〔非公開〕

ＴＲ－Ｃ－００８１

仮想協調作業空間における物体の管理

北　村　　喜　文
Yoshifumi　KITAMURA

１９９３　１．１４

ＡＴＲ通信システム研究所

仮想協調作業空間における物体の管理

Object Management in Virtual Cooperative Workspace

北村喜文
Yoshifumi KITAMURA
ATR 通信システム研究所 知能処理研究室
kitamura@atr-sw.atr.co.jp

概要

本稿では仮想空間での協調作業における物体の管理方法について検討する．仮想空間中で，物体間の衝突時に反射や破壊などの物理法則を適用したり，物体の接触配置作業などを容易にするために，仮想空間中の複数の物体の位置を管理し，物体間の衝突／干渉チェックする方法を考える．まずロボットシミュレーションにおける衝突／干渉チェックの手法を調査した．これらの手法は，(1) 多面体表現を用いた静的な干渉チェック，(2) 運動物体の軌道を用いた動的な衝突チェック，(3)octree を用いた静的な干渉チェック，に大きく分類することができた．これを踏まえて，仮想空間での協調作業における物体の管理する方法を検討した．その結果，octree と多面体の 2 種類の形状表現を両立させた干渉チェック法が有効であろうとの検討結果を得た．

目次

1 はじめに

遠隔地にいる複数の人間が1つの仮想空間を共有し，この空間に対して実空間と同じように働きかけることにより協調作業を行なうことを目指して，仮想空間操作の研究が進められている．

例えばブロックの並べ替えやレイアウト変更などの作業を，複数の人間が手でつかんで協調してこれを行なう状況を想定する [TK92]．この場合，作業者が存在する実空間と仮想空間が違和感なく同様に扱えることが望ましい．この問題点の1つとして，物体に物理法則が適用されていないなどの点がある．例えば，

1. 物体に体積や質量が表現されてないので，物体が他の物体の存在を無視して突き進んでしまう．そのため，複数の物体を接触させて配置させたり，協調作業時に物体同士をはめ合わせるなど，高度な作業が困難である．
2. 物体間の衝突，反射運動などが正確には表現できない．
3. 無重力であるので，物体は中空に浮かんだままで自由落下しない．

仮想空間中で，物体間の衝突時に反射／破壊などの物理法則を適用したり，物体の接触配置作業などを容易にするためには，仮想空間中の複数の物体の位置を管理し，物体間の衝突／干渉チェックを考える必要がある．現状のシステムでは，物体は単なる surface model で記述され，計算機のパワーがリアルタイム表示にかかりっきりのせいもあり，きっちりと考えられた例はない．

この仮想空間内の物体間の衝突／干渉チェックの問題は，ロボットの行動計画を立てるための要素技術として長く研究されてきた，ロボットシミュレーションにおける衝突／干渉チェックの問題解決法が参考になる．2章では，まずこの分野の研究を調査した結果を述べる．3章では，この結果を踏まえて，仮想空間での協調作業における物体の管理する方法を検討する．また，4章では，octree を用いた仮想協調作業空間における物体間のグローバルな干渉チェック法を検討する．

2 ロボットシミュレーションにおける衝突・干渉チェック

この分野の解説としては，日本ロボット学会論文誌を中心に文献 [尾崎 84] [有本 88] [木村 88] [新井 92] がある．これらの研究例を，問題設定とそれへのアプローチ，物体の表現法の観点から整理する．表 1参照.

2.1 概要

2.1.1 問題設定とそれへのアプローチ

これらが扱っている問題を大きく分けると，次のように分類できる．

1. 静止物体間の干渉 (interference) ··· 静止物体同士が重なっている状態
2. 運動物体間の衝突 (collision) ··· 移動物体が他の物体に接近・接触し，さらに重なる一連の動作

2 はさらに

(a) 静止物体と移動物体間の衝突

(b) 移動物体間同士の衝突

に分類できる．以下，それぞれについて説明する．

静止物体間の干渉

静止物体間の干渉の問題は，立体間の集合演算による形状作成操作 (CSG) などに必須の機能として解決されており，最近の3次元 CAD モデラには基本機能の1つとして備わっている．この実現手法などについては，立体モデリングの教科書 [Man88] などに詳しい．また [Boy79] でも扱われている．[重松 83] では，CSG による CAD システムの中で，物体間の積集合部分を抽出している．

運動物体間の衝突

運動物体間の衝突の問題では，運動物体の扱い方によって，次の3つのアプローチがある．

1. 物体の連続的な動きを一定の（空間的，時間的）間隔でサンプルされた離散位置で近似し，各位置で立体間の静的な干渉を検出する
2. 運動物体が空間を掃引してできる立体（通過する領域）と他の物体との干渉を静的に検出する

表 1: 主な衝突チェック法の分類

	物体の表現		
運動の表現	円柱や四角柱など単純な形状による近似	B-rep など多面体による近似	octree などの体積表現
物体の連続的な動きを一定の間隔でサンプルされた離散位置で近似し，各位置で立体間の静的な干渉を検出	[尾崎 82], [小沢 86]	[Boy79]	[ACB80], [Hay86], [登尾 87], [劉 89]
運動物体が空間を掃引してできる立体と他の物体との干渉を静的に検出		[Her86]	
4 次元の時空間での解析により干渉を検出		[ER84]	
運動物体の動きを方程式で表し，これを解析的に解くことにより衝突する場所と時刻を算出する	[水垣 84]	[Boy79], [Can86], [岡野 88]	

