leq 2^{l_1-1} \ and \ |z_2 - cz_1| \leq 2^{l_1-1}$$

が q の 8 つの角の座標 (x_2, y_2, z_2) に対して満足

(b) Outside (S が Q の外側)： if

$$|cx_1 - bcx_2| > disb \ or \ |cy_1 - bcy_2| > disb \ or \ |cz_1 - bcz_2| > disb.$$

3. Partial (部分的な重なり)： その他の場合
— 8 つの children に分割し，再帰的に intersection を調べる

平均的な計算量は，K を source tree の node 数，l を全作業空間の 1 辺の長さの対数 (すなわち木の深さ，階数) としたとき，$O(Kl)$ 以下となることが報告されている.

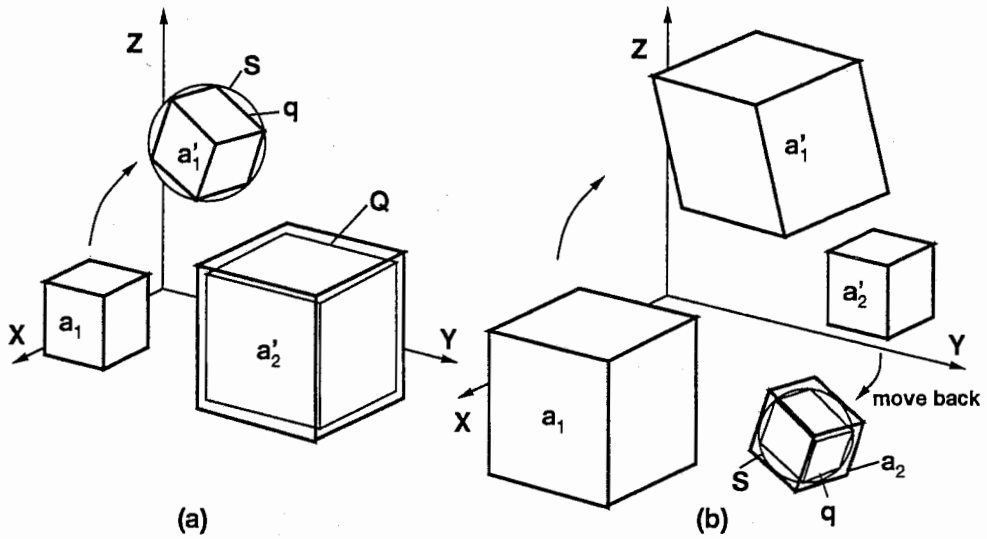


Fig. 5.14: Approximate intersection test between transformed tilted cube and upright cube.

第 6 章

面間の動的拘束を用いた仮想物体の操作補助法

仮想環境における物体配置操作に，自然なユーザインタフェースを提供する方法について述べる．実時間で検出された物体の衝突面を用いることにより，複雑な形状の物体の接触面を動的に決定し，この面で物体の運動を拘束する．ここで，操作者の手の動きは制限せずに，物体の仮想空間内での位置のみの自由度に制限を設け，この位置を視覚的に操作者に提示する（視覚的位置の制限）．本手法は力フィードバック機構など特別なハードウエアを用いる必要がなく，疑似的な磁石を模擬することにより，自然なユーザインタフェースを提供する．これにより，操作者は仮想物体の配置操作を精度良く，また効率的に遂行することができる．本稿ではまず，仮想物体操作を補助する方法について述べる．続いて配置精度と作業効率に関して，提案した操作補助法の有効性を被験者実験を通して確認する．最後に応用例として，本操作補助法を利用した仮想積木の組み立て作業を，実世界の積木組み立て作業と比較し，議論する．

6.1 背景

仮想現実の技術は，人間の 3 次元空間知覚能力と日常生活を営む実世界での体験を利用して，直観的で洗練されたユーザインタフェースとして利用されることが多くなってきた．しかし計算機能力の限界から，実世界と全く違わない仮想環境を生成することは困難であり，通常は一部の限定された物理法則などのみを模擬することが多い．したがって実世界では簡単な作業も，仮想環境ではしばしば熟練を要する特殊な技能が必要な作業となる．例えば，箱を手で直接つかんで机の上に置くという実世界では簡単な作業も，物体の位置などに何の拘束も設けない最も単純な構成の仮想環境では，難しくまた時間がかかる作業となるであろう．このような仮想環境での作業を，実世界の作業と同じように簡単かつ効率的に遂行するためには，物体間の干渉の検出と回避，仮想物体に働く重力，物体と机の間の摩擦など，物体間の相互作用を計算し，模擬することが必要である．

仮想環境における物体配置操作に，自然なユーザインタフェースを提供するための 1 つの有力な方法は，物体または操作者の手の動きの自由度を制限することである．これには大きく分けて 2 つの考え方がある．第 1 の考え方は，操作者の手の動きの自由度を，互いに接触した面間に反力を発生させるような力フィードバック機構などの装置を利用して制限する方法である [Iwa90, 広田 91, 平田 93, SP94]．この方法では，操作者は特別な装置を身に付ける必要があり，自然で自由な動きを妨げる原因となる．第 2 の考え方は，操作者の手の動きは制限せずに，物体の視覚的位置のみの自由度に制限を設けようという考え方である．この方法では特別なハードウエアは必要ないが，操作者は自分の手の位置と操作対象物体の間に不自然なズレが生じないよう，注意深くシステムを設計し，評価する必要がある．この視覚的な自由度の拘束方法について検討した例は多いが [吉村 91, BV93, FFD93a, FFD93b, 竹村 94, SP94, 木島 95]，物体間の正確な衝突や干渉の計算を避けるため，物体の接触を予め決めておいた面のみに限定したり，単純な形状の 2 物体のみを考え，作業を操作対象物体の並行移動のみに限定するなど，計算量を軽減する試みが見られる．そのためこれらは環境や作業に制約が多く，複雑な形状の物体を複数用いた組み立て作業などに柔軟に応用することができなかった．効率的な衝突検出の結果見つかった頂点の位置を拘束しようとする例 [Sny95] もあるが，仮想物体の配置において点のみによる拘束は不自然であり，またこれらのすべての例は，試作した物体配置システムについて，その操作性や実作業と比較した場合の（不）自然さなどを評価していない．

本章では，3章で述べた実時間衝突面検出の方法によって検出された衝突面を用い

ることにより，複雑な形状の物体の接触面を動的に決定し，この面で物体の視覚的な運動を拘束する仮想物体操作を補助する方法について述べる．本手法は力フィードバック機構など特別なハードウエアを用いる必要がなく，自然なユーザインタフェースを提供する．これにより，操作者は仮想物体の配置操作を精度良く，また効率的に遂行することができる．本章ではまず，仮想物体操作を補助する方法について述べる．続いて配置精度と作業効率に関して，提案した操作補助法の有効性を被験者実験を通して確認する．最後に応用例として，本操作補助法を利用した仮想積木の組み立て作業を，実世界の積木組み立て作業と比較し，議論する．

6.2 面間拘束を利用した物体配置操作

6.2.1 配置操作における物体運動の自由度

一般に，人間が手を使って物体の配置操作を行なう場合，次の 4 つの段階の動作をすると考えられる [竹村 94].

1. 把持：手を使って操作対象物体を把持する
2. 運搬：その物体を目的の場所（空間）に移動する
3. 位置決め：正確な位置と方向を決める
4. 解放：操作対象物体から手を離す

例えば基本的な内容を理解するため，立方体を机の上の決められた場所に置くという簡単な配置操作を考える．まず **Fig. 6.1**(a) に示すように物体を 1 つ机に置く場合，人が調整しなければならない物体の運動の自由度は，上の第 2 の段階では 6（空間内の並進 3 自由度と回転の 3 自由度）であるが，第 3 の段階では，操作対象物体の運動は机の上面で拘束されるため，その自由度は 3（机の上の並進 2 自由度と回転 1 自由度）である．続いて同図 (b) に示すように，2 つ目の立方体を 1 つ目の立方体の隣に接触させて並べて机上に置く場合，上の第 3の段階では，操作対象物体の運動は机の上面と第 1 の立方体との接触面の合計 2 面で拘束されるため，残された自由度は 1（両接触面に平行な方向の並進）である．また 3 つ目の立方体を，既に図 (c) に示すように置かれた 2 つの立方体に合わせて置く場合，操作対象物体の運動は机の上面と 2 つの立方体との接触面の合計 3 面で拘束されるため，残された自由度は 0 である．

ところが，物体の位置などに何の拘束もない単純な仮想環境で同様の操作をする場合，**Fig. 6.1**(a) に示す物体 1 つを机に置く操作の第 3 の段階の動作においても 6

自由度の調整が求められ，詳細な位置決めが困難なものとなる．同様に，**Fig. 6.1**(b)(c) に示す配置操作の場合も，第 3 の段階の動作においてもやはり 6 自由度の調整が求められ，いずれの場合も詳細な位置決めが困難なものとなる．そこで，仮想環境における物体の配置操作を容易にかつ自然に行なわせるためには，物体の運動の自由度を本来の配置操作が持つ自由度にまで制限することが必要である．**Table 6.1** にこれらの自由度の関係を示す．本論文では，**Fig. 6.1**(a) ～ (c) の物体配置操作を各々，1 面拘束 (one face constraint)，2 面拘束 (two faces constraint)，3 面拘束 (three faces constraint) と呼ぶこととし，これらの物体配置操作の補助について考える．

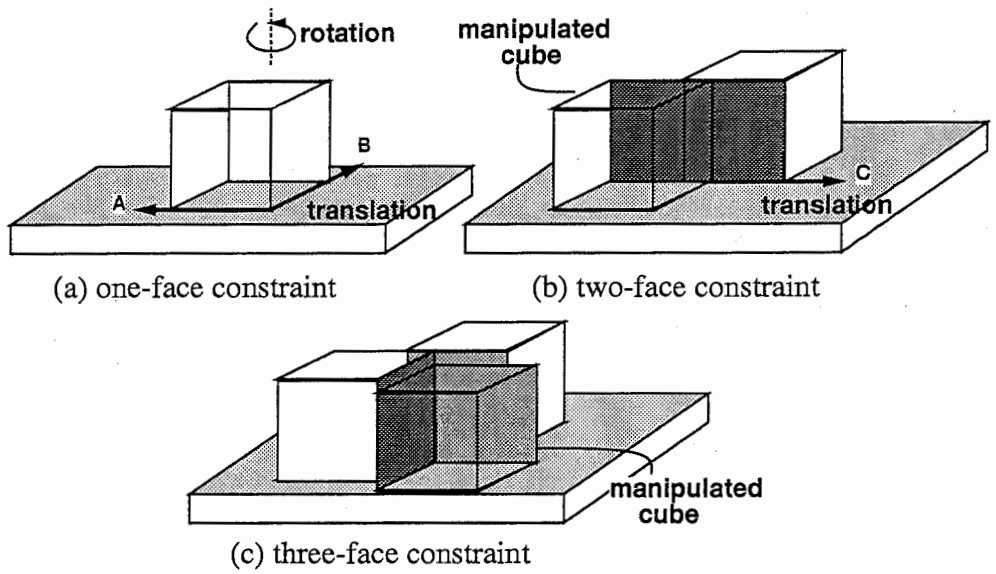


Fig. 6.1: Constraints among faces for object manipulation.

