エネルギー科学研究科

エネルギー社会・環境科学専攻修士論文

題目: 機器保修訓練環境のためのPCクラスタを用いた
剛体挙動シミュレーションの並列処理

指導教官： 吉川 榮和　教授

氏名： 社領　一将

提出年月日： 平成14年2月6日(水)

1

論文要旨

題目：機器保修訓練環境のための PC クラスタを用いた剛体挙動シミュレーションの並列処理

吉川榮和研究室　社領　一将

要旨：

近年、人工現実感技術を用いた機器保修の訓練環境が、効率的な訓練を実施するための 1 つの手法として注目を集めている。人工現実感技術を用いて、発電プラント等の大規模エネルギー機器の保修作業を体験できる環境を構築できれば、エネルギー機器の取り扱い上の安全性・信頼性の向上が期待できる。

このような訓練環境を実現する際、仮想物体の挙動を合成する手法として、主に Rule-base の手法と Physically-base の手法の 2 つの手法が用いられているが、より高い訓練効果を得るためには、この 2 つの特徴を同時に実現することが望ましいとされる。しかし、実際には Rule-base の手法に比べて Physically-base の手法はあまり用いられていないのが現状である。Physically-base の手法は、仮想物体の挙動を物理法則に基づいて厳密に計算する手法であるため、膨大な計算タスクをリアルタイムに処理する必要があり、現在の単一の計算機環境では実現が困難だからである。そこで本研究は、機器保修の訓練環境に PC クラスタを導入することを試みる。PC クラスタは、メモリ分散型の並列計算機であり、剛体挙動シミュレーションに必要な計算タスクを、ネットワークを介して分散し、複数の計算機を用いて並列処理することを可能にする。

本研究は、機器保修の訓練環境を実現する際に必要な Physically-base の手法に基づく剛体挙動シミュレーションの実現と、計算機上への実装を目的とする。さらに、実装したシミュレーションの処理の一部を、PC クラスタを構成する複数台の計算機に分散し並列処理する手法を開発し、現時点で容易に入手可能な計算機環境上でのリアルタイムシミュレーションの実現可能性の検討を目的とする。

具体的には、まず、剛体挙動のモデル化をおこない、単一の計算機環境上で動作する剛体挙動シミュレーションシステムを構築した。次に、剛体挙動シミュレーションの処理の一部を分散し並列処理する手法を開発し、PC クラスタ環境上で動作する剛体挙動シミュレーションシステムを構築した。さらに、現時点で容易に入手可能な計算機環境上でのリアルタイムシミュレーションの実現可能性を検討するため、同一の仮想空間のシミュレーションを、単一の計算機環境で処理する場合と PC クラスタ環境で処理する場合に要する処理時間を比較する実験をおこなった。

評価実験の結果、50 個程度のナットとボルトが床に向かって落下する例では、計算機を 2 台接続した環境で約 55%、4 台接続した環境で約 127% の高速化を実現し、分散・並列化の効果が PC クラスタ環境の性能に依存することを確認した。また、高負荷時の計算負荷の分散によりシミュレーション速度が安定するため、分散・並列化がリアルタイムシミュレーションの実現に有効であることを確認した。

目 次

図 目 次

表 目 次

第 1 章 序論

ほとばしる水しぶき、飛び散る火花、風に舞う髪、揺れる衣服。Playstation2 用ゲームソフト「ファイナルファンタジー X」は、実世界と区別が付かないほどのリアルなアニメーションを実現し話題を集めた。これらの表現は、流体力学や剛体力学に基づいて物体の挙動を厳密に計算しシミュレーションすることで実現されている。このような計算機を用いた物理シミュレーションは、よりリアルなアニメーションを合成できるだけでなく、合成に要する人間の手間を軽減することができるため、今や 3 次元コンピュータグラフィクスに欠かせない技術の 1 つとなっている。

近年の計算機技術の発達により、3 次元コンピュータグラフィクスをはじめとする人工現実感技術は、コマーシャル、映画、テレビ等、様々な分野で利用され、我々の生活に欠かせない技術となっている。そして、人工現実感技術がさらに発展し、物理シミュレーションを用いて現実と区別がつかないほどの仮想空間を実現することができれば、我々の生活を根本から変えてしまう程の効果があると期待されている。例えば、家から一歩も外に出ることなく、インターネット上に構築された仮想都市に集い、仕事やショッピングすることができるようになる。また、好きな時に好きなだけ世界各国を旅行し、世界中の人達と同じ仮想空間内で自由にコミュニケーションすることができるようになる。このような技術が実現すれば、エネルギー問題、CO_2 排出問題、過疎化問題といった現代の社会が抱えている様々な問題を解決することができると期待されている。

現実と区別がつかないほどの仮想空間を構築する技術が期待されているのは機器保修の訓練環境も同様である。人工現実感技術を用いた機器保修の訓練環境は、効果的な訓練方法の 1 つとして近年注目を集めているが、まだまだ自由度が低く実用的とはいえない。しかし、機器保修の訓練環境に物理シミュレーションを導入し、より現実に近い物体の挙動を合成することができれば、訓練生に実際の訓練に近い訓練環境を提示することが可能となり、訓練効果を大幅に高めることができる。

しかし、物理シミュレーションは、あらかじめ定められたパラメータを用いて物体の挙動を計算しアニメーションとして合成する技術として利用が進められているものの、ユーザが自由に操作を加え擬似体験できる 3 次元仮想空間内の仮想物体の挙動を制御する技術としてはあまり利用されていないのが現状である。その理由の 1 つとし

て、物理シミュレーションを実行する際、膨大な計算が必要であることが挙げられる。つまり、擬似体験が可能な仮想空間を実現するためには、ユーザの入力がリアルタイムに仮想空間へ反映される必要があるが、物理シミュレーションに必要な膨大な計算タスクを単一の計算機環境を用いてリアルタイムに処理することは困難だからである。そこで本研究では、物理シミュレーションにPCクラスタを導入することを試み、リアルタイムシミュレーションの実現可能性を検討することとした。PCクラスタとは、パーソナルコンピュータを用いたメモリ分散型の並列計算機であり、膨大な計算タスクを分散・並列処理することで高速化することが可能な計算機環境である。

以上により、本研究では、物理法則に基づいたリアルタイムシミュレーションを実現する第一段階として、より訓練効果が高い人工現実感技術を用いた機器保修の訓練環境の実現を目標に、既存の剛体挙動のモデル化手法を拡張する剛体挙動シミュレーションを実現し、計算機上へ実装することを目的とする。さらに、実装した剛体挙動シミュレーションの処理の一部を、PCクラスタを構成する複数台のノードマシンに分散し並列処理する手法を開発し、現時点で容易に入手可能な計算機環境上でのリアルタイムシミュレーションの実現可能性の検討と、実用化のための今後の指針を得ることを目的とする。

以下に、本論文の構成について述べる。まず、第2章では、本研究の背景と目的について述べ、本研究と従来研究の目標の相違を明確にする。第3章では、剛体挙動のモデル化手法について述べ、それを実装した単一の計算機環境で動作する剛体挙動シミュレーションシステムについて述べる。また、構築したシステムを用いて実施した剛体挙動シミュレーションの負荷計測実験について述べる。第4章では、分散・並列処理の定義や用語を説明し、剛体挙動シミュレーションの処理の一部を分散・並列化する手法について述べる。また、それを実装したPCクラスタ環境で動作する剛体挙動シミュレーションシステムについて述べる。第5章では、構築したシステムを用いて実施したシミュレーションの速度の評価実験について述べ、剛体挙動のリアルタイムシミュレーションを実用化するための指針を示す。最後に、第6章で本研究の結論を述べる。

第 2 章 研究の背景と目的

本章では、まず、人工現実感技術を用いた機器保修の訓練システムを実世界における訓練環境と比較した場合の利点と研究開発の現状について述べる。次に、近年の計算機の性能向上の傾向と、その計算機を用いた PC クラスタの実用化の現状について説明し、本研究の目的を述べる。最後に、PC クラスタを用いた分散・並列処理に関する従来研究について説明し、本研究の特徴について述べる。

2.1 人工現実感技術を用いた機器保修の訓練システム

近年、工学プラントが大規模・複雑化するとともに、機器保修作業の訓練の重要性が増大している。特に、原子力発電プラント等の大規模エネルギー機器の保修作業や運転作業の訓練は、エネルギー利用技術の安全性・信頼性の向上の観点からも重要視されている。その中で、次世代の訓練環境として人工現実感技術を用いた訓練 [1] が注目されている。仮想空間に訓練環境を構築し、訓練を実施する利点として、以下の点が挙げられる。

1. 事故が起こる可能性が低く非常に安全である。
2. 訓練を実施する際に必要な空間が比較的小さい。
3. 実世界では実現が困難な機器の故障等を容易に模擬できる。
4. 訓練の記録を残すのが容易である。
5. 訓練環境の複製が容易であり、同じ訓練環境を多数構築する場合に費用が安い。
6. 訓練環境が劣化や故障をしないので維持費用が安い。
7. 訓練環境の管理・輸送が容易である。
8. 訓練生を支援するための各種機能を付加することで訓練効果を高めることができる。

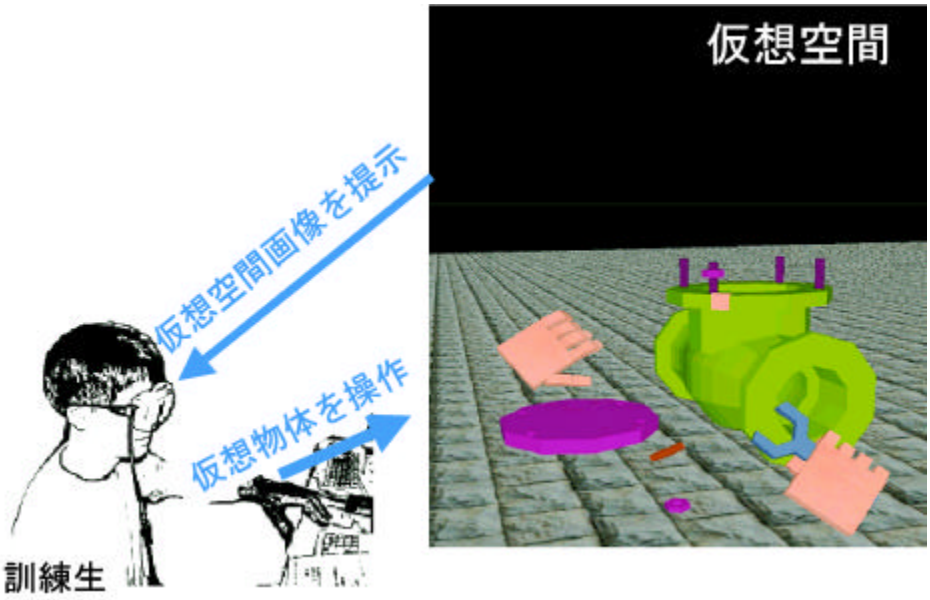


図 2.1: 仮想現実感技術を用いた訓練システム

この様に、人工現実感技術を用いた保修作業の訓練環境を構築すれば、安全・安価に多様な訓練が実施可能となり、エネルギー機器の取り扱い上の安全性・信頼性の向上が期待できる。

図 2.1 に人工現実感技術を用いた機器保修の訓練環境の概要を示す。訓練生は、データグローブ [2] 等の入力デバイスを用いて仮想空間内に構築された仮想物体を操作し訓練を行う。入力デバイスとしては他にも、3 次元マウス、ジョイスティック、2 次元マウス、キーボード等が挙げられるが、これら入力デバイスによって訓練生が入力した操作情報は、訓練生の手や道具を表す仮想物体の位置や姿勢の情報に変換され、仮想空間に反映される。操作の結果は、ディスプレイやヘッドマウントディスプレイ等の出力デバイスを通じ、訓練生に提示される。この様なシステムを実現し、仮想空間内で現実空間内と同様の作業を可能とするためには、ユーザの入力動作に応じて仮想空間内の仮想物体が現実空間と同様に動く仕組みが必要である。例えば、データグローブを用いる場合は 3 軸方向の平行と回転の 6 自由度に加え、手を開く、閉じる等の数種類の自由度を持つ数値がユーザの動作に応じてシステムに入力されるが、この値をもとに仮想空間内の仮想物体の挙動を計算機でどの様に制御するかが問題となる。

ユーザの入力動作に応じて実空間と同様の仮想物体の挙動を合成する手法の 1 つとして Rule-base と呼ばれる手法がある。Rule-base の手法では、仮想物体が取りうる状態と、各状態におけるユーザの入力動作に応じた仮想物体の挙動を、あらかじめデータベースとして定めておき、仮想空間のシミュレーションをおこなう際には、そのデー

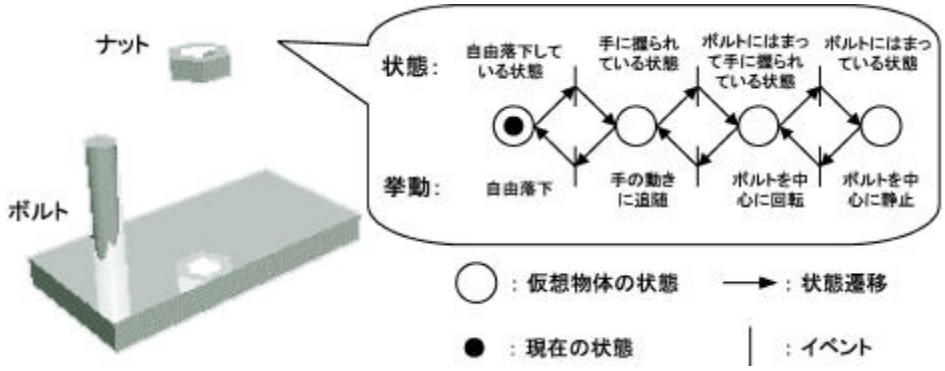


図 2.2: Rule-base の手法によるナットの実現

タベースを用いて、仮想物体の挙動を合成する [3]。例えば、「ナットを手で掴んでボルトに締める」作業をおこなうことが可能な仮想空間を構築する場合には、図 2.2 に示す様に、ナットが取りうる状態として「自由落下している状態」「手に握られている状態」「ボルトにはまって手に握られている状態」「ボルトにはまっている状態」を定める。そしてナットが「自由落下している状態」の時には、ナットが他の物体に接触するまでは、ユーザの動作入力の内容に関わらず、常に自由落下する様に定め、「手に握られている状態」にある時には、ユーザが手を開く動作をおこなうまでは常にユーザの手の挙動にナットの挙動が追随する様に定める。この様に Rule-base の手法では、仮想物体の状態を常に管理するため、訓練の進み具合や作業の過程などをシステムがリアルタイムに把握することが容易となり、作業の誤り修正など訓練生に操作結果をフィードバックできるシステムを容易に実現できるという特徴がある。また、シミュレーションをおこなう際に必要な計算量を少なくできるという特徴がある。

しかし、「ナットを手で掴んでボルトに締める」という単純な例では、仮想物体の状態を管理しその状態における挙動を実現する Rule-base の手法が有効であるが、この手法では構築することが難しい訓練環境も数多く存在する。例えば「机の上に転がり落ちるナット」を仮想空間内に実現する場合、実空間と同様の挙動をさせるためには、落下速度や落下角度に応じてナットがどの様に跳ね返るかを考慮しなければならず、これを Rule-base の手法で実現するためには、非常に多くの状態と各状態に応じたナットの挙動をあらかじめ定めなくてはならない。この様に、仮想物体が様々な挙動をする可能性がある仮想空間を実現する際には、その全てを個別の状態として定義するのではなく、仮想物体に作用する力を運動方程式から算出し、その結果をもとに仮想物体の挙動を決定する手法が望ましい。

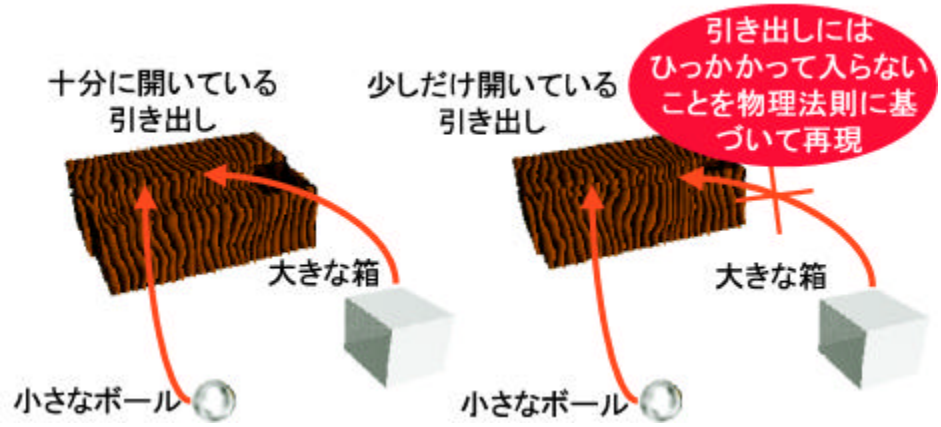


図 2.3: Physically-base の手法による引き出しの実現

物理法則に基づいて厳密に仮想物体の挙動を計算する手法は、Physically-base の手法と呼ばれ、より実際の訓練に近い訓練環境を実現できるという特徴がある。例えば、図 2.3 に示す様に「引き出しに小さなボールと大きな箱を収納する」という操作が可能な仮想空間を構築する場合、Rule-base の手法では、引き出しの開き具合を何段階にも定義し、それぞれの状態に対しボールや箱が「収納できる」「収納できない」を人の手で設定しなければならない。しかし、Physically-base の手法では、引き出しの開き具合とボールや箱の大きさから、「収納できる」「収納できない」を自動的に判別できる。また、引き出しの開き具合が同じ状態でも、収納する仮想物体の傾きにより収納できる場合と収納できない場合が同時に存在する仮想空間を容易に実現できる。この様に、Physically-base の手法を用いれば Rule-base の手法では困難もしくは十分に実現できなかった環境が実現可能となる。また、Rule-base の手法の様に、仮想物体の状態を人の手で個々に定義する必要がないため、仮想空間の構築も容易となる。

ただし、あらゆる操作が可能な訓練環境を Physically-base の手法だけを用いて構築することは非常に難しく、また得策でもない。それは、仮想空間内に配置する仮想物体は一般にポリゴンと呼ばれる手法を用いて三角もしくは四角平面の集合体として表現されるため、図 2.4 に示すように、曲面を完全に表現できないためである。例えば、実世界においてボルトにはまったナットを回転させる場合、ナットが回転する方向や回転のしやすさは、ナットの内側に刻まれた溝とボルトの外側に刻まれた溝の間の接触状態によって決まる。しかし、この様にわずかな形状の違いにより仮想物体の挙動が変化する現象を、仮想物体の形状情報等を用いて計算する Physically-base の手法を用いて実現するためには、莫大な数のポリゴンを用いる必要があり非常に困難である。よって仮想空間内においてボルトに沿ったナットの回転運動を実現する場合、Rule-base の

図 2.4: ポリゴンによる曲面の表現

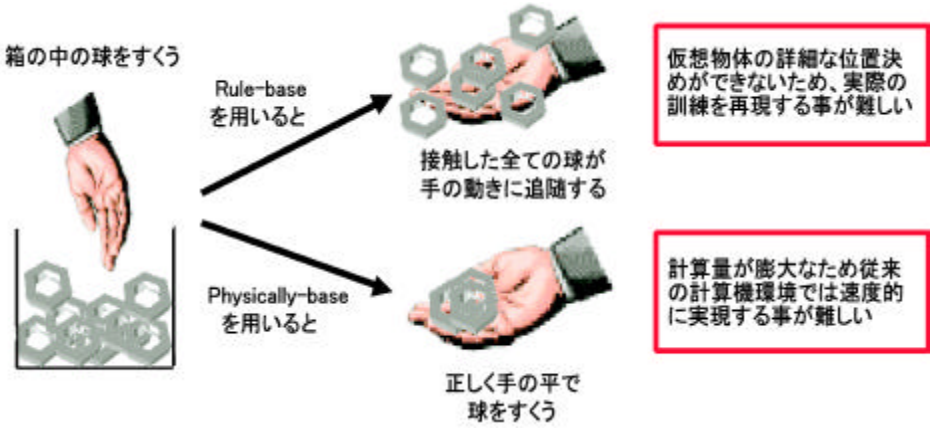
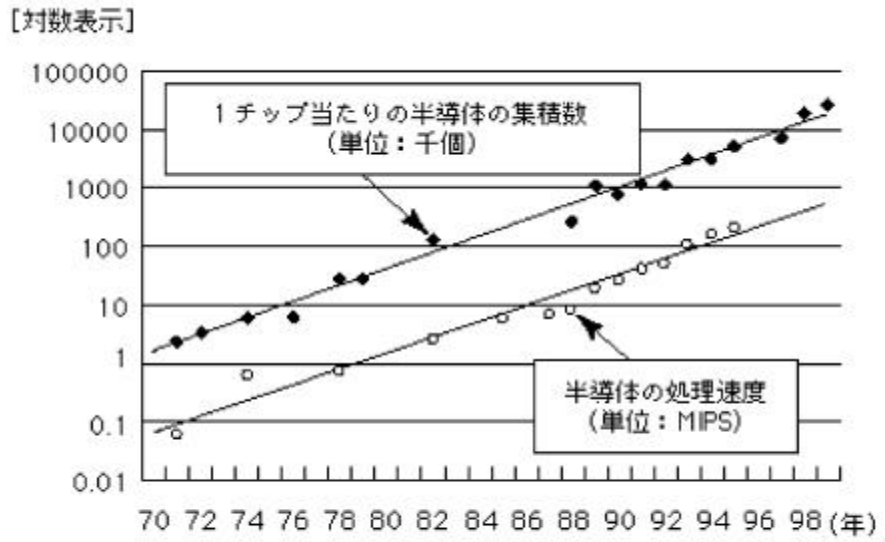


図 2.5: Physically-base の手法と Rule-base の手法の欠点

手法を用いて、ナットの挙動をボルトの長軸を中心とした回転平行運動に限定し、ナットを握っているユーザの手の挙動に応じて挙動させる様に定める方が現実的である。

以上の様な状況を考慮すると、人工現実感技術を用いた機器保修の訓練環境を実現する場合、訓練に使用可能な機器保修の訓練環境を、現実的な範囲内での労力で構築可能にするためには、Rule-base の手法と Physically-base の手法を同時に用いることが理想である。しかし、Physically-base の手法を用いて機器保修の訓練環境を実現するためには膨大な計算をリアルタイムに実行する必要があり、単一の計算機環境上に実現することは困難である。そのため、両手法を同時に実現する訓練環境を構築する際には、図 2.5 に示す様な欠点をお互いに補い合う形で、構築する仮想空間の状況に応じて両手法を使い分ける必要がある。

なお、これまでに開発された人工現実感技術を用いた機器保修の訓練環境として、南雲ら [4] よる原子力ドライウェルにおけるウォークスルー環境、阿部ら [5] による機器の修復支援システム、新井ら [6] による変電所保守員向け体感型シミュレータ、吉川ら [7] による仮想空間シミュレーションシステム OCARINA 等が挙げられる。これらの研究は全て Rule-base の手法を用いて仮想空間が構築されており、訓練環境を構築する際の



（備考）MIPS＝Million Instructions Per Second（1秒間の命令実行回数）。
（資料）インテル社ホームページより作成。

図 2.6: ムーアの法則

労力が大きいこと、シミュレーションのリアル性が低いことが問題となっている。

2.2 計算機性能の向上とPCクラスタの実用化

インテル社の創業者のひとりであるゴードン=ムーアは1995年、「トランジスタの集積度は1年6ヶ月ごとに2倍になる」というムーアの法則[8]を提唱した。1990年から1998年におけるトランジスタの集積度の変化を図2.6に示す。トランジスタの集積度が高まれば、同じ大きさのシリコンウェハ上に載せることができるトランジスタの数を増やすことができるため、計算機の性能を向上させることが容易となる。そして現在、ムーアの法則に示されたとおりトランジスタの集積度が著しく高まった結果、単一の計算機の性能は飛躍的に向上し、一般的なパーソナルコンピュータですら数年前のスーパコンピュータに匹敵する処理能力を有するに至った。しかし、一方で、この様な計算機の急速な発展にも数年後には限界が来るのではないかと言われており、その理由の1つとして微細加工技術の限界が指摘されている。つまり、シリコンウエハ上の配線幅が$0.13\mu m$という極限に微細な領域にまで達した現在、トランジスタの集積度を上げることで計算機の性能を向上させてきた従来の手法では、これまでと同様のペースで計算機の性能を向上させることが、今後は困難になると予想されるからであ

図 2.7: 小規模 PC クラスタ (35 台接続)

る。また、現在用いられているほぼ全ての計算機は、ハンガリーの数学者フォン=ノイマンが考案したノイマン型 [9] と呼ばれる方式に基づいて設計されているが、ノイマン型計算機は基本的に 1 度に 1 つの命令しか処理することができず、高速化にも限界があることが指摘されている。現在、この様な集積化技術の限界とアーキテクチャ技術による制限に対する将来的な危惧を背景に、単一の計算機の性能に依存しない超並列計算機や、ノイマン型と異なる方式に基づいて設計された量子計算機 [10] や DNA 計算機 [11] の研究開発が活発に進められている。

その様な状況の中、計算機の新たな形態として、PC クラスタが注目されている。PC クラスタとは、パーソナルコンピュータを用いたメモリ分散型 [12] の並列計算機であり、大規模な計算タスクを分散・並列化して実行することが可能なシステムである。この PC クラスタは、ネットワーク接続された複数台のパーソナルコンピュータに専用のライブラリを導入することで比較的容易に構築することが可能なため、既存の計算機環境をそのまま流用することが可能であり、導入の際の技術的な容易さとコストの低さから現在急速に普及している。また、当初はスーパコンピュータに代わる計算システムとして、構成する計算機が 100 台を越える様な大規模な PC クラスタが普及したものの、最近は大学の研究室レベルを中心に図 2.7 に示す様な 20 台程度の小規模な PC クラスタの普及が進展し、比較的容易に入手可能な高性能計算機環境として広く認知されつつある。

PC クラスタを導入する利点として、以下の様な点が挙げられる。

表 2.1: PMv2 を用いたメッセージ往復時間の比較

Network	Bandwidth (MByte/sec)	RTT Latency (μ sec)
Myrinet(1.2Gbps) Lanai9 in 33MHz PCI64	146.9	20
Gigabit Ethernet (Syskonect)	73.4	61
Fast Ethernet (EEPRO100)	11.9	100
Fast Ethernet x 2 (EEPRO100)	23.9	105
Fast Ethernet x 3 (EEPRO100)	30.7	117

1. スーパコンピュータに匹敵する計算機環境を得ることができる。
2. 接続する計算機を増やすだけで性能を上げることができる。
3. 既存の計算機環境が流用可能であるため導入が容易である。
4. 安価な PC と無料の OS を用いることで導入コストが低い。
5. 技術進歩によるシステムのバージョンアップが容易である。
6. 古い計算機を有効利用することができる。

PC クラスタで動作するソフトウェアの開発には、SCore クラスタシステムソフトウェア [13] をはじめとする、並列プログラミング環境のためのソフトウェアパッケージを用いることができる。SCore クラスタシステムソフトウェアには、高性能通信ライブラリ PMv2 が含まれており、これを用いれば一般に用いられている TCP/IP に比べより高速なクラスタ間通信を実現できる。表 2.1 に、 64 台の NEC 社製 ExpressServer(Dual Pentium-III 800 MHz, 512MByte Main Memory) を用いて構築した PC クラスタにおいて PMv2 を用いた際のメッセージ往復時間の比較を示す。LAN 環境として一般的な 100BASE-T において PMv2 を用いた場合、単位メッセージが計算機間を往復するのに

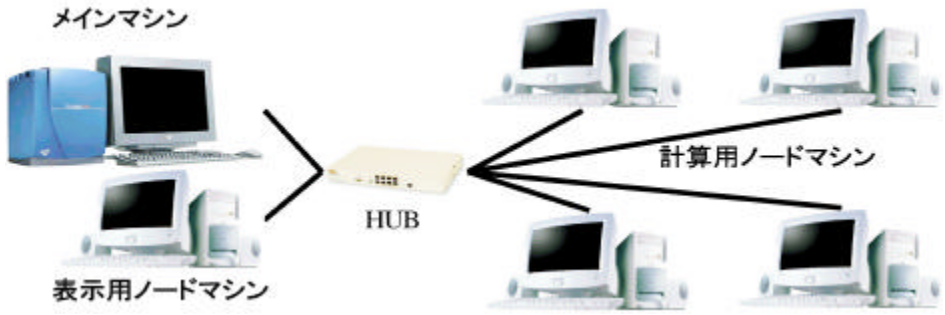


図 2.8: PC クラスタの構成図

要する時間はおよそ $100\mu s$ となり、1 秒間に約 10000 回のメッセージを高速に交換することができる。この様な高性能なソフトウェアパッケージと強力な通信ライブラリを備えたことで、PC クラスタは今や、自然現象シミュレーション [14]、高度医療情報処理 [15]、ゲノム解析 [16] といった最先端の情報処理技術に欠かせない重要な基盤技術となりつつある。例えば、近年注目を集めている Web 検索エンジン Google が PC クラスタを採用している事例が指し示す様に、限られた分野にだけ許されていたスーパコンピュータレベルの高性能な計算機環境が、それほど専門性を必要としない分野にまで新たに開拓されつつある。そして、PC クラスタの普及により、今後その傾向は益々強くなるものと予想される。

仮想現実感技術の分野においても、PC クラスタの利用は急速に進展している。特に、人物の頭髪や海の波面といった 3 次元画像をよりリアルに合成する際に必要な計算タスクや、物体の変形や流体の移動などの物理シミュレーションをより正確におこなう際に必要な計算タスクを、PC クラスタを用いて分散・並列処理する試みが活発におこなわれている。

2.3 本研究の目的

本研究は、機器保修の訓練環境を実現する際に必要な Physically-base の手法に基づく剛体挙動シミュレーションを、既存の剛体挙動のモデル化手法を拡張することで実現し、計算機上へ実装することを目的とする。さらに、実装した剛体挙動シミュレーションの処理の一部を、図 2.8 に示す様な PC クラスタを構成する複数台のノードマシンに分散し並列処理する手法を開発し、現時点で容易に入手可能な計算機環境上でのリアルタイムシミュレーションの実現可能性の検討と、実用化のための今後の指針を得ることを目的とする。具体的には、まず、仮想物体の移動、回転、衝突等の物理現

象・物理法則を計算機を用いてシミュレーション可能とするためのモデル化をおこない、高速かつ速度が安定的なシミュレーションを実現するための機能を検討し、単一の計算機環境で動作する剛体挙動シミュレーションシステムを構築する。次に、単一の計算機環境で動作する剛体挙動シミュレーションシステムを用いて、仮想空間のシミュレーションをおこなうのに必要な各種の処理のうちどの処理にどの程度の計算が必要かを計測する実験を実施する。そして、実験から得られた結果に基づき、剛体挙動シミュレーションの処理の一部を分散し並列処理する手法を開発し、PC クラスタ環境で動作する剛体挙動シミュレーションシステムを構築する。最後に、構築したシステムを用いて評価実験を実施し、単一の計算機環境におけるシミュレーションと PC クラスタ環境におけるシミュレーションを比較検討することにより、PC クラスタを用いた剛体挙動シミュレーションの有効性を確認する。

なお、本研究は、現時点で容易に入手可能な計算機環境として、大学の研究室の様に複数台のパーソナルコンピュータが既にネットワーク接続されており、直ちに小規模 PC クラスタを導入することができる環境を想定する。また、スパナ等の工具を用いた機器の分解・組み立て、バルブ等の機器操作、破砕導管の溶接等の訓練が可能な機器保修の訓練システムを実現するためには、表 2.2 に示す物理法則・物理現象の実装が必要である。しかし、本研究を Rule-base の手法と Physically-base の手法の特徴を同時に実現する訓練環境を将来的に実現するための基礎研究と位置づけ、表 2.2 の左側に示す物理法則・物理現象の実現を前提にシステムを設計する。ただし、実現しない物理法則・物理現象のうち静止摩擦力と動摩擦力は、機器保修の訓練環境を実現する上で重要と考え、システムの一部を入れ替えるだけで容易に実装可能となるようにシステムを設計した。

2.4 PC クラスタを用いた分散・並列処理に関する従来研究と本研究の特徴

2.2 節において、PC クラスタを用いた研究開発が様々な分野で急速に拡大していることについて述べた。本節では、PC クラスタを用いた分散・並列処理に関する従来研究について述べ、本研究が目的とする機器保修の訓練環境のための剛体挙動シミュレーションの分散・並列処理との比較をおこない、本研究と従来研究の目標の違いを明確にする。

分散・並列処理に関する研究の中でも、暗号解析は、解析アルゴリズムの分散・並

表 2.2: 機器保修の訓練環境を実現する際に必要な物理法則・物理現象

機器保修の訓練環境を実現する際に必要な物理法則・物理現象	
本研究で実現する物理法則・物理現象	本研究で実現しない物理法則・物理現象
衝突時の撃力 (力積) 接触力 (垂直抗力) 重力 拘束力	物体の変形 物体の分割・結合 静止摩擦力、動摩擦力 空気抵抗、磁力

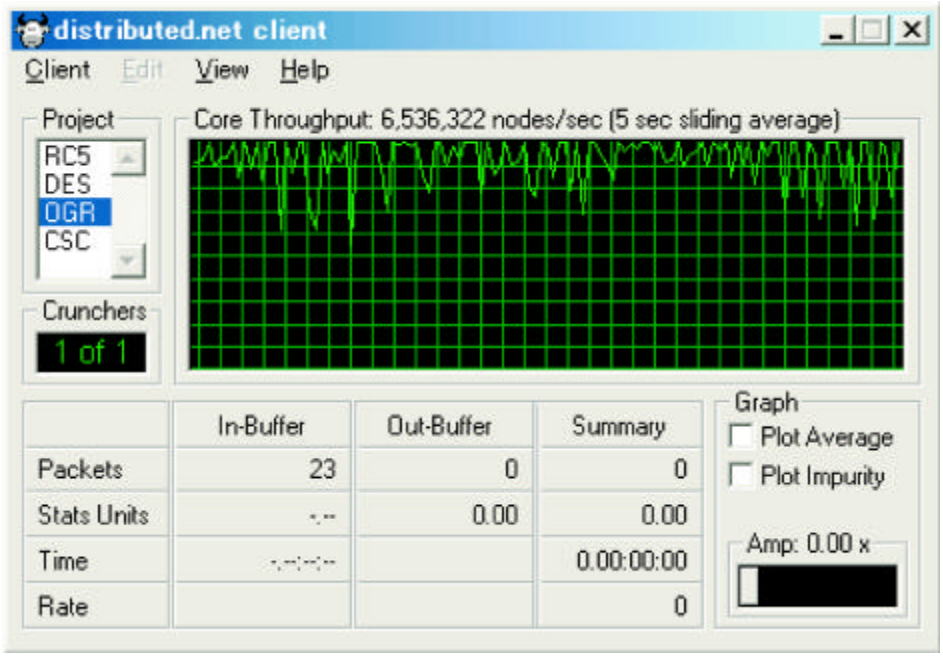


図 2.9: 分散計算の例 (distributed.net client による暗号解析)