3. 運動物体の動きを方程式で表し，これを解析的に解くことにより衝突する場所と時刻を算出する

第 1 のアプローチは立体間の静的な干渉検出法によって容易に実現できるが，もともと膨大な計算を要する干渉検出を多数回繰り返すことになるので，計算時間がかかる．また，運動をサンプルする際の間隔を大きく取り過ぎると，起こり得る衝突も見逃す恐れがある，などの欠点がある．この問題を解く方法としては，次の 2 つがある．

1. 物体間の共有点を求める

2. 近似表現した物体の中心線（基準線）間の距離を比較する

一方，第 2，第 3 のアプローチは，立体の運動が比較的単純な場合には有効であるが，複雑な場合には困難である．第 2 のアプローチによる研究例は，時間の概念が陽に現われないので，運動物体が 2 つ以上ある場合には使いにくい [Cam85]．そこで類似の方法として，3 次元物体の運動を時間軸を加えた 4 次元の時空間内の解析による方法 [ER84] [Cam85] があるが，計算機内の表現が複雑になり直観的に理解しにくいなどの欠点がある．

第 3 のアプローチによる研究例は，[Boy79] [水垣 84] [Can86] [岡野 88] などがある．本来的には，衝突チェックは移動ロボットやマニピュレータの行動計画をたてるためであることを考慮すると，このように衝突する時間と場所が正しく求まる必要があろう．

2.1.2 物体の表現方法

干渉計算を高速化するために，移動物体（ロボット）や環境物体の形状は安全側に近似された多面体 (B-rep) で近似される例が多い [Boy79] [重松 83] [小沢 86] [Can86] [岡野 88]．さらに物体を円柱や四角柱など簡略形状で近似表現した例 [尾崎 82] [水垣 84] [小沢 86] もある．

干渉問題を単純化するために，静止した作業環境に対して移動するロボット（マニピュレータ）を太さのない折れ線で近似し，太さ分の寸法だけ作業対象物の寸法を拡大することもある．そして折れ線が運動してできる掃引面と拡大された静止物体との干渉を検出しようとする例もある．この場合には，面と立体の干渉計算になるので計算が楽になる．これは 2 次元で知られている手法 [LP82] を拡張した試みである．

自由曲面 (Bezier や B-spline) は，パラメータと実空間における位置や距離との対応は一般に複雑で解りにくい，といった理由から，これは用いられていない．また一般化円筒などの sweep による表現も，同じ理由から用いられていない．

CSG による形状表現は，直観的には，一定の時間間隔でサンプルされた離散位置で立体間の干渉を静的に検出に向いているように思えるが，もともと計算量が非常に多い処理であるので，これを繰り返し計算させるには無理がある．

voxel や octree といった体積表現法は，基本演算が単純で理論的に明解ではあるが，一般に形状表現は近似となり，データ量は多くなる．しかし，計算時間は物体の数や形状の複雑さに依存せず，ハードウエアで高速化できる見通しはある．octree を使った干渉チェックの枠組は [ACB80] で報告されている．一般的な運動をする物体に対しては，特に回転運動は木の構造や node 数が変化してしまうため，この表現法を適用することは困難である．そこで，

運動しない静止作業環境を表現するのに用いた例 [登尾 87] [劉 89] [Her86] がある．また，物体の運動に伴って octree 表現を更新する方法について，物体の運動を平行運動のみに限定した例 [AN84]，回転運動について [WA85] [WA87] [HO87] に研究例がある．

2.2 研究例

2.2.1 ロボットシミュレーションにおける衝突・干渉チェックの研究例

[Boy79] は物体を多面体 (B-rep) で表現し，静止 2 物体間の干渉と，静止物体と移動物体間の衝突の 2 つの問題を扱った．前者の場合は，複数の物体の全ての組み合わせに対して，エッジ（稜線）と面の交わり方の状態を調べる．後者についても同じように 2 物体の衝突を，図 1のように

1. 頂点が面上に接触する場合
2. 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，これらと静止物体の面やエッジの交点を，全ての組み合わせについて解析的に解いている．但し数式を解く手続きを簡単にするために，物体の運動は平行移動とある軸回りの回転に制限されている．今，R_e, R_f をそれぞれある運動物体の稜線と面の数，E_e, E_f をそれぞれある静止環境物体の稜線と面の数とすると，これら 2 物体間の衝突チェックに要する計算コストは，$R_eE_e + E_eR_f + E_fR_e$ に比例する [Hay86]．すなわち，物体の複雑さと数が増すにつれて計算量は増大することになる．

この衝突の場合分けによる考え方を，複数の運動物体間の衝突問題に適用した例もある．[Can86] は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扱い，[岡野 88] は物体の運動を 3 次関数で表して，物体の衝突時刻と場所を解析的に求めた．いずれも物体の衝突を上記のように分類して，全ての組み合わせについて計算し，最も衝突までの時間が短いものを選んでいる．[岡野 88] の報告では直方体同士の衝突で，240 の方程式を解くのに 1 秒程度 (IBM System360/370) とのことである．

また [水垣 84] は，ロボットを円柱のリンクで表し，リンク中心線が通過する軌道面と，リンクの半径を無視した分だけ厚みを増した対象物との交点を解析的に求めている．

[尾崎 82] は，物体を直方体で近似し，この中心間の距離から干渉の有無を判定する条件式を導いた．これは近似表現のため，高速の干渉チェックが可能である．サンプリング時刻毎にこのチェックを行なう．また，[小沢 86] は物体を多面体，円柱または四角柱で近似し，それぞれの組み合わせごとに同様の距離の条件式から干渉をチェックした．

