2002年度　卒　業　論　文

複数基準によるシリアライズを利用した衝突判定の高速化アルゴリズムに関する研究

指導教員：渡辺　大地

メディア学部リアルタイム3DCGカーネルプロジェクト

学籍番号　99p096

大森　祐

２００３年3月

2002 年度　卒 業 論 文 概 要

論文題目

複数基準によるシリアライズを利用した
衝突判定の高速化アルゴリズムに関する研究

メディア学部 **学籍番号:** 99p096	**氏名**	大森　祐	**主査**	渡辺　大地
			副査	和田　篤

キーワード	３ＤＣＧ、衝突判定、シリアライズ

リアルタイム３ＤＣＧにおいて物理シミュレーションなどを実現するには、できるだけ高速な衝突判定が求められる。最もシンプルな衝突判定の手法は、すべての物体について１つ１つ判定していく、総当たりによる方法である。しかし、この方法では、物体の数が増えるにしたがって、判定回数、処理時間の増加が大きくなる。これは、規模が大きなシステムや、３Ｄでの処理を考えると、少しでも判定にかかるコストを削減し、高速化を行う必要がある。

本研究では、物体の位置や大きさから得られるデータのシリアライズによって、衝突する可能性のある物体を検出し、判定回数を減らすことで、衝突判定の高速化を行った。高速化には２つのアルゴリズムについて実装した。また、それぞれの手法について処理時間と判定回数の測定を行い、効果の検証をした。

目次

1　序論

近年、ハードウェアの飛躍的な性能向上により、３次元コンピュータ・グラフィックス（３ＤＣＧ）を用いた３Ｄコンテンツが身近なものとなってきた。それに伴い、高度な物理シミュレーションエンジンを持ったソフトウェアの開発も著しい進歩を遂げている[1, 2, 3]。これらは、３ＤＣＧによって作成されたモデルに対して、質量、力、速度といった物理法則情報を与えることで、現実の物体と同じようなシミュレーションを簡単に実現することができる。

物体を扱うプログラムにおいて、しばしば突き当たるのが衝突判定の問題である。シミュレータなどは物体が接触しているかどうかを常に監視する必要があり、処理の負荷が大きい部分の１つと言える。そして、扱う物体の数が多いほど、また、物体の形状が複雑になるほど、衝突検知のための計算量は増加する。さらに、より精度の高い衝突検知も重要な課題である。

物体の形状が多角形や曲面など複雑なものとなれば、物体がどの部分で衝突しているかを検出する手間が増える。物体同士の接触点をいかに正確に検知するかが重要である。接触判定の方法の１つとして、物体と物体を仕切ることができる平面が存在するかどうかを調べるという手法が研究されている[4]。物体と物体との最短距離を求める手法についての研究も行われている[5]。布のように変形を伴う物体の衝突を扱った、クロスシミュレーション技術では、剛体の衝突判定と違って衝突のパターンが複雑であり、正確な衝突検知の方法など、研究がなされている[6,7,8]。また、扱う物体の数が数百、数千と増えていけば、それぞれが互いに衝突判定を行う回数は爆発的に増えていく。判定回数を削減する方法の１つに、空間をグリッドベースのマップに区切り、各マップ内に存在する物体同士で判定を行うという方法がある[9]。異なる大きさの物体を扱う場合は、正確な衝突判定を行う際に注意が必要だが、この方法では異なる分割度のマップを複数用意することで正確な判定を行うことができる。しかし、一部分に物体が密集するような状態では、ほとんどの物体が判定候補となってしまい、判定回数をうまく削減できない場合もある。衝突判定は、より大きなシステムになるほど深刻な問題であり、できる限りの高速な処理が求められる。

本論文では、衝突判定処理において、物体数の増加によるコストを削減し、高速化を実現する手法を提案する。判定回数の削減方法は、複数の指標に基づいたシリアライズを利用し、衝突の可能性のある物体だけで判定を行うものである。次いで、そのアルゴリズムについて実装したプログラムを用いて、衝突判定処理に要した時間と判定回数を測定した結果を示し、効果を検証する。

2　衝突判定について

2.1　衝突判定について

衝突判定は、エンターテインメントや物理シミュレーションなどで、複数の物体を扱う際に必要となる処理である。一般に、任意の３次元形状を持つ物体同士の衝突判定は手間がかかる。例えば、ポリゴンベースで構成されている物体を考える。どんな形状でも、三角形と三角形の衝突検出だけで判定できる[9]。任意の三角形同士の衝突検出は次のようにして行われる。

まず、２つの三角形のうち、一方の三角形（三角形１）が含まれる平面を求める。次に、もう一方の三角形（三角形２）を構成する頂点のうち、２つの頂点で定義される直線をとり、その直線が三角形１の平面と衝突するかを求める。衝突点が２つの頂点間にあるならば、三角形２は三角形１の平面と衝突していることになる。衝突点が２つの頂点間になければ、三角形２の別の２辺についても同様に調べていく。三角形２が三角形１の平面と衝突していることが分かったら、次は、その衝突点が三角形１の内部にあるかどうかを調べなくてはならない。３次元座標を平面に射影し、投影後の衝突点が、投影後の三角形１の内部に入っていれば、２つの三角形が互いに衝突していることになる。このような処理を、物体を構成する三角形すべてについて行う必要がある。物体のポリゴン数や物体数が増加していくと、計算量は膨大なものとなる。よって、可能な部分に対しては少しでも処理を軽減し、高速化を行っていく必要が出てくる。

2. 2　高速化へのアプローチ

一般に、衝突判定部分の高速化を図ろうとした場合、大きく２つのアプローチが考えられる。１つは、衝突判定に至るまでの過程において、無駄な衝突を回避するための、ある仕組みを用意する。そして、一定の手間をかけることで判定回数を減らす、というアプローチである。本研究では、このアプローチに沿った手法で判定回数を削減し、プログラムの高速化を試みた。

そして、もう１つは、衝突したかどうかの判定方法に手を加える、というアプローチである。複雑な形状を持つ物体の衝突判定を行う場合、その物体を単純な形状をした他の物体に置き換えて、衝突判定を行う。この方法は、判定の高速化にはつながるが、そのままでは精密さに欠けてしまう場合もある。他の物体に置きかえる方法としては、バウンディングスフィアやバウンディングボックスに置きかえる方法が代表的である[10]。

