octree と多面体表現を用いた三次元物体間の衝突面検出

北村 喜文^{*1} 竹村 治雄^{*2} アフジャ ナレンドラ^{*3} 岸野 文郎^{*1}Colliding Face Detection among 3-D Objects
using Octree and Polyhedral Shape RepresentationsYoshifumi Kitamura^{*1}, Haruo Takemura^{*2}, Narendra Ahuja^{*3} and Fumio Kishino^{*1}

We propose an accurate and efficient method for detecting potential collisions among multiple objects with arbitrary motion (translation and rotation) in three-dimensional (3-D) space. The algorithm can be used directly for both convex and concave objects. The method consists of two main stages. In the first, coarse stage, an approximate test is performed to identify interfering objects in the entire workspace using octree representation of object shapes. In the second, fine stage, polyhedral representation of object shapes is used to more accurately identify any object parts that might cause interference and collisions. For this purpose, specific pairs of faces belonging to any of the interfering objects found in the first stage are tested, thus performing detailed computation on a reduced amount of data. Experimental results, which demonstrate the efficiency of the proposed collision detection method with adequate octree are given.

Key Words: 3-D Object, Octree, Polyhedral Shape Representation, Interference and Collision Detection

1. はじめに

三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が [1], 「精度」と「計算の効率」のトレードオフの問題があった。三次元移動物体の動きの正確なシミュレーション（例えば簡単に、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三次元環境で、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 [2][3]。例えば、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み合わせに対して、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 [4]。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 [5]~[9]。

さらに、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 [10] 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 [4][11][12]、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 は三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 [13]。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

原稿受付 1995 年 8 月 6 日

^{*1}ATR 通信システム研究所

^{*2}奈良先端科学技術大学院大学

^{*3}イリノイ大学

^{*1}ATR Communication Systems Research Laboratories

^{*2}Nara Institute of Science and Technology

^{*3}University of Illinois at Urbana-Champaign

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 [14]。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。例えば文献 [15] 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。

本論文では、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三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 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. 衝突検出のための形状表現

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み合わせを調べることににより検出できる。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三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、を調べることににより検出できる [4]。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、 $O(n^2 \cdot EF)$ となることから分かる。ただし、 F , E はおのおの平均的な物体の面、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分かる。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 2.4 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。例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。ただ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 [4]、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 [11]、物体の運動を 3 次関数で表す [12] 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、四次元空間 (space and time) を導入した例 [16][17] もあるが、いずれも原理的には、物体と掃引物体を三次元または四次元の 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 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 [13]。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八つの子 node に分割される。

octree 表現された 2 物体間の干渉は、それぞれの物体を表す二つの木を並列にたどることによって検出できる [14]。いま、 N_A , N_B を、それぞれの木のなかで、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、この node が示す空間は干渉している可能性がある、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、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分かる。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 は三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 [13][18]、なかでも任意の並進と回転により octree を実時間で更新するアルゴリズムも提案されている [19]。同様の三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 [20] 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。

2.3 移動物体と静止物体

三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あらかじめ分類し、おのおの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。例えば静止物体を octree で表現し、動物体を多面体を用いて表現した例 [15] や、球や円柱などの単純な形状で表現した例 [21] もあるが、作業空間内の移動物体と静止物体とをあらかじめ区別し、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。しかもこれらは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。

2.4 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み合わ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、しそうでない）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 [3] もあるが、世界座標系のなか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 [22] 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み合わせごと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。例えば文献 [5][6][9] 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。ほかにも、物体間の距離に基づく方法はいくつかあるが [7][8]、いずれも同様の仮定が必要である。ほかにも、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いるか否かを素早く判定する方法 [23] 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。

3. octree と多面体表現を用いた衝突検出

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、そのおのおのの多面体と octree の 2 種類の形状表現はあらかじ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 ごとの時刻 ($\dots, t_{i-1}, t_i, t_{i+1}, \dots$) における各物体の三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おのおの変換される。