[登尾 87] は，静止した作業環境を octree，移動物体（ロボット）を多面体 (B-rep) で表現し，octree の階層構造を利用した静止物体と移動物体間の効率的な干渉チェックアルゴリズムを提案した．しかし，この方法を単純に 2 つの運動する物体間の干渉チェックに適合するように拡張するのは困難である．[Her86] は，ロボットが運動することによってできる sweep 物体を概略形状で近似している．[劉 89] は，同様な体積表現法の 1 つとして，回転方向の操作が楽な階層的球 (HSM: Hierarchical Sphere Model) で表現した物体間の干渉を扱った．これは，物体が回転運動をしても球領域間の距離を逐次計算するだけでそれらの干渉を高速にチェックできる特徴を持つ．

[Cam85] は，CSG で表現される物体の連続的な動きを一定の時間間隔でサンプルされた離散位置で近似し，各位置で物体間の静的な干渉を検出する方法，運動物体が空間を掃引してできる立体（通過する領域）と他の物体との

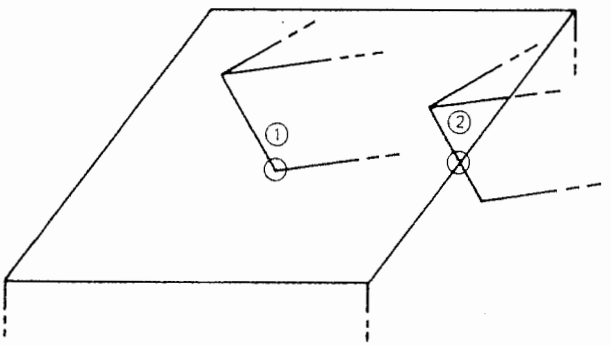

図 1: 衝突状態の分類

干渉を静的に検出する方法，3次元物体の運動を時間軸を加えた4次元の時空間内の解析による衝突検出の方法，の3種類の方法を比較している．[ER84]も4次元の時空間内の解析による衝突検出を扱っているが，詳細な記述がない．

2.2.2 仮想空間への応用例

[吉村91]では[岡野88]の研究を利用して仮想空間の衝突チェックを試みた例が報告されているが，前述のとおり計算量が多く，実際のデモシステムには組み込まれていないようである．

3 仮想空間での協調作業における物体の管理

物体の接触配置など高度な作業を容易にするための要素技術の一つとして，仮想空間中の複数の物体の位置を管理し，物体間の衝突／干渉チェックを考える．

3.1 仮想空間を利用した協調作業における物体の衝突 / 干渉チェック問題の特徴

前章で調べたロボットシミュレーションにおける問題は，もともと移動ロボットやマニピュレータの障害物との衝突を回避した行動計画をたてるために，研究されたものである．これに対し，仮想空間を利用した協調作業における物体の衝突／干渉チェックの問題は次のような異なった特徴を持つ．

1. 作業空間には，複数の人間が独立に操作することによる複数の運動物体が存在する．
2. 手でつかんで運ぶことを想定しているので，移動物体の軌道を正確に数式で表すことができない．また運動を開始する以前には，その物体の運動については何の情報もない．運動して初めてその向きと移動量（速度）がわかる．
3. 物体は操作する人の意思により，突然動き出すし，また止まる．物体の運動属性は突如変化することがある．
4. 物体の表示や人の動きの認識などに時間がかかるので，連続した時間の関数による解析的な方法は使いづらい．どうしても，離散的な時間と場所ごとのチェックになる．
5. 移動するロボットと静止環境物体といった区別はできない．全ての物体を，運動物体と静止物体の区別なく同じ3次元物体表現の枠組で表したい．床や壁など環境物体であることが明らかな対象物は，別の枠組で表現することも考えられるが，あらゆる衝突を同じ枠組で扱う方が場合分けなどをせずに済み，美しい．
6. 空間中の物体数，運動物体数，形状の複雑さに計算時間量が依存しないアルゴリズムが必要．
   - B-rep などで表現された物体を利用したアルゴリズムは，干渉チェックの計算時間が頂点や面の数の組み合わせに比例して増大する．最大計算時間量がある範囲内に収まることが保証される方式が望まれる．
7. 物体のレイアウト変更作業では，物体同士を接触させて置きたいことも多い．

3.2 仮想協調作業空間の衝突 / 干渉チェックへのアプローチ

以上の特徴から，まず運動物体の連続的な動きを方程式で表し，これを解析的に解くことにより衝突する場所と時刻を算出する方法は使いにくいことがわかる．また，運動物体が空間を掃引してできる立体と他の物体との干渉を静的に検出する方法や，4次元の時空間での解析により干渉を検出する方法を利用するとしても，物体運動の初速度やある区間の速度などを利用した概略的なものにならざるを得ない．したがって，離散的な時間ごとの物体間の干渉を調べることが適当である．

前節の特徴5から，物体表現として適当と考えられるのは，

- 円柱または四角柱といった近似図形
- 多面体
- octree などの体積表現法

のいずれかを単独で使用する方法であるが，特徴6を実現するためには，Decomposition 型の形状表現が有利である．これには octree が一般的である．

一方，octree で表現される物体形状は近似形状であるので，特徴7にあげたような精度が求められる作業には向いていない．このような作業には，B-rep のような多角形などによる形状表現が望ましい．そこで両者について，特徴を整理してみる．

3.2.1 octree 表現の特徴

仮想空間における協調作業時の物体の衝突チェックの問題に octree を用いた場合の期待される効果として，

1. 衝突チェックに要する計算量は，空間の解像度（セルの数）によって決まる．
2. 物体の運動速度（サンプル時間当たりの移動量）に応じた空間解像度で干渉の効率的なチェックができる．
   - サンプル時間当たりの移動量が大きい → 同じ時間内ではその移動量に等しい幅の荒い干渉チェックができれば良い
3. 複数の視線や指先による方向指示によって物体の占有空間を指し示すことができる．