バウンディングスフィアとは、物体の中心から最も離れた点を半径とする外接球である（図１）。

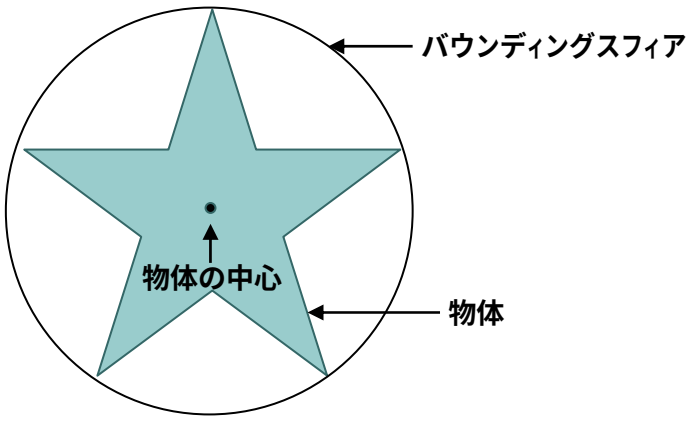


図１：バウンディングスフィア

バウンディングボックスとは、物体を内包する外接直方体である（図２）。

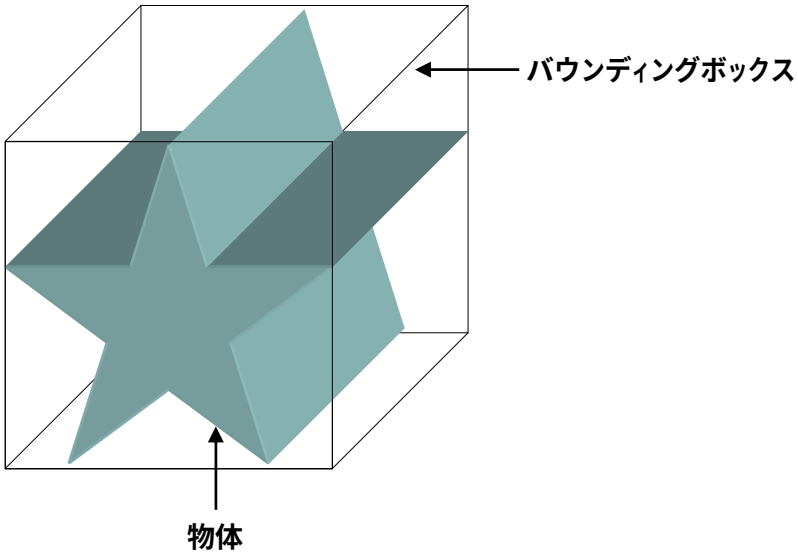


図２：バウンディングボックス

また、複雑な形状を、判定を行いやすい形状に近似させて処理コストを抑える方法も研究されている[11]。

2.3　総当たりによる衝突判定

衝突判定を行おうとしたとき、すべての物体について１つ１つ判定していく、という方法が考えられる。物体Ａから物体Ｄまで、物体の数が４個の場合に必要となる衝突判定の組み合わせは、（Ａ－Ｂ）、（Ａ－Ｃ）、（Ａ－Ｄ）、（Ｂ－Ｃ）、（Ｂ－Ｄ）、（Ｃ－Ｄ）の６つである。すべての物体の組み合わせについて調べていくので、全部で６回の衝突判定を行うことになる（図３）。以後、この手法を総当たり法と呼ぶことにする。

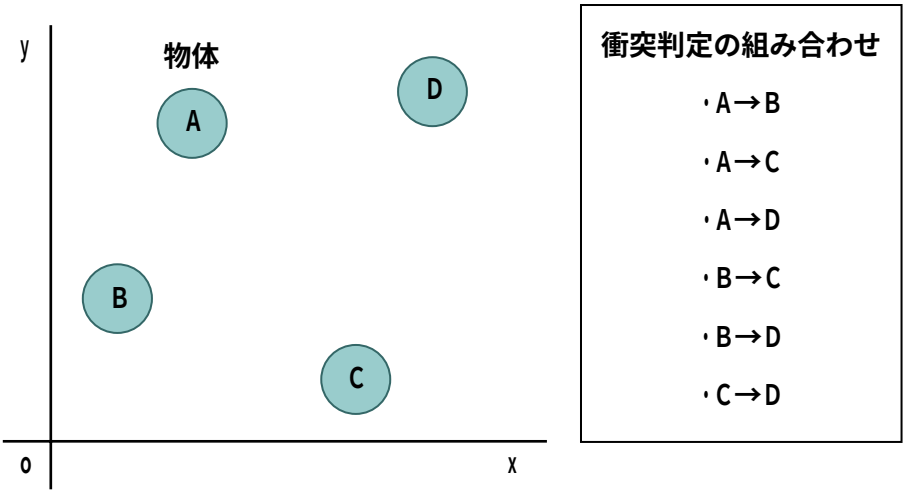


図３：総当たりによる判定

総当たり法のアルゴリズムは次の通りである。

● 物体はn個で、$O_i\left(i=1,2,3,\cdots,n\right)$とする

ある物体O_iについて、

１．$i<j\leq n$を満たすような、O_iとO_jについて衝突判定を行う

２．これをすべての物体について行う

総当たりの判定回数は、n 個の物体の中から 2 個の物体を選び出す、組み合わせの問題と考えられるので、${}_nC_2$ と表わすことができ、

$$ {}_nC_2 = \frac{n(n-1)}{2} $$

$$ {}_nC_2 = \frac{n^2}{2} - \frac{n}{2} $$

となる。すなわち、判定回数は

$$ \frac{n^2}{2} - \frac{n}{2} $$

となり、総当たり法の計算量のオーダーは、

$$ O(n^2) $$

である。

任意の形状の、物体 1 対の衝突判定にかかる時間を平均T とすると、全体では

$$ \frac{n(n-1)}{2} \times T $$

となる。ここで、物体のバウンディングスフィアを考えて、まず球体同士の衝突検知を行い、衝突しないと分かればそこで判定を止め、衝突の可能性があれば、さらに詳しく衝突検知を行うことにする。球同士の衝突判定にかかる時間を平均T_s、衝突の可能性があるとされたペアの数をk とすると、バウンディングスフィアを利用した衝突判定時間は、

$$ \frac{n(n-1)}{2} \times T_s + kT $$

となる。k が $\frac{n(n-1)}{2}$ に比べて小さく、T_s がT に比べて非常に小さい場合、このような手順を取ることで衝突判定の高速化を行うことができる。本研究の目的は、$\frac{n(n-1)}{2}$ の部分を小さくすることで、さらに判定時間を削減することである。

3　シリアライズによる衝突判定の高速化

3.1　データのシリアライズ

ある空間に複数の物体があるとする。そしてプログラム中では、物体個々の位置や速度といった情報は配列によって管理していることとする。多くの場合、物体の位置情報が格納された配列内での順序は、空間的な位置関係とは関連がない。この中から、ある１つの物体と衝突している物体を探すとき、何の手がかりもない状態であれば、配列を１つ１つ調べていくしか方法はない。

例えば、同じ大きさの物体が横一直線上に並ぶように分布していたとする。この状況で、ある物体と衝突している物体を探すとき、該当する可能性は隣り合っている物体が最も高い。隣り合う物体の情報をすぐに手に入れられるのならば、無駄な処理をせずに済む。しかし、物体の位置情報が格納されている配列において、隣のポインタに格納されている物体が、必ずしも空間的に近くに存在する保証はない。そこで、配列内のデータを、座標値の小さい順に並べ替える。すると、空間内で隣り合っている物体の情報が、配列においても隣に格納されることになり、衝突の可能性が高い物体に対して、効率よく判定を行うことができる。