列化が容易であり、解析速度が計算性能に強く依存する、等の理由から古くから研究がおこなわれてきた。例えば、暗号解析プロジェクト「distributed.net」[17] では、懸賞付きの暗号解読コンテストや暗号化鍵の強度の検証、メルセンヌ素数探索等、暗号解析処理に必要な膨大な計算タスクを、インターネットを介して接続されたプロジェクト参加者の計算機に分散・並列化するシステムの開発がおこなわれている。図 2.9 に distributed.net による暗号解析ソフトウェアの画面例を示す。

医療分野における分散・並列処理の実用化も進められており、具体的には、ヒトゲノ

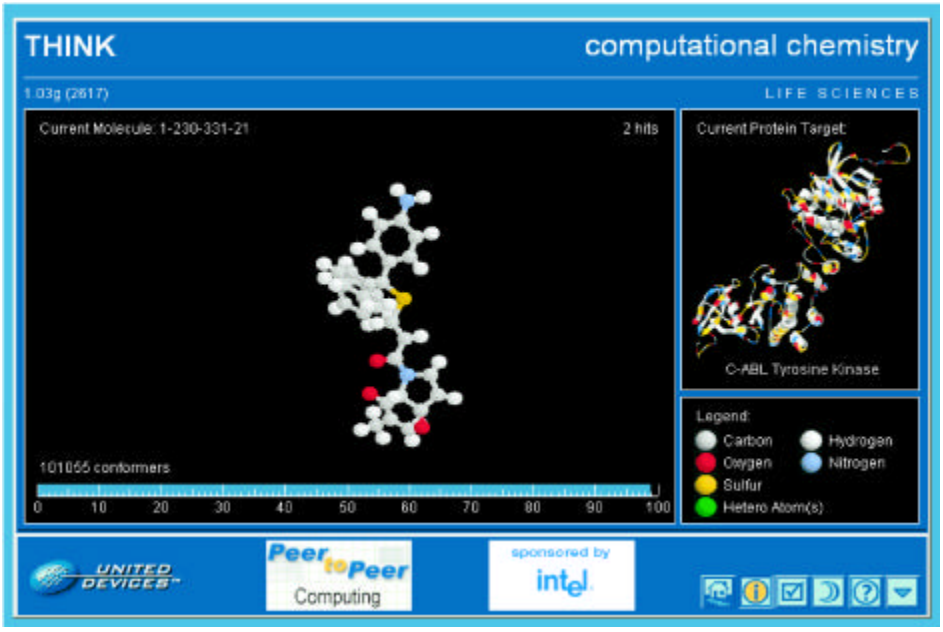


図 2.10: 分散計算の例 (UD Agent による新薬開発)

ム解析、エイズ特効新薬の開発 [18]、アルツハイマーの原因となるタンパク質の異常折り畳みのシミュレーション [19] 等が挙げられる。これらの医療分野における分散・並列処理の特徴は、Web 上での簡単な登録手続きをおこない、自分の計算機にクライアントソフトウェアを導入するだけで、誰でも簡単にプロジェクトに参加することが可能な点である。例えば、ガン治療薬開発プロジェクト「UD Agent」は、ガンと関係の深いタンパク質に有効と思われる数億種類の分子モデルから薬として有効なものを検証し、ガン治療薬の開発に生かすことを目的としたプロジェクトであるが、プロジェクト参加者は各自の計算機に導入した図 2.10 に示す様なクライアントソフトを用いて、プロジェクトのサーバから自動的に割り当てられたデータの解析を担当する。なお、解析は参加者が計算機を利用していない空き時間を用いて自動的におこなわれ、解析が終了すると解析結果は自動的にサーバに送信される。この UD Agent では、自分の計算機がどれほど解析に貢献しているかを Web 上で確認することができ、医療技術の発展に貢献しているという実感を得ることができるため、近年プロジェクト参加者が急増している。

他にも PC クラスタを用いた分散・並列処理に関する研究として、分子・原子の挙動計算 [20] や流体シミュレーション [22] をはじめとする物理シミュレーション、分散レンダリング [23] やビデオエンコーダ [24] をはじめとする画像処理、宇宙電波の分析 [25] やデータベース検索をはじめとする解析処理 [26] 等、数多くの研究が報告されている。こ

れら全ての研究に共通する特徴は、膨大な計算処理をネットワークで接続された複数の計算機で分散・並列処理し、少しでも処理を完了するまでの時間を短くすることを目的としている点である。例えば、単一の計算機を用いて 24 時間かけて得られた結果と、複数の計算機を用いて 1 分かけて得られた結果は全く同一であるため、結果を得るまでにかかる時間を短縮することだけを目的に分散・並列処理をおこなっており、どこまで短縮すべきかという明確な目標はない。

一方、本研究で対象とする機器保修の訓練環境のシミュレーションでは、シミュレーションの開始から終了までの合計の計算時間を短縮すればよいというものではなく、訓練生が仮想物体に操作を加えた影響がリアルタイムに仮想空間内の他の物体に反映される必要がある。

すなわち、仮想空間のシミュレーションを一定の時間間隔だけ進めるために必要な計算量は、仮想空間の状態に応じて大きく変化するが、時間間隔 T_1 だけシミュレーションを進めるために必要な計算時間を T_2 とした場合、常に以下の式が成り立つ必要がある。

$$T_1 \geq T_2 \tag{2.1}$$

また、仮想空間を更新する時間間隔 T_1 は、仮想空間を観察する訓練生が違和感を感じないようにするために十分小さい必要があり、一般に毎秒 10 回の仮想空間の更新が必要であると考えられることから、以下の式が成り立つ必要がある。

$$100ms \geq T_1 \geq T_2 \tag{2.2}$$

上記の式を成立させることを考えた場合、シミュレーション全体を通じた T_2 の最大値 T_{2max} を如何に低く抑えるかが問題となる。一般に機器保修の訓練環境において、シミュレーションで必要となる計算をあらかじめ予測し、先行して計算処理することは非常に困難であるため、本研究では、仮想空間のシミュレーションをおこなうにあたり必要となる計算処理を PC クラスタを用いて分散・並列化することにより、T_{2max} の値を低く抑えることが大きな目的となる。

一方、本研究では、分子・原子の挙動計算や流体シミュレーションの様に完全に実世界を反映する正確なシミュレーションは必要ではなく、合成される仮想物体の動きに対し訓練生が違和感を感じなければ十分機能を満たしていると言うことができる [27]。そのため、PC クラスタを用いた機器保修の訓練システムにおいて、訓練生が違和感を感じることなくリアルタイム性を確保できる仮想空間の複雑さの限界を探ることが本研究の主眼となる。

第 3 章 剛体挙動のモデル化と単一計算機を用いたシミュレーションシステムの構築

本章では、まず剛体挙動のモデル化について説明する。その後、本モデル化手法を実装した単一の計算機環境で動作する剛体挙動シミュレーションシステムについて説明する。最後に、単一の計算機環境でシミュレーションを実行する際の各種処理の計算負荷を計測するために実施した、計算負荷計測実験について述べる。

3.1 剛体挙動のモデル化

剛体挙動を計算機上でシミュレーションするためには、計算機上に数学的モデルとして実装しなければならない[28]。また、本研究の剛体挙動シミュレーションは機器保修の訓練環境に用いることを想定しているため、シミュレーション速度が十分に速いだけではなく、その速度が大きく変化しないようにする必要がある[29]。

本節では、まず剛体挙動シミュレーションの概要について述べ、次に、3 次元仮想空間における剛体の状態遷移と剛体間の衝突判定方法、剛体間の接点に働く力の算出方法、位置と姿勢の離散的処理について説明する。最後に、高速かつ安定的なシミュレーションを実現するために本研究で開発した各種手法について説明する。

3.1.1 剛体挙動シミュレーションの概要

単一の計算機環境における剛体挙動シミュレーションの処理の流れを図 3.1 に示す。まず、シミュレーションを開始する前に、(1) 剛体の初期位置、初期速度等を設定する。次に、(2) ユーザからの入力を処理した後、(3) ある時間幅 t だけ時間積分をおこない、剛体の位置、姿勢を更新し、剛体同士の衝突判定をおこなう。衝突判定では、まず、(4) 計算負荷を軽減させるための球近似による簡易な衝突判定をおこない、接触する可能性がある剛体の組み合わせを抽出し、その組み合わせに対してのみ (5) ポリゴンレベルの厳密な衝突判定をおこなう。

シミュレーションはシステムが定めた時間幅 t ずつ進行するが、時間幅 t だけシミュレーション時間を進めた際、衝突した剛体同士が干渉を起こす可能性がある。干渉と

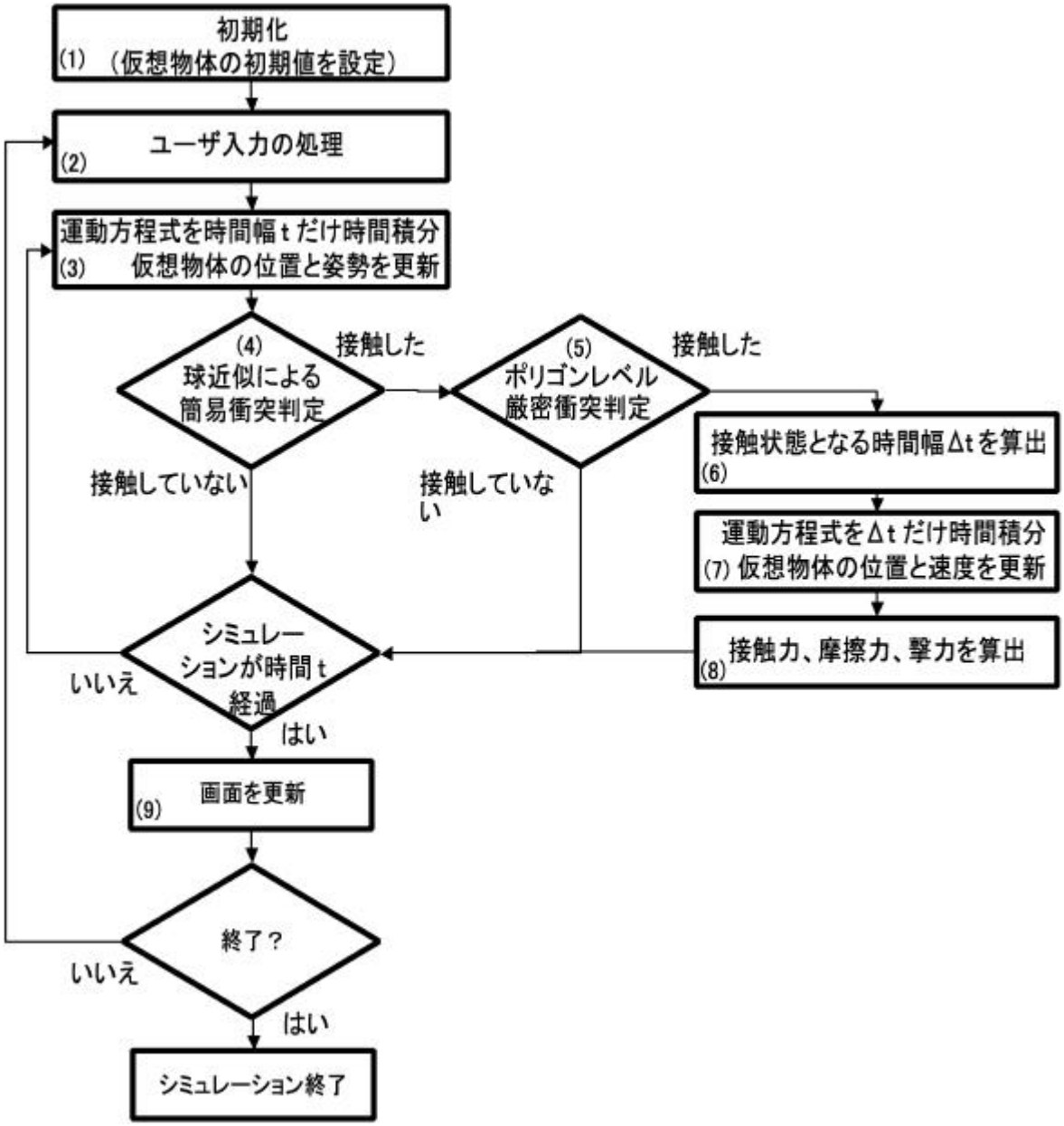


図 3.1: 単一の計算機環境におけるシミュレーションの処理の流れ

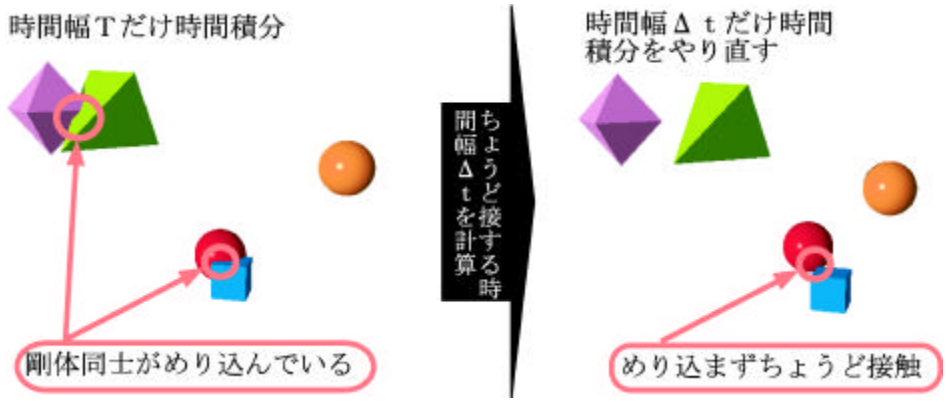


図 3.2: 時間積分のやりなおし

は、剛体同士がお互いにめり込んでいる状態を指す。干渉が発生すると、剛体間の接点を正確に算出することが難しく、接点に働く力を正しく算出することができない。また、たとえ正確な接点を算出することができたとしても、剛体の中に別の剛体が埋まっているようなアニメーションを出力し、ユーザに違和感を感じさせてしまう。そこで、剛体同士の干渉が発生しないようにするため、衝突判定において干渉が検知されると、(6) 剛体同士がちょうど接する時間刻み $\Delta t(0 < \Delta t < t)$ を算出し、(7) 時間幅 Δt だけの時間積分をやりなおし、シミュレーションを Δt だけ進行させる。時間積分のやりなおしを図 3.2 に示す。なお、時間幅 t の間に複数の剛体が同時に干渉している場合、算出した時間刻みのうち最も短い時間刻みを Δt とする。

そして、(8) 衝突が起こった時点での剛体のそれぞれの接点について、接触力や摩擦力など剛体に働く力を算出する。以上 (3)〜(8) までの処理を繰り返し、(9) シミュレーションがある一定時間を経過する度に画面を更新する。なお、本研究では、(3)〜(9) までの一連の処理を便宜的に 1 フレームと呼ぶこととする。

3.1.2 3 次元空間における剛体の状態遷移

本研究では、図 3.3 に示すように、剛体の運動状態を 3 つの状態に分類して扱う。まず、剛体同士の接点が無い場合 (図 3.3 の a)、剛体は重力からの外力を受け、自由落下運動をおこなう。ユーザが 3 次元マウス等の入力デバイスを用いて剛体を操作する際、剛体が受ける力も重力と同様に外力として扱う。次に、剛体同士が衝突する場合 (図 3.3 の b)、接点では衝突が起こった瞬間に衝突による撃力が衝突面と垂直方向に働き、次の瞬間お互いの剛体は離れる方向に移動する。そして、剛体同士がお互いに接触状態にある場合 (図 3.3 の c)、接点では接触力が働き物体同士が干渉するのを防ぎ、接触状

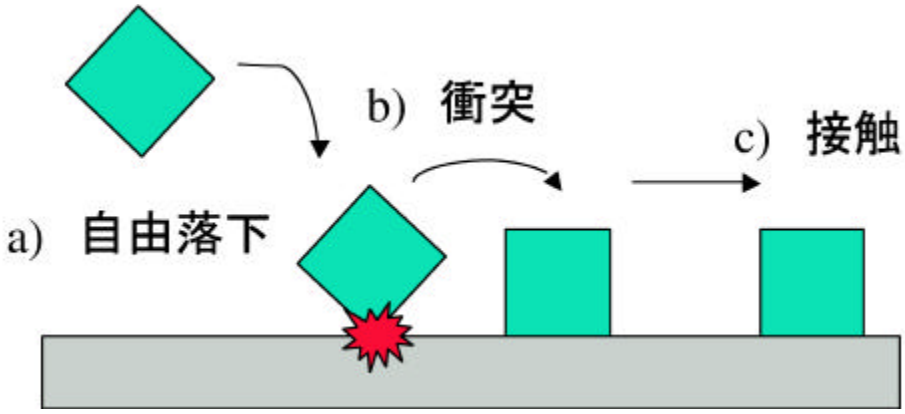


図 3.3: 剛体の状態

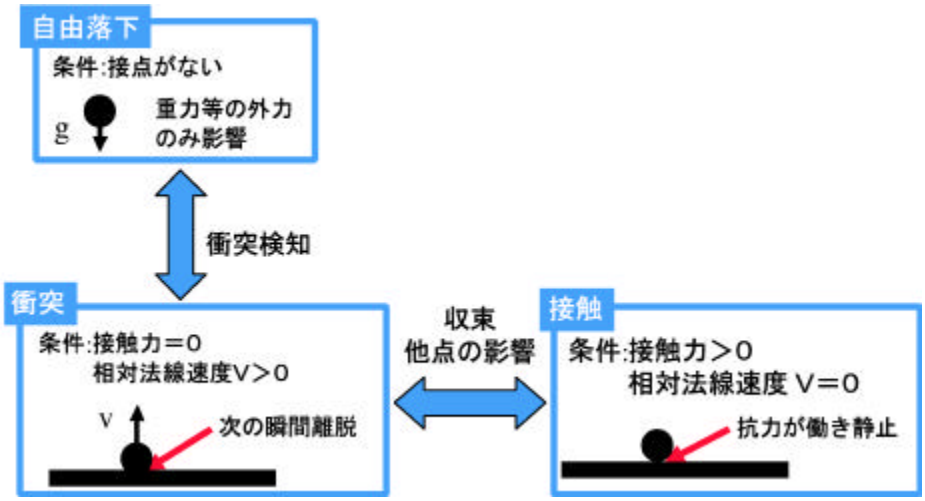


図 3.4: 剛体の状態遷移

態を維持する。以上のように剛体の状態を定めることで、個々の剛体の挙動をシステムが管理することが可能となる。なお、Rule-base の手法と剛体の状態を設定し管理する点が似ているが、ユーザ自身の手で状態と状態間の遷移条件を記述する必要がなく、システムが自動的に判別し設定する点が異なる。

図 3.4 に状態間の遷移条件を示す。自由落下している剛体は他の剛体との衝突を条件に衝突状態へ状態遷移する。衝突の際、剛体間の相対法線速度がある閾値以上だった場合、お互いに跳ね返り、次の瞬間には自由落下状態に戻る。しかし、剛体間の相対速度がある閾値以下だった場合、剛体は更に接触状態にまで状態遷移し、他の剛体による新たな衝突が発生するまで接触状態を維持する。本研究では、剛体の状態管理として、以上に述べた 3 つの状態と状態遷移条件を用いる。

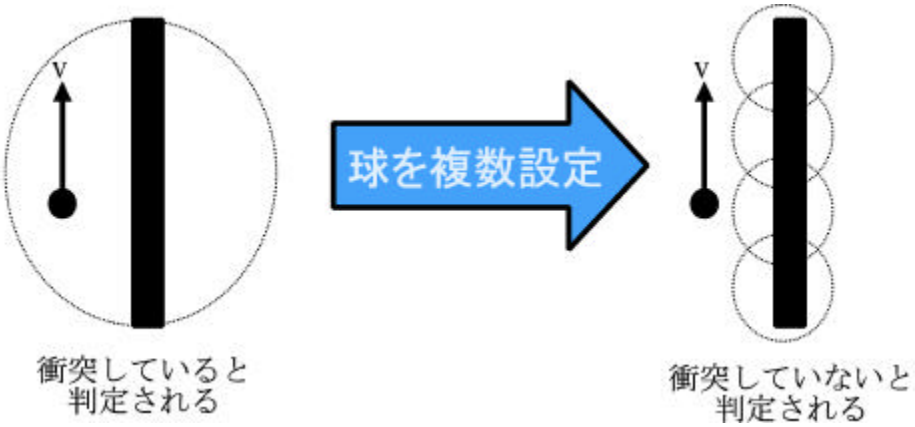


図 3.5: 複数の球による球近似の接触判定

3.1.3 3次元空間における剛体の衝突判定

剛体の状態遷移を正しくシミュレーションするためには、剛体同士の衝突を正確に判定しなければならない[30]。また、ユーザが不自然さを感じることがないように剛体の挙動をシミュレーションするためは、剛体間の接点に瞬間的に働く撃力や持続的に働く接触力が作用する方向を正確に求め、運動量と角運動量を適切に変化させなければならない[31]。

しかし、仮想空間内のすべての剛体の組み合わせに対しポリゴンレベルで厳密に衝突を判定をおこなうことは望ましくない。衝突判定をおこなう剛体の形状が複雑になった場合、判定をおこなうポリゴン数が増大し、シミュレーション速度を著しく低下させる恐れがあるためである。従って、シミュレーションを安定して実行するためには、厳密な衝突判定の回数を極力減らすことが重要となる。そこで本研究では、厳密な衝突判定をおこなう前に、まず、球近似による簡易な衝突判定により衝突が予想される剛体の組み合わせをあらかじめ抽出し、その組み合わせに対してのみポリゴンレベルの厳密な衝突判定をおこなうものとした。

球近似による簡易な衝突判定は、2つの近似球の中心間の距離と、2つの近似球の半径の和を比較するだけで判定をおこなうため、全ての剛体の組み合わせに対しておこなっても、計算負荷は大きくならないと予想される[32]。しかし、図3.5に示すように、細長い物体に対して球近似による簡易な衝突判定をおこなう場合、1つの剛体に対し1つの近似球しか設定しなければ、近似球が剛体に対して非常に大きくなってしまい、その結果、必要以上の剛体の組み合わせに対し衝突する可能性があると抽出してしまう。そこで、本研究では、1つの剛体に対し複数の近似球を設定可能とし、ポリゴンレベルの厳密な衝突判定をおこなう剛体の組み合わせの数を減らす試みをおこなった。

図 3.6: 凹形状の分割

球近似による簡易な衝突判定によって衝突する可能性があると抽出された剛体の組み合わせは、衝突判定アルゴリズム Voronoi Clip[33] を用いてポリゴンレベルで厳密な衝突判定をおこなう。Voronoi Clip は 2 つの多面体間の衝突判定をおこなう数多く提案されたアルゴリズムの内のひとつであり、他のアルゴリズムと比較して安定して精度の高い結果を導きだせるため、数多くの仮想現実感技術を用いたアプリケーションで用いられている。Voronoi Clip の特徴を以下に示す。

1. 非常に頑健なアルゴリズムであり、誤動作や無限ループに陥る可能性が低い。また、計算誤差が小さく信頼性が高い。

2. ハッシュテーブル [34] を使用するため、同じ剛体同士の衝突では 2 回目以降の判定が高速化される。

3. 特別な前処理等を必要としないため実装が容易である。

4. 凸形状のみ適用可能である。

以上に述べた特徴からも、Voronoi Clip は本研究が想定している機器保修の訓練環境のような剛体が激しく運動し衝突する運動シミュレーションに適したアルゴリズムと言える。ただし、Voronoi Clip は凸形状にのみ適用可能であるため、凹形状に対しては図 3.6 に示すように、これを複数の凸形状に分割することで適用する。具体的には、Voronoi Clip は ptree 形式という独自の形状データを用いて衝突判定をおこなうが、凹形状の衝突判定をおこなう場合、この ptree 形式の形状データの中に凹形状を分割してできた複数の凸形状の形状と位置の情報を記述する。例えば、図 3.7 に示すように、凹形状である机の ptree 形式の形状情報を記述する場合、まず、凸形状である天板と脚の形状を記述し、続いて、天板と脚の位置の情報を記述する。

本研究では、剛体同士が接している接点に働く力をそれぞれ算出し、各剛体毎の運動量と角運動量の変化を計算することで剛体の挙動をシミュレーションする。そのた

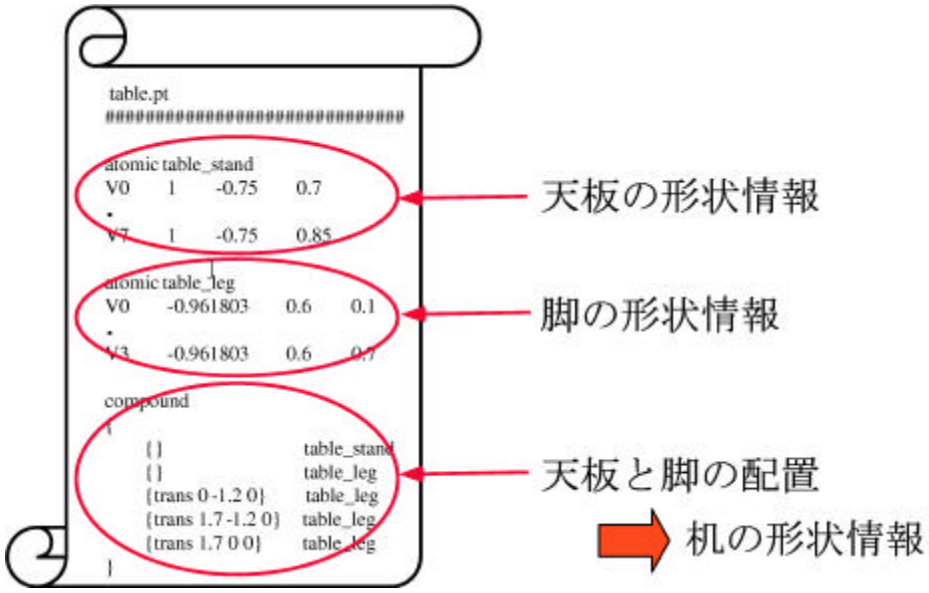


図 3.7: PolyTree データの書式

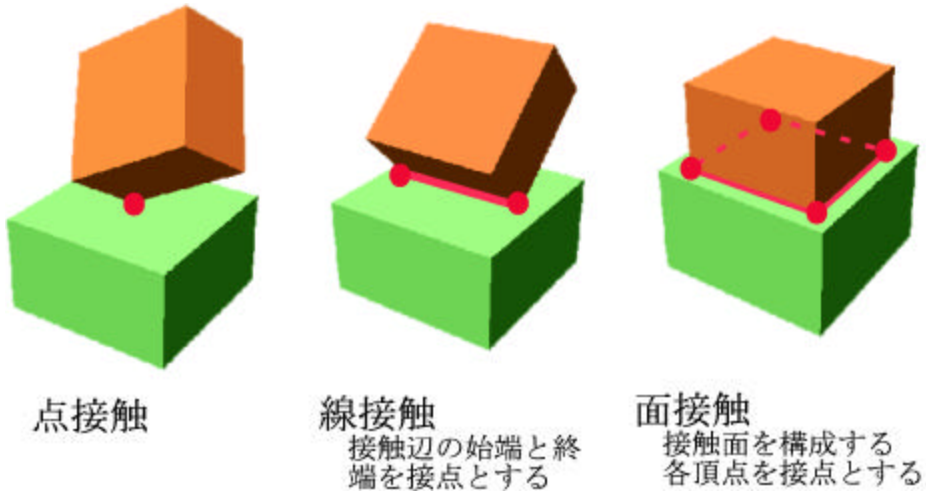


図 3.8: 衝突状態と接点

め、剛体同士が線と線で衝突している場合や面と面で衝突している場合、これを点による衝突に置き換えなければならない。図 3.8 にそれぞれの衝突状態と点による衝突への置き換え方法を示す。剛体同士が点で衝突している場合、剛体間の接点は 1 つに定まる。線で衝突している場合、線を構成する両端の 2 点を接点とする。面で衝突している場合、面を構成する各頂点をそれぞれ接点とする。接点が求まれば、それぞれの接点における衝突後の速度の法線方向と相対法線速度を算出する。

図 3.9 に示すように、複数の剛体が同時に衝突する多点衝突の際、接点 A における衝突が接点 B における衝突に影響を及ぼす場合がある。そのため、剛体同士の接触状

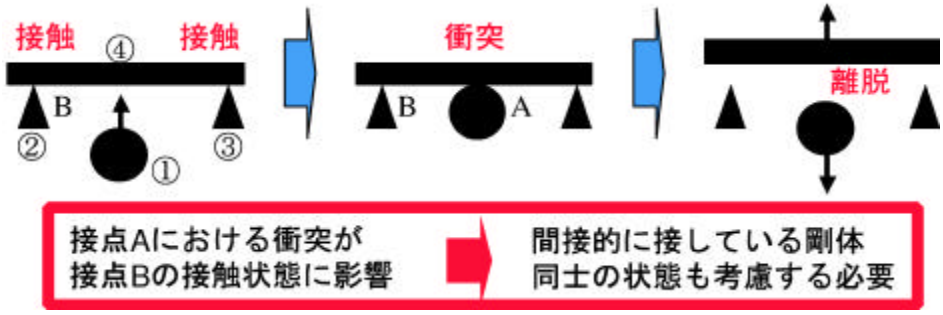


図 3.9: 多点衝突

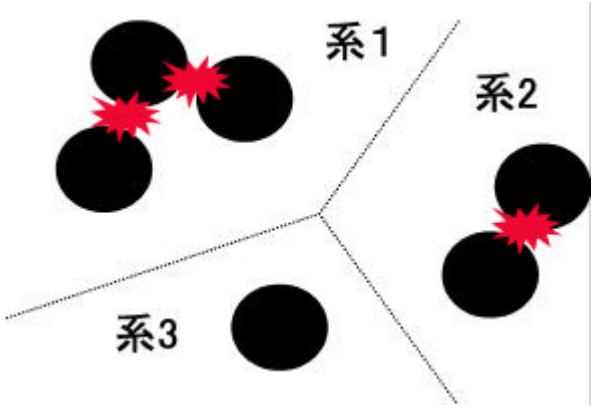


図 3.10: 系

態は、直接的に接している剛体同士 (図 3.9 における①と④) の状態のみならず、間接的に接している剛体同士 (図 3.9 における①〜④) の状態も含めて考えなければならない [35]。本研究では、図 3.10 に示すように、直接・間接に接している全ての剛体の組を便宜的に系と呼ぶこととする。

3.1.4 剛体間の接点に働く力の算出手法の検討

剛体挙動をシミュレーションするためには、剛体間の接点において働く力を算出しなければならないが、接点に働く力のモデル化手法には、撃力による手法 [36]、ベクトル場を利用する手法 [37]、バネとダンパによる手法 [38]、撃力と接触力による手法 [39]、撃力と接触力と摩擦力による手法 [40] など様々な手法が提案されている。そこで、本研究のシミュレーションの対象が機器保修の訓練環境であることを前提に、これらの手法の利点と欠点を検討し、本研究に最も適したモデル化手法を決定した。

表 3.1 に各モデル化手法の特徴の比較を示す。

まず、撃力による手法は、剛体間の接点において作用する力を全て撃力によって表

表 3.1: 接点に働く力の算出手法の比較

	撃力による手法	ベクトル場から抗力を計算する手法
接点に働く力	撃力	接触力
算出方法	衝突と接触を全て衝突と扱う	ベクトル場を用いて近似
長所	シミュレーションが単純	計算量が少ない
短所	シミュレーション速度が不安定	挙動が不自然 摩擦力を実現できない

	バネとダンパによる手法	撃力と接触力による手法
接点に働く力	接触力	撃力、接触力
算出方法	バネとダンパを用いて近似	Dantzig のアルゴリズムで算出
長所	計算量が少ない	シミュレーション速度が安定
短所	挙動が不自然 摩擦力を実現できない	摩擦力を実現できない

	撃力と接触力と摩擦力による手法	
接点に働く力	撃力、接触力、摩擦力	
算出方法	Dantzig のアルゴリズムで算出	
長所	シミュレーション速度が安定	
短所	撃力と接触力による手法と比べ アルゴリズムが複雑	

現する手法であり、接触と衝突を統一的に扱うことができるためシミュレーションを単純化することができるという利点がある[41]。しかし、剛体の数が増加すると計算量が指数関数的に増加する可能性があり、機器保修の訓練環境の様に多数の剛体が衝突、接触するシミュレーションには適さない。特に、複数の剛体が積み重なる状態になるとシミュレーション速度が著しく低下するため、機器の分解組み立て訓練等を実現することは速度的に難しい。

次に、ベクトル場から抗力を計算する手法は、剛体の周囲に斥力を与えるベクトル場を定義して剛体同士の衝突を回避する手法であり、剛体の位置によってのみ接触力を計算することができるため、計算量が少なくてすむという利点がある。しかし、剛体が衝突した際の撃力が考慮されていないため、剛体同士が干渉してしまう可能性がある。また、本モデル化を用いて計算された剛体は現実とかなりかけ離れた挙動を示すため、訓練を疑似体験させるという本研究の目的に適さない。

バネとダンパによる手法はペナルティ法とも呼ばれ、商用ソフトウェアにおける剛体挙動シミュレーションに広く用いられている手法である。剛体間の微少な干渉を許し、干渉点にバネおよびダンパが設定されていると近似して接触力を算出するため、ベクトル場から抗力を計算する手法と同様に、現実とはかけ離れた挙動を示すという欠点がある。また、適切な挙動を得るためには、対象に応じて接触力を決定するバネ定数を適宜調整する必要があり、仮想空間を構築する手間も増大する。

撃力と接触力による手法と撃力と接触力と摩擦力による手法は、剛体間の接点毎に条件式をたて、Dantzig のアルゴリズムを用いて解くことで接点に働く力を算出する手法である。両手法は、接触力と撃力の算出方法はほぼ同様であり、摩擦力を算出することができるか否かという点だけが異なる。両手法は多点衝突の際もシミュレーション速度が安定しているという利点があり、リアルタイム性が必要不可欠な機器保修の訓練環境の構築に適している。また、剛体同士が干渉する可能性が低く、他の手法と比べてもより現実に近い挙動を得ることができる。

以上のような検討の結果、撃力と接触力と摩擦力による手法 (以下、川地らの手法) を用いることにする。なお、本研究では摩擦力は実装しないが、川地らの手法を用いることで実装は比較的容易にできる。以下、川地らによる接点に働く力のモデル化手法について説明する。