Table 6.1: DOF of object motion in object alignment tasks.

constraints among faces	real environment	simple virtual environment	insufficient DOF
(a) one face	3	6	3
(b) two faces	1	6	5
(c) three faces	0	6	6

6.2.2 実時間衝突面検出

面間の拘束を用いて仮想物体の操作補助を行ない，複雑な形状の物体を複数用いた組み立て作業などに柔軟に応用するためには，拘束すべき面のペアを予め決められたものだけに限定するのではなく，操作対象物体の動きに合わせて柔軟にこれらを決定する必要がある．そのためには，物体間の衝突や干渉を正確に検出することが必要であるが，複雑な形状の運動物体を多く含むような3次元環境では，その計算量の多さから実時間で処理することが困難であった [Pen90]．多面体表現された物体間の衝突や干渉を効率良く検出するための研究も多くなされてきたが，物体形状や環境，運動物体の軌跡等に制限を加えたものが多く，一般的な環境で物体間の正確な衝突を実時間で検出できるものは少ない．

本章では，3章および [SKTK95] で述べた，複数の複雑な多面体表現された物体間の衝突する面のペアを実時間で検出する方法を利用する．この方法は効率的な空間分割法を用いることにより，4,000 面程度の物体間の衝突面ペアを約 70(ms) で検出することができる．例えば **Fig. 6.2**は，1,816 面を持つ物体 (Venus) と 528 面の物体 (space shuttle) の衝突であるが，この場合衝突面のペア（黒く塗りつぶした部分）を約 40–60(ms) で検出できる（6.4節で用いた計算機による）．これにより，人間の知覚プロセッサの周期時間の平均値 100(ms)[CMN83] 以内に全ての処理を終了し，操作者に自然な運動印象を与える処理速度で，面間の動的拘束を利用した仮想物体の操作補助が設計できる．

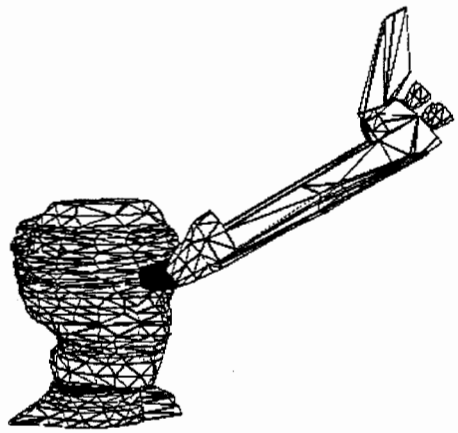


Fig. 6.2: A result of detecting colliding face pairs.

6.3 操作補助の方法

仮想環境中の全ての物体は多面体で表現されており，剛体であるとする．また形状は凸，凹いずれでも構わないとする．この仮定の下で仮想物体の配置操作を補助する方法について述べる．

6.3.1 概 要

Fig. 6.3に仮想環境において物体の配置操作を補助する処理の流れを示す．ここで提案する操作補助の方法は，操作者の手の位置を制限するのではなく，それによって操作される仮想物体の運動の視覚的な位置のみを制限する方法である．したがって，操作対象物体の実際の位置と方向は，操作対象物体の実際の位置と方向は操作者の手によって制御され，操作者の視覚に与えられるディスプレイ上の表示とは異なるものである．ここで，操作者が物体の配置操作を行なっている際に，不自然でなく物体の位置と方向を修正する操作補助システムをいかにして実現するかが重要な課題である．本論文では，両物体間の状態の遷移を表すメタファとして疑似的な磁石を考える．つまり，両物体が補助状態に入る場合には，あたかも拘束面に貼られている磁石が引き寄せられるような感覚を操作者に与え，逆に補助を解除する場合にも，あたかも拘束面に貼られている磁石を引き離すような感覚を操作者に与えることを目標とする．

まず物体の運動を拘束する方法を決定するためには，正確な衝突検出の結果が必要であり，ここでは実時間で複雑な物体間の衝突面を検出するアルゴリズム [SKTK95] を用いた．これにより，拘束面の候補を動的に発見することができる．続く操作補助アルゴリズムは，拘束面選択，運動拘束，拘束解除の 3 つの部分から構成される．これらについて詳しく説明する．

6.3.2 拘束面選択

複雑な形状をした物体間の場合，衝突面検出 [SKTK95] によって衝突面の組合せとしてただ 1 組のみが検出されることは稀であり，多くの場合は拘束面の候補として近傍の複数の衝突面の組が検出される．そしてこれら衝突面ペアの空間的関係や物体の運動速度などを考慮することにより，操作者の意図（どの面をどの面に接触させようとしているのか など）を推測し，拘束すべき面として最良の面を動的に選択することができる．本章では面間の拘束のみを考えるので，各衝突面ペアに対して次のような条件を適用することにより，その候補の数を減少させることができる．

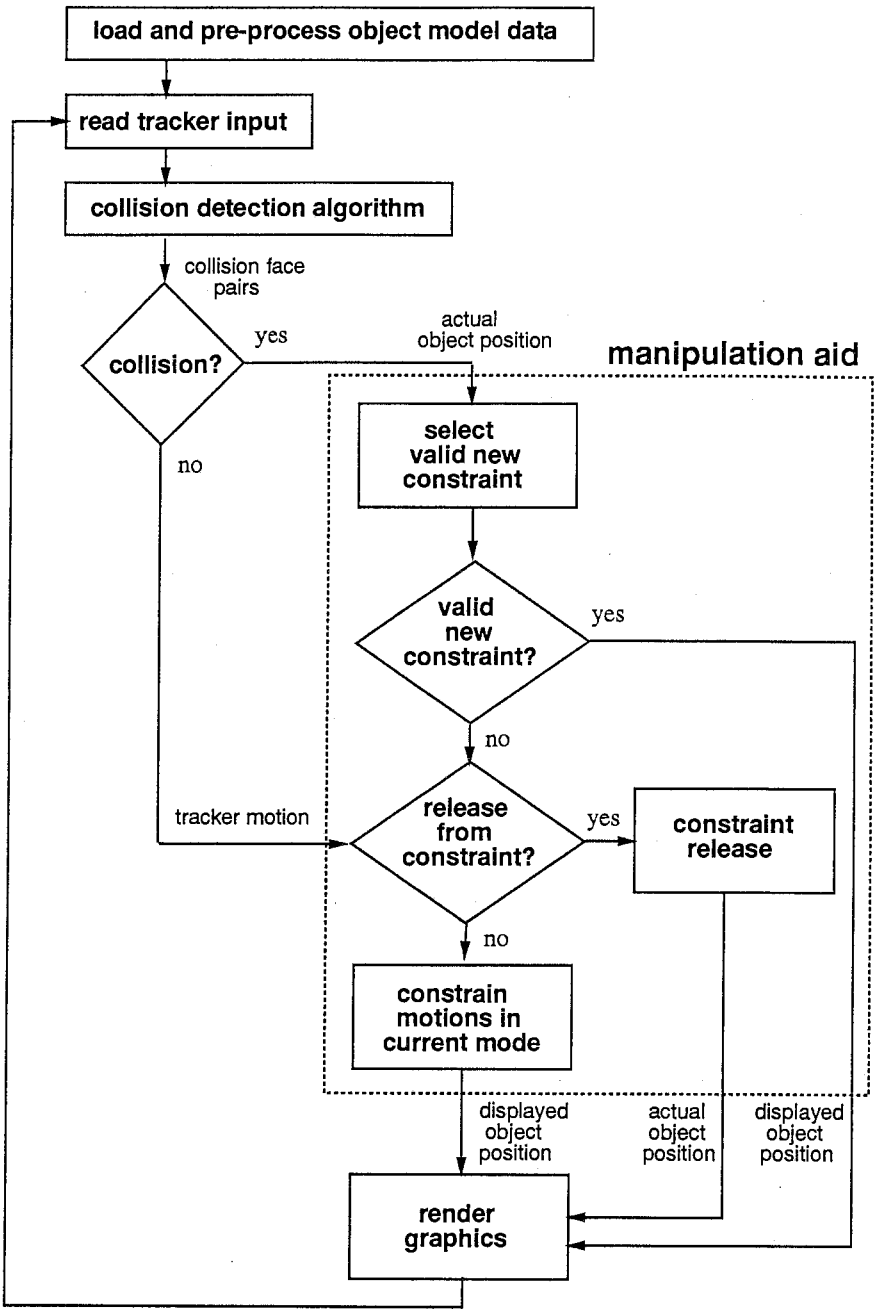


Fig. 6.3: Role of virtual object manipulation aid in the execution flow

- ペアのうち少なくとも片方の面が移動している
- ペア両面の法線ベクトルのなす角度が 120 度以上
- 両面の重なり面積比（6.3.4節で説明）があるしきい値よりも大きい

この条件を満足する全ての面ペアに対して，次式で与えられる誘引力 ($attraction$) を計算する．

$$attraction = rC_r + vC_v \tag{6.1}$$

ここで，r はペア両面の法線ベクトルのなす角度，v は操作対象物体の側の面（以下，操作面）の速度ベクトルと配置対象物体の側の面（以下，配置面）の法線ベクトルのなす角度，C_r, C_v は係数である．最大の誘引力を持つ面ペアが，拘束面として選択される．

6.3.3 運動拘束

妥当な面ペアが上で述べた方法により見つかると，物体の表示位置は運動が拘束された位置に修正される．まず操作面が配置面と並行になるように回転され，続いて両者が一致するように並行移動される．回転成分は次式の M_{rot} で与えられる．

$$M_{rot} = T(vtx)A(\theta, \vec{v})T(-vtx) \tag{6.2}$$

ここで，vtx は操作面の重心，$A(\theta, \vec{v})$ は操作面を配置面に平行にするマトリックス，$\vec{v}$ は，操作面と配置面の両方の法線ベクトルに垂直な単位ベクトル，θ は操作面と配置面の法線ベクトルのなす角度である．一方，平行移動量は，操作面の重心から配置面に下ろした垂線の長さに相当する．物体の位置が配置面上に拘束されてから後の運動は，操作者が 6.3.4に述べる方法で故意に拘束を解除しようとするまで，この面上で拘束される．

続いて，物体の運動を 1 面または複数の面で拘束する方法について述べる．1 面拘束の場合，操作対象物体は拘束面を 1 つ持つ．この時，運動は 3 自由度（配置面上の並進 2 自由度とその法線ベクトル回りの回転 1 自由度）に制限されるが，並進に関しては，操作対象物体の本来の（拘束を加えない）運動成分を配置面に投影することによって求める．具体的には，物体の絶対位置 $\vec{X}$ は，$\vec{A}$ と $\vec{B}$ を配置面上の互いに直行する単位ベクトル，s と t を操作対象物体の本来の（拘束を加えない）並進運動成分の配置面への投影として，

$$\vec{X} = s\vec{A} + t\vec{B} \tag{6.3}$$

で与えられる．一方回転に関しては，操作対象物体の本来の（拘束を加えない）回転運動の成分 ω_a, ω_e, ω_r と各軸方向の単位ベクトル (n_x, n_y, n_z) を用いて，

$$ang = n_z\omega_a + n_y\omega_e + n_x\omega_r \tag{6.4}$$

で与えられる角度 ang に拘束される.

2 面拘束の場合，操作対象物体は拘束面を 2 つ持ち，この物体の自由度は 1 （両拘束面に平行な方向の並進）である．運動の方向は両拘束面の法線ベクトルに垂直な方向であり，(6.3) 式を簡略化した次式で計算される.

$$\vec{X} = s\vec{C} \tag{6.5}$$

ただし，$\vec{C}$ は拘束された運動の方向の単位ベクトル，s は操作対象物体の本来の運動成分のこの直線への投影である.

3 面拘束の場合は操作対象物体は拘束面を 3 つ持ち，この物体は角に拘束されるため自由度を持たない.

6.3.4 拘束解除

物体の運動を特定の面によって拘束された状態を解除するため，“重なり領域比”と”拘束面からの距離” の 2 つのパラメータを用いる．これらのパラメータは，操作対象物体と配置対象物体の両物体間の拘束状態と非拘束状態の遷移を決定する重要なパラメータである.