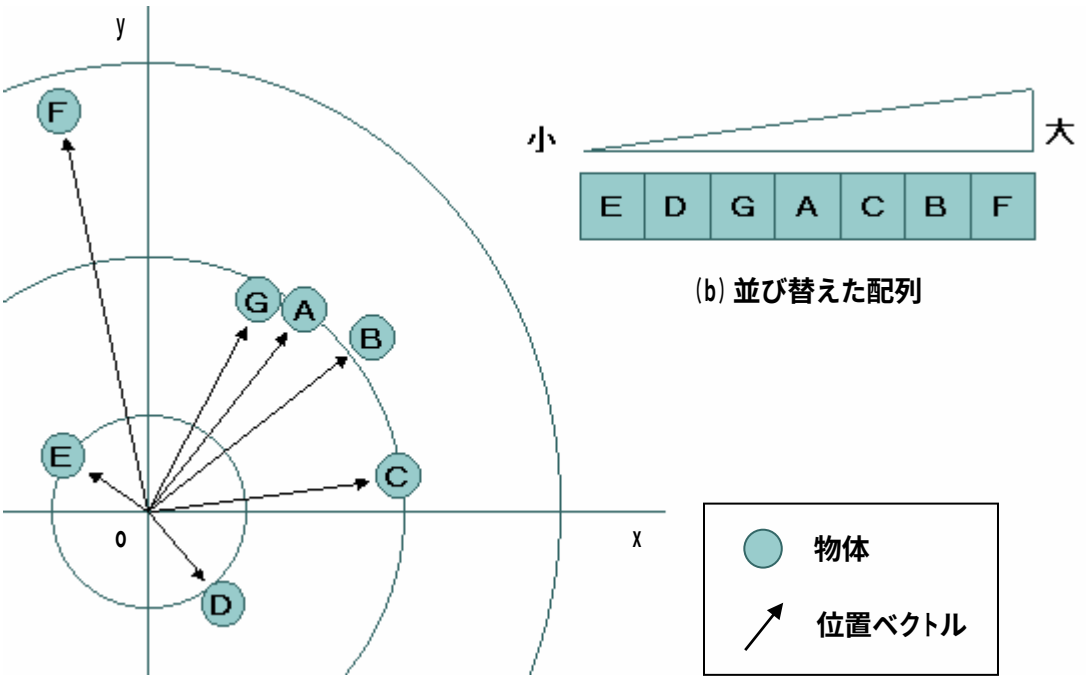
このように、欲しい情報に対して無作為に並んでいるデータを並び替え、整列させることで、複雑な問題を解きやすくすることができる[12]。

データの整列化、つまり、シリアライズを利用して、衝突判定の高速化を行う手法を次節以降で述べる。

3. 2　手法１：原点からの距離を指標とするシリアライズ

3. 2. 1　概念

この手法は、原点からの距離を指標として整列させる手法である。衝突する可能性のある物体同士は、空間的に接近しており、位置ベクトルもほぼ等しい。よって、原点からの距離もかなり近い値となる。すべての物体 n 個について、原点からの距離を求め、長さ n の配列に格納する。それを昇順に並べ替える。これで、距離の近い物体同士が、配列中でも近くに集まることになる。ここで注意すべき点は、原点を中心とした同心円上にある物体すべてが配列中で近くに集まるという点である。図４に物体の分布状態と、並べ替えた配列の例を示す。物体Ａ－Ｂ間より、物体Ａ－Ｃ間のほうが空間的に離れているが、並べ替えの指標は原点からの距離であるので、同心円上にあるＣのほうが配列内においてＡの近くに配置される。



(a) 物体の分布　　(b) 並び替えた配列

図４：原点からの距離を基準に整列

原点からの距離を基準に整列させたら、次にその距離の差を確認する。物体O_iの原点からの距離をD_i、バウンディングスフィアの半径をr_iとする。物体O_iと物体O_jとの距離をd_{ij}とすると$d_{ij} \geq \left|D_i - D_j\right|$であり、また$d_{ij} > r_i + r_j$を満たす場合、$O_i$と$O_j$は衝突しない。

よって、$\left|D_i - D_j\right| > r_i + r_j$を満たす場合、$O_i$と$O_j$は衝突しないことが分かる。

最大のバウンディングスフィアの半径を$r_{\max}$とすると、$\left|D_i - D_j\right| > 2r_{\max}$を満たす場合、$O_i$と$O_j$は衝突しない。したがって、$\left|D_i - D_j\right| \leq 2r_{\max}$を満たす場合のみ、衝突判定を行えばよい。さらに、$D_i$は昇順に並んでいるから、$\left|D_i - D_{i+h}\right| \leq \left|D_i - D_{i+h+1}\right|$が保証される。ここで、$h$は正の整数である。よって、$\left|D_i - D_{i+h}\right| > 2r_{\max}$を満たす$h$が検出された場合、$D_{i+h+1}$以降の値を参照する必要はない。

図５に具体的な例を示す。物体Ａについて衝突判定を行うとき、並び替えた配列から、Ｇ→Ｄ→Ｅ、Ｃ→Ｂ→Ｆの順で、原点からの距離の差を確認する。この状況で詳しい衝突判定が必要な物体は、Ｇ、Ｃ、Ｂの３つである。