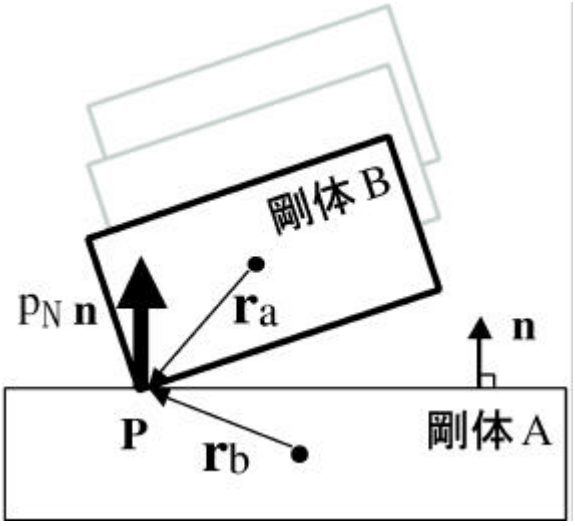


図 3.11: 接点に働く力のモデル化

3.1.5 剛体間の接点に働く力の算出

本研究は、川地らの手法[40]を用いて剛体間の接点に働く力を算出する。図 3.11 に示すように、剛体 A と剛体 B とが 1 点 P で衝突している場合を考える。衝突面の単位法線ベクトルを n とし、衝突による力は n に平行に大きさ p_N で働くとする。ただし、n は衝突面において剛体の外側を指しているものとする。この衝突を記述するために以下に示す記号を用いる。添え字の A または B は、その記号が剛体 A または剛体 B に関するものであることを示す。

m_A, m_B	質量
$\mathbf{I}_A, \mathbf{I}_B$	慣性テンソル
$\mathbf{u}_A, \mathbf{u}_B$	衝突前の剛体の並進速度ベクトル
$\mathbf{u'}_A, \mathbf{u'}_B$	衝突後の剛体の並進速度ベクトル
ω_A, ω_B	衝突前の剛体の角速度ベクトル
ω'_A, ω'_B	衝突後の剛体の角速度ベクトル
P	接点
$\mathbf{G}_A, \mathbf{G}_B$	重心
$\mathbf{r}_A, \mathbf{r}_B$	重心から接点への位置ベクトル

衝突している時間が十分短く、衝突の前後で剛体の位置および姿勢が変化しないと仮定すると、原点を基準とする絶対座標系において、剛体 A 上の接点の衝突前の速度 $\mathbf{v_A}$ と衝突後の速度 $\mathbf{v'_A}$ は次式のようになる。

$$\mathbf{v}_A = \mathbf{u}_A + \omega_A \times \mathbf{r}_A \tag{3.1}$$

$$\mathbf{v'}_A = \mathbf{u'}_A + \omega'_A \times \mathbf{r}_A \tag{3.2}$$

衝突時に働く力 $p_N\mathbf{n}$ による速度変化は、運動量保存則より次式のようになる。

$$\mathbf{u'}_A = \mathbf{u}_A + \frac{p_N\mathbf{n}}{m_A} \tag{3.3}$$

$$\omega'_A = \omega_A + \mathbf{I}_A^{-1}(\mathbf{r_A} \times p_N\mathbf{n}) \tag{3.4}$$

式 (3.1)、式 (3.2)、式 (3.3) および式 (3.4) より、衝突時の力による剛体 A 上の接点の速度変化は次式のようになる。

$$\begin{aligned}\mathbf{v'}_A &= \mathbf{v}_A + (\frac{\mathbf{n}}{m_A} + (\mathbf{I}_A^{-1}(\mathbf{r_A} \times \mathbf{n})) \times \mathbf{r}_A)p_N \\ &= \mathbf{v}_A + \mathbf{c}_A p_N \end{aligned} \tag{3.5}$$

この $\mathbf{c}_A$ は剛体 A の質量と姿勢および接点位置によって定まる定数ベクトルである。

ここで、接点における 2 剛体の相対法線速度 v_N を考える。v_N は剛体上の接点の速度 $\mathbf{v}_A$ および $\mathbf{v}_B$ と、衝突面の法線方向の単位ベクトル $\mathbf{n}$ によって次式のようになる。

$$v_N = \mathbf{n} \cdot (v_A - v_B) \tag{3.6}$$

ここでもし、$v_N < 0$ であったならば、これは接点において剛体同士がお互いに近づく方向に運動していることを意味している。同様にして、$v_N = 0$ ならば接触状態にあることを意味し、$v_N > 0$ ならばお互いに離れていくことを意味する。これと同様に、衝突後の相対法線速度 v'_N は次式のようになる。

$$\begin{aligned} v'_N &= \mathbf{n} \cdot (\mathbf{v'}_A - \mathbf{v'}_B) \\ &= \mathbf{n} \cdot (\mathbf{c}_A - \mathbf{c}_B)p_N + v_N \\ &= cp_N + v_N \end{aligned} \tag{3.7}$$

なお、c は定数である。

次に、図 3.12 に示すように、ある系において同時に n 点での衝突が起こった場合を考える。衝突前後の相対法線速度の変化は式 (3.7) を拡張して次式のようになる。

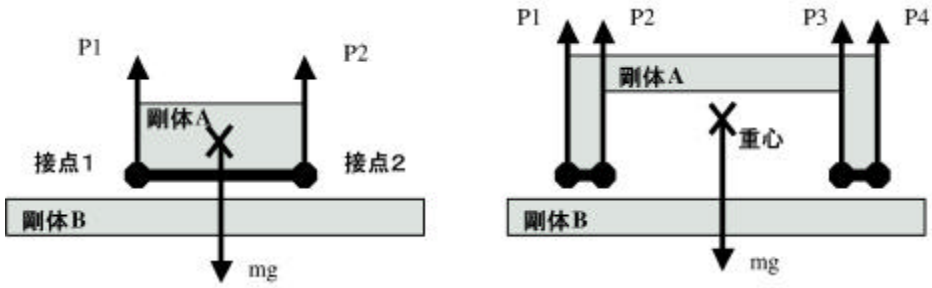


図 3.12: 多点接触時に接点に働く力のモデル化

$$v'_{Ni} = \sum_{j=1}^{n} A_{ij} p_{Nj} + v_{Ni} \tag{3.8}$$

ただし、v_{Ni} は接点 i における衝突前の相対法線速度、v'_{Ni} は接点 i における衝突後の相対法線速度、p_{Ni} は接点 i で働く力の大きさ、A_{ij} は接点 j における力が接点 i における相対法線速度に与える影響を表す定数で、剛体の質量と姿勢および接点の位置で定まる。

ここで、n 個の接点について、v_{Ni}、v'_{Ni}、p_{Ni}、A_{ij} をまとめて

$$\begin{aligned} \mathbf{v}' &= (v'_{N1}, \ldots, v'_{Nn})^T \in \mathbf{R}^n \\ \mathbf{v} &= (v_{N1}, \ldots, v_{Nn})^T \in \mathbf{R}^n \\ \mathbf{p} &= (p_{N1}, \ldots, p_{Nn})^T \in \mathbf{R}^n \\ \mathbf{A} &= (A_{ij}) \in \mathbf{R}^{n \times m} \end{aligned}$$

と表記すれば、式 (3.8) は次式のように簡単な形に書き直すことができる。

$$\mathbf{v}' = \mathbf{A}\mathbf{p} + \mathbf{v} \tag{3.9}$$

つまり、衝突による剛体の挙動の変化を知るためには、既知の値 $\mathbf{v}$、$\mathbf{A}$ から、接点に働く力 $\mathbf{p}$ および衝突後の速度 $\mathbf{v}'$ を計算すればよい。

跳ね返り係数 ϵ を用いれば、衝突前の相対法線速度 v_N と衝突後の相対法線速度 v'_N との間には $v' = -\epsilon v$ なる関係がある。これを多点衝突に拡張すると、$i = 1, \cdots, n$ に対して次の式が成り立つ。

$$v'_{Ni} \geq -\epsilon_i v_{Ni} \tag{3.10}$$

式 (3.10) において等式にならないのは、ある接点における衝突が別の接点における相対法線速度の変化に影響を及ぼす場合があるためである。

また、接点で働く力はお互いを押し返す方向にしか働かないので $p_{Ni} \geq 0$ である。$v'_{Ni} > -\epsilon_i v_{Ni}$ の場合には、接点 i 以外での力によって接点 i で物体はお互いに離れていくと考えられるので、接点 i では力は働かず、$p_{Ni} = 0$ である。この条件は次式のようになる。

$$p_{Ni}(v'_{Ni} + \epsilon_i v_{Ni}) = 0 \tag{3.11}$$

以上、式 (3.9)、式 (3.10) および式 (3.11) をまとめると次式のようになる。

$$\begin{aligned} \mathbf{v}' &= \mathbf{A}\mathbf{p} + \mathbf{v} \\ v'_{Ni} &\geq -\epsilon_i v_{Ni} \\ p_{Ni} &\geq 0 \\ p_{Ni}(v'_{Ni} + \epsilon_i v_{Ni}) &= 0 \end{aligned} \tag{3.12}$$

ここで

$$V_{Ni} = (1 + \epsilon_i) v_{Ni} \tag{3.13}$$

$$V'_{Ni} = v'_{Ni} + \epsilon_i v_{Ni} \tag{3.14}$$

なる $\mathbf{V}' = (V'_{N1}, \ldots, V'_{Nn})^T$ および $\mathbf{V} = (V_{N1}, \ldots, V_{Nn})^T$ を用いて v_{Ni} と v'_{Ni} を置き換えれば、式 (3.12) は次式のようになる。

$$\begin{aligned} \mathbf{V}' &= \mathbf{A}\mathbf{p} + \mathbf{V} \\ V'_{Ni} &\geq 0 \\ p_{Ni} &\geq 0 \\ p_{Ni} V'_{Ni} &= 0 \end{aligned} \tag{3.15}$$

式 (3.15) の条件の下で実行可能な解 $\mathbf{V}'$ および $\mathbf{p}$ を求める問題は、線形計画法における線形相補正問題であり、Dantzig のアルゴリズム [42] を用いて解くことができる。

3.1.6 剛体の位置と姿勢の更新

接点に働く力や重力等の外力が求まれば、運動方程式から剛体の運動量と角運動量の変化量を算出することができる。そこで、本項では、接点に働く力や重力に基づき剛体の運動量や角運動量および位置や姿勢を更新する方法について述べる。式を記述するにあたり以下に示す記号を用いる。

m	質量	ϵ	跳ね返り係数
$\mathbf{r}$	接点位置	$\mathbf{I}$	慣性テンソル
$\mathbf{x}$	位置	$\mathbf{q}$	姿勢
$\mathbf{v}$	速度	ω	角速度
$\mathbf{P}$	運動量	$\mathbf{L}$	角運動量
$\mathbf{F}$	力	$\mathbf{T}$	トルク

なお、慣性テンソルとは 3 次元空間内において剛体の回転のしやすさを表すパラメータであり、3 行 3 列 Matrix を用いて以下のように求めることができる。なお、剛体を質点の集まりと考え、質点 i の原点を基準とする絶対座標を x_i, y_i, z_i、質量を m_i と表す。

$$\mathbf{I} = \begin{pmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{pmatrix} \tag{3.16}$$

$$\begin{aligned} I_{xx} &= \sum m_i(y_i^2 + z_i^2) \\ I_{yy} &= \sum m(z_i^2 + x_i^2) \\ I_{zz} &= \sum m(x_i^2 + y_I^2) \\ I_{xy} &= I_{yx} = -\sum m_i x_i y_i \\ I_{yz} &= I_{zy} = -\sum m_i y_i z_i \\ I_{zx} &= I_{xz} = -\sum m_i z_i x_i \end{aligned}$$

2 つの剛体が衝突する際、接点に瞬間的に働く撃力を衝突時の速度、角速度、質量、慣性テンソル、および跳ね返り係数を用いて次式に示す。なお、添え字の A または B は、その記号が衝突している 2 つの剛体 A および B に関するものであることを意味する。

$$F = \frac{-(1+\epsilon)v_N}{m_A^{-1} + m_B^{-1} + \mathbf{n} \cdot (\mathbf{I}_A^{-1}(\mathbf{r}_A \times \mathbf{n})) \times \mathbf{r}_A + \mathbf{n} \cdot (\mathbf{I}_A^{-1}(\mathbf{r}_b \times \mathbf{n})) \times \mathbf{r}_A} \cdot \mathbf{n} \tag{3.17}$$

接点に瞬間的に働く撃力が求まれば剛体の速度や角速度の変化量を算出することができる。算出の際には、重心の移動にニュートンの運動方程式、回転にオイラーの運動方程式を用いるのが一般的である。まず、重心の移動を考えると、ニュートンの運動方程式は次式のようになる。

$$\mathbf{F} = \frac{d\mathbf{P}}{dt} = m\frac{d\mathbf{v}}{dt} \tag{3.18}$$

式 (3.17) および式 (3.18) より、容易に速度の変化量が求められるので、位置の変化量は次式のようになる。

$$\frac{d\mathbf{x}}{dt} = \mathbf{v} \tag{3.19}$$

次に、姿勢の回転を考える。オイラーの運動方程式は次式のようになる。

$$\mathbf{T} = \frac{d\mathbf{L}}{dt} = r \times \mathbf{F} \tag{3.20}$$

式 (3.17) および式 (3.20) より、角運動量の変化量が求まるので、角速度は次式のようになる。

$$\omega = \mathbf{I^{-1}L} \tag{3.21}$$

なお、本研究では、姿勢に quaternion と呼ばれる 4 元数を用いている。quaternion を用いた回転は 3 次元行列を用いる場合に比べ、計算機の有効桁数の問題から発生する誤差が少ないという利点がある。quaternion で表現した姿勢 $\mathbf{q}$ を角速度 ω で回転させた時の姿勢の変化量は次式のようになる。

$$\frac{d\mathbf{q}}{dt} = \frac{1}{2}(o, \omega) \cdot \mathbf{q} \tag{3.22}$$

以上に述べた並進運動と回転運動をまとめると、剛体の状態ベクトル $\mathbf{Y}$ とその微分 $\frac{d\mathbf{Y}}{dt}$ は次式のようになる。

$$\mathbf{Y} = \begin{pmatrix} \mathbf{x} \\ \mathbf{q} \\ \mathbf{P} \\ \mathbf{L} \end{pmatrix} \quad \frac{d\mathbf{Y}}{dt} = \begin{pmatrix} \mathbf{v} \\ \frac{1}{2} \cdot \omega \cdot \mathbf{q} \\ \mathbf{F} \\ \mathbf{T} \end{pmatrix} \tag{3.23}$$

本研究では、式 (3.23) に示す状態ベクトルを毎フレーム計算し、位置と姿勢を更新することで、剛体挙動をシミュレーションする。

3.1.7 高速かつ安定なシミュレーションの実行

本研究では、物体を剛体に限定し、衝突は瞬間的に発生すると仮定して、モデル化をおこなった。しかし、実世界では、衝突の際にはある時間継続して物体同士がお互いに力積を及ぼし合う現象が起こっている。そのため、このようなある一定時間継続して発生する現象を計算機を用いて離散的に処理した結果、剛体の挙動が不自然にな

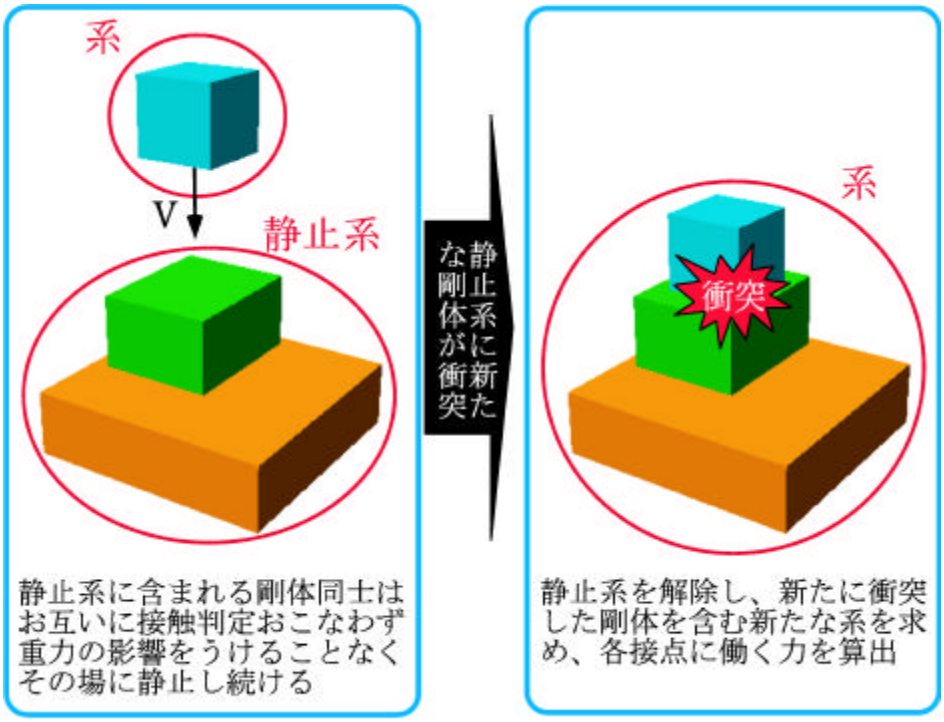


図 3.13: 静止系

る場合が生じる [43]。また、これまで述べてきた剛体挙動のモデル化手法を計算機に実装しただけでは、例えば、剛体がお互いに積み重なり短時間に衝突を繰り返すシミュレーションをおこなうと、シミュレーション速度が著しく低下する。そこで、高速かつ速度が安定した剛体挙動シミュレーションを実現するため、シミュレーションの問題点を検討し、いくつかの処理を実装した。以下、実装した処理について説明する。

静止系

系に属する全ての剛体が静止状態にある場合、本研究ではその系を便宜的に静止系と呼ぶ。静止系に含まれる剛体は、接触を維持しつつ静止しているため位置や姿勢を更新する必要がなく、その系に含まれる剛体同士の衝突判定や接点に働く力の算出を省略することができる。しかし、静止系に別の剛体が衝突すると、図 3.13 に示すように、新たな系を導出して接点に働く力を算出し位置と姿勢を更新する。接点に働く力の算出の結果、系に含まれる全ての剛体の状態が静止状態となれば、その系は再び静止系となり静止し続ける。

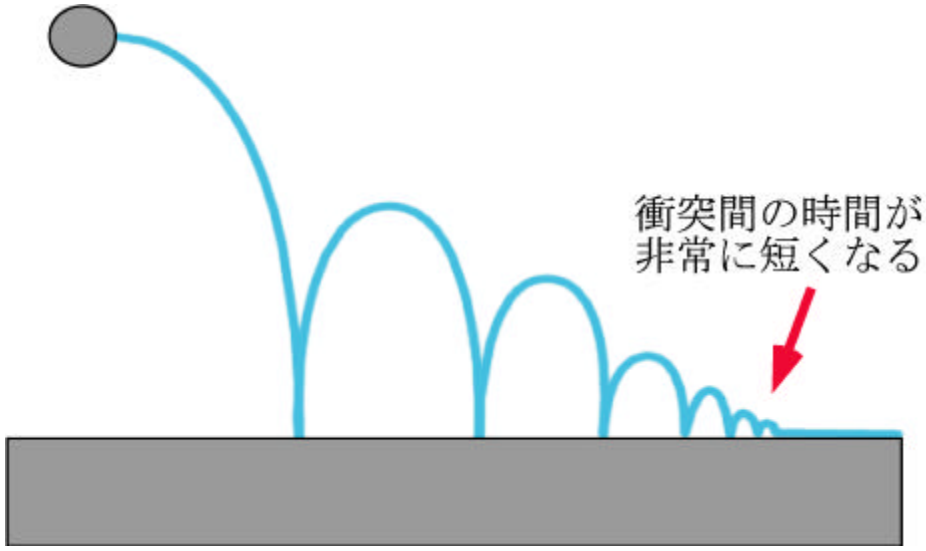


図 3.14: 衝突の減衰と接触への収束

衝突の減衰と接触への収束

図 3.14 に示すように、剛体が床に向かって自由落下して床との衝突を繰り返すと、跳ね返り係数が 1 ではない場合、衝突までの時間間隔が次第に短くなり、シミュレーションが進まなくなる。そこで、衝突の際の相対法線速度がある閾値以下だった場合、その衝突を強制的に接触へ収束させる機能を実装した。接触に収束した剛体を含む系が静止系になると、自分の系に含まれる剛体に対する接触判定を省略することができ、短時間に衝突を繰り返しシミュレーションが進まなくなる現象を回避できる。

なお、閾値は次のように算出する。剛体が床に衝突した時の相対法線速度を v_N、重力加速度を g とすると、次回の衝突までの時間 t_n は次式のようになる。

$$t_n = \frac{v_N - (-v_N)}{g} = \frac{2v_N}{g} \tag{3.24}$$

次回の衝突までの時間 t_n が時間積分をおこなう時間幅 t 以下の値になると、同じ剛体同士の衝突が連続して発生し、シミュレーション速度が低下する。このような現象を回避するため、閾値を $gt/2$ に設定する。

衝突後の追い越し現象の回避

本来は連続的な現象である剛体の挙動を計算機で離散的に処理しているため、例えば、静止している質量が非常に小さい剛体に質量が非常に大きい剛体が高速で衝突した場合、質量の大きな剛体が質量の小さな剛体を追い抜いてしまう現象がおこる。

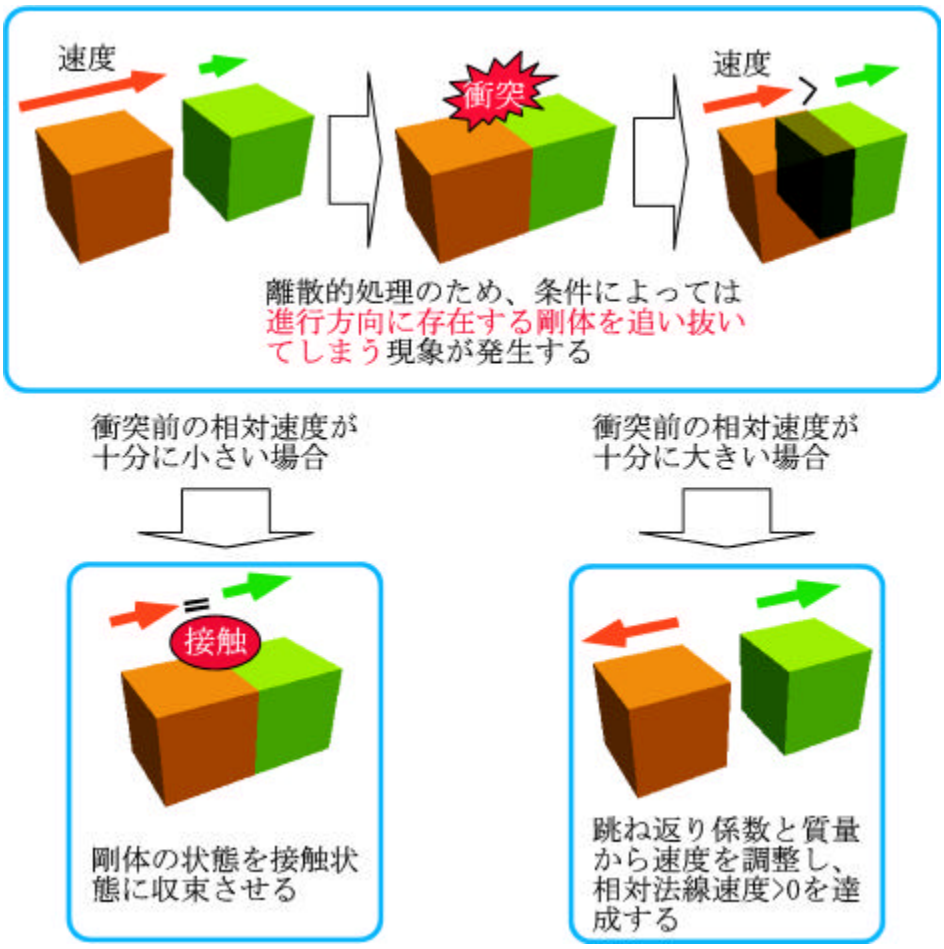


図 3.15: 追い越し現象の対処

そこで、このような現象が発生した際、図 3.15 に示すように、衝突前の相対法線速度が先に定めた閾値以下だった場合は、剛体の状態を接触状態に収束させ、等速で運動するように速度を調整する。また、衝突前の相対法線速度が閾値以上だった場合は、跳ね返り係数とお互いの質量に応じて、自然な跳ね返りに見えるように速度を調整する。

2 個の剛体が衝突する瞬間の時刻を求めるアルゴリズム

本研究では、前述のように、ポリゴンレベルの厳密な衝突判定に Voronoi Clip を用いる。Voronoi Clip は、2 つの剛体の形状、位置、姿勢の情報を元に、それらの剛体が干渉しているかどうかを判定し、干渉している場合はその干渉の深さを結果として返す。しかし、現時点での剛体の干渉の深さだけでは、2 個の剛体が衝突する瞬間の時刻を求めることはできない。また、衝突する直前の剛体の位置と速度が判明していたとしても、複雑な形状の剛体が回転しながら衝突する可能性がある状況では、2 つの剛体が衝突する瞬間の時刻を解析的に正確に求めることは困難である。

そこで、本研究では、前回の時間積分をおこなった時刻 t_0、及び、その時刻における剛体の位置、姿勢、速度、角速度と、衝突判定をおこなう時刻 t_c、及び、その時刻における剛体の位置、姿勢、速度、角速度、そして前回の時間積分をおこなった時刻 t_0 と衝突判定をおこなう時刻 t_c の時間差 t の情報を用いて、2 つの剛体が衝突する瞬間の時刻を求める。図 3.16 に、衝突する瞬間の時刻を求めるアルゴリズムを示す。アルゴリズムの流れは以下のようになる。

1. 1 回目の衝突判定として、衝突判定時刻 $T = t_0 + \frac{t}{2}$ における剛体同士の衝突判定をおこなう。

2. 剛体の衝突が検知された場合には、2 回目の衝突判定として衝突判定時刻 $T = t_0 + \frac{t}{2} - \frac{t}{4}$ における衝突判定をおこない、衝突が検知されなかった場合には、衝突判定時刻 $T = t_0 + \frac{t}{2} + \frac{t}{4}$ における衝突判定をおこなう。

3. 以下、同様に、n 回目の衝突判定において衝突が検知された場合には、$n+1$ 回目の衝突判定として衝突判定時刻 $T_{n+1} = T_n - \frac{t}{2^n}$ における衝突判定をおこない、衝突が検知されなかった場合には衝突判定時刻 $T_{n+1} = T_n + \frac{t}{2^n}$ における衝突判定をおこなう。

4. n 回目の衝突判定が終了したときの次回の衝突判定時刻 T を、2 つの剛体が衝突した瞬間の時刻とする。

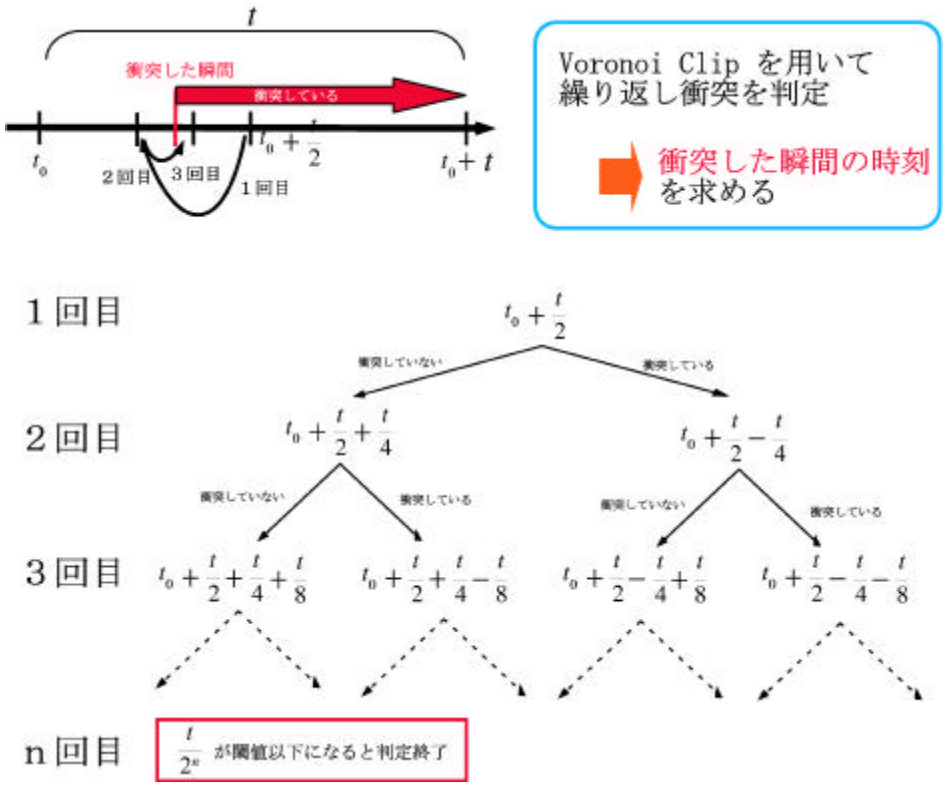


図 3.16: 衝突する瞬間の時刻の算出

ただし、以上に述べたアルゴリズムは、衝突判定をおこなう剛体同士が時刻 t_0 の時に衝突していないことが前提となる。しかし、実際のシミュレーションでは、例えば図 3.17 に示すように、時刻 t_0 の時に衝突した剛体が、再び時刻 t_0+t の時に衝突する場合がある。このような衝突がおこった場合、先に示したアルゴリズムでは、衝突する瞬間の時間を正確に求めることができない。そこで本研究では、図 3.17 に示すような衝突に対しては、これを衝突していないと仮定してシミュレーションを続けることにする。

3.2 単一の計算機環境における剛体挙動シミュレーションシステム

これまで述べた剛体挙動のモデル化手法を計算機上に実装し、単一の計算機環境で動作する剛体挙動シミュレーションシステムを構築した。これは、単一の計算機環境でシミュレーションを実行する際の各種処理の計算負荷を計測し、剛体挙動シミュレーションの効率的な分散・並列化を実現するための検討をおこなうためである。

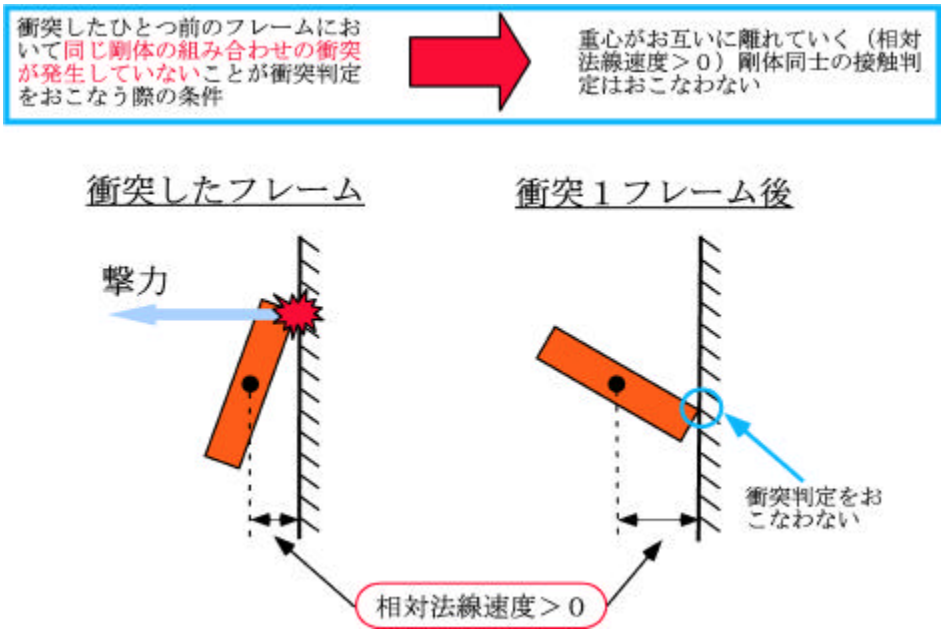


図 3.17: 衝突判定をおこなわないケース

表 3.2: 剛体挙動シミュレーションシステムの構築環境 (単一の計算機環境)

CPU	PentiumIII 1.26GHz × 2
メモリ	768MB
ネットワーク	100BASE-T
OS	RedHatLinux 7.1
開発言語	C 言語、OpenGL

本節では、まず、単一の計算機環境上に構築した剛体挙動シミュレーションシステムの構成と各部の機能について説明する。次に、本システムにおける仮想空間の構築方法とこれまでの構築方法と比べた場合の利点について説明する。最後に、本システムを用いておこなった負荷実験について説明し、効率的な分散・並列化を実現するための検討をおこなう。

3.2.1 剛体挙動シミュレーションシステムのシステム構成

3.1 節で述べた剛体挙動のモデル化手法を計算機上に実装し、単一の計算機環境で動作する剛体挙動シミュレーションシステムを構築した。使用したハードウェア環境およびソフトウェア環境を表 3.2 に示す。

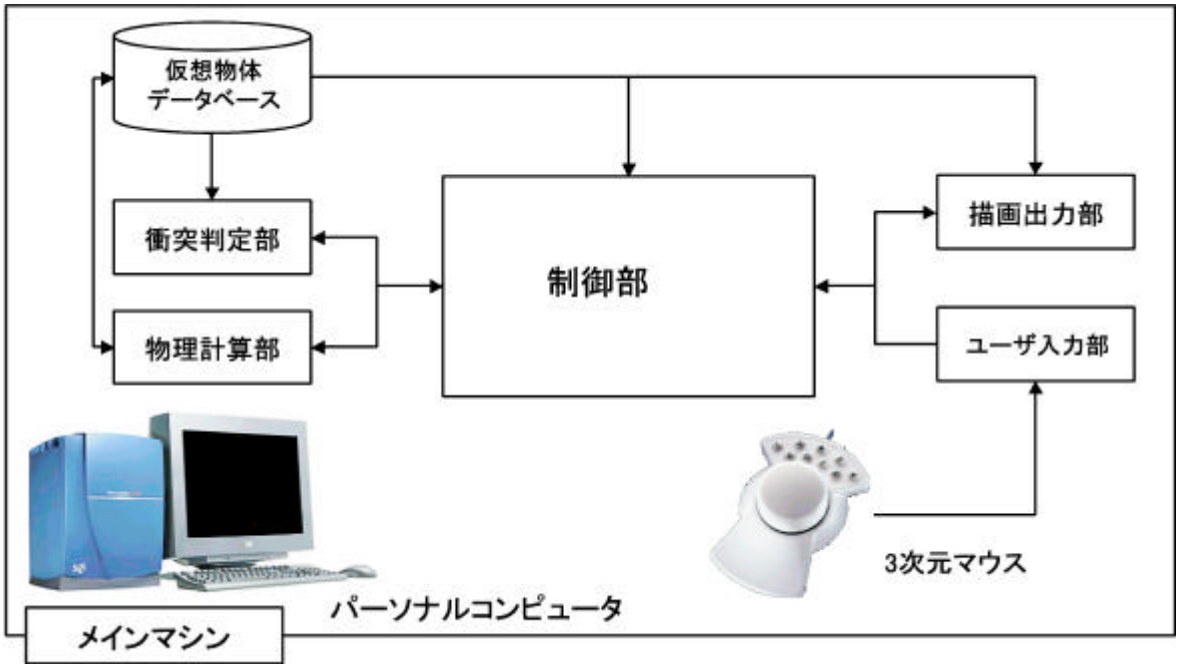


図 3.18: システム構成（単一の計算機環境）

本システムの構成を図 3.18 に示す。本システムは、剛体の情報を格納する仮想物体データベースと、データベースに基づきシミュレーションを実行する 5 つの処理部から構成される。

