

## 第 4 章

# モデルと実世界の視覚インタフェース・マルチメディアディスプレイ

### 4.1 マルチメディアディスプレイの目的

全章までの枠組によって、ロボットとその作業環境および作業の内容を記号的に記述する枠組ができたことになる。言語による記号的な記述は、厳密であり汎用性も高いが、人間にとって常にわかり易い表現であるとは限らない。それは、一つには、ロボットのプログラムには関係のような記号的な情報だけでなく、座標による物の位置・姿勢のように量的情報がふんだんに含まれ、その数値表現が直観的にはわかりづらいことがある。また、記号情報は全体が階層的な構造をもっているとしても、一つ一つは局所的な表現の羅列であり、それらをパターンとして大局的にふかんするには必ず推論が必要になるからである。

さらに、言語によって記述されたモデルはすでに現実の物体とは遊離した存在である。言語がいかに厳密な表現を持っていようと、現実の世界を忠実に反映していないのでは用をなさない。現実の環境物体をモデルに結び付け、その正当性を確認する手段が必要になる。

本研究では、これらの人間、モデル、実環境を結び付ける手段として視覚的なインタフェースを用いることにした。視覚は人間の持つ最大の入力チャネルであり、大量の情報を人間に送り込み、その情報処理能力を活用できること、また、視覚によって状況のパターンの解釈が可能になり、モデルと実際の物体の違いも簡単に提示できることに注目したからである。

視覚的なインタフェースを経由して、人間を中心に現実の環境とモデルを統合する装置としてマルチメディアディスプレイ (MMD) を開発した。MMD は、実際の環境から送られてくる TV カメラ画像の表示、モデルの高速グラフィックス表示、これらの重畳表示と立体視表示を実現する。これらの機能により、環境モデルの作成、モデルを用いたグラフィックスシミュレーション、実作業の監視などの一連のロボットプログラミングを円滑に進めることを目的としている。これらの、MMD のロボットに対する応用は次章に譲り、本章では MMD の機能の詳細、ハードウェア構成、およびホスト上のソフトウェアについて述べる。

## 4.2 ロボットプログラミングのための視覚的インタフェース

### 4.2.1 3次元の視覚インタフェース

ロボットのプログラムが一般的な記号処理プログラムと大きく異なる点は、ロボットが3次元の物理的実世界を対象としていることである。この違いを原点として、マンマシンインタフェースにもさまざまな特徴が派生してくる。

まず、3次元の量的情報を豊富に含むため、テキストで記述されたプログラムの可読性が問題となる。記号的なプログラムは、記号の位相的な関係と構造を記述することが主な目的である。従って、リスト表現等を用いれば関係や構造を人間にとってもわかり易い表現で提示することができる。さらに進んで図式表現を取るとしても、記号プログラムは2次元的なグラフ表示によって効果的な表現が可能であり、そのためのプログラムも簡単に構成することができる。一方、ロボットのプログラムは3次元世界における動作の記述を目的としており、1次元的なテキスト表現、2次元のグラフ表示では不十分である。このことは、プログラムのデバッグを考えてみれば一層はっきりする。記号プログラムは、システムに用意されたシンボリックなデバッガを用いることで快適なプログラミング環境が提供できるのに比べ、ロボットプログラムにはそれらのデバッガは無効であり、時々刻々と変化するロボットの姿態、作業環境の状況を変数に代入されたベクタから思い浮かべることは困難な作業である。

もう一つの、実世界を扱わねばならないという要請から、すべてを決定的な方法で記述することが困難になるという問題が生ずる。記号的プログラムは、計算機の上で開発され、その上で実行される。その動作は外部の事象によって左右される要素が少なく、ほとんどの場合決定的な論理が支配する。これに対しロボットプログラムは外部の物理的実体を操作することが目的である。しかし、物理的実体の振舞いを完全に記述することは不可能であり、予期できないエラーが発生し得る。つまり、ロボットは計算機から外部に開いた世界を扱わなければならない、そのために外界をセンスし監視する機能が必要になる。

以上の考察から、ロボットのためのマンマシンインタフェースには、(1)3次元の物体の形状、動き、関係を高速に表示する機能、(2)実世界の状況を表示し、それらをモデルと比較提示する機能、が重要な役割を演ずることがわかる。