重なり面積比

拘束面間の重なり面積比は，操作対象物体が他の物体に触れているかどうかを判断するパラメータである．拘束状態にある 2 面の重なり領域の面積の，これらのうち小さい方の面の面積に対する割合が，あるしきい値 $overlap\_threshold$ よりも小さくなった場合，この 2 つの面間の拘束を解除する．ここで重なり面積比のしきい値 $overlap\_threshold$ は，拘束状態と解除状態を安定に分離するため，ある幅のヒステリシスを持たせている．つまり，拘束を解除する場合はより小さめのしきい値 $overlap\_threshold-H$ を利用し，6.3.2で述べた拘束を開始する場合は大きめのしきい値 $overlap\_threshold+H$ を利用している.

拘束面からの距離

拘束面からの距離は，操作対象物体の本来の（拘束を加えない）重心位置と配置対象物体の拘束面の間の垂直距離である．この距離があるしきい値 $dist\_threshold$ よりも大きくなった場合に，この 2 つの物体間の拘束を解除する．ここで，$dist\_threshold$ は重なり面積 A と係数 k を用いた次式で表される動的なしきい値である.

$$dist\_threshold = k\sqrt{A} \tag{6.6}$$

つまり拘束面からの距離のしきい値は，拘束状態にある 2 つの物体間の接触面積が大きければしきい値はそれだけ大きくなり，両物体間の拘束を解除するために操作者はより遠くに操作対象物体を運ばなくてはならない．反対に，拘束状態にある 2 つの物体間の接触面積が小さければしきい値はそれだけ小さくなり，操作者はそれほど遠くに操作対象物体を運ばなくも両物体間の拘束を解除することができる．

6.4 実験手順

本節ではまず，本研究で用いた仮想物体配置操作実験環境の構成を述べる．続いて 6.3章で述べた操作補助の方法の効果を確認するための実験の方法と，応用例として本操作補助法を利用した仮想積木の組み立て作業を，実世界の積木組み立て作業と比較する実験の方法について述べる．

6.4.1 実験環境の構成

Fig. 6.4に，本研究で用いた仮想環境において物体の配置操作を行なうための実験装置の構成を示す．グラフィックスワークステーションで生成した仮想環境の表示には，70 インチ背面投影型 3 管式プロジェクタと 3 次元位置センサを付加した液晶シャッターめがねから構成される視点追従型立体表示装置を用いた．このため，操作者には奥行き感と運動視差をもった歪みのない仮想環境の画像を提示できる．利用者は 6 自由度の機械式位置・方向検出装置 (ADL-1™) の先端を握ってこれを操作することにより，仮想物体を把持，運搬，回転，解放などの直接操作をすることができる．これは 3 次元環境における人の空間知覚能力を活かして仮想物体の配置操作を実行することができる一般的な構成の 1 つであり [竹村 94]，操作者の手の並進・回転の移動量と操作対象物体の並進・回転の移動量が等しく 1 対 1 に対応するように設計されている．

6.4.2 配置精度と作業効率に関する実験の方法

配置精度と作業効率に関して，提案した操作補助法の有効性を確認する被験者実験の方法を述べる．まず，実験で用いた物体配置操作の作業内容と，それぞれにおいて操作者に提示する条件について説明した後，各実験の方法を述べる．

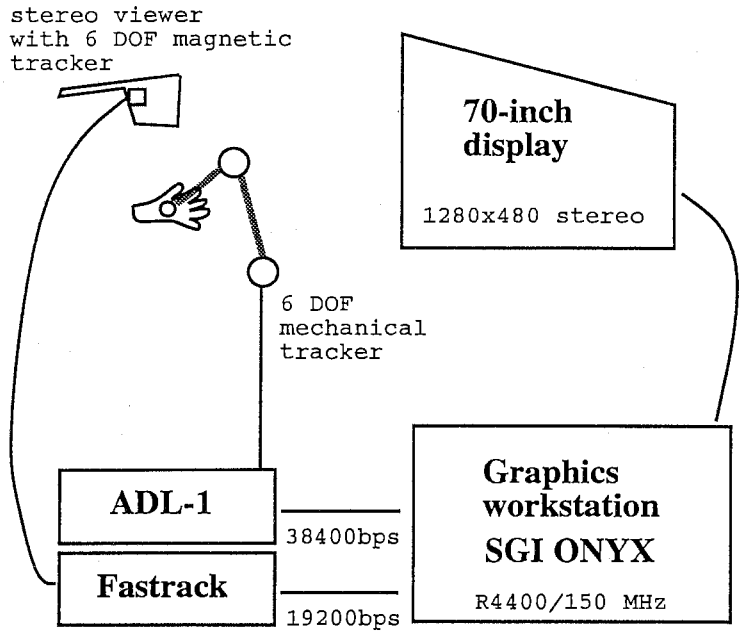


Fig. 6.4: Hardware configuration for manipulation aid.

作業内容

ここでは提案した操作補助法の有効性を確認するため，1面拘束，2面拘束，3面拘束を実現する最も単純な形状の物体として立方体を用意し，3種類の作業（Task A，Task B，TaskC）を設定する．**Fig. 6.5**に実験に用いた作業の内容を示す．

Task A: 1面拘束 — **Fig. 6.5**(a) のように水平方向に配置された7(cm) の大きさの2つの立方体のうち，手前側の立方体を掴んで他方の上面に4つの頂点の位置が正確に重なるように載せる．

Task B: 2面拘束 — **Fig. 6.5**(b) のように1辺を共有して互いに直角になるようにして置かれた7(cm) の大きさの2つの立方体の双方に接触するように，また6つの頂点の位置が正確に重なるように，手前に置かれた同じ大きさの立方体を配置する．

Task C: 3面拘束 — **Fig. 6.5**(c) のように互いに辺を共有して直角に置かれた3つの立方体の全てに接触するように，また7つの頂点の位置が正確に重なるように，手前に置かれた同じ大きさの立方体を配置する．

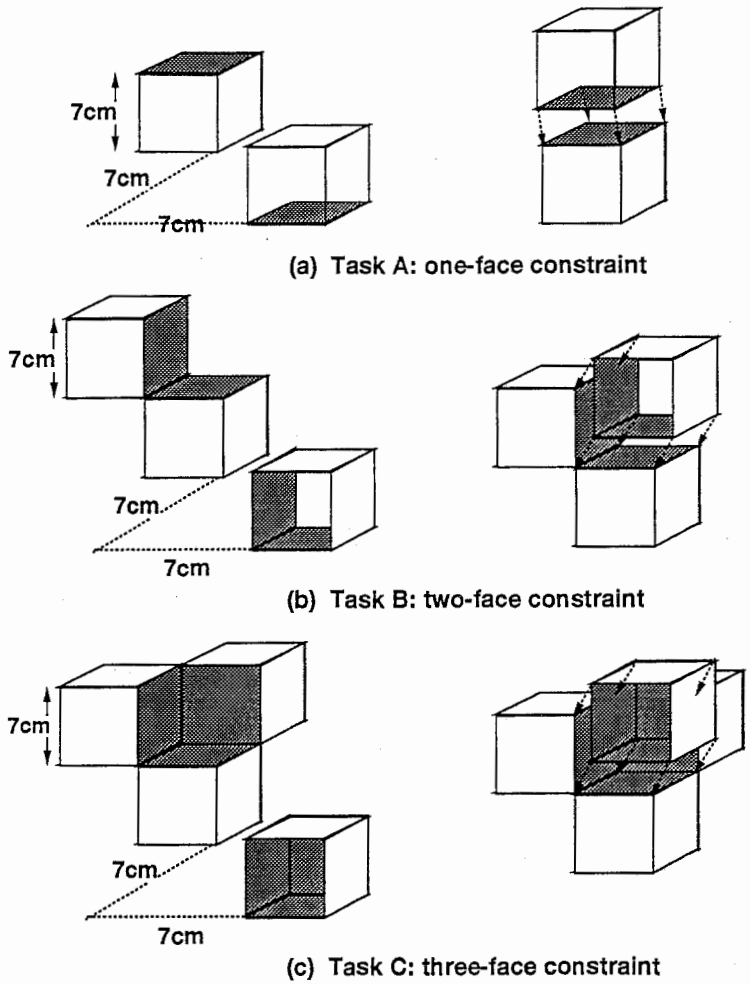


Fig. 6.5: Experimental tasks for manipulation aid.

操作者へのフィードバック

操作補助の効果を調べるため，各作業に対して次の 3 種類のモードを用意する.

Mode 1: 何の操作補助も，視覚的なフィードバックも操作者に与えない.

Mode 2: 操作者に対して操作補助は与えないが，衝突検出の段階で見つかった衝突面を色を変化させることによって提示する.

Mode 3: 操作者に対して 6.3章で述べた操作補助を与える. 拘束面は色を変化させることで提示する.

配置精度に関する実験

この実験の目的は，上で述べた3種類のモードについて，物体配置の精度を比較することである．被験者には各試行において，できるだけ正確にかつ素早く作業を完了することを求め，作業完遂時間，距離誤差，角度誤差 を計測する．作業完遂時間は，物体を把持してから解放するまでの時間であり，計算機内部のクロックを用いて測定する．したがって一度解放した物体を再度摑んで配置し直すことは許さない．各10回作業をする．距離誤差は， Task A, B, C で重ねて置くべき4，6または7頂点間のずれ（長さ）の和である．角度誤差は，目標位置（方向）に対する角度 (azimuth, elevation, roll) の誤差の和である．

作業効率に関する実験

この実験では，物体をある与えられた精度で配置するのに要する時間を比較する．各頂点の距離誤差の平均が決められた作業終了条件値よりも小さくなった場合に，作業が終了したとして物体の表示色を変化させてこれを操作者に提示する．したがって小さい作業終了条件値を与える方が作業は困難なものとなる．被験者は Task A, B, C のそれぞれに対して， 3種類の作業終了条件値 4(mm), 3(mm), 2(mm) で Mode 1 と 3 の 2 種類の作業を行なう．それぞれに対して 10 回の試行の各作業完遂時間を計測する．

6.4.3 実世界の作業と比較する実験

6.3で述べた操作補助の方法の応用例として，本操作補助法を利用した仮想積木の組み立て作業を，実世界の積木組み立て作業と比較する実験の方法について述べる．

作業内容

ここでは，予め用意した5種類の積木を用いて，かたつむりの形に積み上げる作業を題材とした． **Fig. 6.6**と **Fig. 6.7**に積木の初期配置と，完成されたかたつむりを示す．個々の仮想物体（積木）は，形状，大きさ，色を実物のそれに似せてモデリングした．完成されたかたつむりの幅，奥行き，高さはそれぞれおよそ 145(mm)（底辺）， 47(mm)， 72(mm) である．積木を用いたかたつむりの組み立て実験では，次の3種類のモードを用意した．

仮想作業・補助なし (virtual without assist) — 仮想積木を用いて，操作補助も衝突検出の段階で見つかった衝突面を色を変化させることもなしで，かたつむりを組

み上げる.

仮想作業・補助あり (virtual with assist) — 仮想積木を用いて，操作補助を利用してかたつむりを組み上げる．このとき拘束面は色を変化させることで，操作者に提示する.

実作業 (real) — 実物の積木を用いて，かたつむりを組み立てる.

積木の初期配置からかたつむりを組み立てるためには，仮想の積木，実物の積木，いずれの場合も，5 つの積木のうち 3 つはその方向を回転させて配置する必要がある．仮想積木は 6.4.1で述べた通り，6 自由度の機械式位置／方向検出装置 (ADL-1^{TM}) の先端を握ってこれの位置と方向を変化させることによって 1 つずつ操作される．そこで仮想作業と実作業を同じ条件で比較するため，実物の積木を把持・運搬に対して次のルールを設けた.

- 積木を親指と他のもう 1 本の指で把持する.
- 指を滑べらせながら物体を回転させず，手首か腕で回転させる.
- 把持した積木以外の物体には触れない.
- 積木を把持してから解放するまでの間は，被験者の肘や手を机や他の積木の上に置いて休むことはできない
- 既に置かれた積木を，他の積木を配置している間に動かすことはできない.