- 制御部

  制御部は、システム全体を制御し、データの流れとその処理方法を決定する。例えば、衝突判定部にて剛体同士の衝突が検知されれば、物理計算部にその結果をメモリから読み出すように命令し、接点に働く力を算出させる。また、シミュレーションが一定時間を経過する毎に描画出力部に画面を更新するように命令する。他にも、シミュレーションを開始する前の設定ファイルの読み込み、剛体の状態判定、系や静止系の導出、シミュレーション時間の制御等をおこなう。

- 衝突判定部

  衝突判定部は、剛体同士の衝突判定をおこなう。まず、球近似による簡易な衝突判定をおこない、衝突している可能性がある剛体の組み合わせを抽出する。その際、同じ静止系に含まれる剛体同士の組み合わせは厳密な衝突判定をおこなう必要がないため除外する。次に、ポリゴンレベルの厳密な衝突判定をおこない、衝突している剛体の組み合わせと剛体間の接点を正確に算出する。剛体同士の干渉

が起こっている場合、前回の時間積分から衝突する瞬間までの正確な時間幅 Δt を算出する。

- 物理計算部

  物理計算部は、接点に働く力の算出と位置と姿勢の更新をおこなう。まず、接点に働く力の算出では、系に含まれる全接点の条件式を同時に満たす解を Dantzig のアルゴリズムを用いて求め、接点に瞬間的に働く撃力と持続的に働く接触力を算出する。次に、位置と姿勢の更新では、接点に働く力を用いて剛体の運動量と角運動量を算出し、位置と姿勢を更新する。その際、連続的な現象である物理現象を計算機を用いて離散的に処理することによる不具合が発生すると、位置や姿勢、速度や角速度をユーザが違和感を感じないよう適切に調整する。

- ユーザ入力部

  ユーザ入力部は、3 次元マウスを用いたユーザの入力を処理する。ユーザは、3 次元マウスを用いて指や工具の形をした剛体を操作し、仮想空間内に配置した剛体に直接の操作を加えることができる。3 次元マウスによる入力値は、外力の方向と大きさに変換され、重力と同様に処理される。また、合成された 3 次元画像の視点を変更することもできる。

- 描画出力部

  描画出力部は、物理計算部で算出された剛体の位置と姿勢の情報、DXF 形式の形状ファイルによる形状情報、RGB 形式の画像ファイルによるテクスチャ情報に基づき、画面を更新する。

- 仮想物体データベース

  仮想物体データベースには、剛体の初期位置や初期速度等の情報を記した仮想空間定義ファイル、剛体の質量や慣性テンソル等の情報を記した仮想物体定義ファイル、剛体の形状情報を記した DXF 形式の形状ファイル、剛体の表面材質を表現する RGB 形式の画像ファイル、剛体同士のポリゴンレベルの接触判定に用いる ptree 形式の形状ファイルが格納されている。これらデータベース内の情報は、シミュレーション開始時に制御部によって計算機のメモリに読み込まれる。

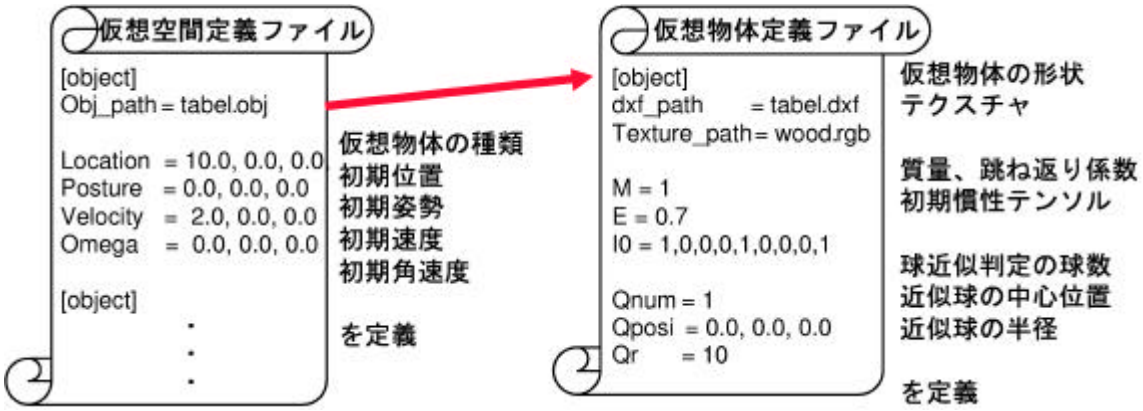


図 3.19: 定義ファイル

3.2.2 仮想空間の構築方法

本システムでは、仮想空間内に配置する剛体の種類や初期位置、初期速度等を定義する仮想空間定義ファイルと、質量や慣性テンソル等のシミュレーション中に変化しない剛体固有の情報を定義する仮想物体定義ファイルの 2 種類の定義ファイルを記述し、さらに商用アプリケーションを用いて DXF 形式の形状ファイルと RGB 形式のテクスチャファイルを用意することで、仮想空間を構築する。図 3.19 に定義ファイルの一例を示す。仮想空間を構築するには、まず、仮想空間定義ファイルに、仮想空間内に配置する剛体の仮想物体定義ファイル名と初期位置、初期姿勢、初期速度、初期角速度を記述する。複数の剛体を仮想空間に配置する場合、この記述を配置する剛体の数だけ繰り返す。次に、仮想物体定義ファイルに、DXF 形式の形状ファイルのファイル名、RGB 形式のテクスチャファイルのファイル名、質量、跳ね返り係数、初期位置における慣性テンソル、簡易な接触判定に用いる近似球の個数、近似球の重心に対する相対位置、近似球の半径を記述する。近似球を複数設定する際は、相対位置と半径の情報を近似球の数だけ続けて記述する。なお、異なる種類の剛体を仮想空間内に配置する場合、種類の数だけ仮想物体定義ファイルを記述する必要があるが、同じ種類の剛体を仮想空間内に複数配置する際には、1 つの仮想物体定義ファイルを繰り返し用いることができる。

Rule-base の手法では、仮想空間内において剛体が取りうる状態とその各状態における挙動の全てをユーザの手でモデル化し設定する必要がある。そのため、仮想空間を構築する際に要する労力は、構築する仮想空間が大規模複雑になるほど膨大になる。しかし、本システムでは、Physically-base の手法を用いて剛体の挙動を物理法則に基づき自動的に計算し、剛体の状態を自動的に判別しているため、ユーザの手で状態や挙

動を設定する必要が無く、仮想空間の構築に要する労力を低く抑えることができる。また、定義ファイルには質量や位置といった簡単なパラメータしか記述する必要が無く、人工現実感技術に対する特別な知識が無くても容易に記述することができる。更に、仮想空間内に同じ仮想物体を複数構築する場合、一度定義した仮想物体定義ファイルを再利用することができるため、構築に要する労力は仮想空間の規模に必ずしも比例しない。そのため、本研究が想定している機器保修の訓練環境のように大規模複雑な仮想空間も、本システムでは比較的容易に構築することができる。

構築した仮想空間の例として、図 3.20 に作業机の転倒シミュレーションを示す。このように、本システムは、構築した仮想空間を物理法則に基づいてシミュレーションし、3 次元アニメーションとして出力する。

3.2.3　シミュレーション負荷の評価

本研究では、従来は単一の計算機環境で実現されていた剛体挙動シミュレーションの処理の一部を、PC クラスタを構成する計算用ノードマシンに分散し並列処理することを目的とする。しかし、分散・並列化が可能な全ての処理を分散・並列処理すると、計算機間の通信量が膨大となり、シミュレーション速度が低下すると予想される。そのため、効率的な分散・並列処理を実現するためには、計算負荷と通信負荷のバランスを考慮しなくてはならない。

そこで、剛体挙動シミュレーションの処理のうち、どの処理にどの程度の計算量が生じるのか調べるため、仮想空間内において球形状の剛体 27 個がお互いに衝突を繰り返すシミュレーションを単一の計算機環境で実行し、時間積分や接触判定等の各処理にかかる負荷を計測する実験をおこなった。実験の様子を図 3.21 に示し、実験の条件を表 3.3 に示す。なお、システムの環境は先に表 3.2 において示したとおりである。

本研究において剛体挙動シミュレーションの分散・並列化を実現する理由の 1 つは、計算負荷が最大時の各処理を複数の計算機に分散し、シミュレーション速度を一定の速度に安定させることである。そこで本実験は、計算負荷が最大の時の各処理の負荷の割合を調べることを目的とし、システムに十分な負荷がかかるように条件設定した。剛体として球形状を用いた理由は、ポリゴン数が比較的多く、ポリゴンレベルの厳密な接触判定の処理に十分な負荷を課すことが期待できるためである。また、多数の剛体が限られた空間で連続的に衝突を起こすように剛体の数と仮想空間の広さを設定した。終了条件として一連の処理を 5000 フレーム繰り返すように設定したが、これは条件設定した仮想空間における平均的な計算負荷を計測するのに十分な処理数である。

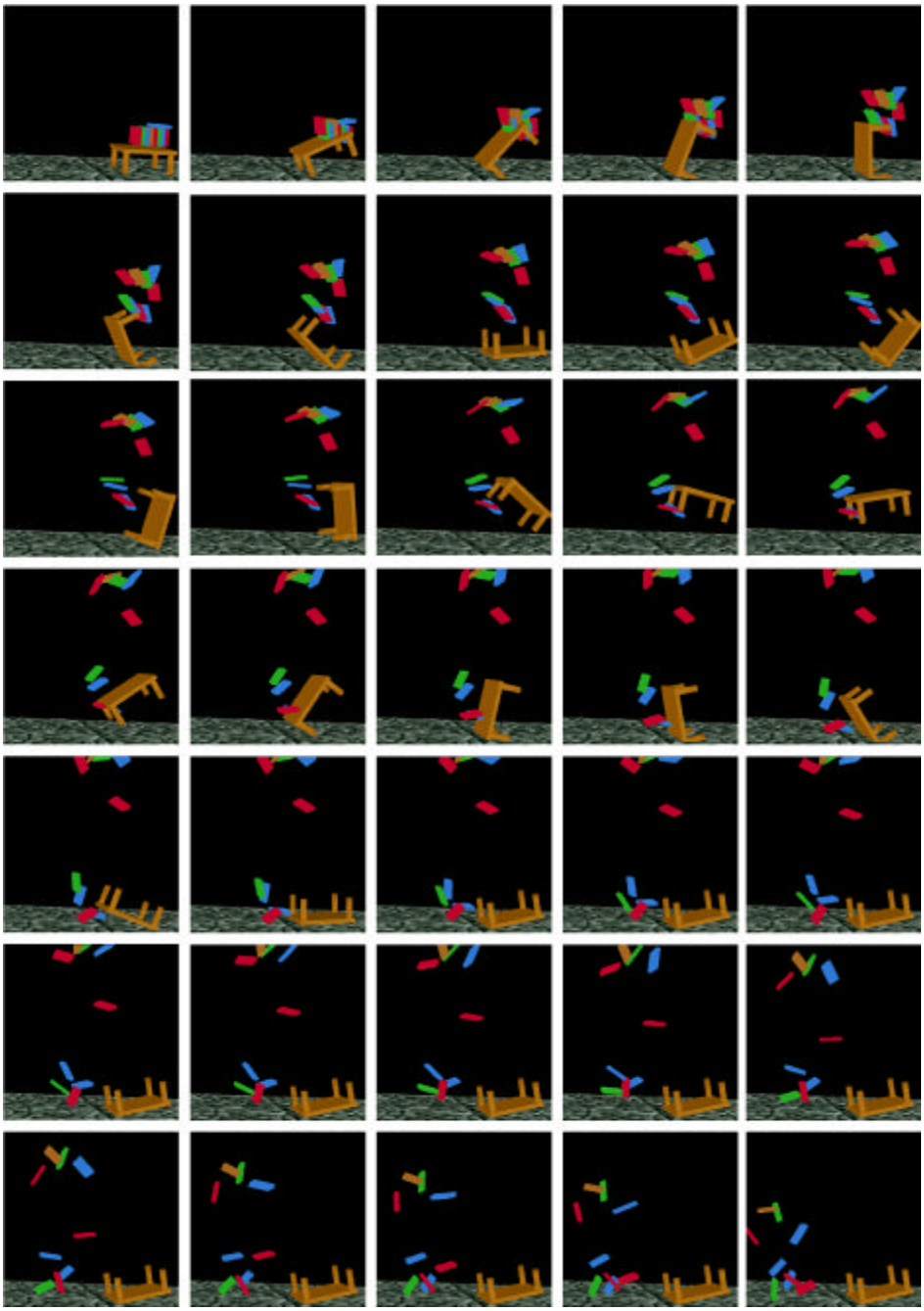


図 3.20: 作業机の転倒シミュレーション

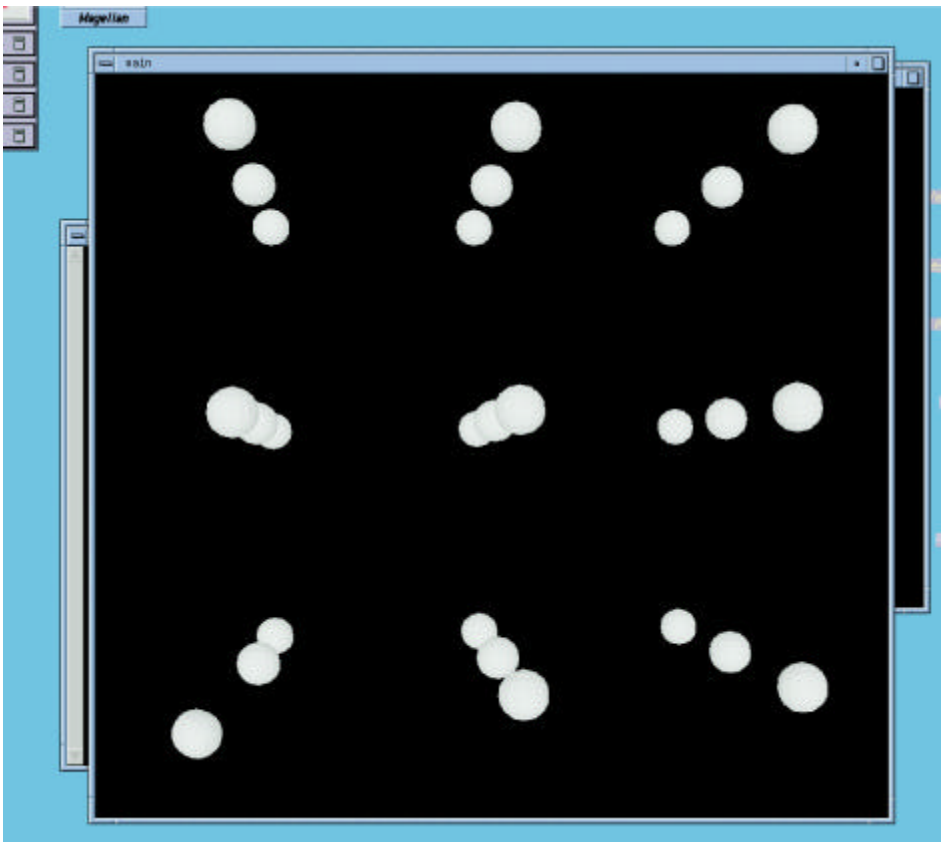


図 3.21: 負荷実験の様子

表 3.3: 負荷実験の条件

剛体の種類	球形状 (半径 10cm、224 ポリゴン、跳ね返り係数 1.0)
剛体の数	27 個
剛体の配置	x,y,z 軸それぞれの方向について 3 個ずつ
	隣接する剛体と 100cm の距離をあけて配置
仮想空間	広さ $(400cm \times 400cm \times 400cm)$ の直方体状
	空間の端は跳ね返り係数 1.0 の壁面
終了条件	一連の処理を 5000 フレーム繰り返した場合
時間積分幅	0.01 秒

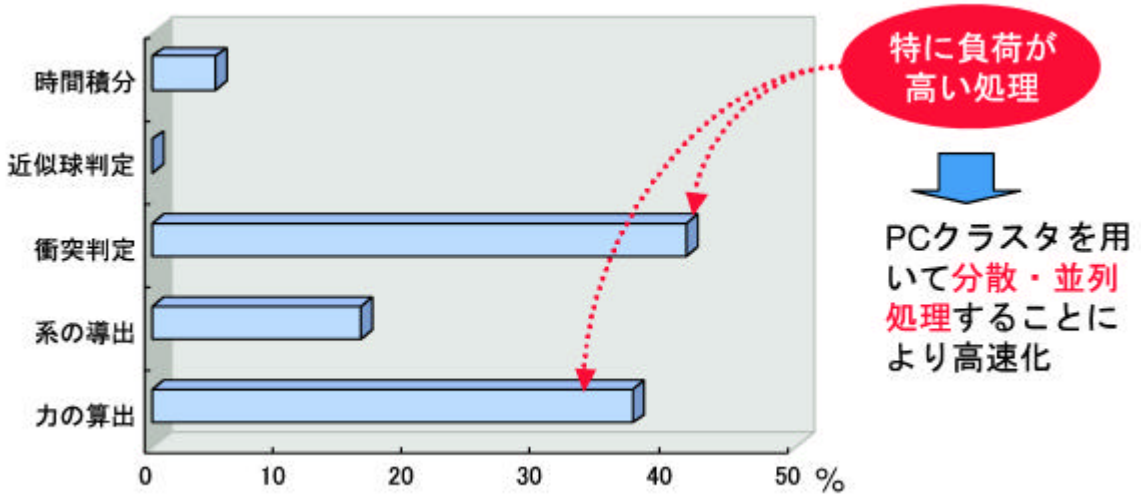


図 3.22: 各処理にかかる負荷の割合

負荷実験の結果を図 3.22 に示す。縦軸は処理を、横軸はシミュレーション全体の計算負荷のうちその処理が占める割合を示す。負荷実験の結果から、高負荷時には剛体挙動シミュレーションに必要な処理のうち、ポリゴンレベルでおこなう厳密な衝突判定と Dantzig のアルゴリズムを用いた接点に働く力の算出に特に高い計算負荷がかかっていることを確認した。この 2 つの処理だけで計算負荷全体の約 80 %を占めていることから、分散・並列化を実現すれば計算負荷が最大の時のシミュレーション速度の低下を抑制する上で大きな効果があると期待できる。逆に、球近似による簡易な衝突判定と時間積分は、計算負荷が低いため分散・並列化の必要はない。なお、系の導出は、計算負荷全体の 17 %を占めており決して計算負荷が低い処理ではないが、先に述べた 2 処理に比べて計算負荷が約半分である点や、分散・並列化が難しい処理である点を考慮して本研究では分散・並列化の対象としない。

3.3 本手法の限界

本研究は、Rule-base の手法と Phisically-base の手法を用いた理想的な機器保修の訓練環境を実現するための基礎研究として、まず、Physically-base の手法に基づいた剛体挙動のリアルタイムシミュレーションを目標に定め、剛体挙動のモデル化と計算機への実装をおこなった。そのため、実際の機器保修の訓練環境で実現が望まれるシミュレーションの機能や物理現象・物理法則の全てを本手法が実現したわけではない。

機器保修の訓練環境を実現する際に必要な物理法則・物理現象は表 2.2 にて示したとおりである。本研究は表 2.2 の左側で示した物理法則・物理現象を実現し、物体の衝突や崩落といった撃力と接触力を用いたシミュレーションを可能とした。しかし、表 2.2 の右側で示した物理法則・物理現象は実現していないため、現時点ではバルブ等の軸を中心とした回転を伴う機器操作や物体の変形といったシミュレーションはできない。

第 4 章 PCクラスタを用いた剛体挙動シミュレーション

本章では、まず、分散・並列処理の分野で使用される用語やその定義について説明する。次に、本研究で開発した剛体挙動シミュレーションの処理の一部を分散・並列化する手法について説明する。最後に、本手法を用いて構築した PC クラスタを用いた剛体挙動シミュレーションシステムについて説明する。

4.1 分散・並列処理

一人では長時間を要する作業も集団で取り組めば短時間に完遂することができる。分散・並列処理とは、この考え方を計算機に当てはめた概念であり、「コンピュータで一連の処理を複数台の処理装置で同時に並行しておこなうこと」(大辞林) と定義される。具体的には、図 4.1 に示すように、1 つの大きな計算タスクを計算機 (CPU) の数と同じ数に分割し、ネットワーク (バス) を介して接続された個々の計算機 (CPU) に分散して割り当て、同時に平行して処理し、結果を 1 つにまとめ出力する、計算手法を指す。なお、分散・並列処理に似た用語として並行処理があるが、分散・並列処理が複数台の計算機がそれぞれのタスクを完全に平行して処理する計算手法を指すのに対し、並行処理は 1 台の計算機が複数のタスクを交互に実行し擬似的に平行して処理しているように見せる計算手法を指す。つまり、分散・並列処理は複数の計算機を用いる処理で

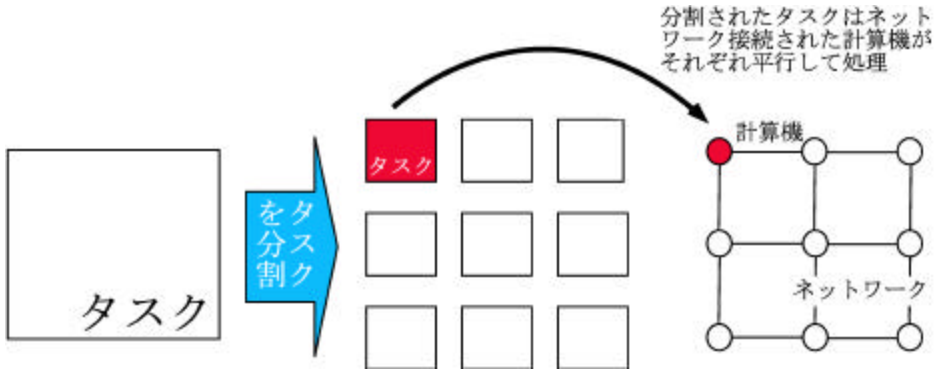


図 4.1: 分散・並列処理

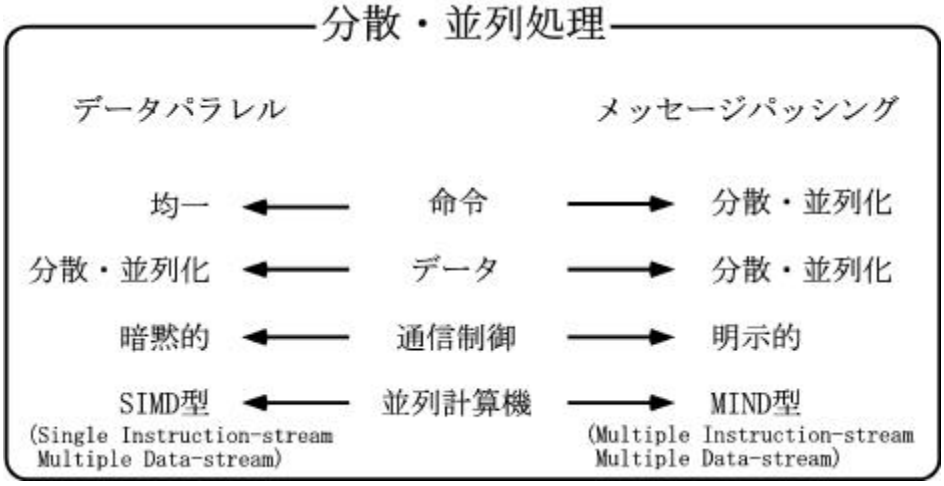


図 4.2: データパラレルとメッセージパッシング

あり、並行処理は 1 台の計算機を用いる処理である。書籍によっては、分散・並列処理と並行処理が混同して用いられている場合が見受けられるが、本研究では以上に述べたように明確に区別して用いる。

分散・並列処理は、1965 年のセイモア=クレイによる並列処理コンピュータの設計を原点とする。セイモア=クレイは、分散・並列処理の基礎であるベクタプロセッシング技術の生みの親であることから「スーパコンピュータの父」と呼ばれ、今日の分散・並列処理を確立した 1 人とされている [44]。1970 年代に入ると、スーパコンピュータを専門に開発・販売する企業が現れ、大学や企業の研究機関を中心にスーパコンピュータの導入が加速し、分散・並列処理の技術は急速に発展した。しかし、スーパコンピュータは高価な機器であったため、画像処理や偏微分方程式の求解計算等その用途が非常に限られてしまい、広く普及するまでには至らなかった。しかし、1990 年代に入ると、比較的安価なマルチプロセッサマシンの登場に伴い、シミュレーションや数値解析の分野をはじめとして様々な分野で分散・並列処理の応用が進められた。そして現在、PC クラスタの登場により、高価なスーパコンピュータを導入することが難しかったデータベース処理や暗号解読といった分野にまで分散・並列処理は広まりつつある。以下、分散・並列処理の種類や用語について詳しく説明する。

分散・並列処理は大きく分けて、データパラレルと呼ばれる手法と、メッセージパッシングと呼ばれる手法の 2 つの手法が存在する。図 4.2 に両手法の特徴を示す。データパラレルは、それぞれの計算機が同じ命令で異なるデータを処理する手法であり、主に大規模シミュレーションに用いられる。一方、メッセージパッシングは、それぞれの計算機が異なる命令で異なるデータを処理する手法であり、主に数値解析に用いられ

る。データパラレルは、データは分散・並列化されるが命令は分散・並列化されないため、PC クラスタをはじめとするメモリ分散型の並列計算機に適しており、メッセージパッシングは、データのみならず命令も分散・並列化されるため、スーパコンピュータをはじめとするメモリ共有型の並列計算機に適している。なお、本研究は、PC クラスタに適したデータパラレルを用いて分散・並列化を実現する。データパラレルを用いたシステムは、全ての計算機が同時に同じ命令を実行するため、個々の計算機を明示的に管理する必要がなく、設計が比較的容易である。

データパラレルを用いた効率的な分散・並列化について、再び人間による作業を例に説明する。集団で作業に取り組む際、無計画にあまりにも多くの人間が集まると、お互いの意志の疎通が困難となる場合が考えられる。また、やるべき仕事を見つけられない人が多数生じ、作業効率が低下することも考えられる。そのため、集団で効率的に作業に取り組むためには、大きな作業をどのように小さな作業に分割するのか、分割した作業を能力の異なる一人一人にどのよう割り当てるのか、の 2 点が重要になる。計算機によるデータパラレルを用いた分散・並列処理も同様で、多くの計算機を使うことが必ずしも計算速度の高速化に結びつかず、どのタスクを分散・並列化させるのか、分散・並列化したタスクをどの様に割り当てるのか、の 2 点が効率的な分散・並列処理を実現する上で特に重要となる。

まず、どのタスクを分散・並列化させるのか、を検討するための指標として、並列化率と粒度について説明する。

- 並列化率

  図 4.3 に単一の計算機を用いた逐次処理、および複数の計算機を用いた分散・並列処理をそれぞれ実行した際の計算時間を示す。このとき、単一の計算機を用いた時の処理時間を t_1、n 個の計算機を用いた時の処理時間を t_n とすると、分散・並列化によるスピードアップ比率 S_n は次式のように定義される。

  $$S_n = \frac{t_1}{t_n} \tag{4.1}$$

  このスピードアップ比率 S_n を用いると、n 個のプロセッサによる並列効率 E_n は次式のように定義される。

  $$E_n = \frac{S_n}{n} = \frac{t_1}{n \cdot t_n} \tag{4.2}$$

  並列効率 E_n は、分散・並列化がどれほど効率的に実現されているかを示す指標であり、上限値を持つ。図 4.3 に示すように、分散・並列化を実現するタスクに

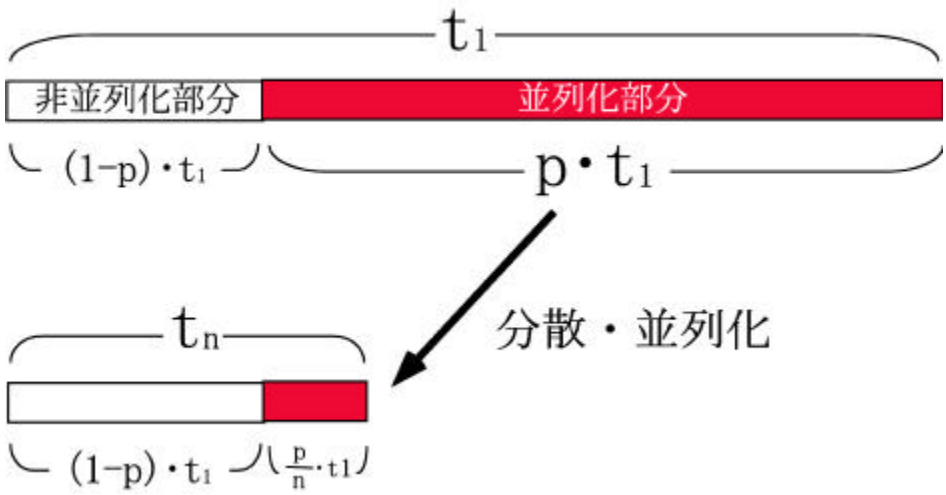


図 4.3: 並列効率

は分散・並列化が可能な部分と分散・並列化が不可能な部分に分類できる。タスク全体に占める分散・並列化が可能な部分の割合を並列化率といい、並列化率を $p(0 \leq p \leq 1)$ とすると n 個の計算機による計算時間 t_n は次式のようになる。

$$t_n = \frac{p \cdot t_1}{n} + (1-p)t_1 = (\frac{p}{n} + (1-p))t_1 \tag{4.3}$$

式 (4.1) および式 (4.3) より、スピードアップ比率 S_n は次式のようになる。

$$S_n = \frac{1}{\frac{p}{n} + (1-p)} \tag{4.4}$$

よって、スピードアップ比率 S_n と並列効率 E_n の上限値は次式のようになる。

$$\lim_{n \to \infty} S_n = \frac{1}{1-p} \tag{4.5}$$

$$\lim_{n \to \infty} E_n = \frac{1}{n(1-p)} \tag{4.6}$$

式 (4.5) を Ware の法則と言い、並列化率 p が高いタスクであればあるほど、分散・並列化による高速化の効果が高くなることを示す。並列化率 p とプロセッサ数 n に対するスピードアップ比率 S_n の変化を表 4.1 に示す。

- 粒度

並列化率が同じタスクでも、図 4.4 に示すように、分散・並列化が可能な部分の分布状況は大きく異なる。タスク全体に対する分散・並列化が可能な部分の分布状況を粒度と言い、分散・並列化が可能な部分が細かく分散して存在している分

表 4.1: 並列化率とプロセッサ数に対するスピードアップ比率

p	n=1	n=2	n=4	n=16	n=256	n=∞
0.500	1.00	1.33	1.60	1.88	1.99	2.00
0.900	1.00	1.82	3.08	6.40	9.66	10.00
0.990	1.00	1.98	3.88	13.91	72.11	100.00
0.999	1.00	2.00	3.99	15.76	203.98	1000.00
1.000	1.00	2.00	4.00	16.00	256.00	∞

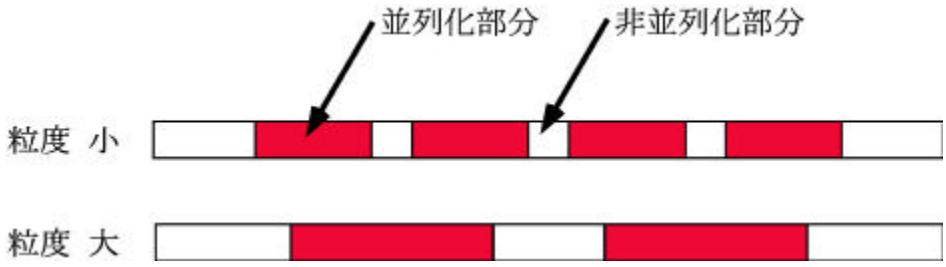


図 4.4: 粒度

布状況を粒度が小さい、かたまって存在している分布状況を粒度が大きいという。並列化率が同じタスクの場合、粒度が大きいタスクは粒度が小さいタスクに比べ通信や同期待ちに要する時間が少なくなるため、より効率的な分散・並列化を実現できる。

並列化率が高く粒度が大きいタスクは、効率的な分散・並列化が実現可能なタスクである。逆に言えば、効率的な分散・並列化を実現するためには、並列化率が高く粒度が大きくなるように、全体のタスクのうち分散・並列処理をおこなう部分とおこなわない部分を決定しなければならない。しかし、本研究のような剛体挙動シミュレーションの場合、画面の描画や系の算出等、分散・並列化が難しく 1 台の計算機でおこなわなければならない処理が多数存在する。そのため、並列化率が高くなるように多数の処理を分散・並列化すると、粒度が小さくなり計算機間の通信回数が増大する。つまり、分散・並列化をおこなう処理の数を可能な限り少なくすることで粒度を大きく保ち、かつ並列化効率が高くなるように分散・並列化をおこなう処理を決定しなければならない。

タスク全体のうち分散・並列化をおこなう部分が決定すると、そのタスクをどのように割り当てるかが重要となる。例えば、全ての計算機の性能が均一である場合、タス

クを可能な限り均等に割り当てることが望ましいが、計算機の性能が均一ではない場合、タスクを計算機の性能に応じて割り当てることが望ましい。なお、本研究は、大学の研究室のように計算機の性能が均一ではない環境を想定しているため、計算機の性能に応じて動的に処理を割り当てる必要がある。

4.2 剛体挙動シミュレーションの分散・並列手法

本節では、まず、剛体挙動シミュレーションの処理を分散・並列化する基本方針について述べ、処理の負荷の推定方法や計算機の計算性能の推定方法について述べる。次に、分散・並列化した剛体挙動シミュレーションの処理の流れについて述べる。そして、本研究で定めた計算機ノード間の通信プロトコルや処理の動的割り当て手法について説明する。