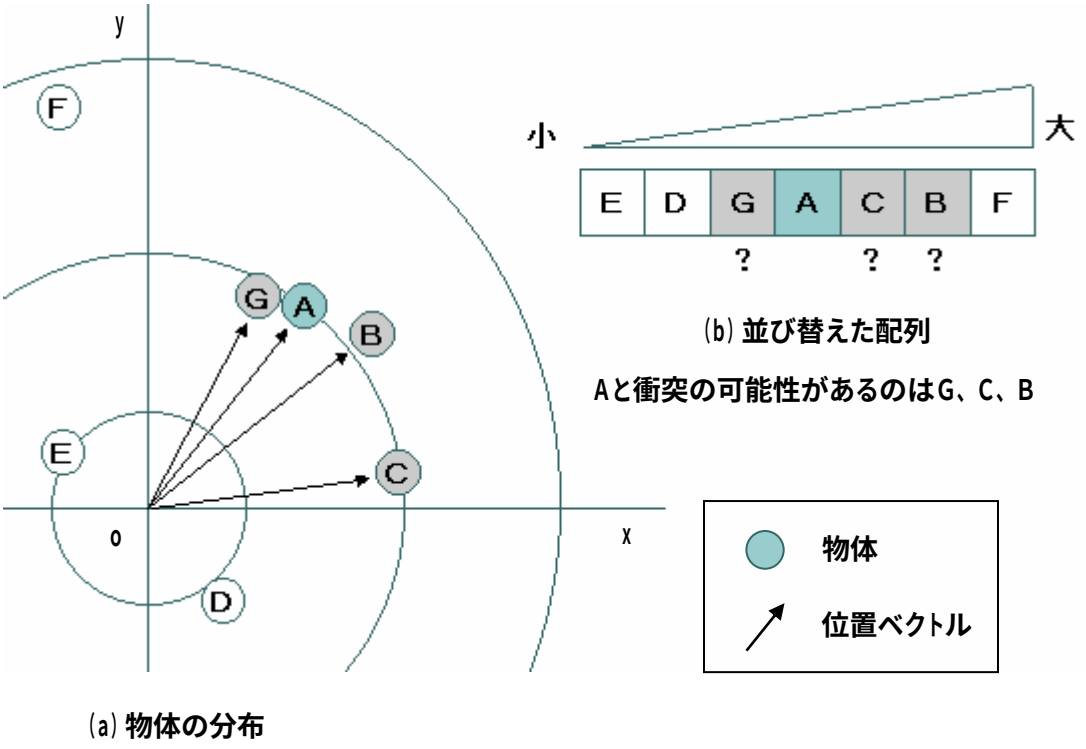


図５：距離の値が近いものだけ判定

原点からの距離を指標とする手法のアルゴリズムは次の通りである。

● 物体は n 個で、$O_i\left(i=1,2,3,\cdots,n\right)$ とする

１．最大のバウンディングスフィアの半径 $r_{\max}$ を設定する

２．原点からの距離 D_i を計算する

３．D_i を基準に、物体を昇順に並べ替える

４．ある物体 O_i について、次の処理を行う

Ⅰ．$O_{i+h}\left(h=1,2,3,\cdots\right)$ との距離の差を、D_i と D_{i+h} から求める

Ⅱ．差が $2r_{\max}$ 以下ならば衝突判定を行う

3. 2. 2　計算量

次に、この手法についての計算量を考える。

まず、原点からの距離を基準に並べ替える作業の計算量は$O(n\log n)$である。その後、衝突の可能性のある物体について衝突判定を行っているが、どのくらいの数に絞れているかを決めるのは難しい。ある物体について判定対象となる物体が平均k個とすると、計算量は

$$O(n\log n + kn)$$

ただし、$2 \le k < n$

であり、$k < \log n$ならば、計算量は

$$O(n\log n)$$

となり、$k > \log n$ならば、計算量は

$$O(kn)$$

となる。物体が空間内に偏りなくばらついているときに、kの値はnに比べて非常に小さくなるはずである。

3.3　手法２：ｘ、ｙ、ｚ軸を指標とするシリアライズ

3.3.1　概念

２つ目の手法について説明する。

この手法は、物体をｘ、ｙ、ｚそれぞれの座標値について整列させ、衝突する可能性のある集団の中で判定を行う、という手法である。物体 O_i に (sx_i, sy_i, sz_i), (ex_i, sy_i, sz_i), (sx_i, ey_i, sz_i), (sx_i, sy_i, ez_i), (ex_i, ey_i, sz_i), (ex_i, sy_i, ez_i), (sx_i, ey_i, ez_i), (ex_i, ey_i, ez_i) で構成されるバウンディングボックスを考えて処理を行う（図６）。ただし、 $sx_i + \Delta x_i = ex_i$, $sy_i + \Delta y_i = ey_i$, $sz_i + \Delta z_i = ez_i$, $\Delta x_i, \Delta y_i, \Delta z_i > 0$ である。

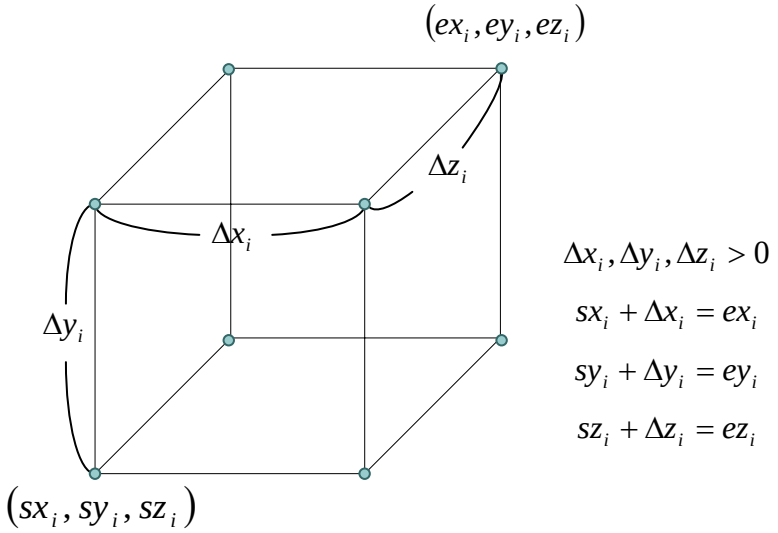


図６：バウンディングボックス

まず、すべての物体 n 個について、sx_i, ex_i を長さ $2n$ の配列に格納し、昇順に並べ替える。配列を順に見ていき、ある物体 O_i の sx_i があり、その物体の ex_i が見つかるまでに sx_j が検出された場合、物体 O_j はｘ軸から見て、物体 O_i と重なり合っており、衝突の可能性があることになる。逆に、sx_i と ex_i が配列内で隣り合っている場合、O_i から見て重なり合う物体は存在しない。

ｘ軸について整列し、どれとも衝突しないと分かった物体を判定対象からはずしたら、次に、残った物体に対してそれらの sy_i, ey_i を求め、ｙ軸について整列させる。同様に処理して、ｙ軸上で重なり合う物体を確認したら、それらをｚ軸について整列させ、最終的に残った物体について衝突判定を行う。

ｘ軸について重なり合う物体の例を図７に示す。重なり合っている物体は O_1 と O_2、O_4 と O_5 である。物体 O_3 は独立しており、判定候補からはずすことができる。残りの４つの物体については、次にｙ軸、ｚ軸と順に処理を続ける。