### 4.2.2 マルチメディア表示技術

ディスプレイ技術の進歩に伴い、高解像度ビットマップディスプレイを備えたグラフィックスワークステーションが一般的になりつつある。また、これらの表示装置を有効に活用するためのソフトウェアとして、GKS (Graphical Kernel System)[2], PHIGS (Programmer's Hierarchical Interactive Graphics System)[1]などのグラフィックス標準が制



定され、Xwindow[42]などのウィンドウシステムが普及してきている。したがって、ロボットの視覚インタフェースを実現するための最も手軽な方法としてこれらを組み合わせることが考えられる。しかし、これらの装置とソフトウェアで前節の機能を実現するためには、次のような問題を解決しなければならない。

### 高速3次元グラフィクス

ビットマップディスプレイは、本来テキストあるいは2次元図形の表示を目的としており、ハードウェア的な図形表示プリミティブとしては、単純なBITBLT機能<sup>1</sup>を備えるに過ぎない。3次元の隠面処理、座標変換をホストワークステーションの汎用CPUで処理する場合は、3次元グラフィクスの表示速度は非常に小さい。これらを高速で処理するハードウェアも存在するが、その場合は次の画面構成の自由度が低下し、マルチビューが処理できなくなることがある。3次元描画速度は、ピッキング操作<sup>2</sup>や、ウィンドウの配置が変化した時の再描画の速度にも影響し、低速のデバイスでは対話性に支障を来す。

### マルチウィンドウ管理

3次元の環境をよりよく理解するためには、一つのシーンを表示するだけでは不十分であり、多くのウィンドウに複数の視点からの絵を表示する必要がある。従来のビットマップディスプレイでは、カーネル内のデバイスドライバ、ライブラリ、ウィンドウサーバプロセスなどのソフトウェアが、重なりあったウィンドウを多くの矩形領域に分割し、矩形毎のクリッピングを施すことでマルチウィンドウを管理している。しかしこのような方法は、背面に隠されたウィンドウに対するグラフィクス表示、ウィンドウの配置や優先度が変化した時の処理に時間がかかり、性能が低下する。また、ウィンドウの描画の起動、ウィンドウ間のモデルの共有などはすべてアプリケーションに委ねられ、上位プログラムの負担が増す。

### 拡大・縮小表示

情報を重要度に応じて最も効果的に提示する方法は、表示の寸法を変えることである。多くのウィンドウシステムでは、ウィンドウの寸法を小さくするとその内容の一部を切り出して周辺部の情報を捨て、大きくすると周辺に空白を詰めるような制御を行なう。すなわち、ウィンドウの大きさが変わっても個々の内容の表示サイズは変化せず、情報をズームアップする効果が得られない。これは、濃淡値を持った画像の拡大・縮小の実時間処理が困難なこと、ウィンドウを管理するプロセスが、

<sup>1</sup>bit block transferの略。ビットマップの矩形領域に対してフォントのビットパターンの複写やビット毎の論理演算を施す操作。

<sup>2</sup>カーソルを合わせた点に表示されたモデルのidを取得する操作。

描画の元となった文字、モデルの情報を保持していないので、ビットマップだけに  
対して拡大縮小を繰り返すと次第に情報が失われてしまうことに起因する。

#### TV カメラ画像の入力

通常のワークステーションがTV カメラからの情報を取り込むためには、専用の  
インタフェースを用意する必要がある。そのようなハードウェアが用意できたとし  
て、映像を表示するためには、濃淡画像をウィンドウの大きさに合わせて拡大・縮  
小し、さらにウィンドウクリッピングを施してフレームバッファに複写する操作が  
必要になる。TV カメラのサンプリングは30回/秒の速度で行なわれるのでTV  
カメラの数が複数になった場合、この処理をソフトウェアで行うのは現実的でな  
い。