配置精度に関する実験

この実験の目的は，上で述べた 3 種類のモードについて，積木組み立ての精度を比較することである．被験者には各試行において，できるだけ正確にかつ素早く作業を完了することを求め，作業完遂時間と距離誤差を計測する．作業完遂時間は，最初の積木を把持してから最後の積木を解放するまでの時間である．3 種類のモードを各 8 回作業をする．距離誤差は，重ねて置くべき頂点間のずれ（長さ）の和である．実物の積木を用いてかたつむりを組み立てる場合，各積木の角は面取りが施されているため，距離誤差を正確に計測することはできない．そこで，本実験では各頂点ごとの距離誤差を最大 2(mm) として見積もることとした.

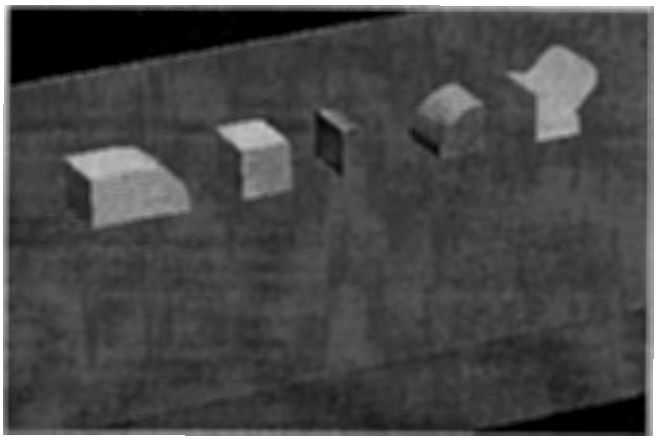


Fig. 6.6: Initial positions of blocks for toy snail.

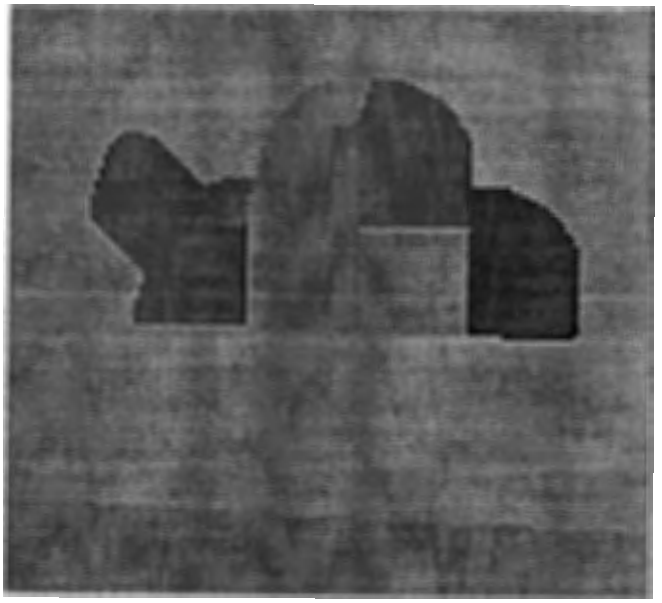


Fig. 6.7: Finished construction of toy snail.

作業効率に関する実験

この実験では，かたつむりを与えられた精度で組み立てるのに要する時間を比較する．各頂点の距離誤差の平均が決められた作業終了条件値 3(mm) よりも小さくなった場合に，その積木の配置が終了したとして物体の表示色を変化させてこれを操作者に提示する（仮想積木の場合）．本実験では実物の積木の場合，各頂点ごとの距離誤差を最大 2(mm) として見積もることとしているので，本来であれば，仮想積木の作業終了条件値を 2(mm) として実験を行ない，両者を比較するべきである．しか

し仮想作業・補助なしのモードでは作業が非常に困難なものとなり，ほとんどの被験者が作業を終了できないことが予備実験の段階で予想できた．そのためここでは，作業終了条件値を 3(mm) とする．それぞれに対して 8 回の試行の各作業完遂時間を計測する．

6.5 配置精度と作業効率に関する実験の結果と考察

配置精度と作業効率に関する被験者実験を行なった．5 名の被験者のうち，4 名は **Fig. 6.4**の環境で仮想物体の配置操作を以前に経験したことがある熟練者で，1 名はその経験がない非熟練者である．いずれの被験者に対しても，実験の開始前には各作業の各モードについて，その操作方法や画面に提示されるフィードバックなどについて習熟させるための練習時間を設けた．また各実験の間には十分な休息をとらせた．なお実験で用いたパラメータは，$C_r = 3.0$, $C_v = 1.0$, $overlap\_threshold = 0.05$, $H = \pm = 0.05$, $k = 0.5$ である．また，実験中は枚秒 15–30 フレーム程度の画像更新速度が得られていた．

6.5.1 配置精度に関する実験結果

配置精度に関する被験者実験を行なった結果を述べ，それに対して考察する．

典型的な被験者の振舞い

Task A における距離誤差と作業完遂時間の関係を示す典型的な例としてある被験者の結果を **Fig. 6.8**に示す．操作補助がある場合 (Mode 3)，距離誤差と作業完遂時間はいずれも平均して，操作補助がない場合 (Mode 1 と 2) に比べて小さく（短く）なっている．しかし，Mode 1 と 2 の間では，それほど大きな差は見られない．これは，衝突検出の段階で見つかった衝突面を色を変化させることによって提示するという Mode 2 での操作者への提示情報が，物体の位置決め操作において役に立たないばかりではなく，逆に妨げとなったとの複数の被験者から実験後聞かれた意見とも一致する．**Fig. 6.9**は Task B に対する同様の結果である．ここでもやはり操作補助がある場合 (Mode 3) の距離誤差と作業完遂時間はいずれも操作補助がない場合 (Mode 1 と 2) に比べて小さく（短く）なっている．しかし **Fig. 6.8**に比べると，これらのデータグループは明らかに分離されている．これは，Task A よりも Task B に対しての方が面の拘束による操作補助による効果が大きい ということを示している．

続いて，**Fig. 6.10**と**Fig. 6.11**は，それぞれ Task A と B の作業について，上と同じ被験者に対する角度誤差と作業完遂時間の関係を示す結果である．いずれの作業についても，操作補助がある場合 (Mode 3) のデータグループと操作補助がない場合 (Mode 1 と 2) のそれとの分離は，上の距離誤差の結果よりも顕著であると言える．これは，角度誤差は必ず距離誤差の原因となるが，逆は必ずしも原因にはならないからであると考えられる．Task B の操作補助がある場合 (Mode 3) には，残された自由度は並進の 1 自由度のみであるため，角度誤差は事実上 0 となる．なお角度誤差に関しても，Mode 1 と 2 の間では，それほど大きな差は見られない．

被験者の一般的傾向

他の被験者の結果も以上の結果と同様の傾向が見られた．Task A と B に対して，各被験者ごとの距離誤差の平均をとったものをそれぞれ**Fig. 6.12**と**Fig. 6.13**に示す．全被験者で，操作補助がある場合 (Mode 3) の距離誤差は操作補助がない場合 (Mode 1 と 2) よりも小さくなっている．そして衝突検出の段階で見つかった衝突面を色を変化させる場合 (Mode 2) の結果はそれがない場合 (Mode 1) よりも悪いか同程度に悪くなっている．5 番の被験者は非熟練者であるため，他の被験者に比べて全体的に大きな誤差となっているが他と同じ傾向を示している．同様に，Task A と B に対する各被験者ごとの角度誤差の平均は，**Fig. 6.14**と**Fig. 6.15**に示すように，上と同様の傾向が見られた．

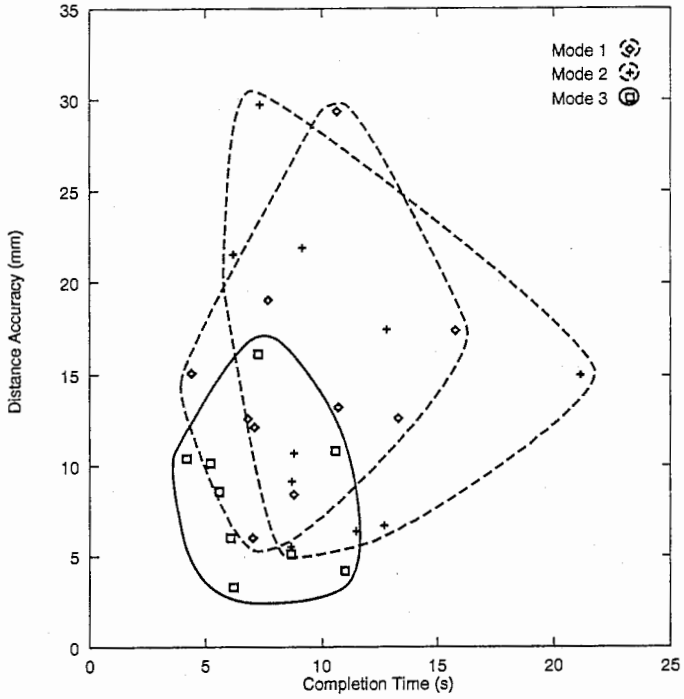


Fig. 6.8: Distance accuracy for Task A of one subject.

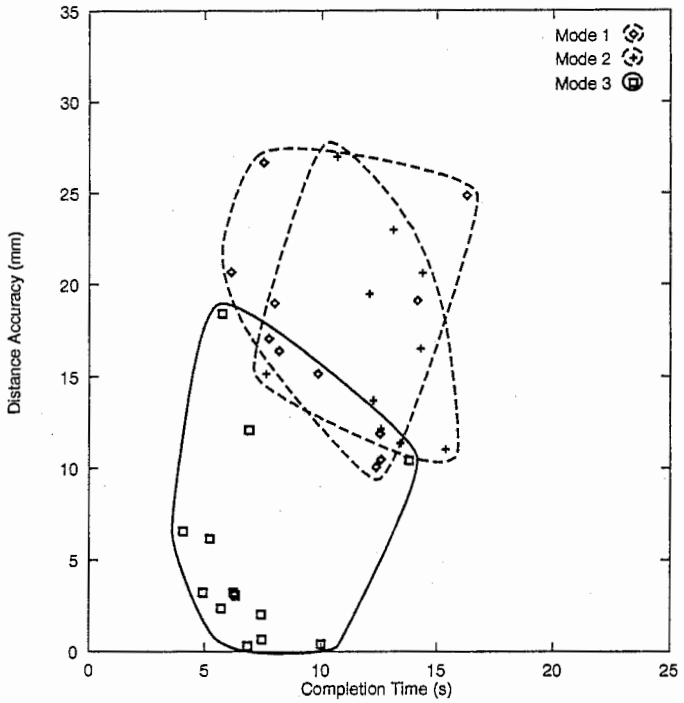


Fig. 6.9: Distance accuracy for Task B of one subject.

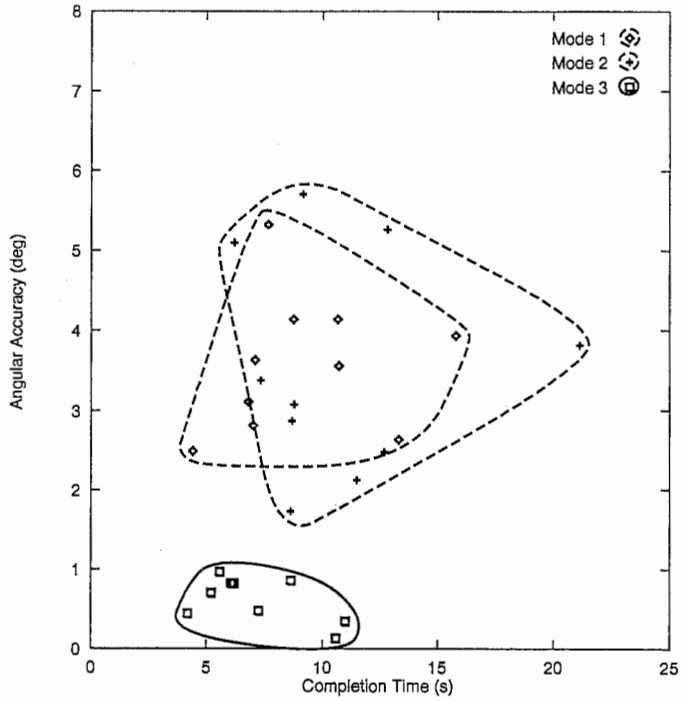


Fig. 6.10: Angular accuracy for Task A of one subject.