4.2.1 剛体挙動シミュレーションの分散・並列化の基本方針

前述のように、効率的な分散・並列化を実現するためには、シミュレーションのどのタスクを分散・並列化するのか、計算機にどの様にタスクを割り当てるのか、の 2 点が重要となる。

分散・並列処理するタスクの種類の検討

3.2.3 項では、負荷実験の結果から「ポリゴンレベルの衝突判定」と「接点に働く力の算出」の 2 つの処理を分散・並列化することが、シミュレーション速度を一定に保つ上で望ましいことを述べた。この 2 つの処理を分散・並列化すると仮定すると、3.2.3 項で述べた負荷実験の結果、全体の処理に対する「ポリゴンレベルの衝突判定」と「接点に働く力の算出」の占める割合は、41.48% および 37.35% であったため、負荷実験で使用した仮想空間に対する並列化率は $p = 0.78$、分散・並列化によるスピードアップ比率 S_n の上限値は $\lim_{n\to\infty} S_n = 4.54$ となり、十分に高い値であるということができる。また、この 2 つの処理は 1 フレームの間に一度ずつしか実行されないため、通信回数は計 4 回となり、粒度は十分に大きくなる。このように、「ポリゴンレベルの衝突判定」と「接点に働く力の算出」を分散・並列化すれば、高い並列化率と大きな粒度を実現することができるため、負荷を分散しシミュレーション速度を一定に保つことが可能であるばかりか、シミュレーション速度を全体的に向上させることができる。また、シミュレーション速度が全体的に向上するため、より大規模複雑な仮想空間が

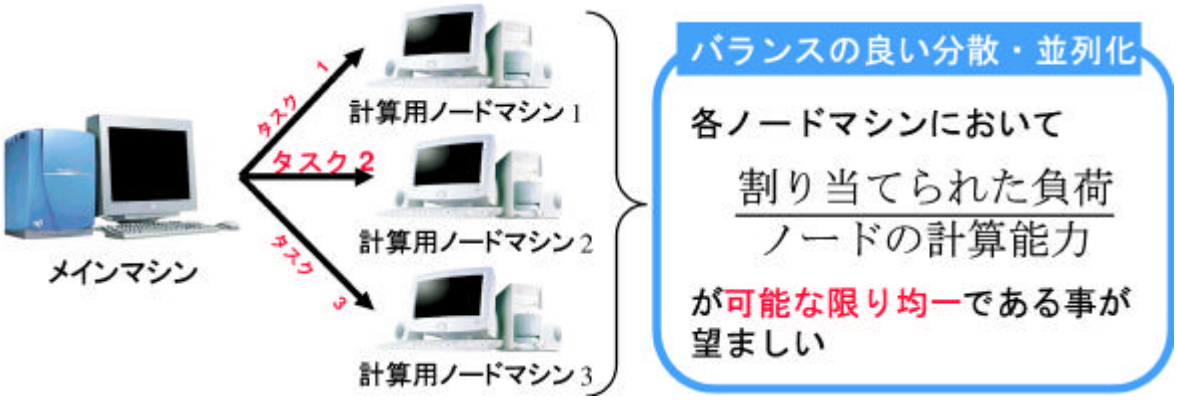


図 4.5: 効率的な分散・並列化

リアルタイムでシミュレーション可能となる。以上の理由から、本研究では、「ポリゴンレベルの衝突判定」と「接点に働く力の算出」を分散・並列化することに決定する。

分散・並列処理するタスクの割り当て方法の検討

本研究は、大学の研究室のような複数台のパーソナルコンピュータが既にネットワーク接続されている環境を想定しているため、計算機の計算能力は均一ではない。また、仮想空間シミュレーションのためのプロセス以外のプロセスが、計算機を利用する可能性もあるため、シミュレーションを実行中に利用可能な計算機の計算能力が変化することも考えられる。従って、分散・並列化の効果を十分に発揮するためには、図 4.5 に示すように、各計算機において処理を完了するまでの時間が可能な限り均一になるように、バランスの良い分散・並列化をおこなう必要がある。例えば、もし、ある計算機の処理が大幅に遅れた場合、他の全ての計算機はその処理が終了するまで処理を止めて待っていなければならず、シミュレーションの速度が低下する。バランスの良い分散・並列化をおこなうためには、割り当てる処理の負荷とその処理を担当する計算機の計算能力を推定し、適切に処理を割り当てなければならない。

処理の負荷の推定方法

処理にかかる負荷の目安を図 4.6 に示す。厳密な衝突判定は剛体の組単位で分散・並列化が可能であり、その負荷はそれぞれの剛体のポリゴン数の積に比例する。これは、厳密な接触判定が剛体を構成するポリゴン面毎に判定をおこなっているからである。接点に働く力の算出は系単位で分散・並列化が可能であり、その負荷は系に含まれる接点の数の 3 乗に比例する。これは、接点に働く力を Dantzig のアルゴリズムを解いて算

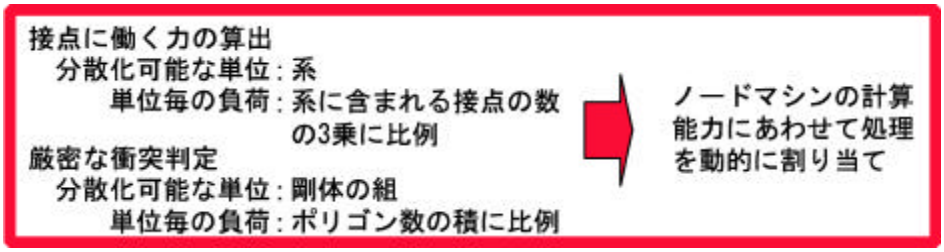


図 4.6: 処理にかかる負荷の推定

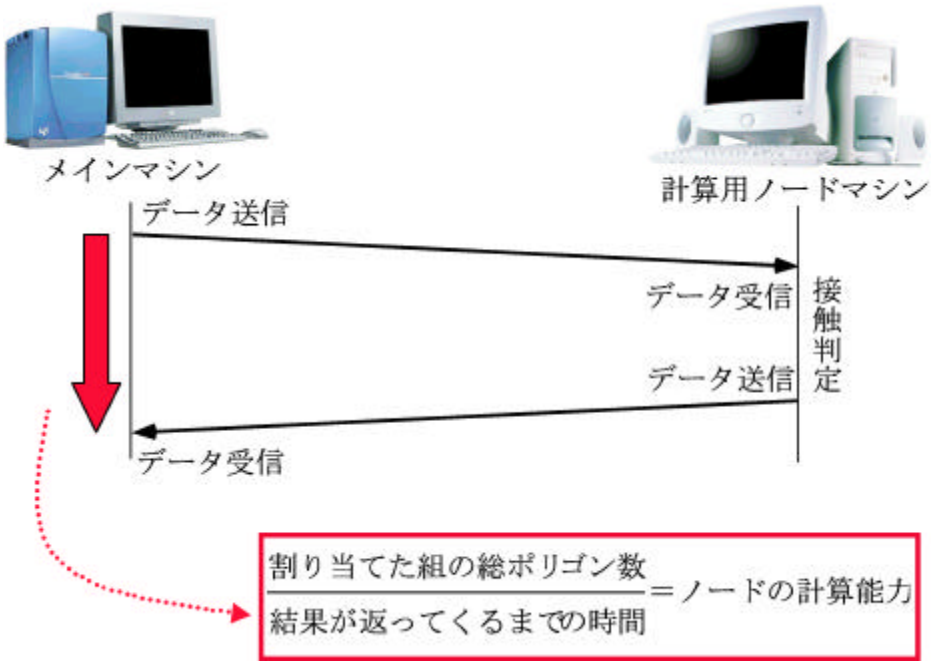


図 4.7: 計算機の能力の推定

出する際、接点の数が n 倍になれば、解かなければならない条件式が n 倍、その 1 つの条件式を解くのに必要な時間が n^2 倍になり、必要な時間が n^3 倍になるためである。

計算機の計算能力の推定方法

計算機の能力は、図 4.7 に示すように、分散・並列化した厳密な衝突判定の結果がメインマシンに返ってくるまでの時間を毎回計測することで推定する。割り当てた接触判定の負荷の大きさと、それを処理するのに要した時間が求まれば、計算機の能力を推定できる。

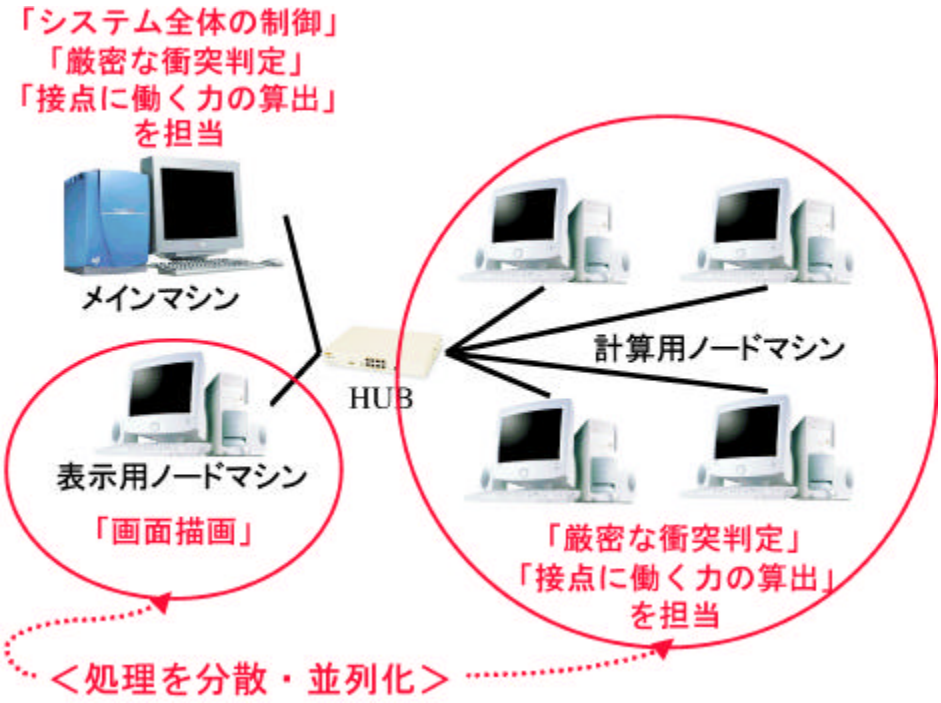


図 4.8: 各計算機の役割

システムの基本構成

本研究では、画面の描画がボトルネックとなりシミュレーション速度を低下させる可能性を考慮して、画面の描画を専用の計算機を用いて処理させることにする。本研究の計算機の構成を図 4.8 に示す。

本研究で開発する剛体挙動シミュレーションシステムは、主にシステム全体の管理を担当するメインマシン、画面の描画のみを担当する描画用ノードマシン、そしてポリゴンレベルの衝突判定や接点に働く力の算出を担当する複数台の計算用ノードマシンで構成される。また、全てのノードマシンとメインマシンはネットワークで接続され、相互にデータの送受信をおこなう。なお、厳密な衝突判定と接点に働く力の算出は計算用ノードマシンとメインマシンの双方に分散・並列化されるため、計算用ノードマシンが 1 台も無い構成でもシステムは動作する。その場合、分散・並列化をおこなわず、メインマシン 1 台が剛体挙動シミュレーションの処理をおこなう。

4.2.2 分散・並列化した剛体挙動シミュレーションの処理の流れ

本研究では、剛体挙動シミュレーションの処理の内、特に負荷が高い「厳密な衝突判定」と「接点に働く力の算出」をメインマシンと複数台の計算用ノードマシンに分

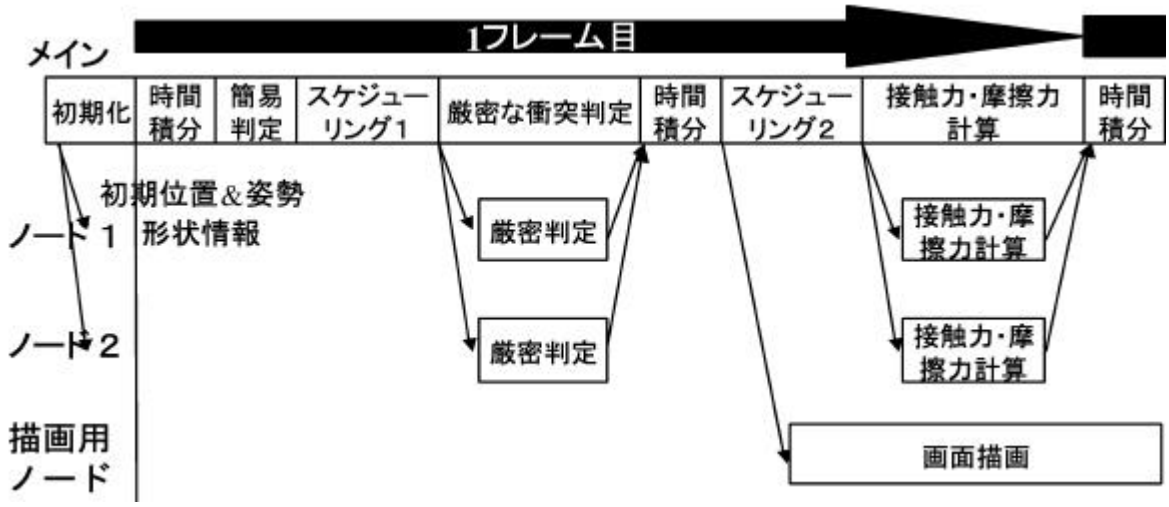


図 4.9: 分散・並列処理の流れ

散し並列処理する。分散・並列化する場合のシミュレーションの処理の流れを図 4.9 に示す。

まず、シミュレーションを開始する前に、メインマシンと描画用ノードマシンにおいて剛体の初期位置、初期速度等を設定する。その後、メインマシンで、ある時間幅 t だけ時間積分をおこない、剛体の位置や姿勢を更新し、球近似による簡易な衝突判定をおこなう。簡易な衝突判定の結果、衝突する可能性がある剛体の組み合わせを抽出すると、メインマシンと計算用ノードマシンに剛体の組み合わせを割り当て、厳密な衝突判定に必要なデータを計算用ノードマシンに送信する。そして、メインマシンと計算用ノードマシンは、それぞれ割り当てられた剛体の組み合わせに対し、厳密な衝突判定を平行して同時に実行する。その後、厳密な衝突判定の結果をメインマシンに集め、衝突が検知されていれば剛体同士がちょうど衝突するまでの時間幅 Δt を算出する。その後、時間幅 Δt だけ時間積分をやりなおし、シミュレーションを Δt だけ進める。ここで、シミュレーションがある一定時間を経過した場合、メインマシンは位置と姿勢のデータを描画用ノードマシンに送信する。描画用ノードマシンが画面の更新をおこなっている間、メインマシンはメインマシンと計算用ノードマシンに系を割り当て、接点に働く力の算出に必要なデータを計算用ノードマシンに送信する。そして、メインマシンと計算用ノードマシンはそれぞれ割り当てられた系に対し、接点に働く力の算出を平行して同時に実行する。以上が 1 フレームの処理の流れとなる。

4.2.3　計算タスクの割り当てアルゴリズム

メインマシンと計算用ノードマシンに処理を割り当てる方法を図 4.10 に示す。まず、①処理を負荷の高い順番に並び替える。次に、②最も負荷が高いタスク 5 をそれぞれの計算機に割り当てたと仮定し、処理を終了するまでに要する時間をそれぞれの計算機毎に算出する。棒グラフの実線部が既に割り当て済みの処理を終了するまでに要する時間、棒グラフの点線部がタスク 5 を終了するまでに要する時間を表す。そして、③既に割り当て済みの処理を終了するまでに要する時間とタスク 5 を終了するまでに要する時間の合計をそれぞれの計算機について算出し、値が最も小さいノード 2 にタスク 5 を割り当てることを決定する。以下、全てのタスクについて同様の処理を繰り返し、全ての処理の割り当てを決定する。

ただし、このアルゴリズムでタスクの割り当てを行った場合、必ずしも最適なタスクの割り当てを行うとは限らない。例えば計算能力が毎秒 10 の計算機と毎秒 8 の計算機の 2 台の計算機があり、タスクの大きさとして 100、60、50 の 3 種類のタスクがあるとする。この場合に前述のアルゴリズムを適用すると、1 番目のタスクは計算能力が毎秒 10 の計算機へ、2 番目と 3 番目のタスクは計算能力が毎秒 8 の計算機に割り当てられ、それぞれの計算時間は次式のようになる。

$$\text{計算機}\,1 \qquad 100 \div 10 = 10\,\text{秒} \tag{4.7}$$

$$\text{計算機}\,2 \qquad 110 \div 8 = 13.75\,\text{秒} \tag{4.8}$$

よって必要となる最大の計算時間は 13.75 秒となる。しかし、2 番目と 3 番目のタスクを計算能力が毎秒 10 の計算機へ、1 番目のタスクを計算能力が 8 の計算機へ割り当てた場合、それぞれの計算時間は次式のようになる。

$$\text{計算機}\,1 \qquad 110 \div 10 = 11\,\text{秒} \tag{4.9}$$

$$\text{計算機}\,2 \qquad 100 \div 8 = 12.5\,\text{秒} \tag{4.10}$$

となる。このように本研究で用いるアルゴリズムは必ずしも最適なタスクの割り当てをおこなわないが、常に最適なタスクの割り当てをおこなう複雑なアルゴリズムを利用すると、そのアルゴリズムを処理する計算負荷自体がシミュレーションの速度を低下させる可能性がある。従って、本研究では、最適なタスク割り当てを行う複雑なアルゴリズムの実装は将来課題とし、計算負荷の低い簡易型のタスク割り当てアルゴリズムを利用する。

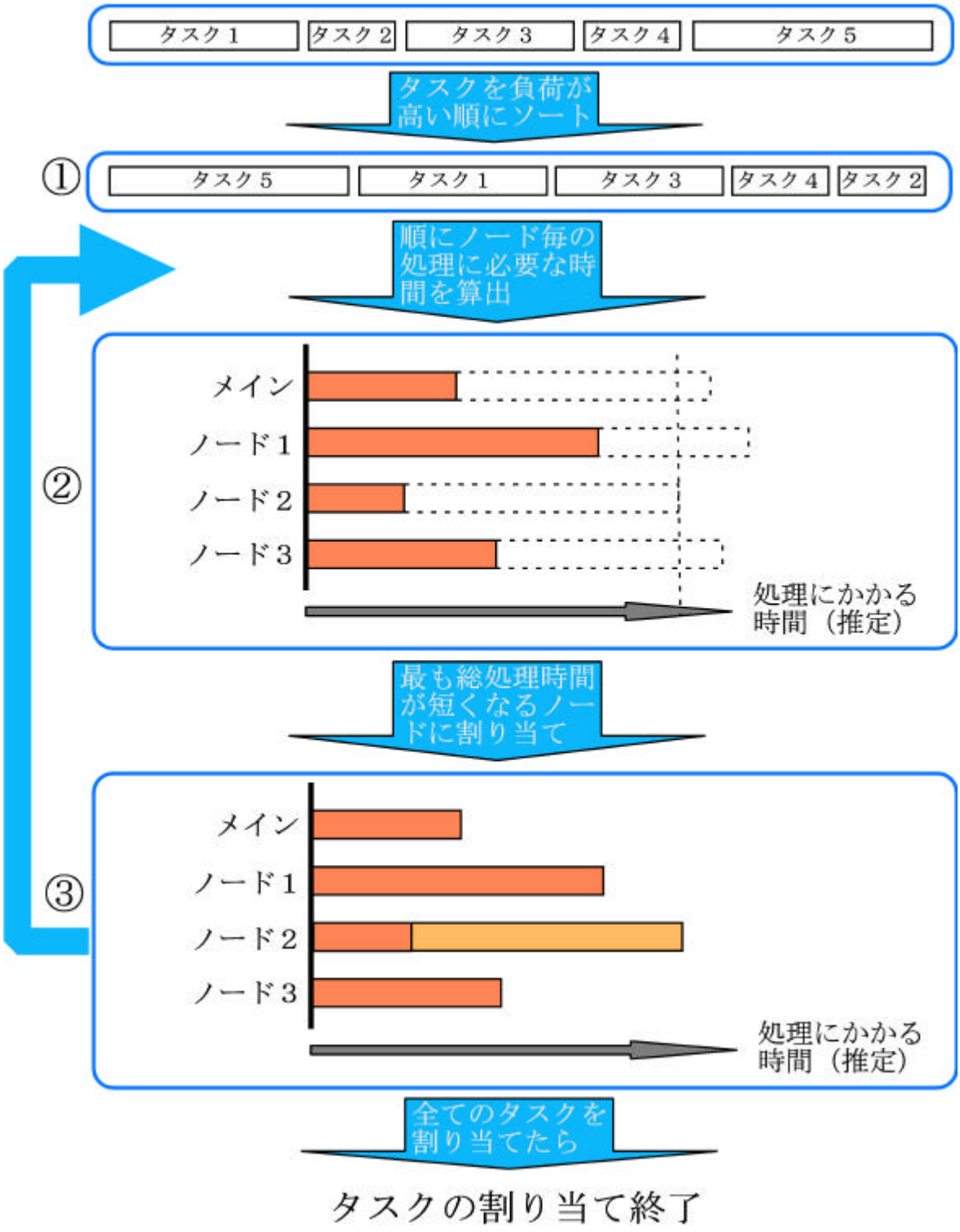


図 4.10: 計算用ノードマシンへのタスクの割り当て

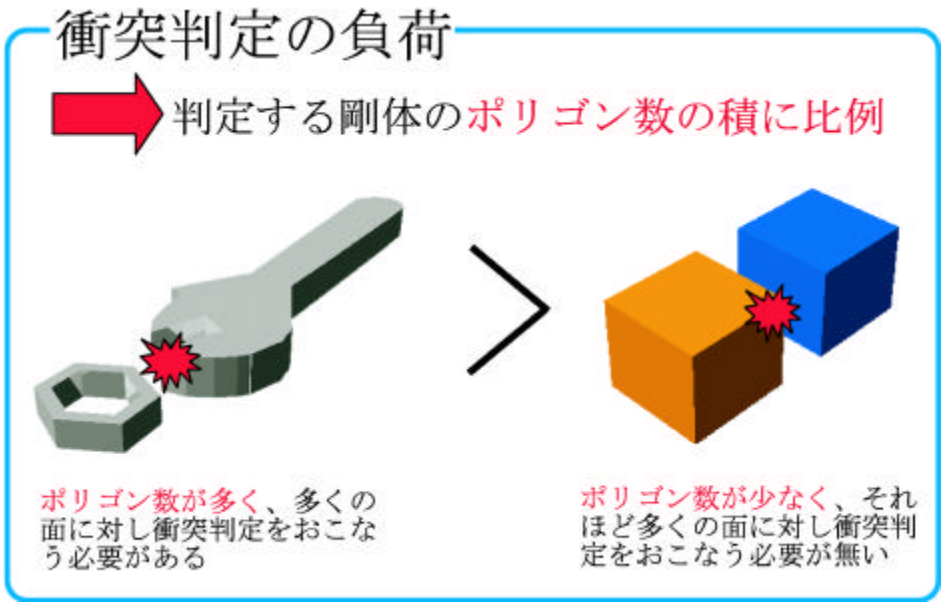


図 4.11: 衝突判定の負荷

4.2.4 ポリゴンレベルの厳密な衝突判定の分散・並列化手法の詳細

分散並列化する際の処理の流れ

ポリゴンレベルの厳密な衝突判定では、剛体の組み合わせ単位で分散・並列化が可能である。1 組の剛体の組み合わせに対して厳密な衝突判定を行う際に必要な計算量を推定できれば、計算用ノードマシンとメインマシンの能力に応じて動的に計算タスクを割り当てることができる。

図 4.11 に衝突判定の負荷の目安を示す。ポリゴンレベルの厳密な衝突判定では、衝突判定アルゴリズム VoronoiClip を用いて、ポリゴン面毎に衝突を判定する。そのため、剛体のポリゴン数が増加するとより多くのポリゴンについて衝突を判定する必要が生じるため、負荷が増大する。厳密な衝突判定の負荷は、衝突判定をおこなう剛体のポリゴン数の積に比例する。なお、ポリゴン数は剛体の形状が複雑であればあるほど増加する傾向にあるが、球形状のように滑らかな曲面を正確に表現しようとする際にも大幅に増加する。

以下にポリゴンレベルの厳密な衝突判定を行うための処理を分散・並列化する際の処理の流れを示す。

1. 衝突判定の負荷とメインマシンおよび計算用ノードマシンの能力を推定する。

2. 厳密な衝突判定をおこなう剛体の組み合わせをそれぞれの計算機に割り当てる。

その際、それぞれの計算機が厳密な衝突判定を終了し、その結果をメインマシンに返信するまでの時間が、可能な限り同じになるように割り当てる。

3. 厳密な衝突判定をおこなうのに必要なデータをメインマシンから計算用ノードマシンに送信する。この際、通信回数を最小限に抑えるため、メインマシンから 1 台の計算用ノードマシンへの送信は 1 回に限定する。1 台の計算用ノードマシンに複数の剛体の組み合わせの衝突判定を割り当てた場合、1 つの送信データの中に複数の剛体の組み合わせのデータを記述する。

4. 計算用ノードマシンは厳密な衝突判定を終了すると、その結果を直ちにメインマシンへ送信する。

なお、厳密な衝突判定は、計算用ノードマシンとメインマシンの両方でおこなう。メインマシンは、計算用ノードマシンへ厳密な衝突判定に必要なデータを送信した後、自身に割り当てられた剛体の組み合わせの厳密な衝突判定をおこなう。そして、厳密な衝突判定を終了すると、計算用ノードマシンから結果のデータを受信するために、受信のための待機状態になる。もし、メインマシンが受信のための待機状態となるより早く計算用ノードマシンが衝突判定の結果のデータを送信すると、データの受信にタイムラグが生じ、分散・並列化の効率が低下する。そこで、剛体の組み合わせを割り当てる際、計算用ノードマシンが計算を終了するよりも先にメインマシンの計算が終了するように、メインマシンには比較的少ない計算量の剛体の組み合わせを割り当てる。そして、計算用ノードマシンが結果のデータを送信する時、メインマシンが必ず受信待機の状態になっているようにする。

メインマシンから計算用ノードマシンへデータを送信する際のプロトコル

図 4.12 に、厳密な衝突判定に必要なデータをメインマシンから計算用ノードマシンへ送信する際の通信プロトコルを示す。データの先頭にはヘッダとして、識別子、剛体の組み合わせの数、時間差を記述する。識別子は、このデータが厳密な衝突判定に用いられるデータなのか、それとも接点に働く力の算出に用いられるデータなのかを示し、厳密な衝突判定に用いるデータならば 0 を記述する。剛体の組み合わせの数は、計算用ノードマシンが衝突判定をおこなうべき剛体の組み合わせの数を示し、受信した計算用ノードマシンがヘッダ以降のデータを解析する際に用いる。時間差は、現在のフレームと 1 つ前のフレームのシミュレーション中の時間差を示し、衝突判定をおこなう際に用いる。

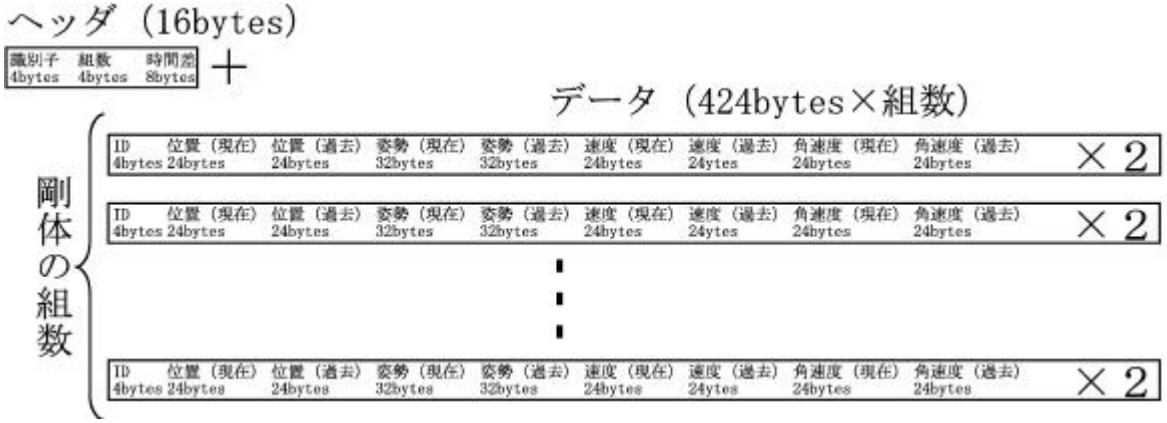


図 4.12: 厳密な衝突判定の通信プロトコル (メインマシンから計算用ノードマシン)

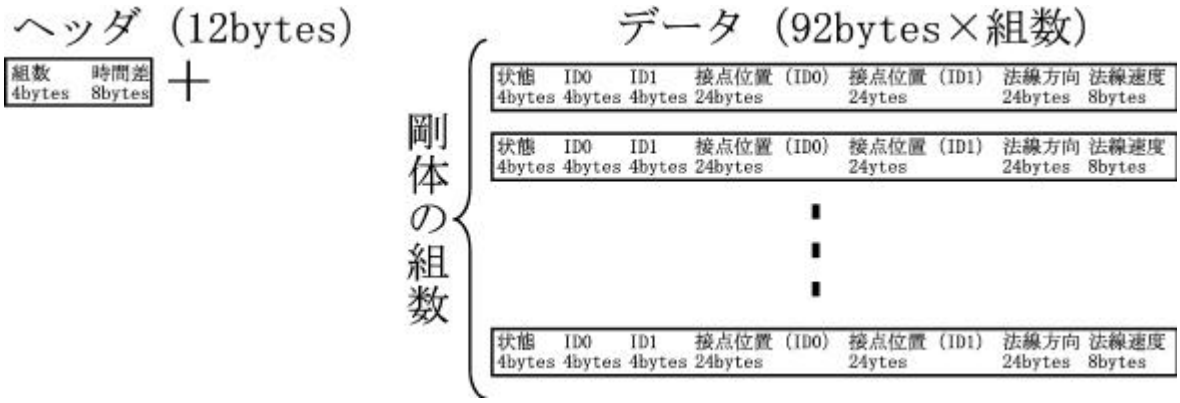


図 4.13: 厳密な衝突判定の通信プロトコル (計算用ノードマシンからメインマシン)

ヘッダ以降は剛体の組み合わせの数だけ厳密な衝突判定に必要なデータを記述する。具体的に記述するデータは、剛体の識別番号（ID）、位置 (現在)、位置 (過去)、姿勢 (現在)、姿勢 (過去)、速度 (現在)、速度 (過去)、角速度 (現在)、角速度 (過去) である。なお、現在とは衝突判定をおこなう時間、過去とは前回時間積分をおこなった時間を示す。

計算用ノードマシンからメインマシンへデータを送信する際のプロトコル

図 4.13 に、厳密な衝突判定の結果データを計算用ノードマシンからメインマシンへ送信する際の通信プロトコルを示す。データの先頭にはヘッダとして、剛体の組み合わせの数と時間差を記述する。剛体の組み合わせの数は、衝突している剛体の組み合わせ数を示し、受信したメインマシンがヘッダ以降のデータを解析する際に用いる。時間差は、最初に衝突する剛体がちょうど接する時間を示す。なお、メインマシンから計算用ノードマシンへ送信する際に用いた識別子は、メインマシンが必ず衝突判定の

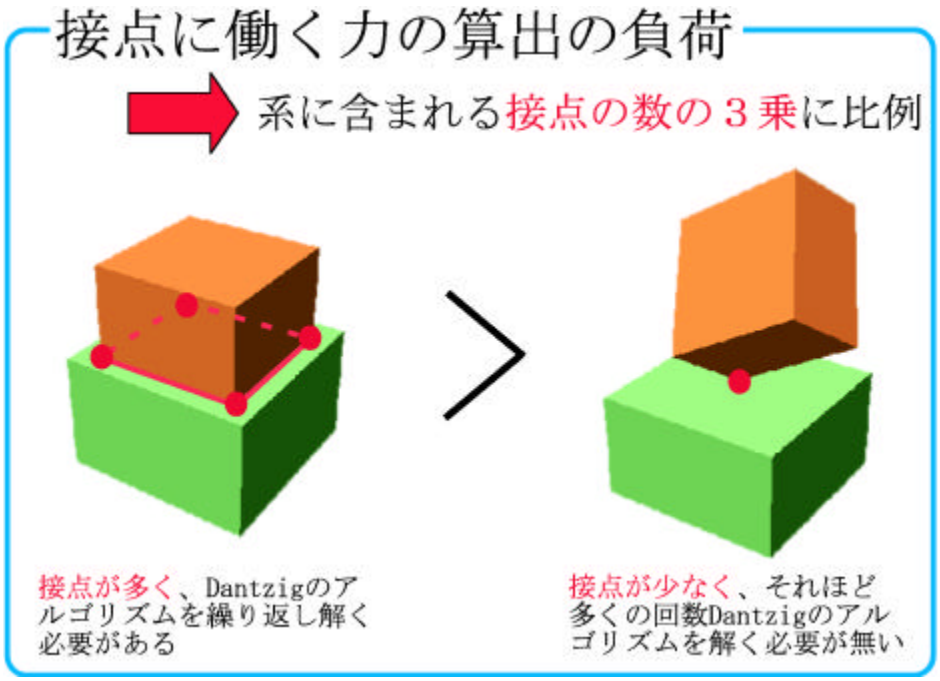


図 4.14: 接点に働く力の算出の負荷

結果として受信するため必要ない。ヘッダ以降は衝突している剛体の組み合わせの数だけ厳密な衝突判定の結果のデータを記述する。具体的に記述するデータは、剛体の状態、剛体の種類を表す ID、接点の位置、法線方向、法線速度である。衝突している剛体の組み合わせが 0 の場合、剛体の組み合わせの数に 0 と記述したヘッダのみメインマシンへ送信する。

4.2.5 接点に働く力の算出処理の分散・並列化手法の詳細

分散並列化する際の処理の流れ

接点に働く力の算出は、系単位で分散・並列化が可能である。1 つの系の接点に働く力の算出に必要な計算量を推定できれば、計算用ノードマシンとメインマシンの能力に応じて動的に計算タスクを割り当てることができる。

まず、図 4.14 に接点に働く力の算出の負荷を示す。本研究は、系に含まれる全ての接点について条件式をたて、それを Dantzig のアルゴリズムを用いて解くことで、接点に働く力を算出する。そのため、系に含まれる接点の数が増加すると、新たに満たすべき条件式が増加するため Dantzig のアルゴリズムによる解法が複雑化し、接点に働く力の算出の負荷が増大する。なお、接点に働く力の算出は、処理をおこなう系に含まれる接点の数の 3 乗に比例する。