#### 重畳 (スーパインポーズ) 表示

モデルの世界と現実の環境を対比させるには、グラフィクス画像とTV カメラ映  
像を重ね合わせて表示するのが効果的である。これによって、実環境のモデリング  
を行ったり、モデルの動きと実マニピュレータの動きを比較することでエラーを検  
出したりすることが可能になるからである。この場合、グラフィクスは隠線消去表  
示を行わなければならない。ところが、高速な隠面処理を行なうハードウェアに  
おいて、面の中塗り塗り色 (fill-color) を単純に背景色あるいは透明色に置き換え  
ただけでは、エッジの欠落が生じ、実用にならない。また、TV カメラからの映  
像を更新しつつ、グラフィクスを重ねて表示するには、フレームバッファを2重  
に備え、各々にマルチウィンドウ処理を施す必要がある。このような高度な機能を  
実時間で実現できるハードウェアが開発された例はない。

#### ダブルバッファリング

特別のグラフィクスワークステーションを除いて、ビットマップ用のメモリは一つ  
だけ実装され、描画と表示が同時に進行する。表示の対象が文字だけの場合、ある  
いはグラフィクスの速度が遅い場合、これは効率的な選択であるが、実時間の3次  
元グラフィクスが要求される場合、ダブルバッファリングは必須の機能である。ダ  
ブルバッファがない場合、モデルの動きを実感することはできない。

#### 立体視

両眼立体視は、少ないウィンドウで3次元の奥行きを理解するための優れた手段で  
ある。立体視が利用できない場合、奥行きを知覚するために多くのウィンドウに異  
なった視点からの画像を表示する必要があるが生じ、スクリーンの解像度を有効に活用で  
きないばかりでなく、視線の移動が頻繁に起こり、オペレータを疲れさせる。ま



た、実環境に多くのカメラを設定することも困難であることが多い。立体視機能を効果的に発揮するためには、表示の動きを利用する必要があり、3次元グラフィックスの速度に対する要求がさらに高まる。また、重畳表示される実画像の格納や表示用のフレームバッファの容量が増大し、ワークステーションに実装することは困難である。

#### ソフトウェア環境

従来のウィンドウシステムは、主にテキスト処理を目的として開発されて来っており、グラフィックスソフトウェアは、これらとは別個に標準化が進められてきた。したがって、ウィンドウを用いて3次元モデルを表示しようとする、異なったインタフェースを持つ2組の大きなソフトウェアパッケージを理解する必要が生ずる。さらに、立体視、実画像を扱う機能は現存するグラフィックスおよびウィンドウソフトウェアには含まれておらず、それらに拡張を加えることは極めて困難である。

これらの技術的課題を個々に解決した装置は存在する。しかし、すべてを統合的に実現したシステムは無く、できたとしてもソフトウェアによる実現になるため、マンマシンインタフェースの向上にはつながらない。

### 4.3 マルチメディアディスプレイの構成と実現

前節で述べたように、既存のワークステーションのビットマップディスプレイを用いたのでは、ロボットのための優れた視覚的マンマシンインタフェースを構成することが不可能であることがわかる。マルチメディアディスプレイは、これらのすべての問題に統合的に対処するために(1)高速3次元グラフィックス、(2)立体視表示、(3)マルチウィンドウ、(4)TVカメラ実画像とグラフィックスの重畳表示、などの機能を備えた専用のハードウェアとして実現することとした。表4.1に性能諸元を掲げる。また、図4.1にMMDの全景を示す。

#### 4.3.1 ハードウェア構成の概要

図4.3.1にマルチメディアディスプレイのハードウェアブロック図を掲げる。MMDのハードウェアは、大きく、管理部、画像生成部、重畳制御部の三つに分けられる[72]。

管理部は、ホストワークステーションとのインタフェース、管理プロセッサ、セグメントメモリ、ウィンドウ重畳制御プロセッサなどからなる。ホストとは高速バスで接続される。バス接続によって、MMD内のほとんどのハードウェア資源がホストのメモリ空間にマップされ、セグメントメモリ、画像メモリ、フレームバッファなどが、あたかもワークステーション自身のメモリであるかのようにアクセスできるようになる。管理プロセッサは、バスインタフェースを通じて送られてくるコマンドを解釈して、セグメントメモリ中にモデルの



図 4.1: MMD の全景

構造体を構築する(詳細は4.4節で述べる)。また、画像生成部のプロセッサにコマンドを送り、画像生成の処理を起動する。重畳制御プロセッサは、ウィンドウのオーバラップ関係が変化した時に、更新が必要なウィンドウを高速に計算する。CPU には、12MHz の 68000 を用いている。