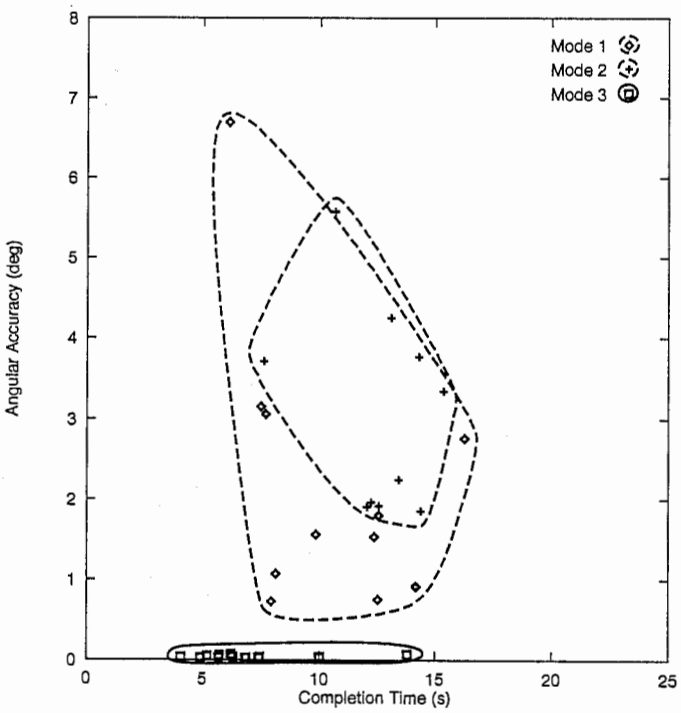


Fig. 6.11: Angular accuracy for Task B of one subject.

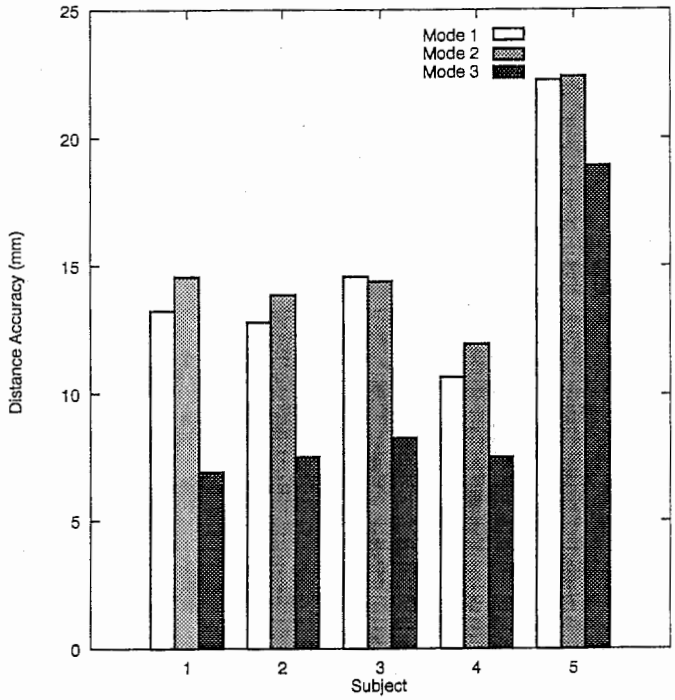


Fig. 6.12: Average distance accuracy for Task A.

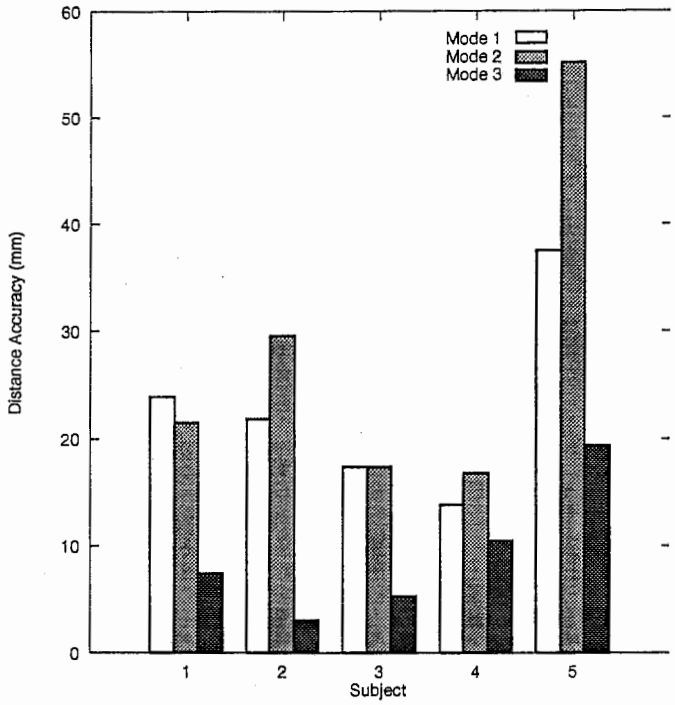


Fig. 6.13: Average distance accuracy for Task B.

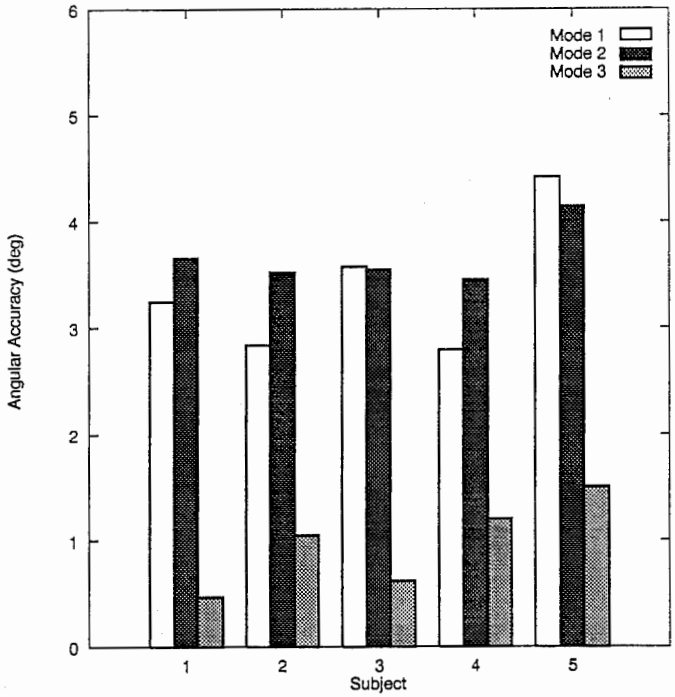


Fig. 6.14: Average angular accuracy for Task A.

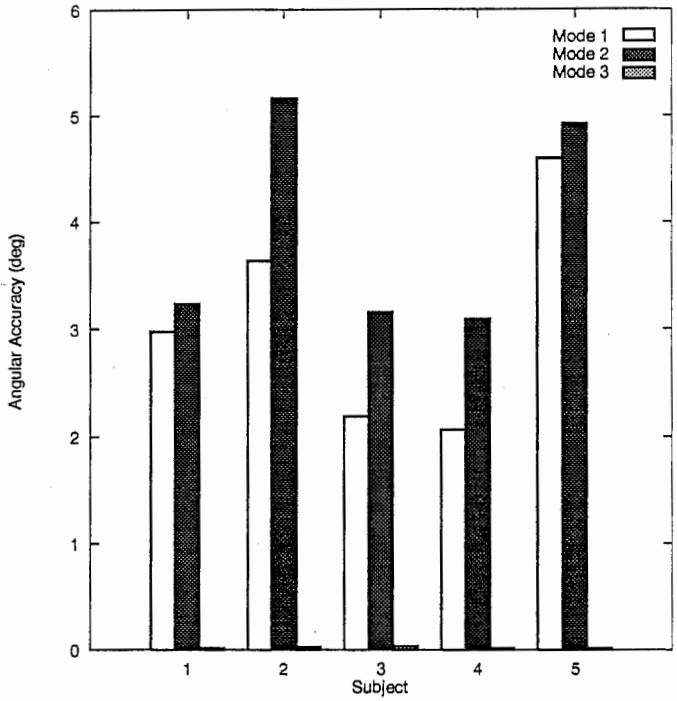


Fig. 6.15: Average angular accuracy for Task B.

操作補助法の効果

Task A と B の作業における操作補助の効果を **Table 6.2**に示す．ここで用いている値は，*ratio* を，操作補助がある場合 (Mode 3) のデータ（距離誤差，角度誤差，作業完了時間）の操作補助がない場合 (Mode 1) のデータに対する割合として，$gain = (1 - ratio) \times 100(\%)$ で求められる値である．それぞれ被験者ごとの平均値を全被験者で平均している．表は，Task A よりも Task B に対しての方がゲインが高い，すなわち，面の拘束による操作補助による効果が大きい ということを示している．これは，Task B はより多くの頂点を重ねる必要があるので操作補助がなければ困難な作業であるが，操作補助があれば操作者が調整すべき自由度は並進の 1 自由度のみであり，Task A で操作補助を受けている場合に操作者が調整しなければならない自由度（3）と比べて小さいためと考えられる．この作業では回転の自由度は許されていないため，Task B の角度に関するゲインは理論上 100(%) となる．以上の実験結果によれば Task B の場合，提案した操作補助によって物体配置操作の距離誤差は約 60(%) 向上し，同時に作業時間も約 40(%) に節約できていることがわかる．

Fig. 6.16は，**Table 6.2**の値をいくつかの理論値と合わせてプロットし，操作補助で拘束される自由度数と得られる効果（ゲイン）の関係について示したものである．つまり，Task A は 3 自由度，Task B は 5 自由度，Task C は 6 自由度を拘束しているので **Table 6.2**のようなゲインが得られるが，操作補助なしで自由度を 1 つも拘束しない場合，得られるゲインも 0 となるはずである．**Fig. 6.16**では，**Table 6.2**の値を標準偏差とともに載せている．少ないデータからではあるが，提案した操作補助によって拘束する物体の運動自由度の数と，その結果得られる効果（ゲイン）の関係についてある程度の傾向がつかめるものと思われる．

Table 6.2: Average gains for Tasks A and B (%).

	Task A	Task B
dist. accuracy	35	59
ang. accuracy	71	99
completion time	21	41

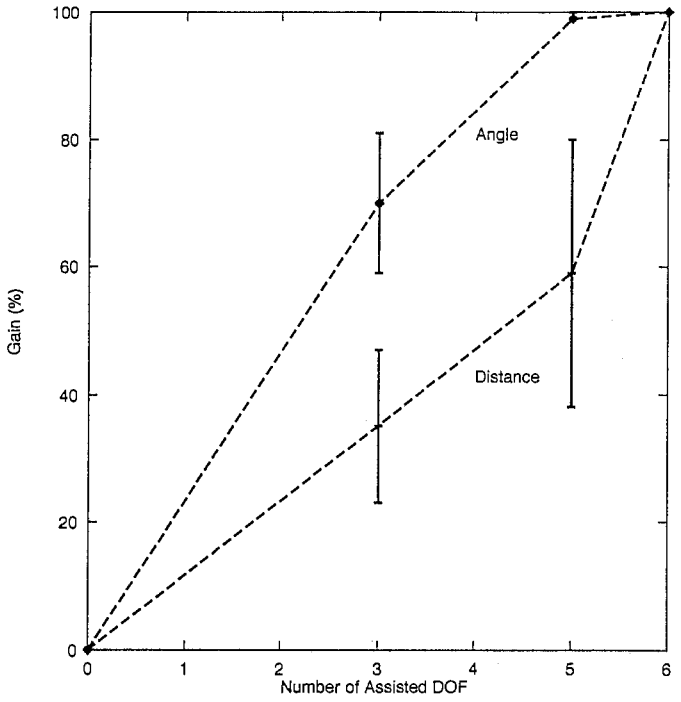


Fig. 6.16: Gain of using dynamic constraints as a manipulation aid.