また欠点もある，

1. octree で表現される物体の形状は，本来の物体の近似表現である．
2. 運動する物体を表現するのが困難である．特に回転については，木の構造が全く変化してしまうので，取り扱いにくい．
3. それぞれの物体の octree 表現を作成するのも困難である．
   - 物体は CG による高速表示するための三角形パッチ表現などの surface model との整合性（どちらか一方があれば他方が生成できるなど）が考えられれば効率的である．

octree で表現された物体の平行移動運動に伴い，octree を高速に更新する研究は [AN84] に見られる．回転については，[WA85] [WA87] [HO87] にある．また octree を用いた衝突チェック，行動計画の例として，[ACB80] [Hay86] がある．

物体の回転運動を考慮すると，[劉 89] で提案されている HSM が良いのかもしれないが，データ量がさらに増す，世界座標系との整合がとりにくいなどの欠点がありそうである．今後の検討が必要である．

欠点 3については今後の課題である．たとえば voxel の表面に対して三角形パッチを張る Marching Cubes 法 [LC87] は知られているが，octree に対してはどのような研究がされているのだろうか．B-rep に変換する例としては [Kun85] があるが，今後調査する必要がある．またその他に octree の操作に関する研究例として，例えば，画像から生成する例 [CA86] [SA90]，距離画像からレイ・トレーシングの要領で octree を作成する例 [Con84]，また octree を表示する方法として [DT81] がある．octree に関するサーベイは [Mea82] にある．

物体は近似的に表現されるから，CAD システムで要求される精度で正確に接触を求めることはできないし，その必要もない．

3.2.2 多角形による形状表現の特徴

B-rep などの多面体を利用した surface model による形状の表現は，CG によって容易に描画できる形状表現の１つであり，現状のデモシステムでも一般的に利用されている．したがって，改めてこの表現を作成する必要はない．しかし，2章で [Boy79] [Can86] [岡野 88] の研究例で述べたように，物体間の衝突／干渉のチェックに必要な計算の量は，物体の個数や複雑さが増すにつれ，増大するという欠点をもつ．他方，（多面体として表現された近似精度の程度での）正確な干渉チェックができるという利点もある．

3.3 仮想空間での協調作業における物体の管理

3.3.1 概要

3.2での議論から，物体の衝突を調べるには，サンプル時間ごとの物体間の干渉チェックが適切であるとの結論を得ることができる．そして，作業空間全体を考慮したグローバルな干渉チェックには，octree を用いた物体表現の方法が適当であるが，物体が十分に近づいた後には，干渉チェックの対象となる物体や面が特定できるので，B-rep などの多面体による形状表現を用いて正確に干渉をチェックし，接触位置を求めるのが適当であろう．

まず第一段階として，協調作業空間に配置されている物体による占有空間を監視し，他の物体の運動がこの空間を浸食しないようにすることを目標とする．この段階では，octree を用いた大まかに干渉をチェックする．

次に仮想空間を利用した協調作業のうち，物体の接触配置などを支援する方法を検討する．例えば，octree を用いた大まかな干渉チェックによって，ある程度物体同士が接近した後に限定された面と稜線の組合せに対して，B-rep などの多面体による形状表現を用いて正確に干渉をチェックする方法を考える．

このようにして仮想空間での協調作業における物体物体が衝突または接触していることが解れば，これらに反射／破壊／自由落下など物理法則を適用する方法を検討し，必要であればそのために必要な拡張を考える．

またoctree表現を他の形状表現から作成する方法（またはその逆の手法）を検討する．たとえば，三角形パッチ表現をoctreeから生成する方法の検討，または逆に三角形パッチ表現や画像からoctree表現を生成する方法を検討する．

3.3.2 octreeを用いた仮想空間での協調作業における物体間の干渉チェックのシナリオイメージの例

前節の第1段階のシナリオを説明する．協調作業空間をoctreeで表現する．aをこの最小セルの辺の長さ，lを木の階層数とすると，表現される空間は一辺$l \cdot a$の立方体である．これが基準世界座標系である．

床，壁，机，人体，作業対象物の全てを，基準世界座標系で表す．ただし，手は今回の衝突チェックの対象から除外する．これは，手と物体の間の干渉は考えないものとし，現在の「掴む」などの動作理解のアルゴリズムとの両立を図るためである．

物体の運動は平行移動だけに限定し，回転は考えないものとすると，[AN84]によれば，物体の移動に伴う木の更新は，物体によって占有される空間（セル）ごとのテーブル参照だけで実行される．回転運動については，[WA87], [HO87]を参考にするか，または[劉89]によるHSMを後に検討する．

描画，通信などのプロセスによって制限される時間間隔ごとに，運動中の物体（手で掴まれている）の位置を知ることができる．このサンプル時間ごとの移動量をx, y, z成分に分け，各成分を最小セルの辺の長さaの整数倍で表す．これをさらに二進数で表すと，octreeの各levelに対する運動を指示することになる．

2つの物体の干渉チェックは，両者の木をたどって，同じセルが同時に占有されているかどうかを見れば良い．運動中の物体とその進行方向の物体を選び出すことができれば，効率的にチェックすることも可能である．また，1つのサイトの利用者が動かせる物体は1つだけであるとすると，各サイトごとにチェック作業をすることができる．ただし，その時点の物体の空間占有状況を把握している世界座標octreeを参照することが必要である．