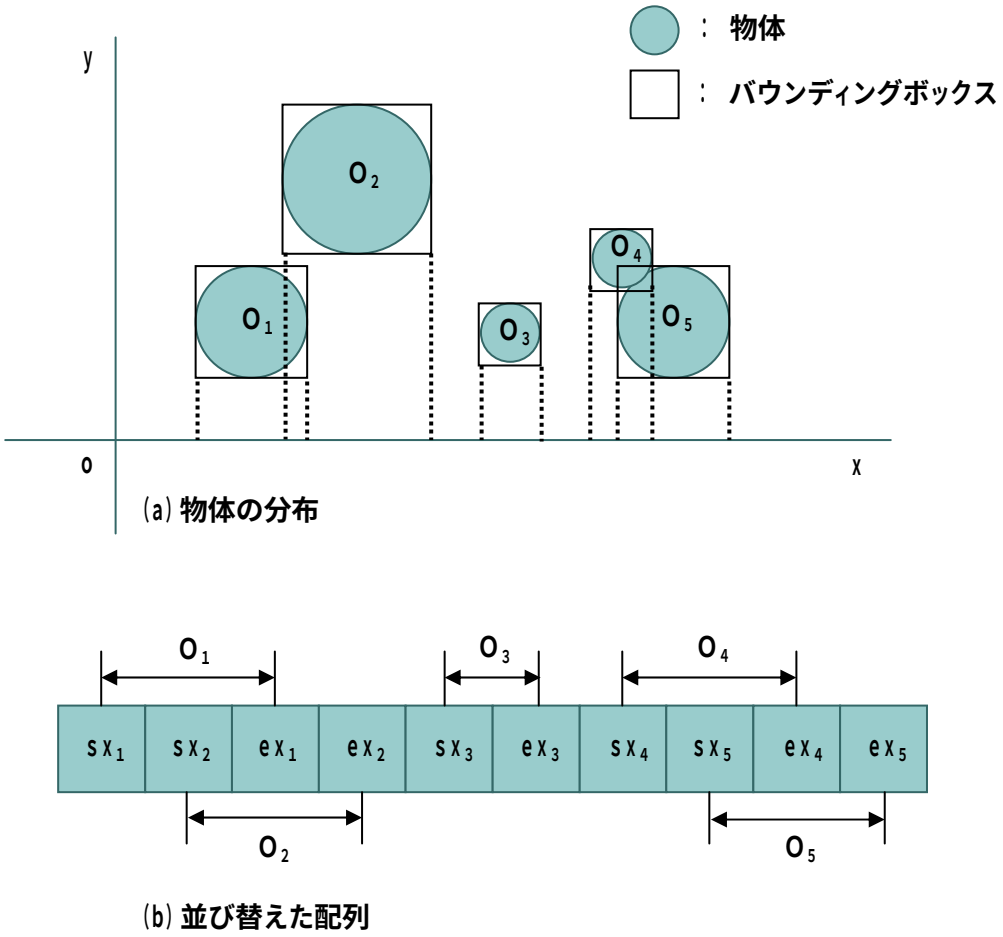


図７：衝突の可能性がある物体は重なり合う

ｘ、ｙ、ｚ軸を指標とする手法のアルゴリズムは次の通りである。

● 物体は n 個で、$O_i\left(i=1,2,3,\cdots,n\right)$ とする

１．sx_i と ex_i を求める

２．長さ $2n$ の配列 $\hat{x}$ を用意し、これに sx と ex を格納する

３．$\hat{x}$ を昇順に並べ替える

４．$\hat{x}_j$ について、次の処理を行う

Ⅰ．$\hat{x}_j = sx_i, \hat{x}_{j+1} = ex_i$ であれば、O_i は判定対象のリストに加えない

Ⅱ．$\hat{x}_j = sx_i, \hat{x}_{j+h} = ex_i$ のとき、$\hat{x}_{j+s}$ に格納されている物体を判定対象のリストに加えていく $\left(0 < s < h\right)$

※格納されている値の中に、同じ値が含まれることもあるため、実装時には注意が必要である

５．リストに加えられた物体について、sy_i と ey_i を求め、同様に処理する

６．リストに加えられた物体について、sz_i と ez_i を求め、並び替える

７．$\hat{z}_j = sz_i$ のとき、O_i について、次の処理を行う

Ⅰ．$\hat{z}_{j+h}\left(h=1,2,3,\cdots\right)$ に格納されている物体との衝突判定を行う

Ⅱ．$\hat{z}_{j+h} = ez_i$ となるまで続ける

3. 3. 2　判定候補の最も少ない座標軸を選択

ある物体に対して衝突しそうな物体の数が、ｚ軸から見た場合よりｙ軸から見た場合のほうが少なかったときは、ｙ軸について判定を行ったほうが、処理が少なくて済む。そこで、まず始めに、ｘ、ｙ、ｚ軸すべてに対して昇順に整列させてしまい、物体がそれぞれいくつの他の物体と衝突する可能性があるのかを調べておくことにする。そして、衝突判定を行う際に、最も判定候補の少ない座標軸について判定を行うようにすれば、全体的な判定回数はさらに削減できるはずである（図８）。この方法についても実装し、測定の対象とした。

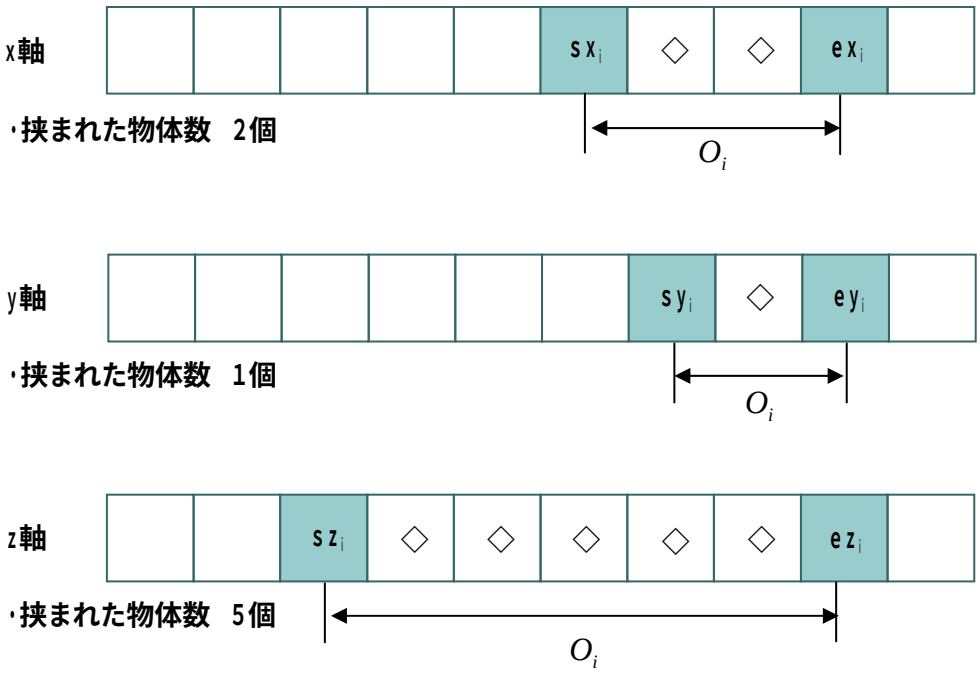


図８：判定候補の最も少ない軸を選ぶ

（この場合は、ｙ軸について判定を行う）

この手法についての計算量も、原点からの距離を指標とする手法と同じように考えることができる。つまり、並べ替えの計算量が$O(n\log n)$であり、ある物体について判定対象となる物体が平均k個として、

$$O(n\log n + kn)$$

ただし、$0 \le k < n$

である。さらに、ｘ、ｙ、ｚ軸の中から、最も判定候補の少ないものを選ぶ場合は、それぞれの判定候補数k_x, k_y, k_zの中での最小値の平均を$k_{\min}$とすると、

$$O(n\log n + k_{\min} n)$$

ただし、$0 \le k_{\min} \le k$

となる。

4　測定結果

前章で述べた２つの手法について実装し、衝突判定に要した時間と判定回数の測定を行った。

4.1　測定環境

測定は、以下のような環境で行った。

ＣＰＵ：Pentium4　1.50GHz
メモリ：512MB
ＯＳ：Windows2000

言語は、C++を使用し、Microsoft Visual C++でコンパイルした。

4. 2　測定条件

測定では、次のような条件を設定した。

物体は、幅 100、奥行 100、高さ 100 の閉じた３Ｄ空間の中を、跳ね返りながら動き回る。物体には初期速度を与え、跳ね返りによる速度ベクトルの変化以外では、力を加えない。初期速度(v_x, v_y, v_z)は、$-0.5 \leq v_x, v_y, v_z \leq 0.5$となるように、物体ごとにランダムに設定した。