6.5.2 作業効率に関する実験結果

本節では作業効率に関する被験者実験の結果を述べ，それに対して考察する．5人の被験者では習熟度に違いがあるため，作業完遂時間を正規化して比較する．まず作業とモードの組合せごとに各被験者の作業完遂時間を平均し，これを次式を用いて正規化した．

$$t_{navg} = \frac{(t - t_{minavg})}{(t_{maxavg} - t_{minavg})} \times 100(\%)$$

ただし t_{minavg}，t_{maxavg} は，平均作業完遂時間の最小，最大である．続いて正規化された作業完遂時間 t_{navg} の5人の被験者の間での平均をとり，これを **Fig. 6.17**に示す．ここでは，最も時間がかかる組合せ（Task C, Mode 1）を100% と考え，他の組合せの作業完遂時間はこれに対する割合としてグラフ化している．図より，操作補助がない場合 (Mode 1) は操作補助がある場合 (Mode 3) に比べて，いずれの作業終了条件値においても高い値（長い作業完遂時間）を示していることがわかる．作業終了条件値が小さくなるほど（困難な作業になるほど）両者の差は大きく広がっていることがわかる．つまり困難な作業ほど，操作補助の効果が現れているわけである．

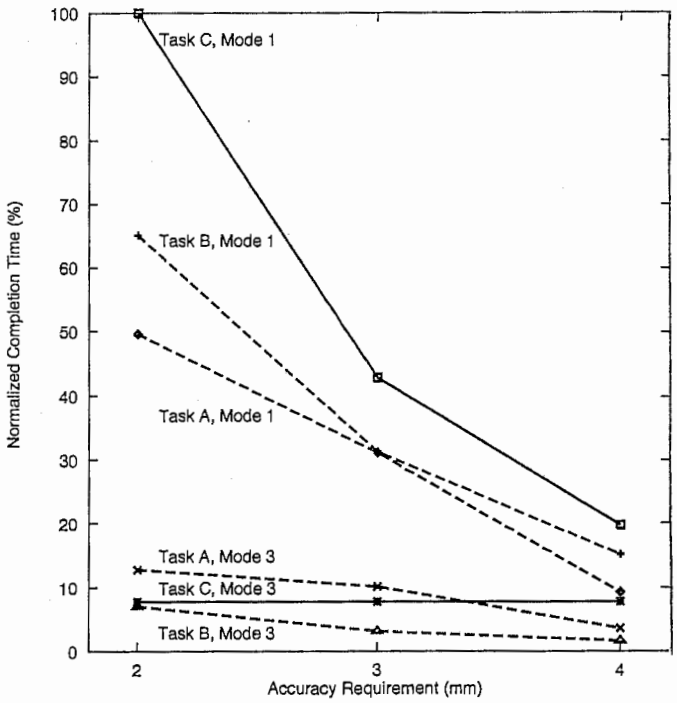


Fig. 6.17: Normalized completion times for three tasks.

6.6 実世界の作業との比較実験の結果と考察

提案した操作補助法を利用した仮想積木の組み立て作業を，実世界の積木組み立て作業と比較する実験の結果について述べる．前章の実験とは異なる 5 名の被験者が実験に参加した．全員，**Fig. 6.4**の環境で仮想物体の配置操作を以前に経験したことがある熟練者である．いずれの被験者に対しても，実験の開始前には各モードについて，その操作方法や画面に提示されるフィードバックなどについて習熟させるための練習時間を設けた．また各実験の間には十分な休息をとらせた．

6.6.1 配置精度に関する実験結果

典型的な実験結果として，ある被験者に対する作業完遂時間と距離誤差の関係を**Fig. 6.18**に示す．先に述べた通り実物の積木の実験では，各頂点ごとの距離誤差を最大 2(mm) として見積もることとしているので，実際の距離誤差は 0 と最大値（14 頂点の合計として 28(mm)）の間の値である．図ではこの最大値をプロットしている．この点を考慮したとしても，図では 3 種類のモードに対応して，明らかに分離した 3 つのデータグループが形成されている．仮想積木の操作補助がある場合のデータグループは，操作補助がない場合よりも実物の積木を用いた場合のデータグループに

近い．正確には，若干時間が余計にかかっているがほぼ同程度の誤差である．仮想積木の組み立てに，操作者の手の位置と方向を検出する機械式トラッカーを用いているなどのオーバーヘッドを考慮すれば，提案した仮想物体操作補助法により，実物の積木を組み立てるのとほぼ同程度の精度と時間で仮想の積木を組み立てることができたと言える．他の被験者からも同様の結果が得られたが，ここでは省略する．

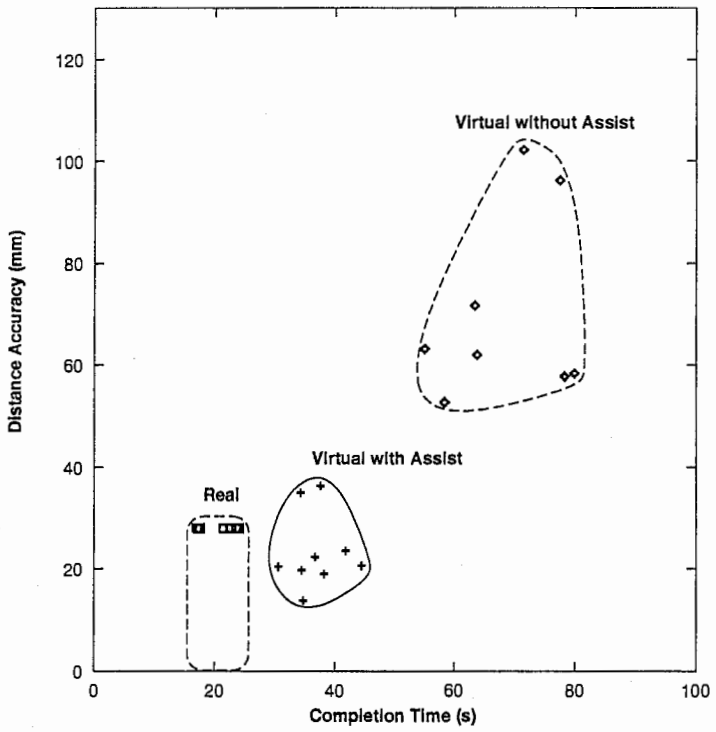


Fig. 6.18: Distance accuracy for virtual/real tasks of one subject.

6.6.2 作業効率に関する実験結果

ここでは，仮想の積木は各頂点の距離誤差が 3(mm) 以下となるようにして組み立てられた．計測された作業完了時間を被験者ごとに平均し，**Fig. 6.19**に示す．仮想積木の操作補助がある場合の平均作業完遂時間は，いずれの被験者でも実物の積木を用いた場合の作業完遂時間に近い．これらの関係を全ての被験者に渡って比較するため，被験者ごとに次式で計算される作業完遂時間の比 $percentage$ を用いた．

$$percentage = \frac{t_M - t_{realavg}}{t_{realavg}} \times 100\%$$

但し，t_M は仮想積木の操作補助あり または なしの平均作業完遂時間，$t_{realavg}$ は実物の積木を用いた場合の平均作業完遂時間である．**Fig. 6.20**にこの値を示す．仮

想積木の場合操作補助を用いれば，いずれの被験者も，実物の積木を用いた場合の 2 倍以内の時間で作業を完了していることがわかる．そして被験者 (subject 1) では，実物の積木を用いた場合とそれほど変わらない時間 (8 (%) だけ余計にかかっているが) で作業を終えている．一方操作補助を用いない場合は，実物の積木を用いた場合の 2.5 倍から最高 5 倍程度の時間がかかっている．この両者を明らかに分離する直線の 1 つとして，**Fig. 6.20**では 100(%) (実物の積木を用いた場合の 2 倍の平均作業完遂時間) に点線を引いている．

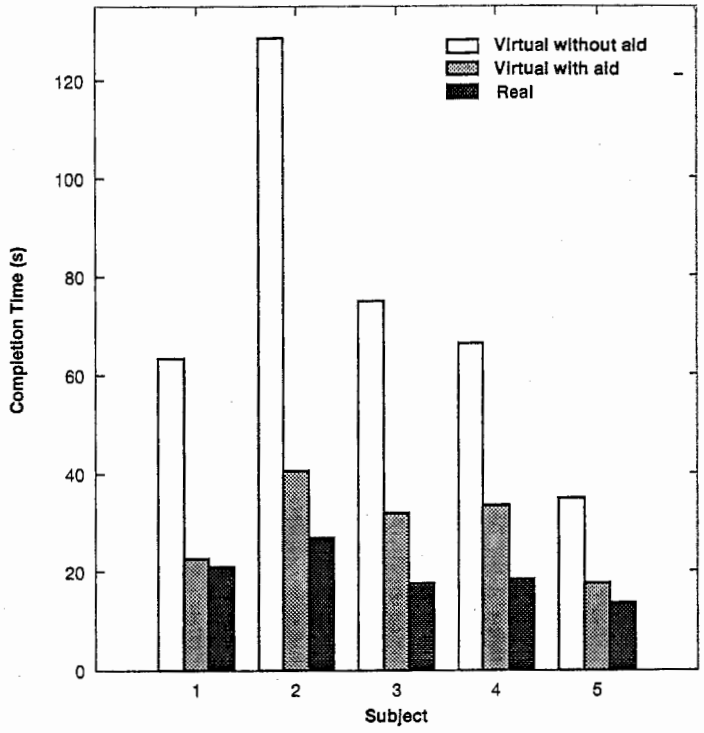


Fig. 6.19: Average task completion time for virtual/real tasks.

6.7 6 章のまとめ

実時間衝突面検出の方法によって検出された衝突面を用いることにより，複雑な形状の物体の接触面を動的に決定し，この面で物体の視覚的な運動を拘束する仮想物体操作を補助する方法について述べた．本手法は力フィードバック機構など特別なハードウエアを用いる必要がない．本章ではまず，提案した操作補助法の有効性を確認するため，配置精度と作業効率に関して，被験者実験を行なった．そして操作補助のやり方として，衝突面検出結果のみを提示する単純な方法では効果がないこと，提案した仮想物体操作補助法の効果は，複雑な作業ほど高いことなどが確認

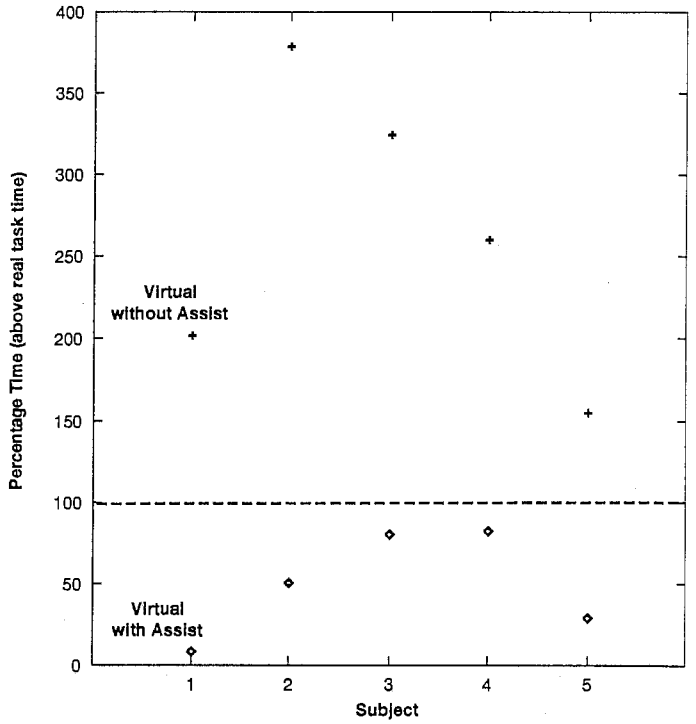


Fig. 6.20: Time percentages of task completion time above real task time.