干渉していることがわかれば，物理法則プロセスに伝える．最も簡単には，例えばその運動はキャンセルされるべきだとして，もとの移動前の（1つ前のサンプリング）位置に描画させるなどが考えられる．

3.3.3 研究課題

仮想空間における協調作業時の物体の管理法に関する研究の方向を，次のようにまとめてみる．

1. 仮想空間を利用した協調作業における物体の衝突チェックの問題に，octreeを使った場合とB-repなどの多角形表現を用いた場合の比較．たとえば項目としては，
   (a) 計算量の見積り
2. octreeを用いたグローバルな干渉チェック機能の試作
   (a) 物体の運動を平行移動に限定
   (b) 運動を回転移動に拡張
3. 2種類の形状表現を両立させた干渉チェック法の検討
   - グローバルな干渉チェック → octreeを用いて大まかに調べる
   - ローカルな干渉チェック → 限定された物体（の面，稜線）に対して，多角形表現を用いて正確に調べる
4. 協調作業支援法の検討
   (a) 物体同士の接触配置，はめ合わせ作業の支援
   (b) 衝突を検知した後，反射／破壊など物理法則に適用させるための属性の記述
5. octree表現を他の形状表現から作成する方法（またはその逆の手法）の検討
   - 三角形パッチ表現をoctreeから生成する方法の検討
   - 三角形パッチ表現からoctree表現を生成する方法の検討

4 octreeを用いた仮想協調作業空間における物体間のグローバルな干渉チェック

以上の研究方針に基づき，まず仮想協調作業空間における物体間のグローバルな干渉チェックに，octreeを用いた場合の効果について検討する．なおoctree表現自体の特徴については，3.2.1で既に述べた．

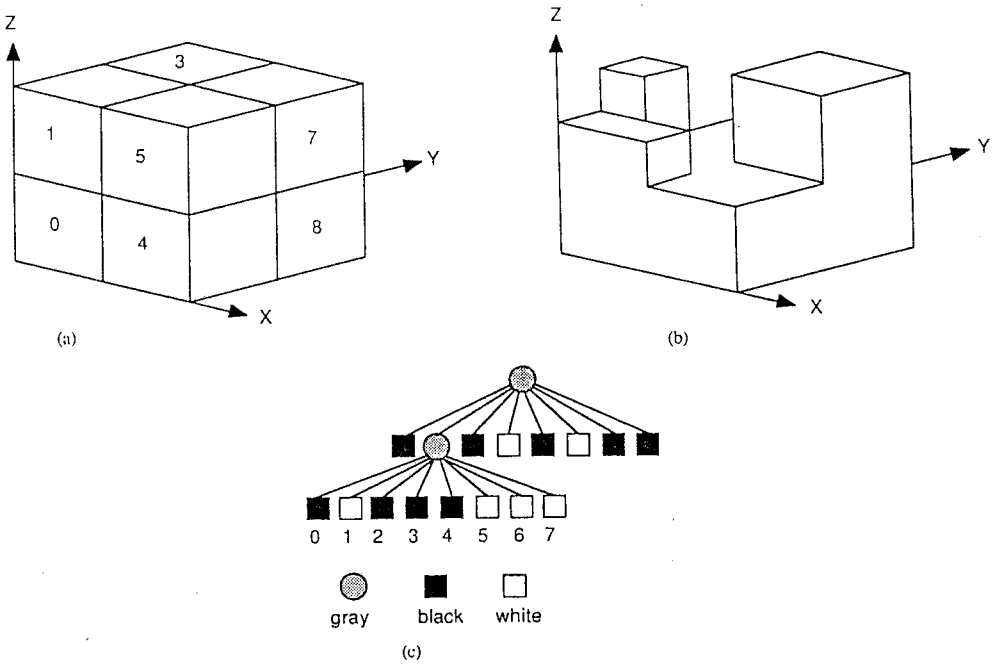

図 2: ある物体とその octree 表現の例

4.1 octree による物体の形状表現

octree で表現された物体間のサンプル時間ごとの干渉チェックを考える．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．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，予め決められた最小の大きさに達するまで 8 つの children に分割される．children を持たない node は leaf node と呼ばれる．

例えば，いま $2^l \times 2^l \times 2^l$ の単位ボクセルの配列を考える．ここで，l は全体空間の解像度に相当する量であり，2^l は root node の 1 辺の長さを示す．図 2(a) の様に，root node は 0, 1, 2...7 にラベル付けされた，1 辺の長さが 2^{l-1} の children に分割される．各々の child は同じように，さらに分割される．図 2(b) に示す物体の octree 表現を (c) に示す．octree 表現において，root の level（階数）を l，level k の node の children は level $k-1$ であるとする．最下位の level は 0 であり，各 node が単位 voxel に相当する．

octree のデータ構造，処理手法については [Sam90a][Sam90b] などに詳しく述べられている．[Man88] で述べられている octree のデータ構造の例を図 3に，octree を作成する手順を図 4に記す．

4.2 octree を用いた干渉チェック

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 つの木を並列に辿ることによってチェックできる [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 leaf 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 children 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，children 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 個の木を辿る必要がある．そして，上と同じように，n 個の対応する node のうち n-1 個の node が全て white の時，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．