以下に接点に働く力の算出処理を分散・並列化する際の処理の流れを示す。

1. 衝突判定により、系とそれに含まれる接点を求める。
2. 接点に働く力の算出処理の負荷を推定する。
3. 接点に働く力の算出処理行う系をそれぞれの計算機に割り当てる。その際、それぞれの計算機が接点に働く力の算出処理を終了し、その結果をメインマシンに返信するまでの時間が、可能な限り同じになるように割り当てる。
4. 接点に働く力の算出処理をおこなうのに必要なデータをメインマシンから計算用ノードマシンに送信する。この際、通信回数を最小限に抑えるため、メインマシンから 1 台の計算用ノードマシンへの送信は 1 回に限定する。
5. 計算用ノードマシンは接点に働く力の算出処理を終了すると、その結果を直ちにメインマシンへ送信する。

なお、接点に働く力の算出も厳密な衝突判定を行う場合と同様に計算用ノードマシンとメインマシンの両方でおこなう。メインマシンは、計算用ノードマシンへ接点に働く力の算出に必要なデータを送信した後、自身に割り当てられた系の接点に働く力の算出をおこなう。そして、接点に働く力の算出を終了すると、計算用ノードマシンから結果のデータを受信するために、待機状態になる。メインマシンには、厳密な衝突判定を行う場合と同様に、計算用ノードマシンが計算を終了するよりも先にメインマシンの計算が終了するように、比較的少ない計算量の計算タスクを割り当てる。

メインマシンから計算用ノードマシンへデータを送信する際のプロトコル

図 4.15 に、接点に働く力の算出に必要なデータをメインマシンから計算用ノードマシンへ送信する際の通信プロトコルを示す。

データの先頭にはヘッダとして、識別子、系の数が記述されている。接点に働く力の算出の識別子は 1 である。系の数は、その計算用ノードマシンが Dantzig のアルゴリズムを用いて接点に働く力を算出するべき系の数を示す。ヘッダ以降は、系に含まれる接点の数や系に含まれている剛体の ID、接点の位置等のデータが記述されている。具体的に記述するデータは、剛体の種類を表す ID、接点の位置、法線方向、法線速度である。

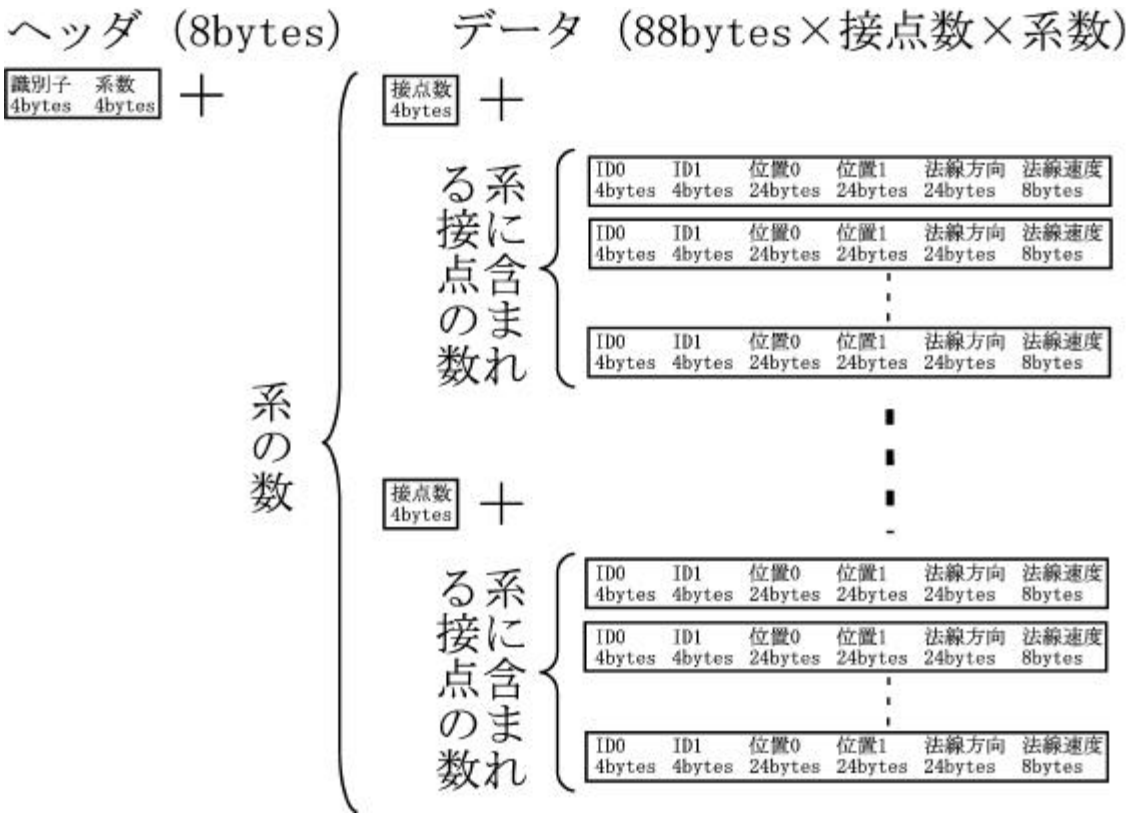


図 4.15: 接点に働く力の算出の通信プロトコル (メインマシンから計算用ノードマシン)

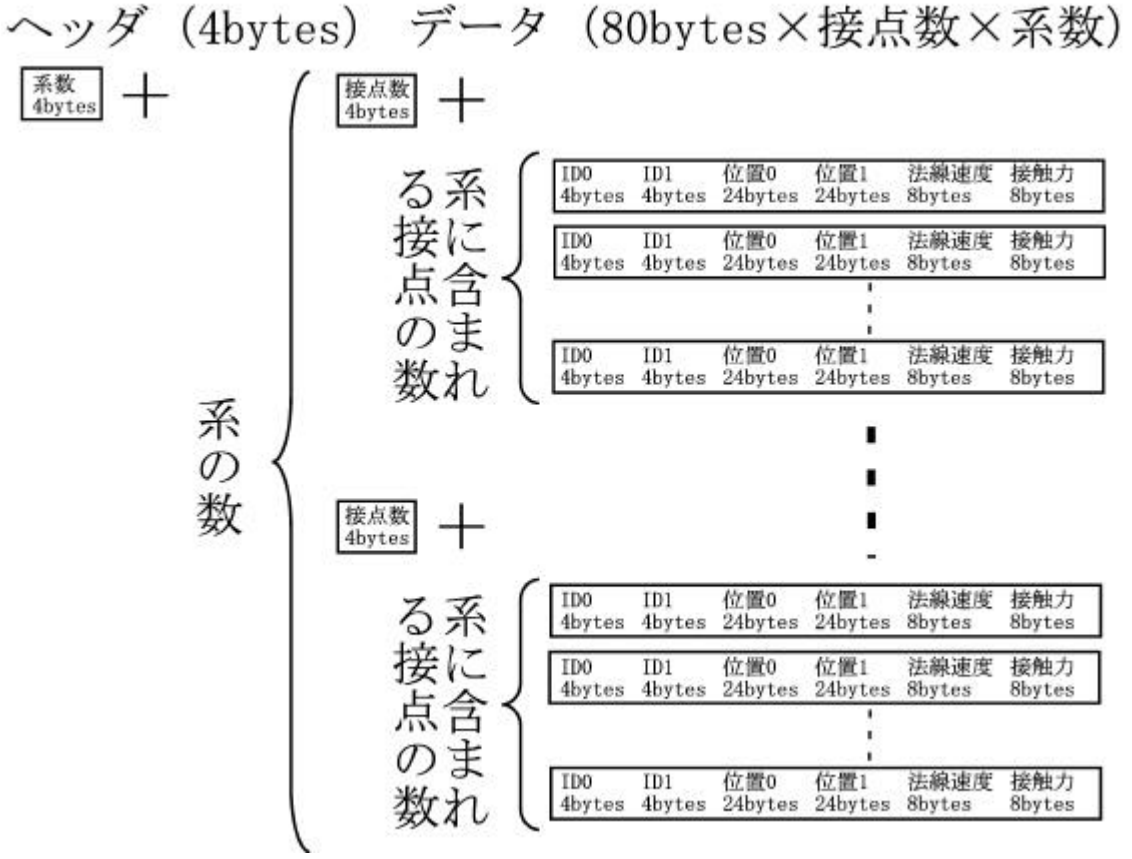


図 4.16: 接点に働く力の算出の通信プロトコル (計算用ノードマシンからメインマシン)

計算用ノードマシンからメインマシンへデータを送信する際のプロトコル

図 4.16 に、接点に働く力の算出の結果データを計算用ノードマシンからメインマシンへ送信する際の通信プロトコルを示す。

データの先頭にはヘッダとして、系の数が記述されている。なお、メインマシンから計算用ノードマシンへ送信する際に用いた識別子は、メインマシンが必ず接点に働く力の算出の結果として受信するため必要ない。ヘッダ以降は、系に含まれる接点の数や系に含まれている剛体の ID、接点の位置等のデータが記述されている。具体的に記述するデータは、剛体の種類を表す ID、接点の位置、法線速度、接触力である。

4.2.6 画面描画の分散・並列化

仮想空間を表示する画面は剛体挙動のシミュレーションが一定時間経過する毎に更新される。しかし、3 次元空間を 2 次元画像として描画する際、形状や表面材質、陰影の計算等のために計算機に高い負荷がかかる。そのため、剛体挙動シミュレーションと画面の描画を同じ計算機でおこなえば、画面描画がボトルネックとなりシミュレー

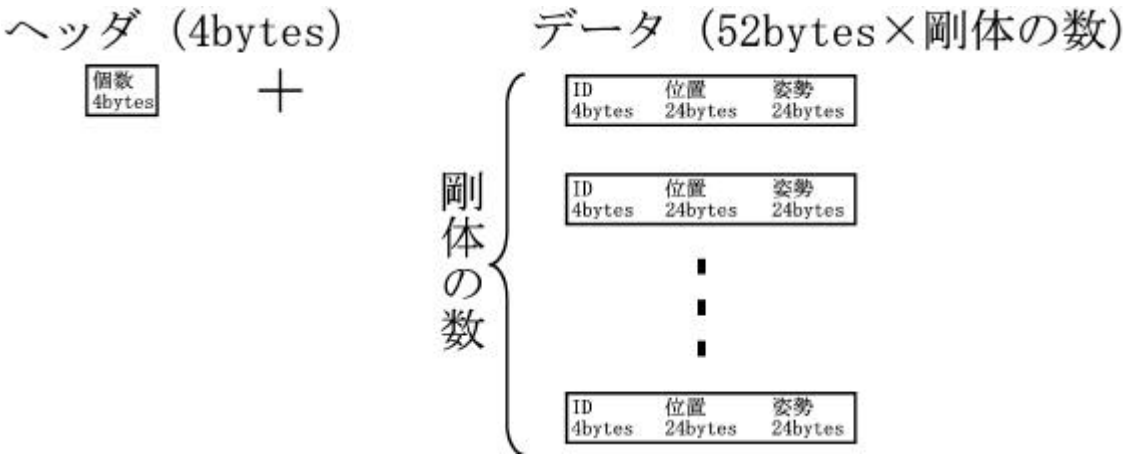


図 4.17: 画面描画の通信プロトコル

ション速度が低下する可能性がある。そこで、本研究では、画面描画に専用のノードマシンを用意した。

画面の描画に必要なデータをメインマシンから描画用ノードマシンへ送信する際の通信プロトコルを図 4.17 に示す。ヘッダとしては描画する剛体の数のみを記述し、描画用ノードマシンはこの値を用いてヘッダ以降に記述される仮想物体の位置や姿勢のデータを解析する。なお、画面を描画する際には処理の結果をメインマシンに返す必要がないため、描画用ノードマシンからメインマシンへはデータの送信は行われない。

4.3 PC クラスタを用いた剛体挙動シミュレーションシステムのシステム構成

本研究で開発した PC クラスタを用いた剛体挙動シミュレーションシステムのシステム構成を図 4.18 に示す。まず、メインマシンにのみ実装される制御部は、どのタスクをどの計算用ノードマシンに割り当てるかを決定するスケジューリング部を持ち、システム全体を制御する役割を担う。タスクの割り当てを決定すると、メインマシンの通信部は計算用ノードマシンに受け渡す剛体の情報をメモリから読み出し、計算用ノードマシンの通信部へ送信する。そして、メインマシンや計算用ノードマシンの接触判定部や物理計算部で解が導き出されると、それらは再び通信部を介して制御部へ集められ、メインマシンのメモリに格納される。以上の処理を経て求められた結果は、シミュレーションがある一定時間経過する毎に描画用ノードマシンへ通信部を介して送られ、描画出力部によって画面に出力される。なお、それぞれの計算用ノードマシン

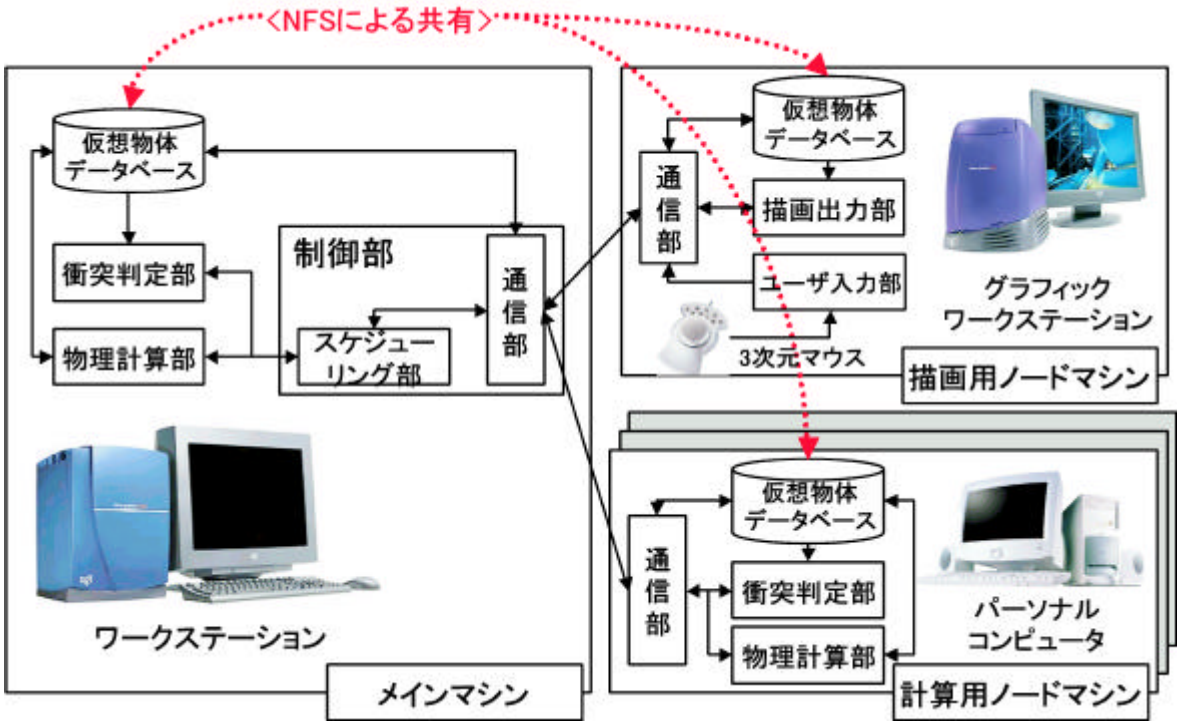


図 4.18: システム構成（分散・並列処理あり）

に仮想物体データベースを持たせているのは、仮想物体の形状や質量などシミュレーション中に変化しないパラメータを、ノードマシンが必要とする度に通信する無駄を省くためであり、UNIX のファイル共有機能 NFS を用いて実現した。

PC クラスタを用いた剛体挙動シミュレーションを実装した計算機環境を表 4.2 に示す。本研究では、1 台のメインマシン、1 台の表示用ノードマシン、3 台の計算用ノードマシンにシステムを実装した。それぞれの計算機はハブを介して LAN 接続され、データの送受信をおこなう。

メインマシンや計算用ノードマシンを構成する接触判定部、物理計算部、制御部等は C++言語を用いて開発し、特に通信部は超並列計算研究会が無料で公開している高速通信ライブラリ PMv2 を用いた。描画用ノードマシンは C 言語と OpenGL を用いて開発した。

本システムは、メインマシンと計算用ノードマシンに一般的な PC と OS を用いているため、単一の計算機環境と比較しても導入の際の手間が少なくて済むという特徴がある。特に大学の研究室のように既に複数の PC が LAN 接続されている環境では、新たに PC やネットワーク機器を用意する必要がなく、PC クラスタを実現するためのライブラリを導入するだけで本システムを導入できる。

表 4.2: PC クラスタを用いた剛体挙動シミュレーションの計算機環境

	メインマシン	計算用ノードマシン 1
CPU	PentiumIII 1.26GHz × 2	PentiumIII 1.0GHz × 2
メモリ	768MB	512MB
ネットワーク	100BASE-T	100BASE-T
OS	RedHatLinux 7.1	RedHatLinux 7.1
	計算用ノードマシン 2	計算用ノードマシン 3
CPU	PentiumIII 1.0GHz × 2	PentiumIII 700MHz × 2
メモリ	512MB	256MB
ネットワーク	100BASE-T	100BASE-T
OS	RedHatLinux 7.1	RedHatLinux 7.1
	描画用ノードマシン	
機種	OCTANE	
CPU	MIPS R10000 256MHz	
メモリ	384MB	
ネットワーク	100BASE-T	
OS	SGI IRIX 6.5	
グラフィック	EMXI	

第 5 章 剛体挙動シミュレーションの分散・並列化の評価実験

本章では、まず、構築したシステムを用いて実施したシミュレーション速度の評価実験について述べる。次に、剛体挙動のリアルタイムシミュレーションを実用化するための指針として、今後解決すべき課題について述べる。最後に、本研究の総括として、計算機性能の向上と人工現実感技術の進歩への展望について述べる。

5.1 シミュレーション速度の評価実験

本節では、まず、シミュレーション速度の評価実験の目的、方法、結果について述べる。そして、得られた結果を考察し、剛体挙動のリアルタイムシミュレーションの実現可能性を検討する。

5.1.1 評価実験の目的

同一の仮想空間の剛体挙動シミュレーションを単一の計算機環境で処理する場合と複数台の PC クラスタ環境で処理する場合のシミュレーション速度を比較し、本研究で開発した剛体挙動シミュレーションの分散・並列化手法を適用することによる効果を評価する。評価の対象は、計算性能の変化に対する分散・並列化の効果の変化、ネットワーク性能の変化に対する分散・並列化の効果の変化、およびリアルタイムシミュレーションが実現可能な仮想空間の複雑さの限界である。なお、本研究では、人が違和感を感じない描画速度の限界とされている毎秒 10 フレームの更新速度をリアルタイムの条件とする。

5.1.2 評価実験の方法

単一の計算機環境と計算用ノードマシンを 1 台から 3 台用いた PC クラスタ環境の計 4 通りの環境に対し、仮想空間内の剛体の数を徐々に増やし、シミュレーション開始から終了までの計算時間と、シミュレーション全体を通して 1 フレームの処理に最も多く時間を費やしたフレームと、その費やした時間を計測する。なお、計算機の環境は

表 5.1: 評価実験の計算機環境

	メインマシン	計算用ノードマシン 1
CPU	PentiumIII 1.26GHz × 2	PentiumIII 1.0GHz × 2
メモリ	768MB	512MB
	計算用ノードマシン 2	計算用ノードマシン 3
CPU	PentiumIII 1.0GHz × 2	PentiumIII 700MHz × 2
メモリ	512MB	256MB

	分散・並列処理に使用する計算機
環境 1	メインマシン (単一の計算機環境)
環境 2	メインマシン、計算用ノードマシン 1
環境 3	メインマシン、計算用ノードマシン 1、同 2
環境 4	メインマシン、計算用ノードマシン 1、同 2、同 3

先に表 4.2 において示したとおりである。それぞれの環境の計算機の構成を表 5.1 に示すが、ここに示す分散・並列処理に用いる計算機に加え描画用ノードマシンを全ての環境で用いた。また、計算機を接続する HUB として 100BASE-T と 10BASE-T の２種類の性能の HUB を用いて計測をおこなう。

評価実験の条件を表 5.2 に示す。また、実験を開始する前、全ての計算機において OS 等、他のソフトウェアの CPU 占有率が 1 %以下でメモリスワップを発生しておらず、計算機の性能が十分発揮できる状態であることを確認した。

本実験では、剛体として多数のナットとボルトが床に落下する場面を解析の対象とするが、これは機器保修の訓練環境において、訓練生が箱を倒して中のナットやボルトがこぼれ落ちる状況に相当する。このように、訓練生の失敗により物が崩れ落ちる等の現象は、訓練生にその失敗を実際に体験させ、実際の作業の現場で失敗させないようにするために、非常に重要であるが、これまでの Rule-base の手法のみを用いた仮想空間の構築方法では、実現することが非常に困難な現象であった。

5.1.3 評価実験の結果

配置する仮想物体の個数が 1 個から 50 個までの各仮想空間に対し、環境 1〜環境 4 までの計 4 環境においてシミュレーション実行した際の計算開始から終了までに要し

表 5.2: 評価実験の条件

剛体の種類	床形状 (一辺 2m の直方体、12 ポリゴン、跳ね返り係数 0.8)
	ボルト形状 (約 4cm 四方、178 ポリゴン、跳ね返り係数 0.8)
	ナット形状 (約 4cm 四方、396 ポリゴン、跳ね返り係数 0.8)
剛体の数	床形状 1 個 ボルト形状およびナット形状 1〜50 個
剛体の配置	床形状:原点に固定
	ボルト形状、ナット形状:床形状上方ランダムな位置
仮想空間	広さ無制限
終了条件	床形状の上で静止するか、床形状からこぼれ落ち落下し続けるか
	いずれかの状態に全ての剛体がなった場合

た時間を、100BASE-T の HUB を用いた場合を図 5.1 に、10BASE-T の HUB を用いた場合を図 5.2 に示す。横軸は仮想空間内の剛体の数を、縦軸は処理を終了するまでに要する時間を表す。

環境 2 から環境 4 までの、それぞれの計算機の構成で 100BASE-T の HUB を用いた場合と 10BASE-T の HUB を用いた場合の計算処理を終了するまでに要した時間の比較を、図 5.3 から図 5.5 に示す。横軸は仮想空間内の剛体の数を、縦軸は処理を終了するまでに要する時間を表す。

配置する仮想物体の個数が 1 個から 50 個までの各仮想空間に対して、単一の計算機を用いてシミュレーションを実行する場合の計算時間と、複数の計算機で分散・並列処理を行いシミュレーションを実行する場合の計算時間の差を、環境 2 から環境 4 までの各計算機環境（HUB は 100BASE-T を使用）に対して求めた結果を図 5.6 から図 5.8 に示す。横軸は仮想空間内の剛体の数を、縦軸は単一の計算機環境で計算処理を終了するまでに要した時間と PC クラスタ環境で計算処理を終了するまでに要した時間の差を表す。時間差が負である場合は単一の計算機環境の方が、PC クラスタ環境より先に計算処理を終了したことを表す。

100BASE-T の HUB を用いた 4 環境において、1 フレームの処理に最も多く費やしたフレームの計算時間とリアルタイムシミュレーションの限界を、図 5.9 と図 5.10 に示す。横軸は仮想空間内の剛体の数を、縦軸は 1 フレームの計算処理に要する時間の最大値を表す。本研究は、毎秒 10 フレームの更新速度をリアルタイムの条件としているため、1 フレームの計算処理に要する時間の最大値が 0.1 秒以下のシミュレーション

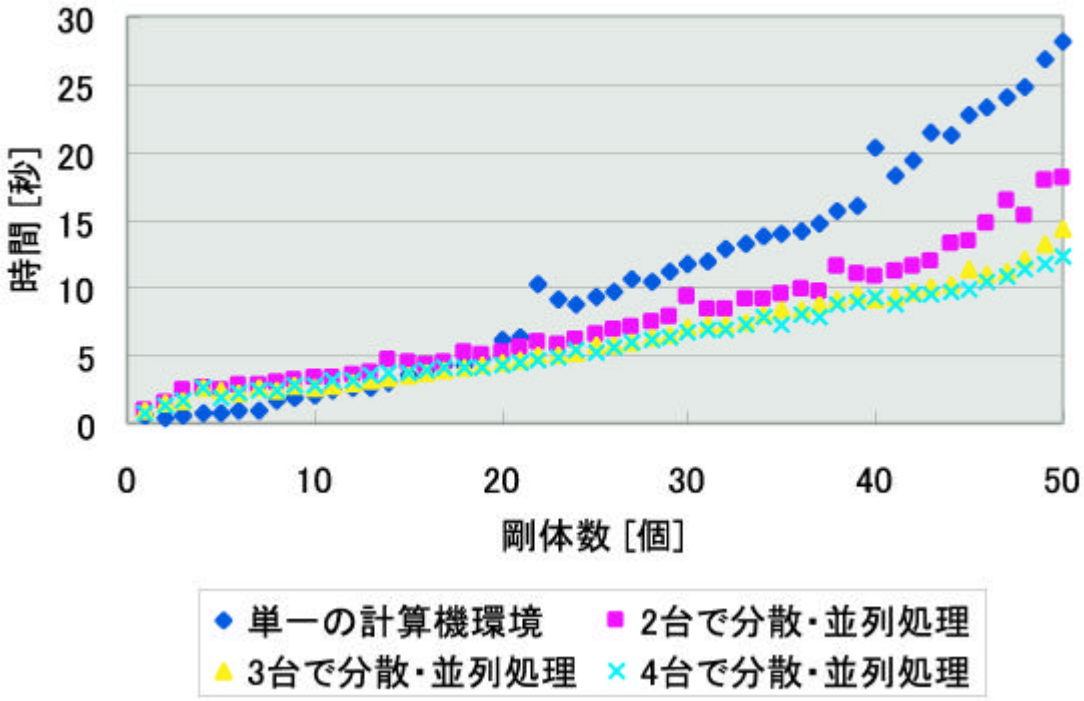


図 5.1: 計算性能に対する分散・並列化の効果 (100BASE-T)

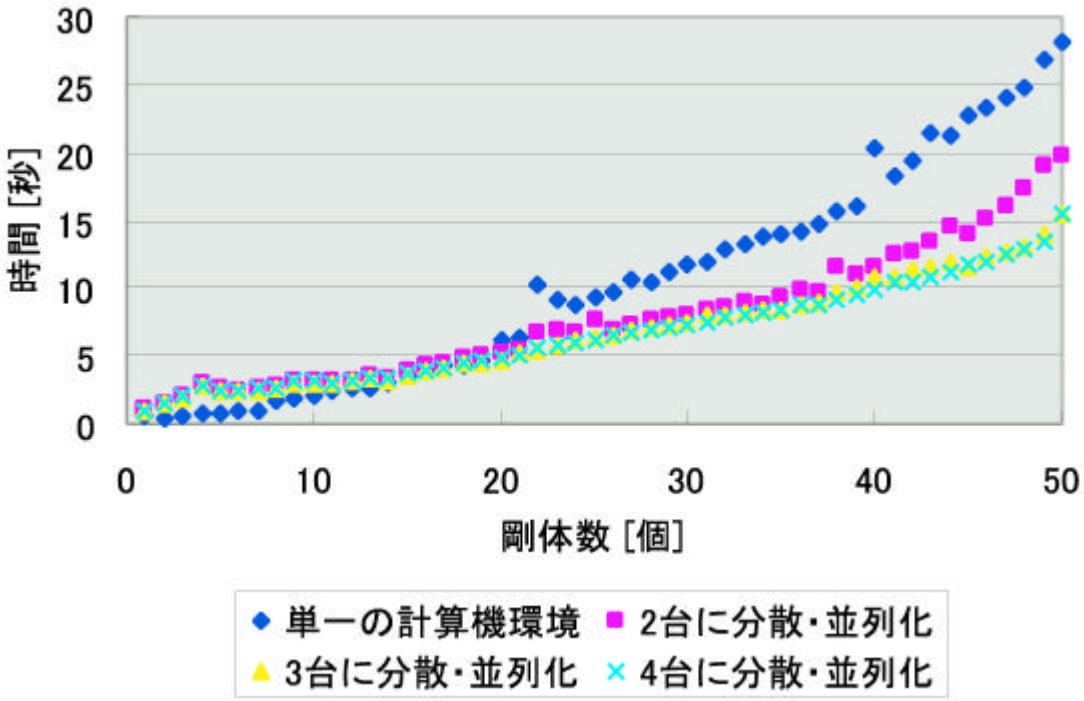


図 5.2: 計算性能に対する分散・並列化の効果 (10BASE-T)

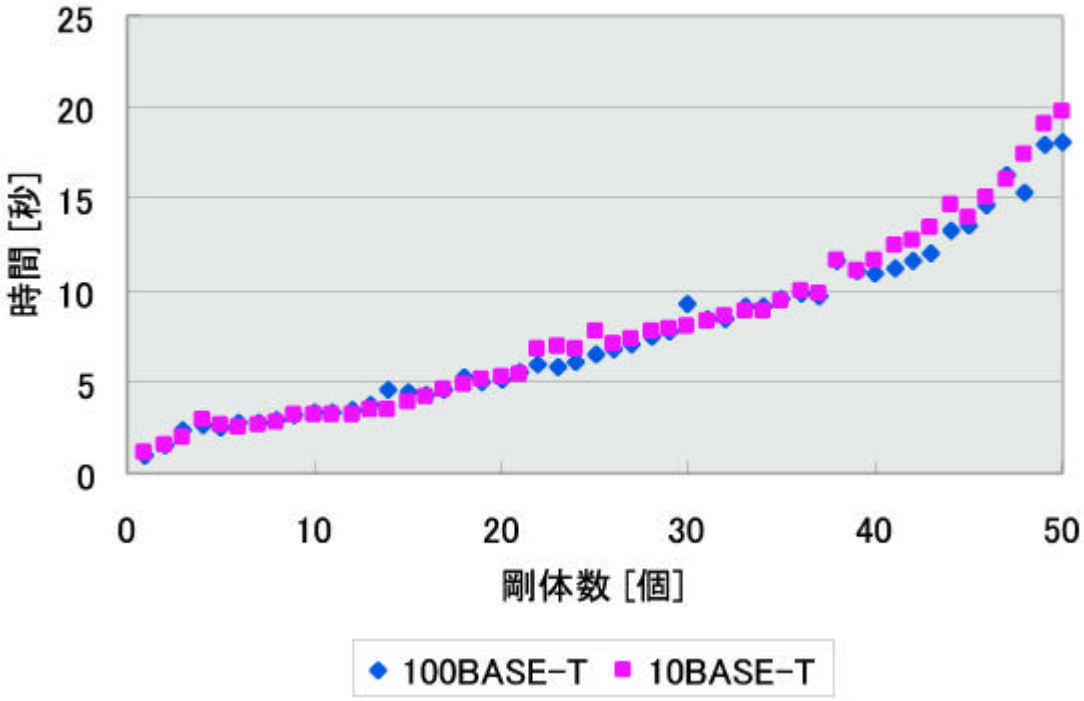


図 5.3: ネットワーク性能に対する分散・並列化の効果 (2 台に分散・並列化)

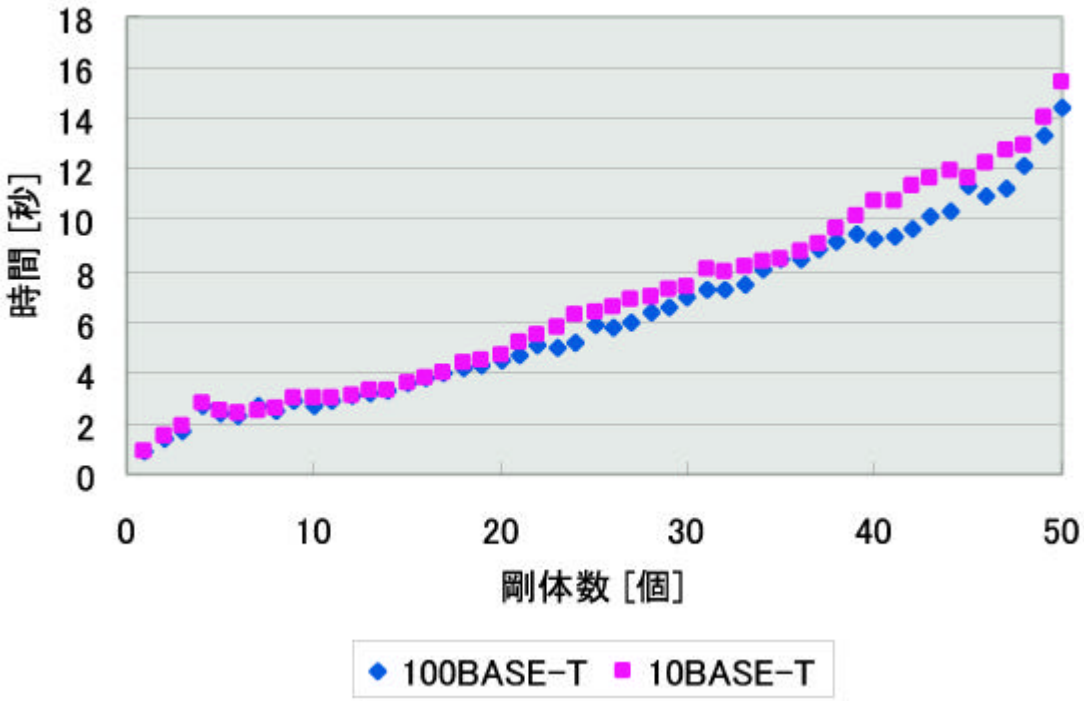


図 5.4: ネットワーク性能に対する分散・並列化の効果 (3 台に分散・並列化)