本研究の目的は、衝突に至るまでの過程において、判定対象の削減と高速化を図ることであるので、物体の形状は球体を使用した。球体同士の衝突判定は、単純なアルゴリズムで求められる。２つの球体間の距離と、それぞれの球体の、半径の和を比較することで、衝突を判定することができる。

閉じた空間でのシミュレーションを行うため、空間に対して物体が大きすぎると、物体の数を増やしたときに空間内に入りきらない可能性がある。物体の数を最大 5,000 個まで増やすことを想定し、球体の半径を 2.0 とした。初期位置は、空間内にランダムに配置し、特に偏りのない分布状況で測定した。

●測定条件

１．幅 100、奥行 100、高さ 100 の閉じた３Ｄ空間

２．物体はすべて半径 2.0 の球体

３．物体の数は 50～5,000 個

４．物体の初期位置：ランダムに配置

５．物体の初期速度：(v_x, v_y, v_z)

※$-0.5 \leq v_x, v_y, v_z \leq 0.5$

※v_x, v_y, v_zは物体ごとにランダムに設定

4. 3　実行画面

図９は、プログラムの実行画面である。

実際には、半径 2.0 の球体でシミュレーションを行っているが、分かりやすいように大きさを変えてある。半径 10.0、物体の数 10 個の様子である。

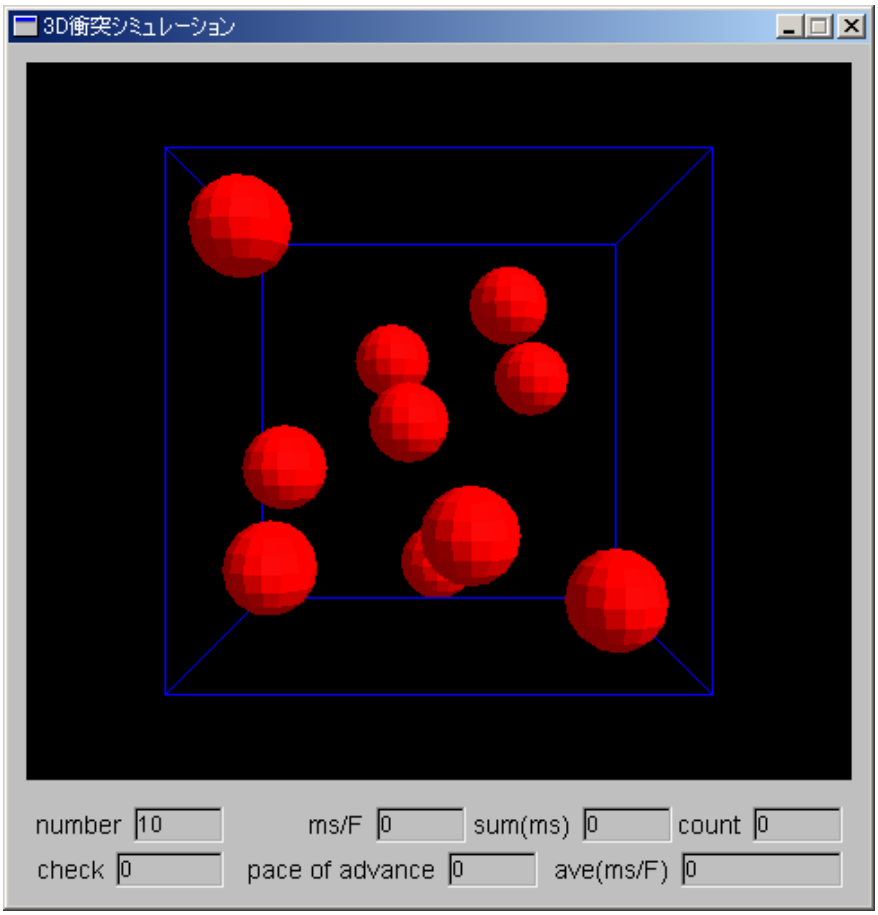


図９：実行画面

4.4　測定１：３つの手法の判定処理時間

以上の条件のもとで、衝突判定処理に使用した時間の測定を行った。得られた数値は、すべての物体が一通りの衝突判定を終えるまで（1 フレーム）の処理を 100 回繰り返した時間である。タイマーの精度は 10ms である。

総当たり法、原点からの距離を指標とする手法、ｘ、ｙ、ｚ軸を指標とする手法についての測定結果を表１、図１０に示す。

表１：３つの手法の衝突判定処理時間

物体数	50	200	500	1,000	3,000	5,000
総当たり法	20	420	2,690	10,690	96,720	268,220
原点からの距離	10	100	570	1,990	17,140	47,540
x、y、z軸	30	220	690	2,440	12,130	32,190