4.3 干渉チェックの計算量

4.2で述べた方法による干渉チェックの計算量は，実際に辿った 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計算量のオーダーは，K_i を i 番目の物体の octree の node 数，その最大値を K^{max} とすると，

$$O(actual\ number\ of\ nodes\ visited) \leq O(K_1 + K_2 + \cdots + K_n) \leq O(K^{max} \cdot n) \quad (1)$$

```
struct octreeroot
{
        float           xmin, ymin, zmin;    /* space of interest */
        float           xmax, ymax, zmax;
        struct octree   *root;               /* root of the tree */
};

struct octree
{
        char            code;                /* BLACK or WHITE or GRAY */
        float           *oct[8];             /* pointers to octants,
                                                preset if GRAY */
};
```

図 3: octree のデータ構造

である．ここで，最大計算時間量 (maximum time complexity) は n 個の物体を表す木の node すべてを辿った場合である．平均的な octree の node 数を K とすると，計算時間量は，$O(Kn)$ である．

一方，物体を多面体で表現した場合，2 物体間の干渉は例えば [Boy79] のように，それぞれの物体の稜線と面，稜線と稜線の位置関係を全ての組み合わせに対して調べることによりチェックすることができる．物体 1，2 の稜線の数を E_1，E_2，面の数をそれぞれ F_1，F_2 とすると，干渉チェックのオーダーは，

$$O(F_1 \cdot E_2 + E_1 \cdot F_2 + E_1 \cdot E_2) \tag{2}$$

である．n 個の物体間の干渉を調べる場合は，i 番目の物体の稜線数を E_i，面の数を F_i とすると，$i \neq j$ の i，j に対して，

$$O\left(\sum_{i=1}^{n}\sum_{j=1}^{n}(F_i \cdot E_j + E_i \cdot F_j + E_i \cdot E_j)_{i\neq j}\right) \geq O\left({}_nC_2 \cdot (2 \cdot F^{min} \cdot E^{min} + {E^{min}}^2)\right) \tag{3}$$

$$= O\left(E^{min}(2F^{min} + E^{min}) \cdot n^2\right) \tag{4}$$

である．ここで，E^{min}，F^{min} は，それぞれ n 個の物体の稜線数のうち最小の稜線数，最小の面数である．平均的な物体の面，稜線の数を各々 F，E とすると，計算時間量は $O\left((2FE + E^2)n^2\right)$ である．対象物体の数が増すにつれ，2 乗のオーダーで計算量が増大することがわかる．

また一般的な物体の場合，octree のノード数は物体の表面積に比例することが報告されている [Mea82]．octree による表現と多面体による表現の場合，いずれも物体が複雑になるにつれて干渉チェックの計算量は増大するが，その割合は同程度であると考えられる．

例えば図 2に示すような物体が 2 つあるとして，これらの干渉チェックを考えた場合，octree を用いた場合は 17 の node で表現されているので，最大計算量は $17 \times 2 \times (interference\ check)$ である．一方，同じ物体を多面体で記述した場合，16 の面と 34 の稜線で表現されるので，計算量は $(16 \times 34 \times 2 + 34^2) \times (interference\ check)$ となる．ここで (interference check) はいずれも座標値の簡単な大小関係の比較である．

4.4 平行移動物体の octree 表現の更新

[AN84] では，平行移動物体の octree 表現を更新する方法が述べられている．ある leaf node の並進運動が与えられると，それを x, y, z 各成分ごとに，最小単位 voxel を基準として 2 進数で表す．そして，octree の各 node ごとに表 2を参照して移動後の表現を得る．たとえば，$r057$ で表される leaf node を y 軸正方向に 5 (2 進数で 101) 移動させる場合を考えると，次のように $r275$ に移動する．

$$\begin{array}{r} r\,0\,5\,7 \\ \quad 1\,0\,1 \\ \hline r\,2\,7\,5 \end{array}$$

```
make_tree(p, t, depth)
    primitive *p;    /* p = the primitive to be modeled */
    octree *t;       /* t = node of the octree,
                            the initial tree with one white node */
    int depth;       /* initially max.  depth of the recursion */
{
        int i;
        switch(classify(p, t))      /* classify octree nodes against primitive */
        {
            case WHITE:
                t->code = WHITE;
                break;
            case BLACK:
                t->code = BLACK;
                break;
            case GRAY:
                if(depth == 0)
                {
                    t->code = BLACK;
                }
                else
                {
                    subdivide(t);   /* divide octree node into eight octants */
                    for(i=0; i<8; i++)
                        make_tree(p, t->oct[i], depth-1);
                }
                break;
        }
}
```

図 4: octree の生成

しかし，この方法を任意の回転運動をする物体に拡張することは困難である．次節で述べる方法で平行移動物体をも扱える．

4.5 回転を含む移動物体の octree 表現の更新

4.5.1 概要

参考文献 [WA87] では，回転と平行移動を含む任意の運動に対して，octree 表現を更新するアルゴリズムが報告されている．この基本的な考え方は，次の 2 点である．

- 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．
- 部分的に物体に占有されている voxel は，その中心が物体内部にある時のみ black とする

これらにより，例えば角度 ϕ 回転させた後に $-\phi$ 回転させたときの octree 表現が一致し，また任意角度の回転に対して black node の数がいたずらに増えることを押えている．

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 $\mathbf{n}^0 = (n_x, n_y, n_z)$ 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，

$$X' = RX + T \tag{5}$$

表 2: 単位平行移動にともなうラベルの更新

initial label	final label after unit displacement		
	+X	+Y	+Z
0	1	2	4
1	10	3	5
2	3	10	6
3	12	11	7
4	5	6	10
5	14	7	11
6	7	14	12
7	16	15	13

ただし,

$$R = \begin{pmatrix} (n_x^2-1)(1-cos\phi)+1 & n_x n_y(1-cos\phi)-n_z sin\phi & n_x n_z(1-cos\phi)+n_y sin\phi \\ n_y n_x(1-cos\phi)+n_z sin\phi & (n_y^2-1)(1-cos\phi)+1 & n_y n_z(1-cos\phi)-n_x sin\phi \\ n_z n_x(1-cos\phi)-n_y sin\phi & n_z n_y(1-cos\phi)+n_x sin\phi & (n_z^2-1)(1-cos\phi)+1 \end{pmatrix}$$

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,

$$X'' = (R'R)X + (R'T + T') \tag{6}$$

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．ここで元の octree を source tree，移動後を target tree と呼ぶことにする.