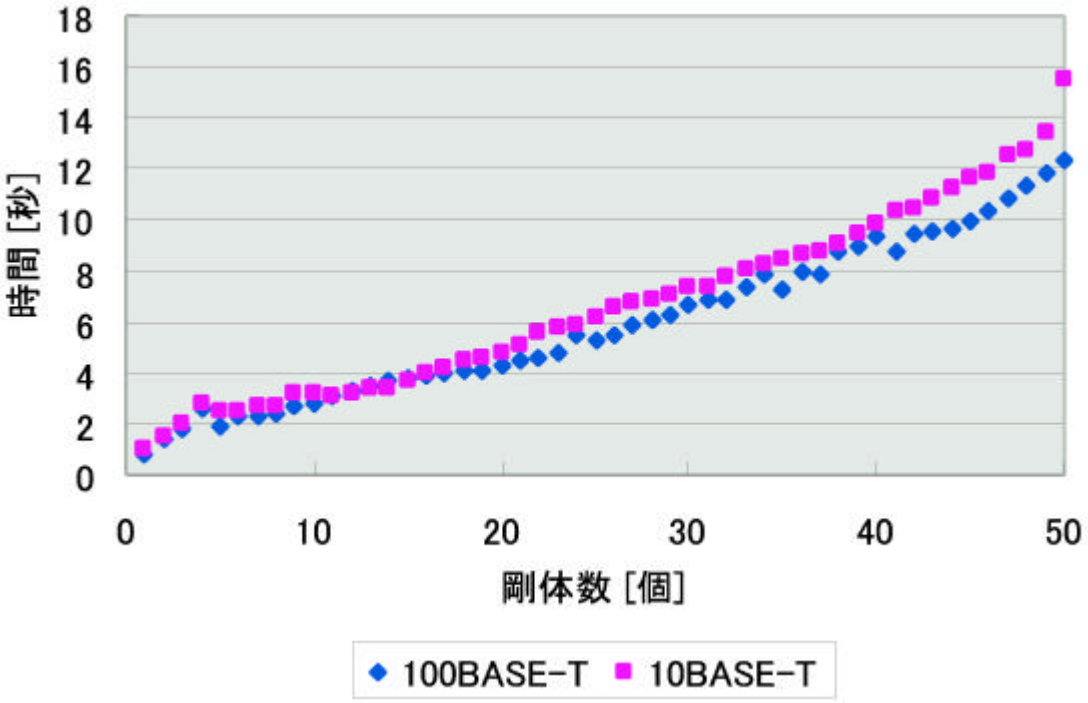


図 5.5: ネットワーク性能に対する分散・並列化の効果 (4 台に分散・並列化)

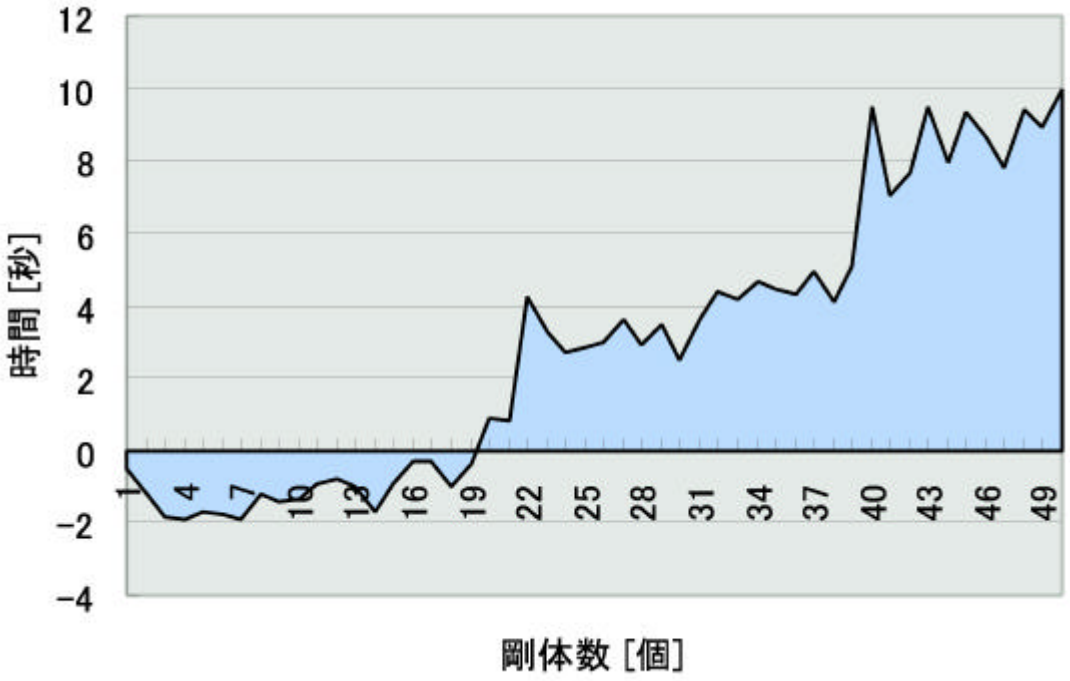


図 5.6: 剛体の数に対する分散・並列化の効果 (2 台に分散・並列化)

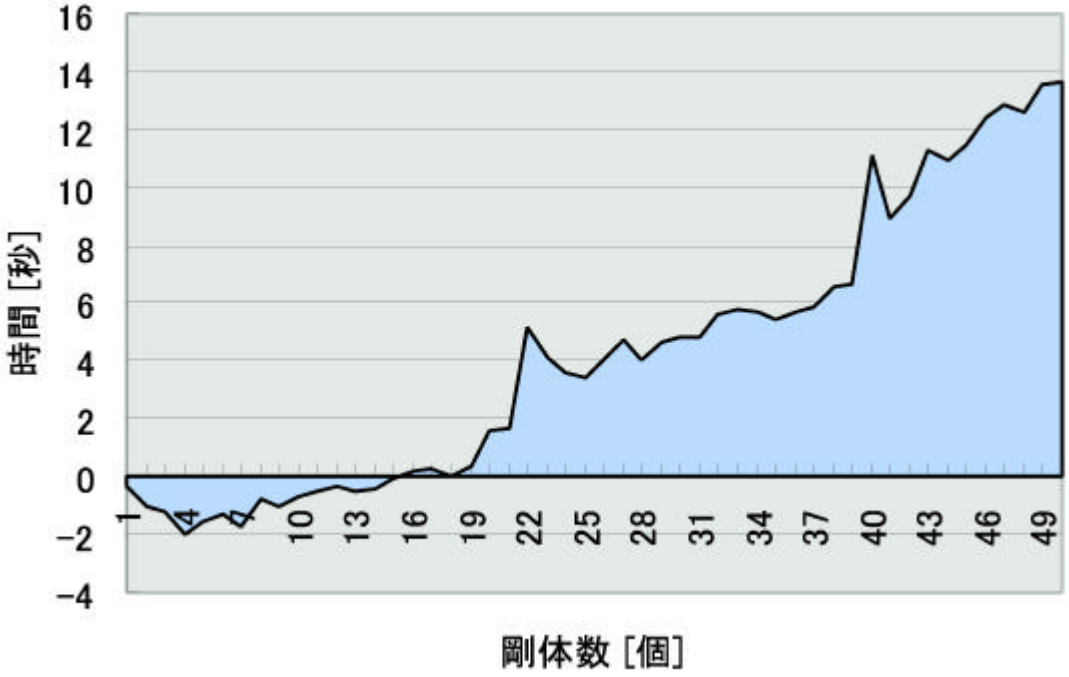


図 5.7: 剛体の数に対する分散・並列化の効果 (3 台に分散・並列化)

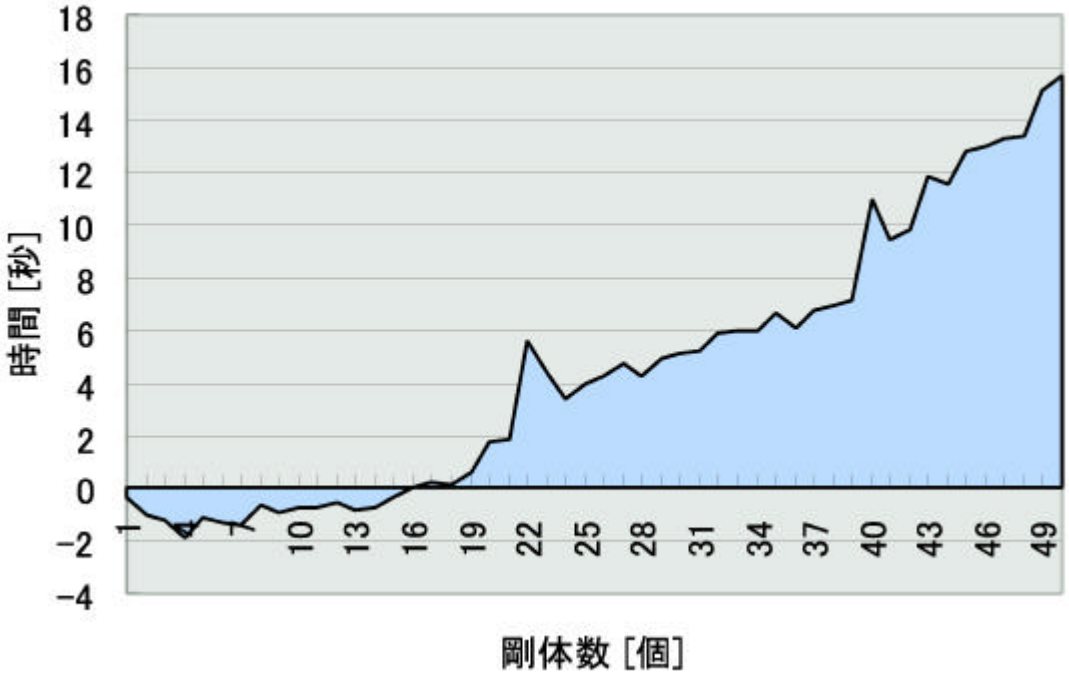


図 5.8: 剛体の数に対する分散・並列化の効果 (4 台に分散・並列化)

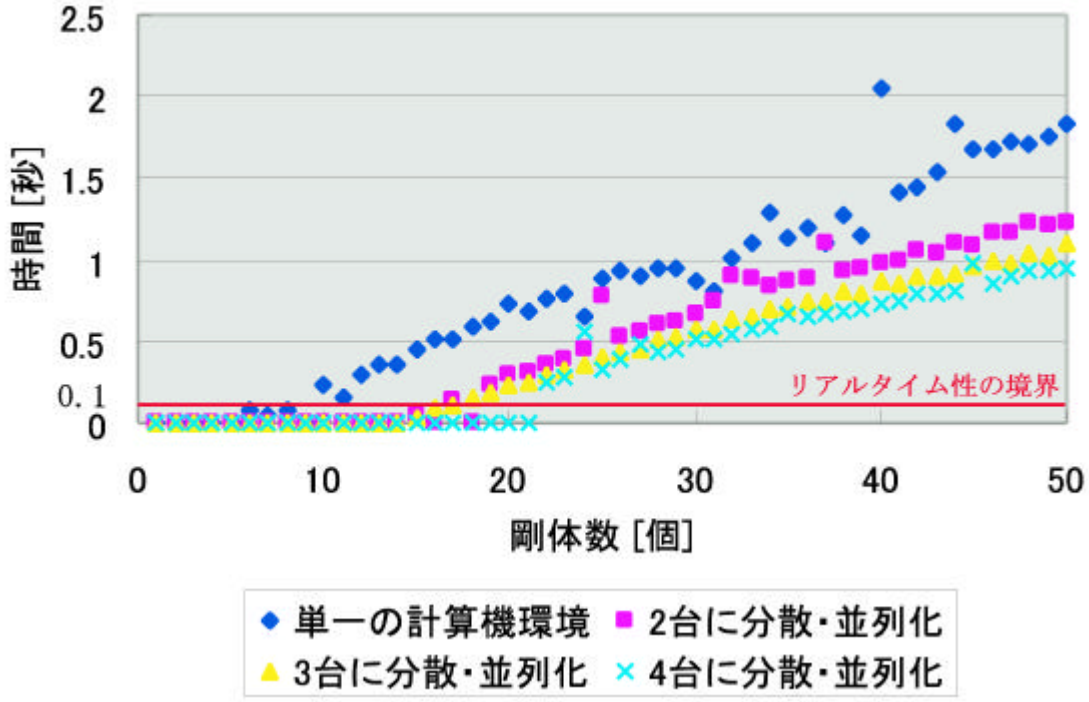


図 5.9: 1 フレームの処理に要した最大時間

をリアルタイム性を実現したシミュレーションと判断する。

5.1.4 評価実験の考察

実験結果の考察を以下に示す。考察する内容は、計算機の性能が変化した場合の分散・並列化の効果の変化、ネットワーク性能の性能が変化した場合の分散・並列化の効果の変化、分散・並列処理の効果が低い仮想空間の特徴、計算処理を終了するまでに要する時間の理論値と計測値の比較、リアルタイムシミュレーションが実現可能な仮想空間の複雑さの限界の 5 点である。

計算機の性能が変化した場合の分散・並列化の効果の変化

図 5.1 および図 5.2 に示したように、分散・並列処理をおこなう計算機の台数が増加するほど、処理を終了するまでに要する時間が短くなる。ただし、剛体の数が少ない時に限り、PC クラスタ環境より単一の計算機環境の方が処理を終了する時間が短い。この点については後に考察する。仮想空間内に配置する剛体の数が増加すると、単一の計算機環境ではシミュレーションの計算処理を終了するまでに要する時間が指数関

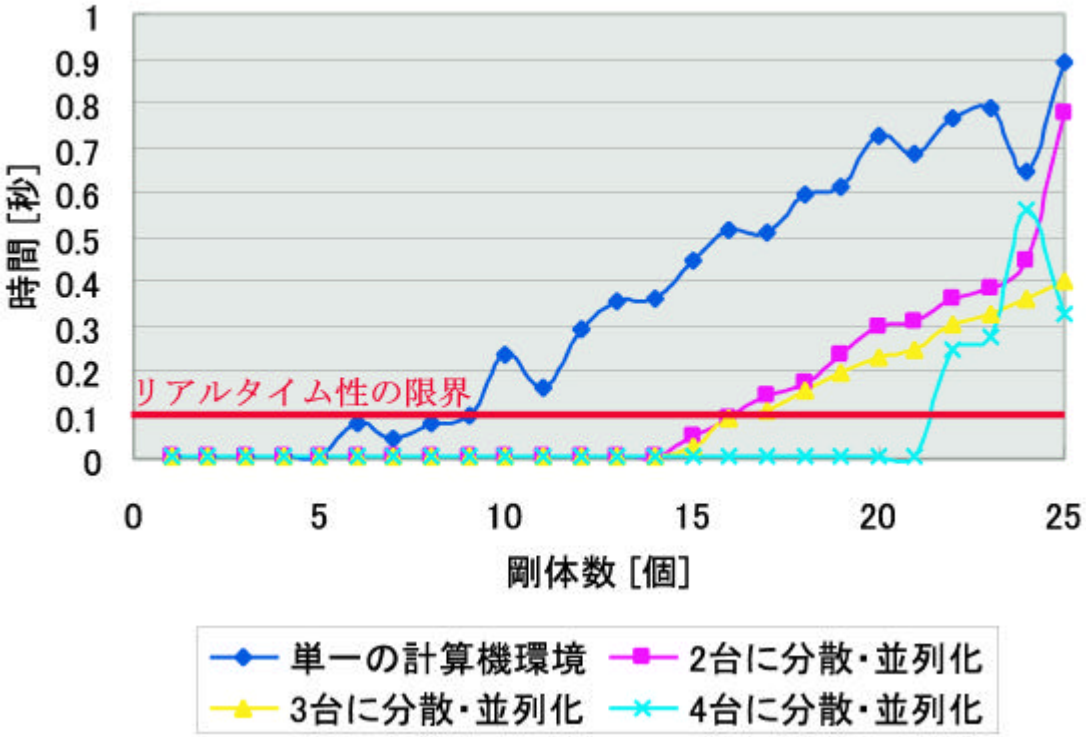


図 5.10: 1 フレームの処理に要した最大時間 (剛体 1 個から 25 個)

数的に増加する。しかし、PC クラスタ環境では計算処理を終了するまでに要する時間は単一の計算機環境に比べてより線形的に増加する。その傾向は、処理をおこなう計算機の台数が増加するほど顕著になり、分散・並列化の効果が計算性能に依存していることを確認できる。なお、ここでいう計算性能とは、計算用ノードマシンの性能やネットワークの性能を含めた PC クラスタ環境全体の性能を指す。

なお、単一の計算機環境で剛体の数が 22 個と 40 個の時、処理を終了するまでに要する時間が飛び抜けて高くなった。これは、例えば、剛体が何重にも積み重なり短時間に非常に数多くの衝突を発生する等の、非常に計算負荷がかかる現象がシミュレーション中に発生したためと予想できる。しかし、同じ仮想空間を分散・並列化してシミュレーションした場合、単一の計算機環境におけるシミュレーションのように飛び抜けて計算処理を終了するまでの時間が大きくなる現象は確認されなかった。これは、計算負荷を複数の計算機に分散する本システムの機能が正しく機能していることを示している。

ネットワーク性能に対する分散・並列化の効果

図 5.3、図 5.4 および図 5.5 に示したように、10BASE-T の HUB を用いた場合に比べ 100BASE-T の HUB を用いた場合の方が分散・並列化の効果が高くなる。この傾向は、PC クラスタを構成する計算機の数が増加するほど、計算機間の通信回数とデータ量が増加するため顕著になる。つまり、効率的な分散・並列化を実現するためには、PC クラスタを構成する計算機の性能はもちろん、個々の計算機を繋ぐネットワークの性能が重要であることを確認できる。

なお、環境 2 で剛体の数が 14 個と 30 個の時、10BASE-T の HUB を用いたシミュレーションの方が、100BASE-T の HUB を用いたシミュレーションよりも先に計算処理を終了している。この現象は、環境 3、4 のように計算機の数を増やした時には見られないことから、100BASE-T の HUB を用いたシミュレーションにおいて、シミュレーション中に計算機の性能が変化し、適切な処理を割り当てることができなかった可能性が考えられる。

分散・並列処理の効果が低い仮想空間

5.1.4 項で述べたように、分散・並列化を実現すればシミュレーションの計算処理に要する時間を必ず短縮できるわけではない。計算機の性能を推定する機能や負荷の割り当てを決定する機能等、分散・並列化を実現するためには単一の計算機環境におけ

るシミュレーションでは必要ない機能を実装する必要があり、シミュレーションする仮想空間によっては計算処理を終了するまでに要する時間がかえって増加する場合がある。図 5.6、図 5.7 および図 5.8 に示したように、仮想空間内の剛体の数が少ない場合は単一の計算機環境の方が効率的にシミュレーションをおこない、剛体の数が増加するほど PC クラスタ環境の方が効率的なシミュレーションをおこなう。なお、本システムを用いた場合は剛体の数 15 個から 20 個が単一の計算機環境と PC クラスタ環境のシミュレーションの効率が入れ替わる境界となり、単一の計算機環境が計算処理を先に終了するシミュレーションは比較的剛体の数が少ない場合に限られた。

リアルタイムシミュレーションが実現可能な仮想空間の複雑さの限界

本研究では、人が違和感を感じない描画速度の限界とされている毎秒 10 フレームの更新速度をリアルタイムの条件としているため、リアルタイムシミュレーションを実現するためには 1 フレームの計算処理を終了するまでに要する時間が常に 0.1 秒以下であることが求められる。図 5.9 および図 5.10 に示したように、剛体の数が増加するほど 1 フレームの処理を終了するまでに要する時間の最大値も増加する。しかし、単一の計算機環境では剛体の数が 9 個の時までリアルタイム性を確保できたのに対し、2 台の計算機を用いた PC クラスタ環境では剛体の数が最低で 16 個の時までリアルタイム性を確保でき、剛体挙動シミュレーションの分散・並列化によりリアルタイムシミュレーションが実現可能な仮想空間の複雑さの限界が向上したことが確認できる。さらに、4 台に分散・並列化した PC クラスタ環境では 21 個の時までリアルタイム性を確保でき、単一の計算機環境に比べ仮想空間に配置することができる剛体の数が 2 倍以上に増えている。そのことからも、ピーク時の負荷を分散しシミュレーション速度を一定に保つ上で、分散・並列化が非常に有効であることが確認できる。

なお、グラフの形が歪になる理由の 1 つは、1 フレームの処理に要する最大の時間が、必ずしも剛体の数に比例せず、シミュレーションする仮想空間の状況に大きく影響されるからである。例えば、10 個の剛体をお互い衝突しないように離れた場所で床に落下させるシミュレーションに要する計算時間は、3 個の剛体がお互い短時間に連続して衝突するように床に落下させるシミュレーションに要する計算時間より、少ない可能性が高い。

しかし、剛体の数が 25 個と 38 個の時、PC クラスタ環境より単一の計算機環境の方が計算処理に要する時間が短くなっている。これは、計算機に処理を割り当てるアルゴリズムか、計算機に情報を送受信する機能にプログラム上の不具合が存在し、不適

切な分散・並列化をおこなっている可能性が考えられる。この問題は、不具合が生じる仮想空間を詳細に検討し、今後解決すべき課題としたい。

考察のまとめ

以上の考察をまとめると次のようになる。

1. 分散・並列化の効果はPCクラスタ環境の性能に依存する。PCクラスタを構成する計算機が多く、計算機を接続するネットワークが高速であるほど、分散・並列化の効果が高くなる。50個程度のナットとボルトが床に向かって落下する例では、計算機を2台接続した環境で約55%、計算機を4台接続した環境で約127%の高速化が実現できた。

2. 分散・並列化の効果はシミュレーションする仮想空間の内容に依存する。仮想空間が複雑であればあるほど分散・並列化の効果が高くなる。

3. 分散・並列化には適さない仮想空間が存在する。仮想空間内の剛体が少ない場合、分散・並列化をおこなわない方が効率的にシミュレーションできる。

4. 分散・並列化の効率は上限がある。上限は並列化率によって定まる。

5. 分散・並列化はリアルタイムシミュレーションの実現に有効である。ピーク時の計算負荷を分散することで、シミュレーション速度を安定させることができる。

本実験では、現在容易に入手できる計算機環境上で剛体挙動シミュレーションの分散・並列化の効果を確認した。また、機器保修の訓練環境のような大規模複雑な仮想空間ほど分散・並列化に適していることを確認した。さらに、シミュレーションのピーク時の負荷を分散し、シミュレーション速度を一定に保つことで、より複雑な仮想空間がリアルタイムシミュレーション可能となることを確認した。しかし、本システムは、剛体21個(ボルト形状10個、ナット形状11個)を床形状に落とし跳ね返らせる程度の仮想空間の複雑さがリアルタイムなシミュレーションを実現する限界であった。そのため、Physically-baseの手法を用いた機器保修の訓練環境を実現するためには、計算性能やネットワーク性能の向上を待つのみならず、より効率的な分散・並列化を実現する手法の開発が不可欠である。

5.2 今後の課題

本システムは、単一の計算機環境で動作する従来のシステムと比べ、より複雑な仮想空間のリアルタイムシミュレーションが可能となったものの、機器保修の訓練環境を実現するまでは至らなかった。そこで、機器保修の訓練環境のような大規模複雑な仮想空間のリアルタイムシミュレーションが可能となるよう、本システムを高速化するために考えられる改良点を以下に示す。

1. 単独の計算機の性能を向上させる。
2. GigabitEthernet や Myrinet 等の次世代 LAN を導入する。
3. PC クラスタを構成する計算用ノードマシンの数を増やす。
4. 並列化率が高くなるように分散・並列化手法を改良する。
5. 効率的に処理を割り当てる機能を実装する。

1 と 2 の、単独の計算機の性能向上と次世代 LAN の導入は比較的容易に実現できる。計算機性能の向上は、PC クラスタを構成する計算機の CPU をより高速なものに取り替えればよく、次世代 LAN の導入は、計算機の LAN カードと計算機を繋ぐ HUB を次世代 LAN の対応製品に取り替えればよい。しかし、この方法では性能向上の限界が入手可能な CPU や HUB 等の市販製品の性能に依存するため、技術革新により高性能な新製品が登場するまで性能を上げることができない。また、本研究は容易に入手可能な計算機環境として大学の研究室のように既にネットワーク接続された環境を想定しているため、PC クラスタの性能向上を目的に新たに機器を導入することは本研究の趣旨として望ましくない。そこで、上に挙げた改良点のうち、3、4 および 5 を本研究が定める今後の課題とする。

- PC クラスタを構成する計算用ノードマシンの数を増やす。

  3.2.3 項で求めた並列化率 $P = 0.78$ および式 (4.3) より、単一の計算機環境で計算処理を終了するまでに要する時間 t_1 と n 台の計算機を用いた PC クラスタ環境で計算処理を終了するまでに要する時間 t_n は次式のような関係がある。

$$t_n = t_1 \times \left(\frac{0.78}{n} + 0.22 \right) \tag{5.1}$$

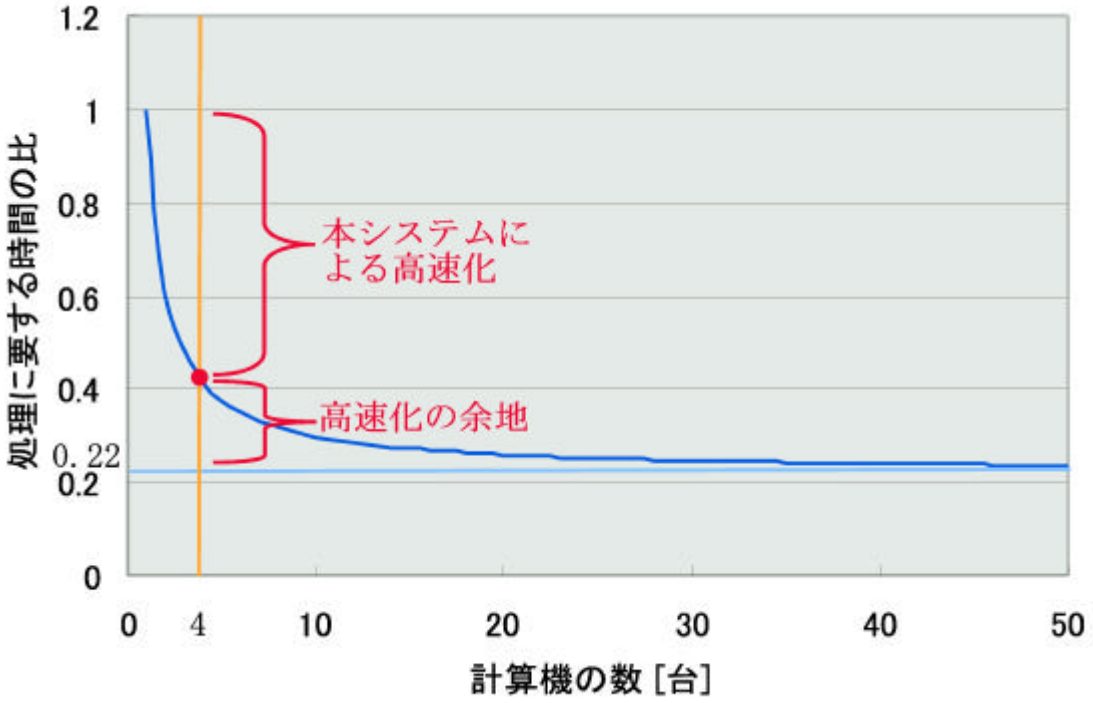


図 5.11: 計算機の数と処理時間の短縮

式 (5.1) において、$t_1 = 1$ とした時の計算機の台数 n と計算処理を終了するまでに要する時間 t_n の関係を図 5.11 に示す。本研究は剛体挙動シミュレーションの処理の一部を最大 4 台の計算機を用いて分散・並列処理をおこなったが、図 5.11 に示すように、本研究が開発した分散・並列化手法の効果を最大限に引き出すためには、さらに多くの計算機を用いて分散・並列処理をおこなう必要がある。

- 並列化率が高くなるように分散・並列化手法を改良する。

  分散・並列化の効果には並列化率によって定まる上限が存在する。つまり、並列化率を高めることは分散・並列化の効果を高めることに他ならない。本研究が実現した 0.78 以上の並列化率を得るためには、厳密な衝突判定と接点に働く力の算出に加え新たな処理の分散・並列化を実現するか、もしくは、分散・並列化をおこなわない処理にかかる負荷を減らさなければならない。ただし、前者の方法を用いて、分散・並列化可能な全ての処理を計算機に分散すると、計算機間の通信量が膨大となり、シミュレーション速度が低下する。そのため、計算負荷と通信負荷のバランスを保つように分散する処理を決定しなければならない。3.2.3 項の負荷実験の結果から、単一の計算機環境における剛体挙動シミュレーションの各

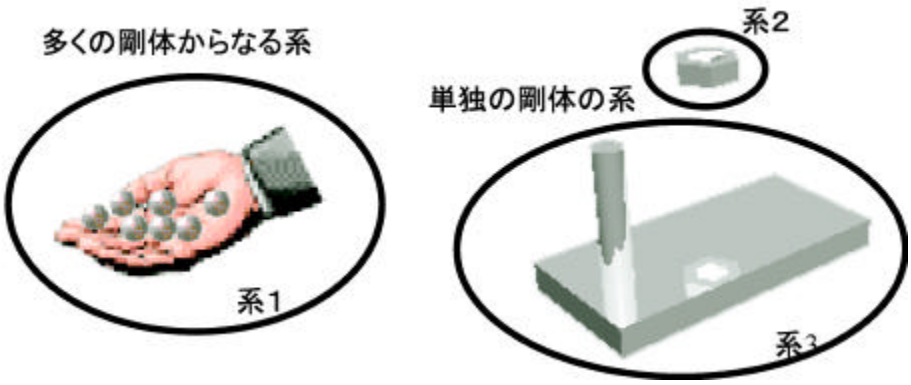


図 5.12: 系に属する剛体の数の偏り

処理のうち、厳密な衝突判定と接点に働く力の算出以外で負荷が大きな処理は系の算出である。系の算出は、剛体間の接点の状態を監視し、仮想空間内にどれほどの系があるのか、その系にはどの剛体が含まれているのかを求める処理である。また、系に含まれる剛体の状態から、その系が静止系であるかどうかを判定する処理も系の算出がおこなう。そのため、剛体の数が多ければ多いほど、系の算出にかかる負荷は増大する。この系の算出にかかる負荷を、アルゴリズムを改善することで減らすことが、並列化率を高める上で有効である。

- より効率的に計算機に処理を割り当てる機能を実装する。

  4.2.3 項で述べたように、計算タスクを割り当てる際、本研究のアルゴリズムでは必ずしも最適なタスクの割り当てをおこなうとは限らない。より効率的な分散・並列化を実現するためには、最適なタスクの割り当てを高速に求めるアルゴリズムを開発する必要がある。しかし、たとえ最適にタスクを割り当てたとしても、割り当てるべきタスクに著しい違いが生じる場合がある。例えば、図 5.12 に示すように、系に属する剛体の数に違いがあると、それぞれの系の計算負荷に大きな違いが生じるため、タスクを均等に計算機へ割り当てることが難しくなる。そこで、比較的質量の大きな剛体を介した間接接触では、お互いの影響が小さいことを利用して系を分割する処理を検討する。例えば、図 5.13 に示すように、質量が比較的大きな手の上に質量が比較的小さな球が乗っている場合、手を介して全ての球は間接接触状態にあっても、離れた球同士が手を介してお互いの接触状態に影響を及ぼし合うことは無いと近似することができる。よって、直接接している球同士と手から構成される複数の系に分割することができる。

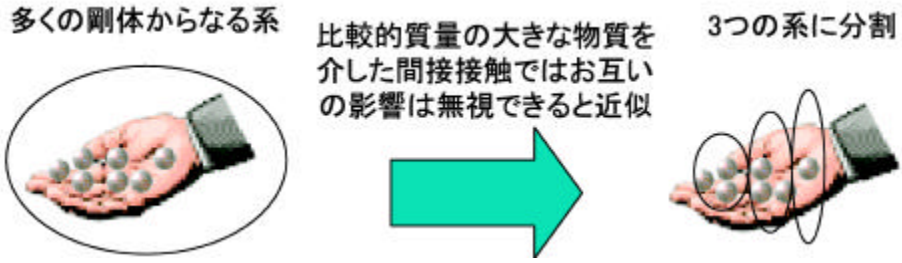


図 5.13: 系の分割

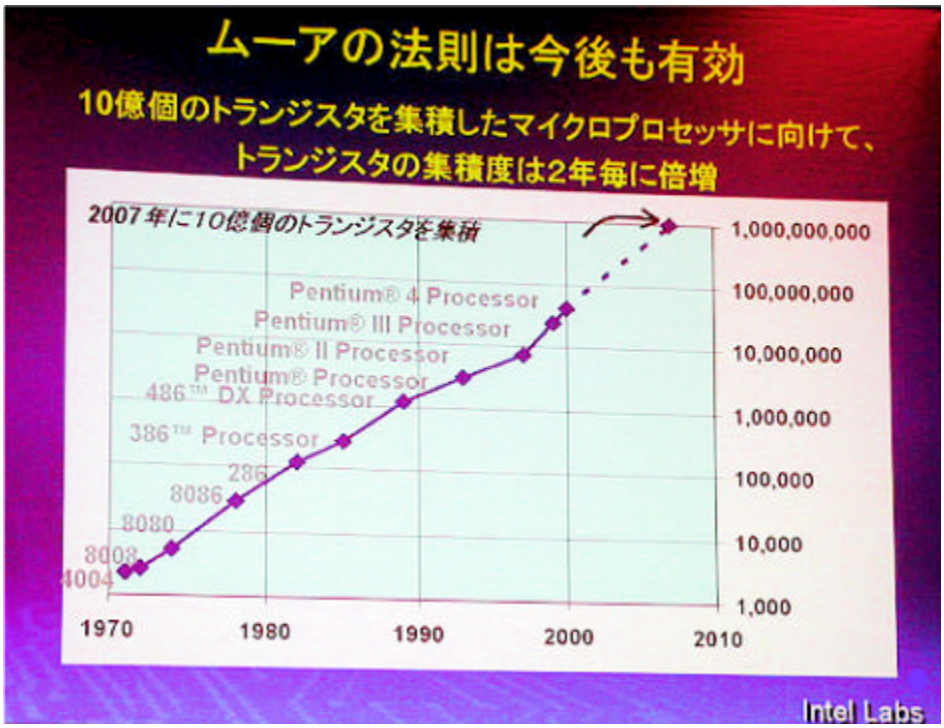


図 5.14: インテルの「テラヘルツ・トランジスタ・アーキテクチャ」

5.3 計算機性能の向上と人工現実感技術の進歩への展望

インテル社は 2001 年 11 月、「テラヘルツ・トランジスタ・アーキテクチャ」を発表した。図 5.14 に示すように、現在のギガヘルツ・トランジスタの 10 倍もの演算能力を持つ CPU を、2007 年に実現する技術的な目処がついたという [45]。これは、今後は成立が困難になるであろうと一般に言われているムーアの法則が、引き続き有効であることを自ら宣言したことに等しい。もちろん、インテル社の発表通り CPU の性能が向上するとは限らないが、このような発表がおこなわれた背景には、CPU の性能向上に寄せられる期待が引き続き大きいことが挙げられる。