Fig. 1 に、衝突面検出の流れを示す。以下、各処理ごとにその内容を詳しく述べる。

Fig. 1 Proposed method for colliding face detection

Fig. 2 An example of white nodes occupied partially by an object

3.1 形状表現の更新

多面体と octree の両形状表現はそれぞれ、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ごとにおのおの変換される。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が、octree 表現の任意の並進と回転による更新はそれほど単純ではない。これにはいくつかの方法が提案されている [19][24] が、基本的には、各 black node ごとに変換してこの部分木を作成し、最後にこれらを組み合わせて全体の octree 表現を作成する方法である。例えば文献 [24] では、変換後に生成される octree の cube (upright cube) は、その中心点に変換された cube (tilted cube) の内部にある場合、black となる。しかし本研究では octree と多面体表現をおのおの独立に更新するので、両者の間に微妙な位置ずれが生じる。二次元の平行移動の例を Fig. 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とき、多角形（三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 を越えたとき、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。本研究では、実時間で octree 表現の任意の並進と回転による更新を終了する方法 [19] を利用する。

Fig. 3 に三次元の octree の場合の例を示す。図 (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。同図 (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。

3.2 octree を用いた大まかな干渉検出

作業空間全体のなかでの物体間の干渉を、おのおの 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 のとき、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

(a) polyhedra (528 faces)

(b) octree

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

Fig. 3 The experimental space shuttle

Fig. 4 Identification of object faces responsible for interference detected using octree representation

- n 個の対応する node のうち少なくとも二つが black のとき、または一つが gray で残りの一つ以上が black のとき、この node は干渉している
- n 個の対応する node のうち少なくとも二つが 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。

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. 4 参照)。干渉が見つかった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。

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. 5 参照)。

Fig. 5 Collision detection between moving faces identified by octree representation as potentially colliding

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 ($a_0^{t_i}, a_1^{t_i}, a_2^{t_i}, \dots$) と $A^{t_{i+1}}$ の頂点 ($a_0^{t_{i+1}}, a_1^{t_{i+1}}, a_2^{t_{i+1}}, \dots$) の凸包 $V_A^{t_i}$ を作成する (文献 [25] 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 をおのおの、 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。おのおの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、次の三つのうち 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とき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 [25] 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. 5 では、最も簡単な例として、三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 ごとの離散時刻 ($\dots, t_{i-1}, t_i, t_{i+1}, \dots$) 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 (例えば直進とか、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。

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。

4.1 標準物体を用いた評価実験

三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

Fig. 6 Examples of experimental objects represented using polyhedra and octrees

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. 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。

4.1.1 面数が異なる運動二物体に対する実験

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. 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 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. 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。

各処理時刻において、各物体の octree と多面体表現はおのおの更新された後、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. 7 に示す。面数 168 の二つの球間の衝突の場合、時刻 $t = 5$ [cycles] 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 は見つかったが、それらから物体の面は抽出されないで、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、

Table 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)

Fig. 7 Computation time for each processing cycle of collision detection for two identical objects having 24, 48, 80, 168, 528 and 728 faces

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 ごとに、そのなかに含まれる面間の衝突を調べた。

また、上と同じく面数 168 の二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. 8**, **Fig. 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分かる。

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 の二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、処理の時刻ごとに面の全組み合わ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

Fig. 8 The number of interfering nodes for each processing cycle of collision detection for two identical objects having 168 faces **Fig. 9** The number of extracted faces for each processing cycle of collision detection for two identical objects having 168 faces

法として考えられるが、このような方法はほか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、

Fig. 10 Computation time of collision detection between two identical objects against the number of object faces

ここではそのような方法は用いていない。

Fig. 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. 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ー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。

4.1.2 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, ...) の運動はおのおの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に、Table 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. 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、複数の物体のうちの二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ごとに 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 をおのおの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ほかに、干渉 node から面を抽出

Fig. 11 Computation time of collision detection among multiple moving objects having 168 faces against the number of objects

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一定となる。Fig. 11 に示す結果はこの様子を示している。

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の環境に応用した。Fig. 3 のように、多面体 (三角形パッチ) 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 [26] 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. 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に、おのおの物体ごと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. 13 に実験結果を示す。図中、a, b, c, d はおのおの物体数が 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 が見つかったと、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. 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. 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。また同じ方法で五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。