できた．次に応用例として，本操作補助法を利用した仮想積木の組み立て作業を，実世界の積木組み立て作業と比較し，本操作補助法が実世界の作業とほぼ同程度の精度と時間で作業を完了できる，自然なユーザインタフェースを提供できていることを確認した．

今後は，逆に力覚や触覚の提示機構が使用できる場合の仮想物体配置作業との比較や，聴覚のフィードバックを付加または視覚に替えて用いた場合の効果などを検討してゆく予定である．

第 7 章

結 論

7.1 3 次元物体間の相互作用

本論文では，3 種類の 3 次元物体間の相互作用を実現するための方法について述べた．以下では本論文の内容を要約し，今後の展望などについてまとめる．

7.1.1 物体間の受動的相互作用

まず，2章と 3章では 3 次元物体間の受動的相互関係について述べた．2章では，並進と回転を含む一般の運動をする複数の 3 次元物体間の衝突面を，octree と多面体表現を用いて効率的に検出する方法を提案した．手法は，静止物体と運動物体を区別することなく，複数の複雑な一般形状の移動物体が存在する空間中で，連続な運動中に生じる衝突物体を離散時刻の検査によって見逃すことなく選びだし，さらに効率的に衝突“面”を特定することができた．ここで，物体の形状としては凸と凹両方の形状を許し，かついずれの形状も特別な前処理などを必要としないで同様に扱うことができた．実験によって，物体が並進と回転の運動をする一般的な環境で有効なことを示した．しかしこの手法では，付加的なデータ構造を用いているため，領域計算量を増大させるといった問題があった．また物体は剛体に限られ，時間とともに形状が変化するような物体には適用できなかった．さらに効率的に複雑な物体間の衝突面を検出できたが，市販の計算機を用いて実時間で検出するには至らなかった．

そこで 3章では，並列計算機を用いることによって実時間で，複雑な形状の非剛体の運動物体間の衝突面を検出する方法について述べた．まず，上の手法の最適化を図り，離散時間の衝突検査で複雑な軌道をもつ物体間の衝突面を見逃すことなく，実際に衝突が起きる直前に効率的に特定する基本アルゴリズムについて説明した．これを更に高速に動作させるため，プロセッサ間の負荷分散の考え方が異なる 2 種

類のデータ並列アルゴリズムを提案した．更に並列アルゴリズムを共有メモリをもつMIMD型の並列計算機を用いて実験を行い，性能を評価した．しかし本並列アルゴリズムではプロセッサ数を増加させても処理速度が向上しない上限があった．これは，共有メモリを介したプロセッサ間の通信がボトルネックになっているものと考えられる．プロセッサ間の通信手段としてメッセージパッシングを利用した場合や，分散メモリをもつ場合など，異なったアーキテクチャのための並列アルゴリズムを検討していくことが今後の課題である．また将来，本手法を実世界で稼働する移動ロボットなどの行動計画などに応用するためには，誤差などについて検討をする必要がある．これらは今後の研究課題である．

ここで述べた，複雑な3次元物体間の衝突を正確に実時間で検出する方法を用いれば，3次元空間内で運動物体が接触した場合の挙動について，様々なシミュレーションを行なうことが可能となる．これは，仮想空間内の要素の挙動を決定するための仕組みの1つとして，仮想現実システムの Interaction [Zel92] を増強するのに必要な技術である．

7.1.2 物体間の能動的相互作用

4章と5章では，3次元物体間の能動的相互関係について述べた．まず4章では，簡単なアルゴリズムで効率的に3次元動的環境での移動物体（ロボット）の経路を探索する方法を提案した．本手法では，ロボット，静止障害物，移動障害物の環境中の全ての物体を，その運動可能性を区別することなくoctreeで表現した．そして障害物を表すoctreeの各 black node を基準としてポテンシャル場を生成し，これを利用して3次元移動ロボットの障害物に衝突しない経路と向きを発見した．いくつかの実験結果を通して，簡単なアルゴリズムでも効率的に3次元の動的環境での経路が発見されることを示した．この手順は簡単な計算の繰り返しとして実現できるため，データ並列アルゴリズムとして容易に並列処理による高速化を図ることができる．これは今後の研究課題である．続いて5章では，4章の手順を効率良く実行するため，回転と並進を含む任意の運動に対して，octreeを実時間で更新する方法について述べた．

能動的相互作用により，3次元空間内の物体が，衝突など外部から与えられる受動的要因の発生に依らず，単にその初期位置・方向と目的位置・方向が与えられた時，この間をつなぎ，空間中に漂う他の物体（障害物）に衝突しない経路を自立的・能動的に効率良く発見することができた．これを用いれば，仮想現実のシステムの Autonomy [Zel92] を強化させることができる．

7.1.3 物体間の人為的相互作用

6章では，3次元物体間の人為的相互関係について述べた．この章では，3章で述べた実時間衝突面検出の方法によって検出された衝突面を用いることにより，複雑な形状の物体の接触面を動的に決定し，この面で物体の視覚的な運動を拘束する仮想物体操作を補助する方法について述べた．手法は力フィードバック機構など特別なハードウエアを用いる必要がなく，主に視覚に基づくフィードバックのみを利用した．まず，提案した操作補助法の有効性を確認するため，配置精度と作業効率に関して，被験者実験を行なった．そして操作補助のやり方として，衝突面検出結果のみを提示する単純な方法では効果がないこと，提案した仮想物体操作補助法の効果は，複雑な作業ほど高いことなどが確認できた．次に応用例として，本操作補助法を利用した仮想積木の組み立て作業を，実世界の積木組み立て作業と比較し，本操作補助法が実世界の作業とほぼ同程度の精度と時間で作業を完了できる，自然なユーザインタフェースを提供できていることを確認した．今後は，本実験の設定とは逆に力覚や触覚の提示機構が使用できる場合の仮想物体配置作業との比較や，聴覚のフィードバックを付加または視覚に替えて用いた場合の効果などに関する検討なども進める必要がある．また本論文では，仮想世界の組み立て作業（全ての物体が仮想空間に存在）と実世界の作業（全ての物体が実世界に存在）とを比較したが，今後は，仮想物体と実物体の相互作用を考え，実・仮想の融合環境での組み立て作業などが可能なシステムを構築していくことも有意義であろう．

6章では，物体間の受動的相互関係の結果を使った物体の挙動に関するシミュレーションの1つの形態として，仮想現実のシステムを試作し，評価を行なった．ここで行なった検討は，仮想現実のシステムの Presence と Interaction の軸 [Zel92] を改善するのに役立つ．

7.2 完全な仮想現実のシステムへ向けて

本論文では，完全な仮想現実のシステムを構築するための要素技術として，3種類の3次元物体間の相互作用について述べた．まず，3次元物体間の受動的相互作用と能動的相互作用を実現する方法について述べた．そして，主に受動的相互作用と人為的相互作用を用いた仮想現実のシステムを試作し，各種の評価を行なった．試作したシステムは，多くの物体間の相互作用を計算し，正確に模擬して操作者に提供することができたが，やはり全ての相互作用と物理法則を模擬する完全なシステムではない．

物体間の干渉の検出と回避，仮想物体に働く重力，物体と机の間の摩擦，反力の提示，接触音の生成など，物体間の相互作用を計算し，正確に模擬する，AIP キューブ [Zel92] で (1, 1, 1) となる完全な仮想現実のシステムを構築するためには，本論文で述べたような要素技術の確立とともに，システムの試作・評価を通して，完全な仮想現実に近いシステムの設計論を確立することが，今後必要である．

参考文献

[ACB80] Ahuja, N., Chien, R. T., and Bridwell, N. Interference detection and collision avoidance among three dimensional objects. In *International Conference on Artificial Intelligence*, pp. 44–48, 1980.

[AN84] Ahuja, N. and Nash, C. Octree representations of moving objects. *Computer Vision, Graphics, and Image Processing*, Vol. 26, No. 2, pp. 207–216, 1984.

[Bar90] Baraff, David. Curved surfaces and coherence for non-penetrating rigid body simulation. *Computer Graphics*, Vol. 24, No. 4, pp. 19–28, 1990.

[Bar92] Barr E. Bauer, editor. *Practical Parallel Programing.* Academic Press, Inc., 1992.

[BF95] Barfield, W. and Furnes, T., editors. *Virtual environments and advanced interface design.* Oxford University Press, 1995.

[Bol80] Bolt, Richard A. Put-that-there: voice and gesture at the graphics interface. *Computer Graphics*, Vol. 14, No. 3, pp. 262–270, 1980.

[Boy79] Boyse, John W. Interference detection among solids and surfaces. *Communications of the ACM*, Vol. 22, No. 1, pp. 3–9, 1979.

[BV91] Bouma, W. and Vanecek, G. Collision detection and analysis in a physical based simulation. In *Eurographics Workshop on Animation and Simulation*, pp. 191–203, September 1991.

[BV93] Bouma, W. J. and Vanecek, G. Jr. Modeling contacts in a physically based simulation. In *Symposium on Solid Modeling and Applications*, pp. 409–418. ACM, 1993.

[Cam90] Cameron, Stephen. Collision detection by four-dimensional intersection testing. *IEEE Transactions on Robotics and Automation*, Vol. 6, No. 3, pp. 291–302, 1990.

[Can86] Canny, John. Collision detection for moving polyhedra. *IEEE Transactions on PAMI*, Vol. 8, No. 2, pp. 200–209, 1986.

[CB90] Connolly, C.I. and Burns, J.B. Path planning using Laplace's equation. In *International Conference on Robotics and Automation*, pp. 2102–2106. IEEE, 1990.

[CGA95] Special Issue. Virtual Reality. Computer Graphics and Applications, Vol. 15, No. 5, IEEE, 1995.

[CH88] Chen, H. H. and Huang, T. S. A survey of construction and manipulation of octrees. *Computer Vision, Graphics, and Image Processing*, Vol. 43, No. 3, pp. 409–431, 1988.

[CLMP95] Cohen, J. D., Lin, M. C., Manocha, D., and Ponamgi, M. I-COLLID: An interactive and exact collision detection system for large-scale environments. In *Symposium on interactive 3D Graphics*, pp. 189–196. ACM, 1995.

[CMN83] Card, S. K., Moran, T. P., and Newell, A. *The Psychology of Human-Computer Interaction.* Hillsdale, NJ: Lawrence Erlbaum Associates, 1983.

[EVJ95] Earnshow, R. A., Vince, J. A., and Jones, H., editors. *Virtual Reality applications.* Academic Press, 1995.

[FFD93a] Fa, M., Fernando, T., and Dew, P. M. Direct 3D manipulation techniques for interactive constraint-based solid modeling. In *Computer Graphics Forum, EuroGraphics*, pp. 237–248, 1993.

[FFD93b] Fa, M., Fernando, T., and Dew, P. M. Interactive constraint-based solid modeling using allowable motion. In *Symposium on Solid Modelling and Applications*, pp. 243–252. ACM/SIGGRAPH, 1993.

[FHA90] Foisy, A., Hayward, V., and Aubry, S. The use of awareness in collision prediction. In *International Conference on Robotics and Automation*, pp. 338–343. IEEE, 1990.

[FS89] Fujimura, K. and Samet, H. A hierarchical strategy for path planning among moving obstacles. *Trans. on Robotics and Automation*, Vol. 5, No. 1, pp. 61–69, 1989.