さらなる進歩が期待されている計算機技術は CPU の性能向上だけではない。例えば、

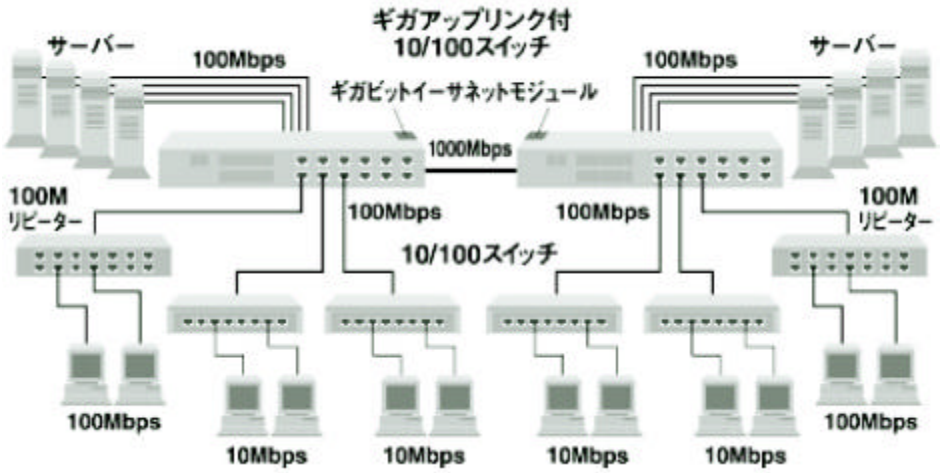


図 5.15: Gigabit Ethernet をスイッチ間の高速リンクに用いる例

近年のネットワークトラフィックの急激な増加にともない、より広帯域なネットワーク環境として Gigabit Ethernet が注目を集めている。図 5.15 に Gigabit Ethernet をスイッチ間の高速リンクに用いる例を示す。このように、光ケーブルや平衡型ケーブルを用いて 1000Mbps を実現する Gigabit Ethernet は、従来の 10BASE-T や 100BASE-T の資産を活かしつつ低コストで広帯域ネットワーク環境を構築できるため、現在急速に普及している。また、PC クラスタの分野では、クラスタ間の高速通信を実現するために開発された Myrinet の導入が進められており、Gigabit Ethernet を越える 2000Mbps もの広帯域を既に実現している。

このように、計算機の性能とネットワークの性能は、今後しばらくの間は現在の発展速度を維持し続ける可能性がある。仮に、インテル社がロードマップに示した通りに CPU の性能が向上し、Myrinet によって実現されている広帯域なネットワーク環境が一般の LAN 環境にも普及すれば、2010 年には本研究で構築した PC クラスタ環境に比べ、計算機の性能で 10 倍、ネットワークの性能で 20 倍も高性能な PC クラスタ環境が実現される可能性がある。これは、1995 年に Microsoft 社が Windows95 を発売した際、標準的とされた Pentium プロセッサと 10BASE-T による計算機環境が、2002 年の現在、Pentium4 と 100BASE-T による計算機環境にまで発展したことに相当する。1995 年当時、主に表計算処理や文章作成に用いられていたパーソナルコンピュータが、現在、リアルな 3 次元画像の合成や音声合成に用いられるまで用途を広げるに至ったことを考えれば、計算機性能の向上がもたらす恩恵の大きさを計り知ることができる。

一方、人工現実感技術の進歩はとどまるところを知らず、仮想空間の再現性は日々向上している。例えば、衣服のような不定形の物体のリアルな挙動を合成する技術 [46] や、

大気の状態をシミュレーションし本物と見間違うほどの空の画像を合成する技術[47]等が開発され、より実世界に近い仮想空間を構築することが可能となっている。また、仮想空間内に構築した仮想物体を本当に手で触っているかのような触感を与えることができるデバイス[48]や、本当に仮想空間内にいるかのように感じさせることができる没入型のディスプレイ[49]など、構築した仮想空間をより実世界と同様に仮想体験することが可能となっている。

以上のように、計算機技術と人工現実感技術は現在まで共に発展してきた。これは、計算機技術の進歩が人工現実感技術の進歩を促してきたからに他ならない。しかし、現在、人工現実感技術は急速な発展を遂げた結果、計算機の性能がいくら向上しても足りないという状況に陥っている。この状況は、仮に計算機性能の向上が減速もしくは停滞するようなことになれば、人工現実感技術の進歩に多大な影響を与える可能性が高いことを示唆している。ムーアの法則に限界が指摘されている今、単一の計算機の性能に頼ることなく人工現実感技術を現在のペースで発展させるための新たな技術の開発が急がれる。

本研究は、機器保修の訓練環境を実現する際に必要な Physically-base の手法に基づく剛体挙動シミュレーションの処理の一部を、PC クラスタを構成する複数台のノードマシンに分散し並列処理する手法を開発し、現時点で容易に入手可能な計算機環境上でのリアルタイムシミュレーションの実現可能性の検討をおこなった。その結果、剛体挙動の分散・並列処理を実現することにより、リアルタイム性を確保することができる仮想空間の複雑さの限界が増加することを確認した。これは、リアルタイムシミュレーションを実現する際、単一の計算機の性能向上に依存することなく人工現実感技術を発展させる技術として、PC クラスタを用いた分散・並列処理が有効であることを示している。

人工現実感技術の発展により、今後さらに実際の訓練に近い仮想体験を訓練生に提示できる技術が確立され、機器保修の訓練環境を実現する際に必要な計算量が益々増加することが予想される[50]。しかし、PC クラスタを用いた分散・並列処理の利点を活用することで、機器保修の訓練環境のような大規模複雑な仮想空間のリアルタイムシミュレーションが実現できるようになると期待する。

第 6 章 結論

本研究では、機器保修の訓練環境を実現する際に必要な Physically-base の手法に基づく剛体挙動シミュレーションを、既存の剛体挙動のモデル化手法を拡張することで実現し、計算機上へ実装した。さらに、実装した剛体挙動シミュレーションの処理の一部を、PC クラスタを構成する複数台のノードマシンに分散し並列処理する手法を開発し、現時点で容易に入手可能な計算機環境上でのリアルタイムシミュレーションの実現の可能性を検討した。

第 2 章では、人工現実感技術を用いた機器保修の訓練環境の利点と研究開発の現状について述べ、仮想空間内の仮想物体の挙動を制御する代表的な手法である Rule-base の手法と Physically-base の手法について述べた。さらに、パーソナルコンピュータを用いたメモリ分散型の並列計算機である PC クラスタの利点について述べ、機器保修の訓練環境に PC クラスタを導入することを提案し、本研究の目的を明らかにした。また、PC クラスタを用いた分散・並列処理の従来研究を紹介し、本研究と従来研究の目標の相違を明確にした。

第 3 章では、まず、シミュレーションを計算機に実装するための剛体挙動のモデル化について述べ、剛体の状態遷移や衝突判定、接点に働く力の算出方法等について述べた。次に、本モデル化手法を用いて単一の計算機環境上に実現した剛体挙動シミュレーションシステムのシステム構成と、構築したシステムを用いた仮想空間の構築方法について述べた。そして、剛体挙動シミュレーションの処理のうちどの処理にどの程度の計算量が生じるのかを調べるため、構築したシステムを用いて実施した負荷計測実験について述べた。実験の結果、高負荷時には剛体挙動シミュレーションに必要な処理のうち、ポリゴンレベルでおこなう厳密な衝突判定と Dantzig のアルゴリズムを用いた接点に働く力の算出に特に高い計算負荷がかかっていることを確認した。

第 4 章では、まず、分散・並列処理の分野で使用される用語やその定義について述べ、効率的な分散・並列化を実現する上で注意すべき点について述べた。次に、剛体挙動シミュレーションの処理の一部を分散し並列処理する手法を提案し、分散・並列化を実現する処理や計算機に処理を動的に割り当てる方法、計算機間の通信プロトコルについて述べた。そして、本手法を用いて PC クラスタ環境上に実現した剛体挙動シミュレーションシステムのシステム構成とシミュレーションの処理の流れについて述べた。

第5章では、本研究で構築した剛体挙動シミュレーションシステムを用いて実施した評価実験について述べた。具体的には、同一の仮想空間の剛体挙動シミュレーションを単一の計算機環境で実現する場合とPCクラスタ環境で実現する場合のシミュレーション速度を比較し、剛体挙動のリアルタイムシミュレーションの実現可能性を検討した。その結果、現在容易に入手できる計算機環境上で剛体挙動シミュレーションの分散・並列化の効果を確認した。また、機器保修の訓練環境のような大規模複雑な仮想空間ほど分散・並列化に適していることを確認した。さらに、シミュレーションのピーク時の負荷を分散し、シミュレーション速度を一定に保つことで、より複雑な仮想空間がリアルタイムシミュレーション可能となることを確認した。その後、より効率的な分散・並列化を実現するための課題を明らかにし、今後の展望についてまとめた。

本研究で実現した剛体挙動シミュレーションの分散・並列化の最大の利点は、ピーク時の負荷をPCクラスタを構成する複数の計算機に分散することで、単一の計算機環境と比べシミュレーション速度が一定の速度に安定する点である。今後、この利点を生かすことで、機器保修の訓練環境のような大規模複雑な仮想空間のリアルタイムシミュレーションが実現できるようになると期待される。

謝 辞

本研究を進めるにあたり、研究全般にわたってご指導を頂きました吉川榮和教授に深く感謝いたします。

本研究を進めるにあたり、数々の貴重な助言を頂きました下田宏助教授に深く感謝いたします。

本研究を進めるにあたり、時には暖かく、時には厳しく、直接のご指導を頂きました石井裕剛助手に深く感謝いたします。

本研究を進めるにあたり、プログラミング作業や論文の校正等、様々な面で協力して頂きました修士課程 1 回生の遠藤啓介君に深く感謝いたします。

本研究を進めるにあたり、暗く寂しい夜の研究生活を、楽しく充実したものに変えて頂きました修士課程 2 回生の岡田芳信君に深く感謝いたします。

研究生活において、ともすれば挫けそうになった際、お互いに慰め、励まし合った修士課程 2 回生の神月匡規君に深く感謝いたします。

研究生活において、時には厳しいつっこみを、時には更に厳しいつっこみを頂きました修士課程 2 回生の近藤寛子さんに深く感謝いたします。

修士課程の 2 年間、非常に個性的な同期達の良心として孤軍奮闘して頂きました修士課程 2 回生の高橋ともさんに深く感謝いたします。

修士課程の 2 年間、同期のリーダーとしていつも力強く導いて頂きました修士課程 2 回生の早瀬賢一君に深く感謝いたします。

本研究室に配属されて以来 3 年間、辛い時は互いに叱咤激励し、苦しい時は互いに切磋琢磨した強敵（とも）、修士課程 2 回生の松﨑剛士君に深く感謝いたします。

修士論文の丁寧な校正を頂きました、修士課程 1 回生の越智和弘君、小林隆君、鮫島良太君、新田和弘君、服部貴司君に深く感謝いたします。

最後に、研究を進める上で何かとお世話頂きました谷友美秘書、吉川万里子秘書および吉川研究室の学生の皆様にも心から感謝いたします。

参 考 文 献

[1] H. Ishii, K. Endou, K. Sharyo：A New Integration Method of Constructiong an Interactive Virtual Environment for the Collaboration between Virtual Agent and Real Human, Semiotica, Proceedings of ROMAN'01 10th IEEE International Workshop, pp.50-55 (2001).

[2] 柏健一郎：機器補修訓練へのデータグローブの適用性の研究, 京都大学工学部電気工学第二学科学士論文 (1993).

[3] 市口誠道：アフォーダンスの概念に基づく人体モーション合成システムの開発, 京都大学大学院エネルギー科学研究科修士論文 (2000).

[4] 南雲俊喜, 中山功, 甘利治雄, 岡田幹夫：小型円筒面スクリーンによる運転・保守作業訓練環境の構築, 日本バーチャルリアリティ学会第 3 回大会論文集, pp.207-208 (1998).

[5] 天野友博, 田中和明, 鄭絳宇, 安部憲広：仮想機械を用いる機器修復法の教示と誤りの検出・修正機構, 日本バーチャルリアリティ学会第 2 回大会論文集, pp.89-92 (1997).

[6] 新井浩一, 阿部慶子, 上地登：VR 技術を用いた変電所保守員向け集合教育用体感型シミュレータの開発, 日本バーチャルリアリティ学会論文誌, **Vol. 2**, No.4, pp.7-16 (1997).

[7] 小牧大輔：オブジェクト指向に基づく仮想空間構築手法に関する研究, 京都大学大学院エネルギー科学研究科修士論文 (2001).

[8] ゴードン・ムーア, 玉置直司 (取材・構成)：「インテルとともに」私の半導体人生, 日本経済新聞社 (1995).

[9] 雨宮真人, 田中譲：コンピュータアーキテクチャ, 知識工学講座 7, オーム社 (1988).

[10] 西野哲朗：量子コンピュータ入門, 東京電気大学出版局 (1997).

[11] J.C.Setubal, J.Meidanis, 五條堀孝 (監訳)：分子生物学のためのバイオインフォマティクス入門, 共立出版 (2001).

[12] 矢川元基, 塩谷隆二：超並列有限要素解析, 朝倉書店, 第 2 章 (1998).

[13] 廣安知之：PC クラスタの作り方, PC クラスタ超入門 2000, 並列計算機研究会講義 1 (2000).

[14] 加藤康義，光成友孝，築山洋：セルオートマトン — 複雑系の自己組織化と超並列処理, 森北出版株式会社 (1998).

[15] 石川幹人：並列計算機を用いたタンパク質の配列のアライメント解析, ICOT TR-0902 (1995).

[16] 川西祐一：遺伝子情報処理と超並列処理, 関西支部セミナー 超並列処理の動向と展望, 情報処理学会関西支部 (1992).

[17] Brian Hayes：Computing Science - Collective Wisdom, March-April 1998, American Scientist (1998).

[18] Mi Li, Garrett M. Morris, Taekyu Lee：Structural studies of FIV and HIV-1 proteases complexed with an efficient inhibitor of FIV protease, **Vol.38**, Issue 1, pp.29-40 (2000).

[19] Michael R. Shirts, Vijay S. Pande：Mathematical Analysis of Coupled Parallel Simulations, PHYSICAL REVIEW LETTERS, **Vol.80**, No.22, pp.4983-4987 (2001).

[20] 小木哲朗, 渡辺浩志, 廣瀬通孝：仮想世界シミュレーションのための並列エンジン, 日本バーチャルリアリティ学会大会論文集, **Vol.3**, pp.211-214 (1998).

[21] 谷村勇輔, 廣安知之, 三木光範：PC クラスタにおける 2 個体分散遺伝アルゴリズムの高速化, 情報処理学会研究報告, 200-HPC-82, pp.161-166 (2000).

[22] 高橋亮一：コンピュータによる流体力学, 構造計画研究所, pp.172-186 (1982).

[23] J. Clyne, J. Dennis：Interactive Direct Volume Renddering of Time-Varying Data, Joint EUROGRAPHICS - IEEE TCVG, Symposium on Visualization (1999).

[24] 堀田大介, 山下陽代：CATV VOD システム開発における技術的課題と解決策について, INTEC Technical Report, **Vol.56** (2001).

[25] Eric Korpela, Paul Demorest, Eric Heien, Carl Heiles, Dan Werthimer：Using SETI@home data to explore hydrogen distribution in the Galaxy, NEWSLETTER 10 (2001).

[26] 瀧和男：第五世代コンピュータの並列処理, bit 別冊 第 III 編 3.8, 共立出版 (1993).

[27] 社領一将：仮想空間内における人の動作の融合方法に関する基礎研究, ヒューマンインタフェース学会研究報告集, **Vol.2**, No.2, pp.63-68 (2000).

[28] 手塚一佳, 藤井太洋, 福地亮介, 西野道人, 山本健介, 金築真：物理シミュレーションの表現に TRY！, Computer Graphics World 1 月号, (2002).

[29] Brian Mirtich: Timewarp Rigid Body Simulation, MERL - A MITSUBISHI ELECTRIC RESERCH LABORATORY (1999).

[30] David Baraff：Analytical Methods for Dyanamic Simulation of Non-penetrating Rigid Bodies, SIGGRAPH '89 Computer Graphics, **Vol.23**, No.3, pp.223-232 (1989).

[31] 小木哲朗, 渡辺浩志, 廣瀬通孝：インタラクティブシミュレーションのための並列予測, 日本機械学会 第 7 回設計工学・システム部門後援会講演論文集, No. 97-69 pp. 280-283(1997).

[32] Philip M. Hubbard：Approximating Polyhedra with Spheres for Time-Critical Collision Detection, ACM Transactions on Graphics, **Vol.15**, No. 3, pp. 179-210 (1996).

[33] Brian Mirtich：V-Clip:Fast and Robust Polyhedral Collision Detection, MERL - A MITSUBISHI ELECTRIC RESERCH LABORATORY, TR-97-05 (1999).

[34] 村田敏幸：X68000 マシン語プログラミング 「探しもの」, Oh!X 1992 年 12 月号, ソフトバンク (1992).

[35] 池井寧, 安部憲広：VR 世界における剛体の力学, VR 世界の構築手法, 培風館, 第 2 章 (2000).

[36] Brian Mirtich, John Canny：Inpulse-based simulation of rigid bodies, In Proceedings of Symposium on Interactive 3D Graphics (1995).

[37] Parris K. Egbert, Scott H. Winkler：Collision-free object movement using vector fields, IEEE Computer Graphics and Application, pp.18-24, July (1996).

[38] 長田隆:マルチボディ・システムの統一的定式化法, 機論, 66-649, C(2000), pp.2917-2923 (2000).

[39] 加藤千典：接触の拘束条件の変化する剛体の運動のモデル化, 東京大学工学部精密機械工学科修士論文 (1995).

[40] 川地克明：機械機構の挙動の直感的な把握を目的とした剛体運動シミュレーション手法, 東京大学工学部精密機械工学科修士論文 (1997).

[41] 株式会社マイクロ・シー・エー・デー：物理法則を融合した仮想空間モデリングシステムの開発, 高度情報化支援ソフトウェアシリーズ育成事業, 98 第 003 号 (1998).

[42] David Baraff: Fast Contact Force Computation for Nonpenetrating Rigid Bodies, SIGGRAPH '94 Computr Graphics Proceedings Annual Conference Series 1994, pp.23-34 (1994).

[43] David Baraff：Coping with Friction for Non-penetrating Rigid Body Simulation, SIGGRAPH '91 Computer Graphics, **Vol. 25**, No.4, pp.31-40 (1991).

[44] チャールズ=マーレイ, 小林達: スーパーコンピュータを創った男, 廣済堂出版 (1998).

[45] DOS/V SPECIAL 編集部: インテルの「テラヘルツ」トランジスタ技術, DOS/V SPECIAL, **Vol. 25**, pp.15-17 (2002).

[46] Jeff Rickel, W.Lewis Johnson：Animated Agents for Procedual Training in Virtual Reality : Perception, Cognition, and Motor Control, Univercity of Southern California (1999).

[47] 田中貴志：3DCG で創造する爆発、雲、水, Computer Graphics World 1 月号 (2001).

[48] 藤田欣也, 片桐宏：指先接触力検出と接触面積制御による物体柔らかさの提示, ヒューマンインタフェース学会研究報告集, **Vol. 2**, No.2, pp.81-84 (2000).

[49] 廣瀬通考, 寺田実, 横井茂樹, 桐山孝司, 佐藤宏介：さまざまな VR 世界の構築例, VR 世界の構築手法, 培風館, 第 2 章 (2000).

[50] 石井裕剛, 吉川榮和：プラント技能訓練へのバーチャルリアリティ技術適用の展望, 計測自動制御学会 システム・情報部門シンポジウム 2000, 講演論文集, pp.55-60 (2000).

付 録 目 次

付 録 図 目 次

付録 A Voronoi Clip

3.1.3 項で述べた、衝突判定アルゴリズム Voronoi Clip について述べる。

概要

Voronoi Clip(又は V-Clip)[1] は、三次元形状を面、稜線、頂点のリストからなるデータ構造で定義する境界線表現 (B-rep:boundary representation) により記述された凸面体のための衝突判定アルゴリズムである。V-Clip は、多面体を構成する面・稜線・頂点をまとめて「フィーチャ」として扱い、多面体間で最も近いフィーチャのペアを追跡することによって２物体間の最短距離を計算する。V-Clip は、Lin-Canny による Closest Feature Algorithm[2] に類似しており、その手法の欠点を克服するために設計された。

1. V-Clip は、干渉している場面も扱える。計算時間は、干渉していないときと殆ど変わらない。

2. V-Clip は、頑健である。縮退した形状は、問題にならなく、無限ループに陥らない。(Lin-Canny アルゴリズムでは、互いに平行な稜線間で最近接点が一意に決定できない場合に無限ループに陥っていた。)

3. V-Clip の実装は、Lin-Canny と比べて非常に簡単である。

また、V-Clip は、凹面体を凸面体の階層構造として構築することによって、凹面体にも適用可能である。この手法は、適度な数の凸面体で、階層構造があまり深くならないように凸分割が出来るときには、非常に有効である。

V-Clip、Lin-Canny、そして拡張 GJK[3] について正確性・頑健性に関するテスト行ったところ、正確性・頑健性で最も良い結果を示し、また、浮動小数点計算回数は、状況によっては他の手法の方が高速なときもあったが、概ね最速値を示した。

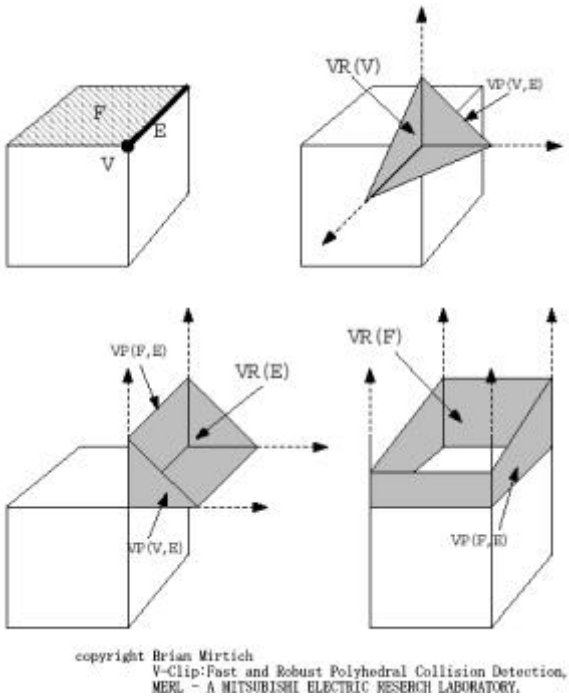


図 A.1: Voronoi Region

原理

凸面体の形状は、フィーチャ（vertice(頂点),edge(稜線),face(面)）により表されていて、フィーチャはVoronoi regionと関係付けられている。ここで、Voronoi regionとは、

定義１:あるフィーチャのVoroni regionとは、そのフィーチャの距離が他のどのフィーチャよりも短い点の集合である。

と定義されているものである。図A.1にその例を示す。

V-Clipは、Voronoi regionに対しての

定理１：２多面体上のフィーチャ対X,Yを考えたとき、x(∈X)とy(∈Y)がX,Y間の最近接点であるとする。もし、xがYのVoronoi regionに含まれ、かつ、yがXのVoronoi regionに含まれたとき、x,yは２多面体間での最近接点であると言える。

を利用している。この定理に従うと、２凸面体上のフィーチャ対X,Yが定理１を満たすとき、X,Yより近接となる点は存在せず、最近接点が求まることになる。このことを活用して、下記の1,2を収束するまで繰り返し、２凸面体上のフィーチャ対X,Yが与えられた時の最近接点を計算する。

1. X,Yが定理１を満足するか検証する。

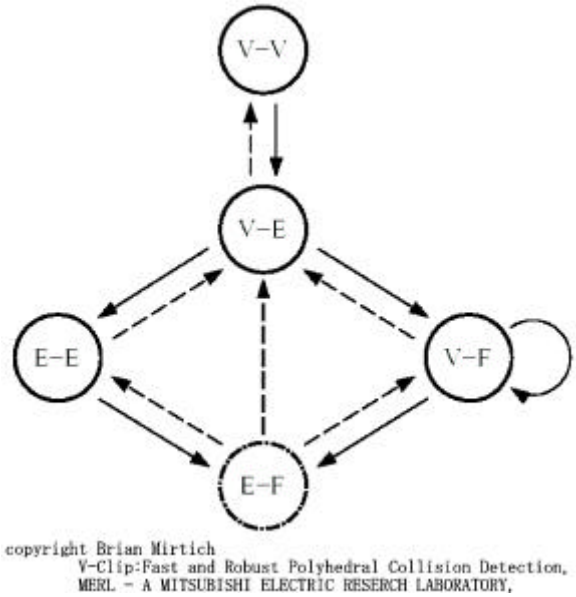


図 A.2: V-Clip の状態遷移図

2. **定理 1 を満足するならば、終了。満足しないときは、X または Y のどちらか一方を隣接する別のフィーチャに遷移させる。その際、フィーチャ間距離が減少、もしくは、距離を保ったまま低次のフィーチャに遷移させる。**

Lin-Canny の手法でも、考え方は同じであるが、実際にフィーチャ間の最近接点を計算してフィーチャ間距離を減少させる方向を決定していた。しかし、V-Clip では、最近接点を計算せず、フィーチャ間の距離関数の導関数を用いてフィーチャ遷移方向を決定している。これが、V-Cliip が頑健である理由である。

図 A.2 の状態遷移図は、最近接点が決定するまでのフィーチャ対の遷移を表す。E,F,V は、各々Edge,Face,Vertex を意味する。それぞれの状態は、可能なフィーチャペアに対応している。例えば、V-F は一方のフィーチャが頂点で、もう一方が面であることを示す。また、実線の矢印は、距離が減少する状態遷移（高次のフィーチャへの遷移）を表し、点線の矢印は、距離が変化しない状態遷移（低次のフィーチャへの遷移）を表す。一般に、E-F 以外の状態でのみ、プロセスは収束する。E-F では、凸面体が干渉しているときにのみ、収束しうる。図 A.2 が示すように、点線の矢印だけではループが形成できなく、また、フィーチャの数は有限であるから、V-Clip は無限ループに陥らず必ず収束することが保証される。

V-Clip では、フィーチャ間の距離関数を用いて定理 1 を満たしているかを検査してい

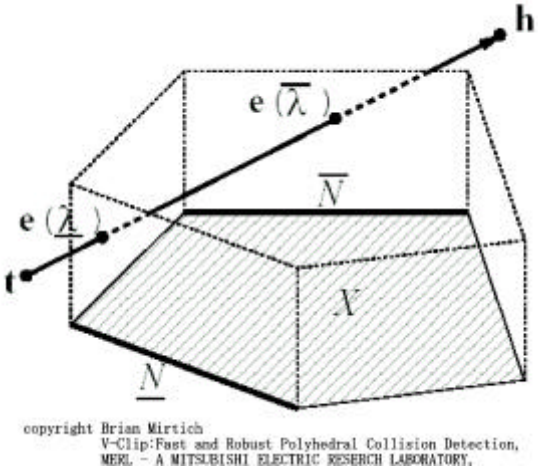


図 A.3: 面 X の Voronoi Region を取り囲む面による Clipping Edge

ることは述べたが、ここではその手法について簡単に述べる。フィーチャ X の Voronoi region,VR(X) を考える。VR(X) は、図 A.3 に示すように面で囲まれており、囲んでいる面の集合を S とする。S によって定義される凸領域を K とする。今、t から h への稜線 E を考えたとき、E∩K は、空であるか、次式で記述される E に沿った線分となる。

$$(1-\lambda)t + \lambda h, \overline{\lambda} \leq \lambda \leq \underline{\lambda}$$

V-Clip では、まず $\underline{\lambda}$、$\overline{\lambda}$ の計算を行い、E を切り取る面に対応した X の隣接フィーチャ $\underline{N}$、$\overline{N}$ を算出する。このとき、E が K を横切るとき、横切らないときで、定理 1 が成立しているかを検証するアルゴリズムが異なり、ここでは、横切る場合のみを説明する。横切らない場合に関しては、参考文献 [1] を参照されたい。

ここで、$e(\lambda)$ と X 間の距離関数 $D_{E,X}(\lambda)$ を定義する。

定義２：稜線 E とパラメータ $e(\lambda)$ を考える。凸面体上のフィーチャ X に関して、距離関数 $D_{E,X}(\lambda)$ を次のように定義する。

$$D_{E,X}(\lambda) = \min_{x \in X} \|x - e(\lambda)\|$$

このとき、$D_{E,X}(\lambda)$ に関して次の定理が成り立つ。

定理２：距離関数 $D_{E,X}(\lambda)$ は、連続的で下に凸である。もし、$e(\lambda_0) \notin X$ ならば、$D_{E,X}(\lambda)$ は、λ_0 で微分可能である。

とりあえず問題となるのは、$D_{E,X}(\lambda)$ が、区間 $[\underline{\lambda}, \overline{\lambda}]$ で最小値をとるかどうかである。定理２から、この問題は $\underline{\lambda}$ および $\overline{\lambda}$ での導関数の符号を調べるだけでよい。区間 $[0, \underline{\lambda})$

で距離関数が最小となるのは、$D'_{E,X}(\underline{\lambda}) > 0$ の場合のみであり、区間 $(\overline{\lambda}, 1]$ で距離関数が最小となるのは、$D'_{E,X}(\overline{\lambda}) < 0$ の場合のみである。E の始点から終点へ向いたベクトルを u とする。$D'_{E,X}(\lambda)$ の符号は、もし、X が頂点ならば、

$$sign[D'_{E,X}(\lambda) = sign[u \cdot (e(\lambda) - v)]$$

であり、もし、X が法線ベクトルを $\hat{n}$ とする面ならば、

$$sign[D'_{E,X}(\lambda)] = \begin{cases} +sign(u \cdot n), D_P[e(\lambda)] > 0 \\ -sign(u \cdot n), D_P[e(\lambda)] < 0 \end{cases}$$

で表せる。結局、λ が定義されている区間の両端でこの符号の検査するだけで、定理１の検査を済ますことが出来る。

参 考 文 献

[1] Brian Mirtich：V-Clip:Fast and Robust Polyhedral Collision Detection, MERL - A MITSUBISHI ELECTRIC RESERCH LABORATORY, TR-97-05 (1999).

[2] Ming C. Lin：Efficient Collision Detection for Animation and Robotics, PhD thesis, University of California, Berkeley, December 1993.

[3] Stephen Cameron：Enhanced GJK: Computing minimum and penetration distance between convex polyhedra, In International Conference on Robotics and Automation. IEEE, 1997.

付録 B Quaternion

3.1.6 項で述べた、Quaternion 同士の演算の内、加法と乗法に関して述べる。

Quaternion は、1 つの実数に 3 つの虚数単位を付け加えたものである。これらの虚数単位は i, j, k で表され、Quaternion q_1 は、

$$q_1 = w_1 + x_1 i + y_1 j + z_1 k \tag{B.1}$$

という形で表現される。ここに、w_1, x_1, y_1, z_1 は実数で Quaternion q_1 の成分を表している。従って、Quaternion 全体の集合 K^4 は $1, i, j, k$ を基底とする 4 次元ベクトル空間になっている。この基底を正規直交基底と見なせば、ベクトル空間 K^4 は 4 次元ユークリッド空間となる。Quaternion q_1 と

$$q_2 = w_2 + x_2 i + y_2 j + z_2 k \tag{B.2}$$

との和 q_{1+2} はベクトルの和として定義され、次の式によって与えられる。

$$q_{1+2} = q_1 + q_2 = w_1 + w_2 + (x_1 + x_2)i + (y_1 + y_2)j + (z_1 + z_2)k \tag{B.3}$$

Quaternion q_1 はその成分 x_1, y_1, z_1 が 0 であれば、実数 w_1 になる。従って、すべての Quaternion の集合 K^4 には、実の Quaternion からなる実数全体 R が含まれている。

実数 w_1 は Quaternion q_1 の実数部分と考えられ、$w_1 = 0$ であるすべての Quaternion q_1 は純虚数と考えられる。それらは空間 K^4 の 3 次元部分空間 I を構成する。部分空間 I と R は互いに直交補空間になっている。このことから Quaternion q_1 は

$$q_1 = w_1 + \hat{q}_1 \tag{B.4}$$

の形に書ける。ここで、$\hat{q}_1 = x_1 i + y_1 j + z_1 k$ である。この時 w_1 は Quaternion q_1 の実数部分、$\hat{q}_1$ はその虚数部分である。同様に Quaternion q_2 を

$$q_2 = w_2 + \hat{q}_2 \tag{B.5}$$

と書く。ここに、$\hat{q}_2 = x_2 i + y_2 j + z_2 k$ である。

Quaternion q_1 に共役な Quaternion $\bar{q_1}$ は、

$$\bar{q_1} = w_1 - \hat{q}_1 \tag{B.6}$$

によって定義される。同様に、

$$\bar{q_2} = w_2 - \hat{q}_2 \tag{B.7}$$

である。

次に、Quaternion の乗法について述べる。掛ける場合には、実数である Quaternion の成分は、虚数単位 i, j, k と交換可能であると考える。するとあとに残るのは、虚数単位同士を掛ける規則を与えることだけである。それらは次のようである。

$$i^2 = j^2 = k^2 = -1, ij = -ji = k, jk = -kj = i, ki = -ik = j \tag{B.8}$$

これらの規則を用いて、まずはじめに純虚の Quaternion の $\hat{q}_1$ と $\hat{q}_2$ を掛けると、次のようになる。

$$\begin{aligned} \hat{q}_1\hat{q}_2 &= -(x_1x_2 + y_1y_2 + z_1z_2) + (y_1z_2 - z_1y_2)i + \\ & \quad (z_1x_2 - x_1z_2)j + (x_1y_2 - y_1x_2)k \end{aligned} \tag{B.9}$$

3 次元ユークリッド空間 I のベクトル $\hat{q}_1$ と $\hat{q}_2$ のスカラー積 $(\hat{q}_1, \hat{q}_2)$ とベクトル積 $[\hat{q}_1, \hat{q}_2]$ は、

$$(\hat{q}_1, \hat{q}_2) = x_1x_2 + y_1y_2 + z_1z_2 \tag{B.10}$$

$$[\hat{q}_1, \hat{q}_2] = (y_1z_2 - z_1y_2)i + (z_1x_2 - x_1z_2)j + (x_1y_2 - y_1x_2)k \tag{B.11}$$

と表されるので、この式は次の形に書き替えることができる。

$$\hat{q}_1\hat{q}_2 = -(\hat{q}_1, \hat{q}_2) + [\hat{q}_1, \hat{q}_2] \tag{B.12}$$

この式を用いれば Quaternion q_1 と q_2 の積は次の形に書くことができる。

$$xy = (w_1 + \hat{q}_1)(w_2 + \hat{q}_2) = w_1w_2 - (\hat{q}_1, \hat{q}_2) + w_1\hat{q}_2 + w_2\hat{q}_1 + [\hat{q}_1, \hat{q}_2] \tag{B.13}$$