4.5.2 アルゴリズム

回転移動物体の octree 表現を更新するアルゴリズムは，次の通りである.

1. source tree を辿る．見つかった black leaf node それぞれに対して，次を実行する.
2. 回転と平行移動により，その node の新しい位置と向きを計算する．この位置がもし作業空間の外部であれば停止.
3. target tree の root から開始して，各 node を表す cube と，上で移動した cube との重なり (intersection) を調べることにより，target tree の node を生成する.

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．（図 5参照） 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.

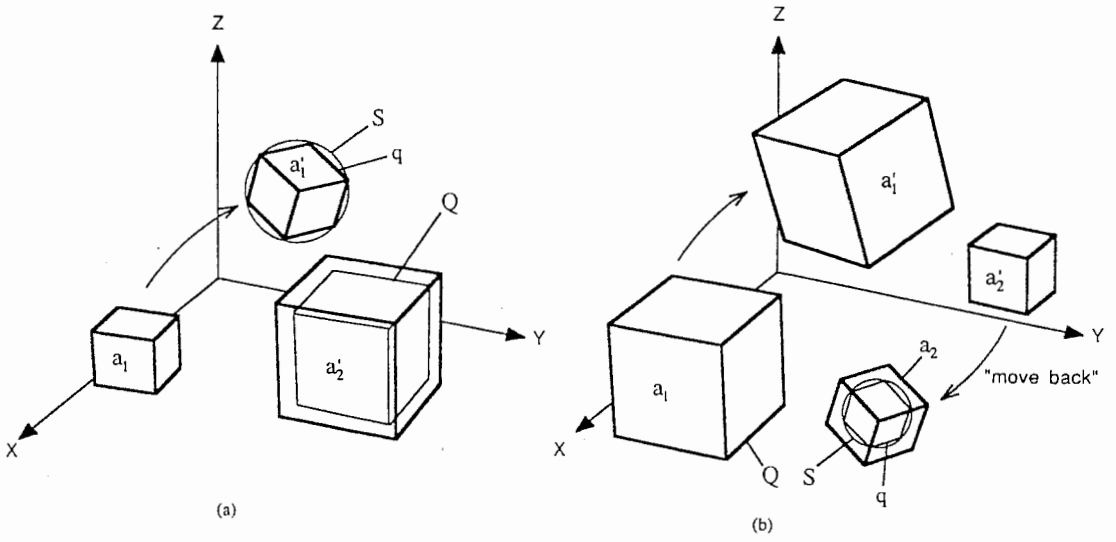

図 5: 回転移動に伴う tilted cube と upright cube の関係

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，

$$disa = \frac{\sqrt{3}\,2^{l_1} + 2^{l_2} - 1}{2} \tag{7}$$

$$disb = \frac{2^{l_1} + \sqrt{3}(2^{l_2} - 1)}{2} \tag{8}$$

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（図 5(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（図 5(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 辺の長さの対数としたとき，$O(Kl)$ 以下となることが報告されている．

```
PROCEDURE fdleaf(node);
BEGIN
        IF black(node) THEN
                IF out-of-space THEN error-stop
ELSE intersect(node)
        ELSE IF gray(node) THEN
                FOR each child of node DO
                fdleaf(child)
END;

PROCEDURE intersect(sourcenode, targetnode);
BEGIN
        IF l1 < l2 THEN
                BEGIN
                IF partial(targetnode, sourcenode) THEN
                        FOR each child of target node DO
                        intersect(sourcenode, child) END
        ELSE { l1 ≥ l2}
                IF l2 = 0 THEN
                        IF center of targetnode in inside sourcenode
                        THEN color targetnode black
                        ELSE color targetnode white
                ELSE
                        IF inside(targetnode, sourcenode)
                        THEN color targetnode black
                        ELSE IF partial(targetnode, sourcenode) THEN
                                BEGIN FOR each child of targetnode DO
                                        intersect(sourcenode, child);
                                        condense(targetnode)
                                END
END;
```

図 6: octree 更新アルゴリズム

4.5.3 インプリメント

5 つのサブルーチン **fdleaf**, **intersect**, **condense**, **black**, **gray**, から構成される.

fdleaf は source tree T_1 を leaf から逆に辿る. 発見された leaf node 各々に対して, 回転と平行移動の変換後の node 位置が作業空間の外部になるかどうかを調べる. もし外部なら error を発する. そうでない場合には **fdleaf** は, この leaf node に対して **intersect** を呼ぶ.

intersect は T_2 の root から始めて, 前節の intersection test をすることにより, **fdleaf** によって与えられた leaf node に相当する node を T_2 に加える. もし前節の intersection test の結果が "partial" であれば, 8 つの children node が生成される. **intersect** は, これら children それぞれに対して再帰的に呼び出され, その後直ちに **condense** が呼ばれる. **condense** はこれら 8 兄弟が同じ色かどうかを調べる. もし同じ色なら, 全 children を削除し, これらと同じ色をその parent node に与える. **black** は, その node が black なら true を返す. 同様に, **gray** は, その node が gray なら true を返す.