単位：ms

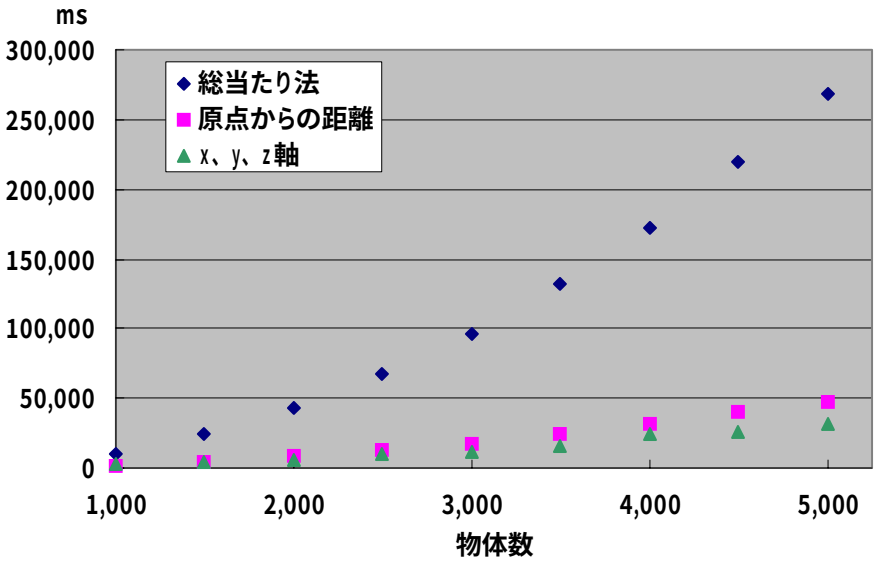


図１０：３つの手法の衝突判定処理時間

また、総当たり法の処理時間を 100 とした場合の、それぞれの手法の割合を図１１に示す。

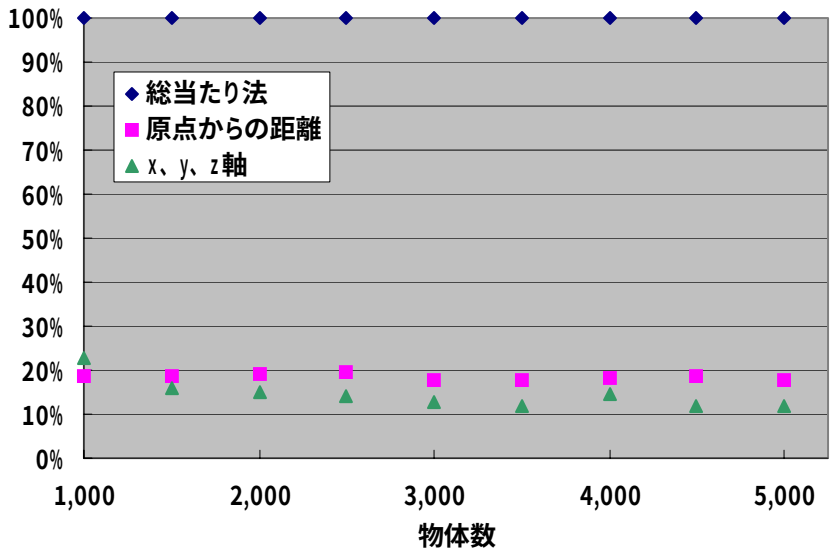


図１１：総当たり法の処理時間で正規化した判定処理時間

物体の数が 50 個未満の少ないうちは、総当たり法の速度は速かった。しかし、その差は数ミリ秒程度のもので、実用段階での不満は、どの手法でも感じられないと思われる。徐々に差が見え始めたのは、物体の数が 50 個を越えたあたりからであった。

物体の数が 1,000 個ぐらいまでは、原点からの距離を指標とする手法が最も速く、総当たり法に対して約 20%の処理時間で判定を行うことができた。また、物体数を増やしてもこの割合は変わることなく、安定していた。この手法については、総当たり法の２割程度のコストで処理できることが分かり、有効性が確認できた。

ｘ、ｙ、ｚ軸を指標とする手法については、物体の数が 1,000 個以上になってから、原点からの距離を指標とする手法を抜いて最も速くなった。こちらは、数が増えるにつれて速度上昇率も上がることが分かった。

4.5　測定２：３つの手法の判定回数

次に、総当たり法、原点からの距離を指標とする手法、ｘ、ｙ、ｚ軸を指標とする手法についての衝突判定回数の比較を行う。総当たり法については、それぞれの個数において回数は固定である。他２つの手法については、10 フレームの平均を取ったものである。測定結果を表２に示す。

表２：３つの手法の衝突判定回数

物体数	50	200	500	1,000	3,000	5,000
総当たり法	1,225	19,900	124,750	499,500	4,498,500	12,497,500
原点からの距離	214	3,209	20,996	83,700	752,273	2,103,731
x、y、z軸	86	1,635	10,156	40,382	365,599	1,020,451

単位：回

判定回数は、処理時間とは違い、数の少ないうちからはっきりとその差を確認することができる。物体の数が 5,000 個では、総当たり法約 1,200 万回、原点からの距離を指標とする手法約 200 万回、ｘ、ｙ、ｚ軸を指標とする手法約 100 万回であった。原点からの距離を指標とする手法は総当たり法の６分の１、ｘ、ｙ、ｚ軸を指標とする手法はさらにその半分の１２分の１となった。この関係は、数がいくつの場合でも変わることはなかった。

原点からの距離を指標とする手法に対する計算量 $O(n\log n + kn)$ についての、k の値について考える。物体数 5,000 個の場合の、物体 1 個あたりに必要な判定回数は 421 回であり、判定対象となる物体は物体総数の約 8.4%に削減したことになる。これは、他の物体数においても同じ割合であった。つまり、$k = n \times 0.084$ とすることができる。また、ｘ、ｙ、ｚ軸を指標とする手法についても同様に考えると、$k = n \times 0.041$ となる（表３）。

表３：k の値について

物体数（n）	50	200	500	1,000	3,000	5,000
k（原点からの距離）	4.280	16.045	41.992	83.700	250.758	420.746
k／n	0.086	0.080	0.084	0.084	0.084	0.084
k（x、y、z軸）	1.720	8.175	20.312	40.382	121.866	204.090
k／n	0.034	0.041	0.041	0.040	0.041	0.041

4.6　測定３：複数軸を扱う有効性について

ｘ、ｙ、ｚ軸を指標とする手法では、３段階にわたって、判定の必要のない物体を省き、衝突の可能性のある集団内で判定を行っているが、複数の軸を扱う有効性を確かめるために、ｘ、ｙの２つの軸について整列させる方法と、ｘ軸１つだけを整列させる方法についても実装し、衝突判定処理に使用した時間の測定を行った。得られた数値は、すべての物体が一通りの衝突判定を終えるまで（1 フレーム）の処理を 100 回繰り返した時間である。タイマーの精度は 10ms である。結果を表４に示す。

表４：指標とする軸の数を変えた場合の判定処理時間

物体数	50	200	500	1,000	3,000	5,000
x、y、z軸	30	220	690	2,440	12,130	32,190
x、y軸	20	140	530	1,640	11,220	28,620
x軸のみ	10	90	440	1,290	9,640	27,040

単位：ms

物体の数に関係なく、指標とする軸の数を減らしていくほど処理時間が短くなった。それぞれについて判定回数を調べてみると、どの手法もほとんど同じ値を出しており、目立った差は見られなかった。ｘ軸、ｙ軸、ｚ軸と順に衝突しない物体を判定対象からはずしていったつもりであったが、効果がなかったことが分かった。

判定回数を削減できなかったのは、物体が空間内にほぼ偏りなく位置していたため、どの座標軸側から見ても同じ数だけの物体がそれぞれ入れ違いながら重なり合ってしまったこと、そして、この状況は物体の数が増えるにしたがってより多く発生してしまうことが原因と考えられる。物体の分布状況によっては、今回とは違った結果が得られるかもしれない。物体数が 50 個未満の少ない状況では、指標とする軸の数が多いほうが判定回数をより削減することができたが、整列させる手間を考えると、１つの軸のみの処理で十分だということが確認できた。

4.7　測定４：判定候補の最も少ない軸を選んだ場合について

ｘ、ｙ、ｚ軸を指標とする手法に対して、最も判定候補が少ない座標軸を選んで衝突判定を行うようにした場合についての判定処理時間を測定した。得られた数値は、すべての物体が一通りの衝突判定を終えるまで(1 フレーム)の処理を 100 回繰り返した時間である。タイマーの精度は 10ms である。測定結果を表５、図１２に示す。

表５：判定候補の最も少ない軸を選んだ場合の判定処理時間

物体数	50	200	500	1,000	3,000	5,000
x、y、z軸	30	220	690	2,440	12,130	32,190
x、y、z軸から選ぶ	30	190	710	1,870	11,760	29,780