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

(a) octree

(b) polyhedra

Fig. 12 Experimental space including five objects (space shuttles) at initial positions

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分かる。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、一つの 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、

Fig. 13 Computation time for each processing cycle of collision detection among objects (space shuttle)

Fig. 14 A snapshot of experiment when colliding faces are detected

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。

例えば、 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] のときの適切な 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. 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、4.1.1 項の実験と同じ要領で、二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、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、こうした衝突条件の変動を考慮して、同一の物体の組み合わせによる衝突をおのおの 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. 15 では、面数

Fig. 15 Computation time (average of 60 collisions with standard deviation) of collision detection between two identical objects against the level of octree shape representations

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分かる。

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三角形は小さくなる (これらの面はほぼ辺の長さが等しいが、必ずしも正三角形ではない)。面数 48, 360, 960 の球で、各面の最長辺の平均をとってみると、おおよそ 0.88 (level 3 octree voxel の一辺の長さ = 1), 0.73 (level 4 octree voxel の一辺の長さ = 1), 0.91 (level 5 octree voxel の一辺の長さ = 1) であった。ここでは物体を構成する面がほぼ正三角形に近く、均一に分布させた球を用いて実験を行ったが、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. 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分かる。

6. おわりに

以上本稿では、並進と回転を含む運動をする複数の三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との間の衝突可能性を局所的に正確に調べた。本稿では、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、三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、3 に述べた一連の衝突面検出の手順のなかで、3.4 節の手順に要する計算量はほか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。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。

謝 辞

本研究の機会を与えて下さった葉原耕平 ATR 通信システム研究所会長ならびに寺島信義同社長に感謝致します。

参 考 文 献

- [1] Y.K. Hwang and N. Ahuja: "Gross motion planning—a survey," ACM Computing Surveys, vol.24, no.3, pp.219–291, 1992.
- [2] Alex P. Pentland: "Computational complexity versus simulated environments," Computer Graphics, vol.24, no.2, pp.185–192, 1990.
- [3] James K. Hahn: "Realistic animation of rigid bodies," Computer Graphics, vol.22, no.4, pp.299–308, 1988.
- [4] John W. Boyse: "Interference detection among solids and surfaces," Communications of the ACM, vol.22, no.1, pp.3–9, 1979.
- [5] Ming C. Lin and John F. Canny: "A fast algorithm for incremental distance calculation," In International Conference on Robotics and Automation, pp.1008–1014, IEEE, 1991.
- [6] M.C. Lin, D. Manocha and J.F. Canny: "Fast contact determination in dynamic environments," In International Conference on Robotics and Automation, pp.602–608, IEEE, 1994.
- [7] G. Gilbert, W. Johnson and S. Keerthi: "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.
- [8] Sean Quinlan: "Efficient distance computation between non-convex objects," In International Conference on Robotics and Automation, pp.3324–3329, IEEE, 1994.
- [9] J.D. Cohen, M.C. Lin, D. Manocha and M. Ponamgi: "I-COLLID: An interactive and exact collision detection system for large-scale environments," In Symposium on interactive 3D Graphics, pp.189–196, ACM, 1995.
- [10] 北村喜文, エイミー, 岸野文郎: "面間の動的拘束を用いた仮想物体の操作補助法," 電子情報通信学会誌, vol.J79-A, no.2, pp.506–517,