図 6 にこれらの概要を示す.

5 おわりに

以上，仮想空間での協調作業における物体の管理方法について検討した．仮想空間中で，物体間の衝突時に反射や破壊などの物理法則を適用したり，物体の接触配置作業などを容易にするためには，仮想空間中の複数の物体の位置を管理し，物体間の衝突／干渉チェックする方法が必要である．そこでまずロボットシミュレーションにおける衝突／干渉チェックの手法を調査した．これらの手法は，(1) 多面体表現を用いた静的な干渉チェック，(2) 運動物体の軌道を用いた動的な衝突チェック，(3)octree を用いた静的な干渉チェック，に大きく分類することができた．これを踏まえて，仮想空間での協調作業における物体の管理する方法を検討した．その結果，octree と多面体の 2 種類の形状表現を両立させた干渉チェック法が有効であろうとの検討結果を得た．つまり，octree を用いた大まかな干渉チェックをした後の限定された面と稜線の組合せに対して，B-rep などの多面体による形状表現を用いて正確に干渉をチェックする方法を検討した．

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, Narendra and Nash, Charles. Octree representations of moving objects. *Computer vision, graphics, and image processing*, Vol. 26, No. 2, pp. 207–216, 1984.

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

[CA86] Chien, C. H. and Aggarwal, J. K. Volume/surface octrees for the representation of three-dimensional objects. *Computer vision, graphics, and image processing*, Vol. 36, No. 1, pp. 100–113, 1986.

[Cam85] S. Cameron. A study of the clash decision problem in robotics. In *International Conference on Robotics and Automation*, pp. 488–493. IEEE, 1985.

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

[Con84] Connolly, C. Cumulative generation of octree model from range data. In *First international conference on robotics*. IEEE, 1984.

[DT81] Doctor, L.J. and Torborg, J.G. Display techniques for octree-encoded objects. *IEEE Computer graphics and applications*, pp. 29–38, 1981.

[ER84] Esterling, D.M. and Rosendale, J.Van. An intersection algorithm for moving parts. In *NASA Symposium on Computer Aided Geometric Modeling*, pp. 119–123, 1984.

[Hay86] Hayward, V. 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, M. Fast three-dimensional collision-free motion planning. In *international conference on Robotics and Automation*, pp. 1056–1063. IEEE, 1986.

[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.

[Kun85] Toshiyasu Kunii. Generation of topological boundary representation fromn octree encoding. *IEEE Computer graphics and applications*, Vol. 5, No. 3, 1985.

[LC87] Lorensen, WE and Cline, HE. Marching cubes: A high resolution surface construction algorithms. *Comoputer Graphics*, Vol. 21, No. 4, pp. 163–169, 1987.

[LP82] T Lozano-Perez. Task planning. In *Robot motion: planning and control*. MIT Press, 1982.

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

[Mea82] D. Meagher. Geometric modeling using octree encoding. *Computer graphics and image processing*, Vol. 19, No. 2, pp. 129–147, 1982.

[SA90] Srivastava, Sanjay and Ahuja, Narendra. Octree generation from object silhouettees in perspective views. *Computer vision, graphics, and image processing*, Vol. 49, No. 1, pp. 68–84, 1990.

[Sam90a] Hanan Samet. *Applications of spatial data structures*. Addison-Wesley, 1990.

[Sam90b] Hanan Samet. *The design and analysis of spatial data structures*. Addison-Wesley, 1990.

[TK92] Takemura, Haruo and Kishino, Fumio. Cooperative work environment using virtual workspace. In *CSCW*, 1992.

[WA85] Weng, Juyang and Ahuja, Narendra. Octree representation of objects in arbitrary motion. In *CVPR, San Francisco*, pp. 524–529, 1985.

[WA87] Weng, Juyang and Ahuja, Narendra. Octree of objects in arbitrary motion : Representation and efficiency. *Computer vision, graphics, and image processing*, Vol. 39, No. 2, pp. 167–185, 1987.

[岡野 88] 岡野, 川辺, 嶋田. 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.

[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.

[重松 83] 重松, 沖野. 3-D 形状モデル間干渉問題の一解法. 精密機械, Vol. 49, No. 11, pp. 89–94, 1983.

[小沢 86] 小沢, 熊本, 明石, 中田. オフラインロボット教示における高速干渉チェックの一 方式. 日本ロボット学会誌, Vol. 4, No. 2, pp. 3–14, 1986.

[新井 92] 新井, 太田. 複数移動ロボット系の計画. 日本ロボット学会誌, Vol. 10, No. 4, pp. 444–449, 1992.

[水垣 84] 水垣, 木村, 佐田. 幾何モデルによるロボットの簡易干渉チェック. 精密機械, Vol. 50, No. 6, pp. 944–950, 1984.

[登尾 87] 登尾, 福田, 有本. 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.

[尾崎 82] 尾崎, 毛利, 高田. マニピュレータの占有空間を考慮した障害物回避動作の決定法. 計測自動制御学会論文集, Vol. 18, No. 9, pp. 942–949, 1982.

[尾崎 84] 尾崎. マニピュレータの障害物回避. 日本ロボット学会誌, Vol. 2, No. 6, pp. 76–82, 1984.

[木村 88] 木村文彦. ロボットシミュレーションのためのグラフィック技法. 日本ロボット学会誌, Vol. 6, No. 2, pp. 130–135, 1988.

[有本 88] 有本卓. ロボットのシミュレーション. 日本ロボット学会誌, Vol. 6, No. 2, pp. 126–129, 1988.

[劉 89] 劉, 登尾, 有本. 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.