[GJK88] Gilbert, G., Johnson, W., and Keerth, S. A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Journal of Robotics and Automation*, Vol. 4, No. 2, pp. 193–203, 1988.

[Gla90] Glassner, Andrew S., editor. *Graphics Gems.* Academic Press Professional, 1990.

[GSF94] Garcia-Alonso, A., Serrano, N., and Flaquer, J. Solving the collision detection problem. *Computer Graphics and Applications*, Vol. 14, No. 3, pp. 36–43, May 1994.

[HA92a] Hwang, Y. K. and Ahuja, N. Gross motion planning – a survey. *ACM Computing Surveys*, Vol. 24, No. 3, pp. 219–291, 1992.

[HA92b] Hwang, Y. K. and Ahuja, N. A potential field approach to path planning. *IEEE Transactions on Robotics and Automation*, Vol. 8, No. 1, pp. 23–32, 1992.

[Hah88] Hahn, James K. Realistic animation of rigid bodies. *Computer Graphics*, Vol. 22, No. 4, pp. 299–308, 1988.

[Hay86] Hayward, Vincent. Fast collision detection scheme by recursive decomposition of a manipulator workspace. In *International Conference on Robotics and Automation*, pp. 1044–1049. IEEE, 1986.

[Her86] Herman, Martin. Fast, three-dimensional, collision-free motion planning. In *International Conference on Robotics and Automation*, pp. 1056–1063. IEEE, 1986.

[HMLS90] Hirose, M., Myoi, T., Liu, A., and Stark, L. Object manipulation in virtual environment – case of fixed stero display. 第6回ヒューマンインタフェースシンポジウム論文集, pp. 571–576. 計測自動制御学会：ヒューマンインタフェース部会, 1990.

[HO87] Hong, T.H. and Oshmeier, M. Rotation and translation of objects represented by octree. In *International Conference on Robotics and Automation*, pp. 947–952. IEEE, 1987.

[Hub93] Hubbard, Philip M. Interactive collision detection. In *Symposium on Research Frontiers in Virtual Reality*, pp. 24–31. IEEE, 1993.

[Iwa90] Iwata, Hiroo. Artificial reality with force-feedback: development of desktop virtual space with compact master manipulator. *Computer Graphics*, Vol. 24, No. 4, pp. 165–170, 1990.

[KD86] Kambhampati, S. and Davis, L. S. Multiresolution path planning for mobile robots. *Journal of Robotics and Automation*, Vol. RA-2, No. 3, pp. 135–145, 1986.

[Kha86] Khatib, Oussama. Real-time obstacle avoidance for manipulators and mobile robots. *International Journal of Robotics Research*, Vol. 5, No. 1, pp. 90–98, 1986.

[Kor90] Korf, R. E.. Real-time heuristic search. *Artificial Intelligence*, Vol. 42, pp. 189–211, 1990.

[Lat91] Latombe, J.C. *Robot motion planning.* Kluwer Academic Publishers, Boston, 1991.

[LC91] Lin, Ming C. and Canny John F. A fast algorithm for incremental distance calculation. In *International Conference on Robotics and Automation*, pp. 1008–1014. IEEE, 1991.

[LMC94] Lin, M. C., Manocha, D., and Canny J. F. Fast contact determination in dynamic environments. In *International Conference on Robotics and Automation*, pp. 602–608. IEEE, 1994.

[MA85] Miyazaki, F. and Arimoto, S. Sensory feedback for robot manipulators. *Robotic Systems*, Vol. 2, No. 1, pp. 53–72, 1985.

[Man88] Mantyla, Martti. *An introduction to solid modeling.* Computer science express, 1988.

[MKT95] Miyasato, T., Kishino, F., and Terashima, N. Virtual space teleconferencing: Communication with realistic sensations. In *International Workshop on Robot and Human Communication*, pp. 205–210. IEEE, 1995.

[MT83] Mantyla, M. and Tamminen, M. Localized set operations for solid modeling. *Computer Graphics*, Vol. 17, No. 3, pp. 279–288, July 1983.

[MW88] Moore, M. and Wilhelms, J. Collision detection and response for computer animation. *Computer Graphics*, Vol. 22, No. 4, pp. 289–298, 1988.

[NNA89] Noborio, H., Naniwa, T., and Arimoto, S. A feasible motion-planning algorithm for a mobile robot on a quadtree representation. In *International Conference on Robotics and Automation*, pp. 327–332. IEEE, 1989.

[Pen90] Pentland, Alex P. Computational complexity versus simulated environments. *Computer Graphics*, Vol. 24, No. 2, pp. 185–192, 1990.

[Prep92] Preparata, F. P. and Shamos, M. I. *Computational geometry: an introduction.* corrected and expanded second edition. Springer-Verlag, 1988. 訳：浅野孝夫, 浅野哲夫. 計算幾何学入門. 総研出版, 1992.

[PT93] Pimentel, K. and Teixeira, K. *Virtual Reality: through the new looking glass.* McGraw-Hill, 1993.

[Qui94] Quinlan, Sean. Efficient distance computation between non-convex objects. In *International Conference on Robotics and Automation*, pp. 3324–3329. IEEE, 1994.

[RK91] Rich, E. and Knight, K. *Artificial Intelligence*, chapter 3. McGraw-Hill, 1991.

[Sam90] Samet, Hanan. *The design and analysis of spatial data structures.* Addison-Wesley, 1990.

[SF91] Shinya, M. and Forgue, M. Interference detection through rasterization. *The Journal of Visualization and Computer Animation*, Vol. 2, pp. 132–134, 1991.

[SH92] Shaffer, C. A. and Herb, G. M. A real-time robot arm collision avoidance system. *IEEE Transactions on Robotics and Automation*, Vol. 8, No. 2, pp. 149–160, 1992.

[SKTK95] Smith, A., Kitamura, Y., Takemura, H., and Kishino, F. A simple and efficient method for accurate collision detection among deformable polyhedral objects in arbitrary motion. In *Virtual Reality Annual International Symposium*, pp. 136–145. IEEE, 1995.

[Sny95] Snyder, John M. An interactive tool for placing curved surfaces without interpenetration. In *Computer Graphics, Annual Conference Series*, pp. 209–218. ACM, 1995.

[SP94] Sayers, C. P. and Paul, R. P. An operator interface for teleprogramming employing synthetic fixtures. *PRESENCE*, Vol. 3, No. 4, pp. 309–320, 1994.

[Ste94] Stevens, Larry. *Virtual Reality Now: A detailed look at today's virtual reality.* MIS:Press, 1994.

[TK92] Takemura, H. and Kishino, F. Cooperative work environment using virtual workspace. In *CSCW*, pp. 226–232, 1992.

[Tur89] Turk, Greg. Interactive collision detection for molecular graphics. M.sc. thesis, Department of Computer Science, University of North Carolina at Chapel Hill, 1989.

[Vin95] Vince, John. *Virtual Reality systems.* SIGGRAPH Series. Addison-Wesley, 1995.

[WA87] Weng, J. and Ahuja, N. Octrees of objects in arbitrary motion: Representation and efficiency. *Computer Vision, Graphics, and Image Processing*, Vol. 39, No. 2, pp. 167–185, 1987.

[Wex93] Wexelblat, Alan, editor. *Virtual Reality: applications and explorations.* Academic Press Professional, 1993.

[Zel92] Zelzer, David. Autonomy, interaction and presence. *PRESENCE*, Vol. 1, No. 1, pp. 127–132, 1992.

[ZPOM93] Zyda, M. J., Pratt, D. R., Osborne, W. D., and Monahan, J. G. NPSNET: Real-time collision detection and response. *The Journal of Visualization and Computer Animation*, Vol. 4, No. 1, pp. 13–24, 1993.

[岡野 88] 岡野 彰, 川辺 真嗣, 嶋田 憲司. シミュレーションによる移動物体間の衝突チェック. 日本ロボット学会誌, Vol. 6, No. 1, pp. 35–41, 1988.

[奥富 83] 奥富 正敏, 森 政弘. ポテンシャル場を用いたロボットの動作決定. 日本ロボット学会誌, Vol. 1, No. 3, pp. 226–232, 1983.

[岸野 89] 岸野 文郎, 山下 紘一. 臨場感通信のテレコンファレンスへの適用. 信学技報, IE89-35, 1989.

[岸野 92] 岸野 文郎. ヒューマンコミュニケーション — 臨場感通信 —. テレビジョン学会誌, Vol. 46, No. 6, pp. 689–702, 1992.

[木島 95] 木島 竜吾, 広瀬 通孝. Virtual physics: 仮想空間の物体挙動計算. In *Human Interface N&R*, pp. 313–322. 計測自動制御学会：ヒューマンインタフェース部会, 1995.

[北村 94] 北村 喜文, マイケル ベイル, 竹村 治雄, 岸野 文郎. 物体の多面体表現からの octree の生成. 電子情報通信学会春季大会, D-604, 1994.

[佐藤 93] 佐藤 圭祐. 極小点のないポテンシャル場を用いたロボットの動作計画. 日本ロボット学会誌, Vol. 11, No. 5, pp. 702–709, 1993.

[竹村 94] 竹村 治雄, 北村 喜文, アラン シャネゾン, 岸野 文郎. 仮想現実環境における物体配置タスクの一操作補助手法. テレビジョン学会誌, Vol. 48, No. 10, pp. 1312–1317, 1994.

[舘 92] 舘 暲, 廣瀬 通孝. バーチャル・テック・ラボ. 工業調査会, 1992.

[登尾 87] 登尾 啓史, 福田 尚三, 有本 卓. オクトツリーを用いた高速干渉チェック法. 日本ロボット学会誌, Vol. 5, No. 3, pp. 189–198, 1987.

[野村 90] 野村 淳二. ショールームにおける人工現実感の応用. 精密工学会第 169 回講演会資料, 1990.

[服部 91] 服部 桂. バーチャル・リアリティ. 工業調査会, 1991.

[平田 93] 平田 幸広, 水口 武尚, 佐藤 誠, 河原田 弘. 組み立て操作のための仮想作業空間. 電子情報通信学会論文誌, Vol. J76-DII, No. 8, pp. 1788–1795, 1993.

[廣瀬 93] 廣瀬 通孝. バーチャル・リアリティ. 産業図書, 1993.

[廣瀬 95] 廣瀬 通孝. バーチャルリアリティ. ヒューマンコミュニケーション工学シリーズ. オーム社, 1995.

[広田 91] 広田 光一, 橋本 剛, 広瀬 通孝. 力覚ディスプレイの CAD/CAE への応用. 第 7 回ヒューマンインタフェースシンポジウム論文集, pp. 95–98. 計測自動制御学会：ヒューマンインタフェース部会, 1991.

[藤村 93] 藤村 希久雄. 行動ストラテジーとアルゴリズム. 日本ロボット学会誌, Vol. 11, No. 8, pp. 1124–1129, 1993.

[吉村 91] 吉村 哲也, 中村 康浩. 作業空間中に不透過性と重力を持つ配置支援システム. 第 7 回ヒューマンインタフェースシンポジウム論文集, pp. 99–104. 計測自動制御学会：ヒューマンインタフェース部会, 1991.

[劉 89] 劉 雲輝, 登尾 啓史, 有本 卓. 移動物体間の干渉が効率的にチェックできるソリッドモデル HSM の提案. 日本ロボット学会誌, Vol. 7, No. 5, pp. 426–434, 1989.

[特集情 92] 小特集. 並列アルゴリズムの現状と動向. 情報処理, Vol. 33, No. 9, 1992.

[特集テ 95] 論文小特集. バーチャルリアリティシステム. テレビジョン学会誌, Vol. 49, No. 10, 1995.

[特集電 95] 特集. バーチャルリアリティ. 電気学会論文誌 C. 電子 情報 システム部門誌, Vol. 115-C, No. 2, 1995.

[特集ロ 92] 特集. 人工現実感. 日本ロボット学会誌, Vol. 10, No. 7, 1992.