- 1996.
- [11] John Canny: "Collision detection for moving polyhedra," IEEE Transactions on PAMI, vol.8, no.2, pp.200-209, 1986.
 - [12] 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.
 - [13] Hanan Samet: "The design and analysis of spatial data structures," Addison-Wesley, 1990.
 - [14] N. Ahuja, R.T. Chien and N. Bridwell, "Interference detection and collision avoidance among three dimensional objects," In International Conference on Artificial Intelligence, pp.44-48, 1980.
 - [15] 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.
 - [16] Stephen Cameron: "Collision detection by four-dimensional intersection testing," IEEE Transactions on Robotics and Automation, vol.6, no.3, pp.291-302, 1990.
 - [17] Philip M. Hubbard: "Interactive collision detection," In Symposium on Research Frontiers in Virtual Reality, pp.24-31, IEEE, 1993.
 - [18] H.H. Chen and T.S. Huang: "A survey of construction and manipulation of octrees," Computer Vision, Graphics, and Image Processing, vol.43, no.3, pp.409-431, 1988.
 - [19] Y. Kitamura, A. Smith and F. Kishino: "Efficient algorithms for real-time octree motion," IEICE Transactions on Information and Systems, vol.E78-D, no.12, pp.1573-1580, 1995.
 - [20] 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.
 - [21] C.A. Shaffer and G.M. Herb: "A real-time robot arm collision avoidance system," IEEE Transactions on Robotics and Automation, vol.8, no.2, pp.149-160, 1992.
 - [22] A. Garcia-Alonso, N. Serrano and J. Flaquer: "Solving the collision detection problem," Computer Graphics and Applications, vol.14, no.3, pp.36-43, May, 1994.
 - [23] David Baraff: "Curved surfaces and coherence for non-penetrating rigid body simulation," Computer Graphics, vol.24, no.4, pp.19-28, 1990.
 - [24] J. Weng and N. Ahuja: "Octrees of objects in arbitrary motion: Representation and efficiency," Computer Vision, Graphics, and Image Processing, vol.39, no.2, pp.167-185, 1987.
 - [25] Preparata, Franco P., Shamos, Michael Ian (著) 浅野孝夫, 浅野哲夫 (訳): 計算幾何学入門. 総研出版, 1992.
 - [26] 北村喜文, ベイルマイク, 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.

北村喜文 (Yoshifumi Kitamura)

1962年7月30日生。85年，大阪大学基礎工学部制御工学科卒業。87年，大阪大学大学院基礎工学研究科博士前期課程修了。同年，キャノン入社。三次元画像処理，コンピュータビジョン等の研究に従事。92～96年，ATR通信システム研究所に出向。三次元ユーザインタフェース等の研究に従事。現在，キャノン株式会社情報メディア研究所研究員。工学博士。電子情報通信学会会員。(日本ロボット学会正会員)

アフジャナレンドラ (Narendra Ahuja)

Narendra Ahuja received the B.E. degree with honors in electronics engineering from the Birla Institute of Technology and Science, Pilani, India, in 1972, the M.E. degree with distinction in electrical communication engineering from the Indian Institute of Science, Bangalore, India, in 1974, and the Ph.D. degree in computer science from the University of Maryland, College Park, USA, in 1979. Since 1979 he has been with the University of Illinois at Urbana-Champaign where he is currently (1988-) a Professor in the Department of Electrical and Computer Engineering, the Coordinated Science Laboratory, and the Beckman Institute. He serves or has served on the editorial boards of the journals; IEEE Transactions on PAMI, Computer Vision, Graphics, and Image Processing, Journal of Mathematical Imaging and Vision, and Journal of Information Science and Technology. He is fellow of the IEEE, AAAI, IAPR, ACM, and International Society for Optical Engineering. He is a member of the Optical Society of America.

竹村治雄 (Haruo Takemura)

1958年4月3日生。82年，大阪大学基礎工学部情報工学科卒業。87年，大阪大学大学院基礎工学研究科博士後期課程単位取得退学。同年，国際電気通信基礎技術研究所入社。ATR通信システム研究所勤務。三次元ユーザインタフェース，CSCW，仮想現実などの研究に従事。93年，奈良先端科学技術大学院大学情報科学研究科助教授。工学博士。IEEE，ACM，電子情報通信学会，情報処理学会各会員。

岸野文郎 (Fumio Kishino)

1946年7月16日生。71年，名古屋工業大学電子工学科修士課程修了。同年，NTT電気通信研究所入所，同ヒューマンインタフェース研究所において，高速・広帯域通信システムの研究・実用化，画像処理の研究に従事。89年，ATR通信システム研究所知能処理研究室室長。画像処理，知能処理，臨場感通信などの研究に従事。現在，ATR知能映像通信研究所所属。工学博士。IEEE，電子情報通信学会，テレビジョン学会各会員。