単位：ms

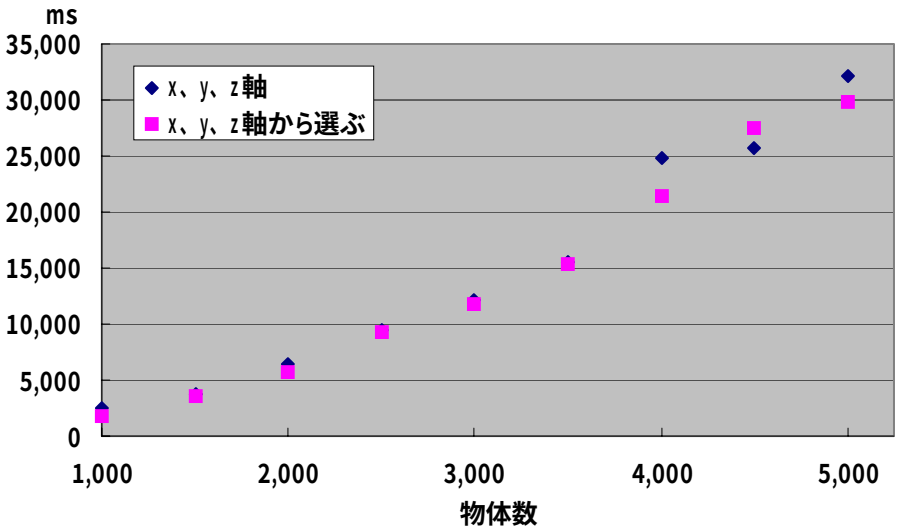


図１２：判定候補の最も少ない軸を選んだ場合の判定処理時間

アルゴリズムの改善により、若干ではあるが処理時間を縮めることができた。ただし、物体数 500 個のとき、4,500 個のときに、速度が逆転してしまっている。このような状況は他の数においても何度か見られた。おそらく、コーディングの仕方の違いによって、ＣＰＵでの計算の過程で、ある部分が最適化されて速度が逆転したのではないかと考えている。

また､判定回数の測定結果を表６に示す｡数値は 10 フレームの平均を取ったものである。

表６：判定候補の最も少ない軸を選んだ場合の判定回数

物体数	50	200	500	1,000	3,000	5,000
x、y、z軸	86	1,635	10,156	40,382	365,599	1,020,451
x、y、z軸から選ぶ	54	1,267	8,605	35,661	336,476	951,047
削減率	37%	23%	15%	12%	8%	7%

単位：回

判定候補の少ない座標軸を選ぶことによって、判定回数削減の効果が確認できた。物体数 5,000 個では 100 万回をきることができた。削減率としては、数の少ないうちは高く、数が増えていくにしたがって低くなっていった。この理由は、先にも述べたように、数が少ないうちは分布密度が低く、衝突しない物体を効率よく省くことができるが、多くなると、ある座標から見て重なり合う物体が増えてしまい、候補からはずせる物体が減ってしまうことが原因と考えられる。物体数 1,000～5,000 個の間では、ある物体について判定対象となる物体は物体総数の約 3.7%になっており、計算量中の $k_{\min}$ の値は、$n \times 0.037$ となった（表７）。

表７：k、$k_{\min}$ の値について

物体数 (n)	50	200	500	1,000	3,000	5,000
k (x、y、z軸)	1.720	8.175	20.312	40.382	121.866	204.090
k／n	0.034	0.041	0.041	0.040	0.041	0.041
kmin	1.080	6.335	17.210	35.561	112.159	190.209
kmin／n	0.022	0.032	0.034	0.036	0.037	0.038

ｘ、ｙ、ｚ軸の順にそのまま判定候補を削減していった場合の判定候補数kの値は$k = n \times 0.041$、ｘ、ｙ、ｚ軸の中から最も判定候補の少ない座標軸を選んだ場合の判定候補数$k_{\min}$の値は$k_{\min} = n \times 0.037$であった。係数の差は0.004と小さいものであった。物体の分布条件を変えた場合、すべての物体が一点に集中しているような極端な状況を除けば、今回のように一様に分布している状況での結果を下回ることはないと考えられる。

5　結論

本研究では、３ＤＣＧを用いた物理シミュレーションなどでしばしば問題となる衝突判定処理に対して、データのシリアライズを利用して高速化を行った。シリアライズの指標には、物体の原点からの距離、バウンディングボックスを考慮したｘ、ｙ、ｚ軸の両端座標、を使用した。

原点からの距離を指標とする手法では、総当たり法の 2 割程度のコストで処理することができた。判定回数は、総当たり法の６分の１に削減できた。ある物体について判定対象となる物体は物体総数の約 8.4%となった。

ｘ、ｙ、ｚ軸を指標とする手法では、物体の数が増えるにしたがって、より高い効果が確認できた。こちらも、総当たり法の２割以下のコストで処理することができた。判定回数は、総当たり法の１２分の１に削減できた。ある物体について判定対象となる物体は物体総数の約 4.1%となった。また、３つの座標軸の中で最も判定候補の少ない軸を選ぶように変更したプログラムでは、さらに削減することができ、物体総数の約 3.7%となった。

今回の測定では、k は n に比例する結果になったが、実際には物体の分布密度に依存するはずである。空間の広さと物体の大きさを固定したため、物体数の増加によって分布密度も高くなり、$O(n\log n + kn)$での k の値が n に依存する形となった。物体の分布密度を固定した状況で測定を行えば、物体数に影響を受けない結果が期待できると考える。

謝辞

本研究を進めるにあたり、終始温かいご指導をいただいた渡辺大地先生、演習講師であり本論文の副査を引き受けていただいた電気通信大学の和田篤氏に心より感謝致します。

研究生活において多々、お世話になりましたすべての方々に、厚く御礼申し上げます。

参考文献

[1] Macromedia, Macromedia Director 8.5 Shockwave Studio, http://www.mac romedia.com/, 2002

[2] MAXON, CINEMA 4D Dynamics, http://www.maxon.net/, 2002

[3] Autodesk, 3ds max, http://www.autodesk.co.jp/, 2002

[4] David Baraff, Rigid Body Simulation, In SIGGRAPH 97 COURSE NOTES 19, Phisically Based Modeling:Prinsiples and Practice, 1997

[5] 川地克明, 鈴木宏正, 離散的ボロノイ領域を用いた非凸多面体間の最短距離計算手法, 精密工学会誌 Vol.67 pp.1782-1786, 2001

[6] David Baraff, Andrew Witkin, Large Steps in Cloth Simulation, Computer Graphics Proceeding SIGGRAPH '98, 1998

[7] Robert Bridson, Ronald Fedkiw, John Anderson, Robust Treatment of Collisions, Contact and Friction for Cloth Animation, Computer Graphics Proceeding SIGGRAPH '02, 2002

[8] Kwang-Jin Choi, Hyeong-Seok Ko, Stable but Responsive Cloth, Computer Graphics Proceeding SIGGRAPH '02, 2002

[9] Mark DeLoura 編, 川西裕幸 監訳, 狩野智英 翻訳, Game Programming Gems, 株式会社ボーンデジタル, 2001

[10] Game Gisp, Brown Computer Science, Collision Detection, http://www.cs.brown.edu/courses/gs007/, 2002

[11] 小堀研一, 中西大輔, 形状近似球群による高速な衝突判定の一手法, 電子情報通信学会論文誌 VOL.J85-DⅡ No.9 pp.1455-1463, 2002

[12] 浪平博人, データ構造とアルゴリズム, CQ 出版株式会社, 1991