画像生成部は合計5本のパイプラインから構成され、各々が担当するメディアを並行的に処理してフレームバッファに画像を生成する。

重畳制御部は、すべてのフレームバッファから同期して画素データを読みだし、メディア毎の優先度をつけてモニタに表示する。同時に立体視メガネの左右のレンズを交互に駆動し、立体視を実現する。

#### 4.3.2 マルチメディア機能

マルチメディアとは、TV カメラ画像、3次元グラフィクス、テキストの3種類の映像メディアを扱えることを意味する。各々の映像は専用のパイプラインで生成される。3種類の映像は、実画像の上にグラフィクス、その上にテキストを重ねて表示される。

##### テキストパイプライン

テキスト系は、セグメントメモリに行毎に構造体として格納された文字コード<sup>3</sup>を読みだし、パイプライン中に備えられたフォントパターンをフレームバッファに展開する。このとき、拡大・縮小の処理と色、点滅などの属性を付加する処理が同時に行なわれる[59]。フォントの種類は多くないが、拡大・縮小が行なわれるので、任意の大きさのウィンドウに、任

<sup>3</sup>文字コードは、ascii, 漢字によらず、すべて16ビットで表現される。

スクリーン 解像度 色 ウィンドウ	水平×垂直=1,280 × 1024 1600 万色中 128 色同時表示 + 点滅表示 最大 60 個
グラフィクス系 セグメントメモリ 座標空間 描画モード 描画速度 TV カメラ実画像系	4MByte 16 ビット固定小数点表現 ワイヤフレーム、隠面消去、隠線消去 10,000 ポリゴン / 秒
カメラ 解像度 階調 画像入力速度 画像表示速度	モノクローム CCD カメラ×4 対 水平×垂直=512 × 384 256 30 フレーム / 秒 20 フレーム / 秒 (ウィンドウ=512 × 384)
テキスト系 フォント 文字属性 表示速度	4 英数字、1 漢字 8 色、反転、点滅 10,000 文字 / 秒

表 4.1: MMD 性能諸元



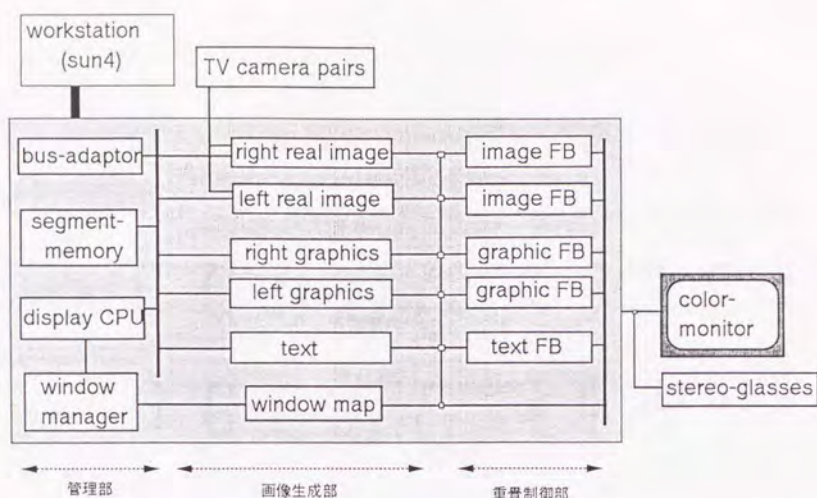


図 4.2: MMD のハードウェア構成

意の数のフォントを配置することが可能になる。拡縮処理は、加重平均法によって行う。これは、新たに発生したドットが 0 か 1 かを決めるために、ドットアドレスとその周囲のフォントパターンとの 4 点との距離によってフォントパターンの輝度を重みづけするものである。二値画像の画素を単純に間引く spc 法に比べると、広い範囲でフォントの品質が保たれる [70]。縮小文字では画素の消失が発生するが、標準のフォントをあらかじめ大きめに定義しておけば、30% 程度の縮小までは問題なく文字が読み取れる。図 4.3 に、拡大・縮小された文字の例を示す。テキストパイプラインは、立体視を行なわないので、ただ一つでよい。処理は、一個のビットスライス型のマイクロプロセッサが制御し、毎秒 10,000 文字の表示速度を得ている。

#### TV カメラ実画像パイプライン

実画像パイプラインの構成を図 4.4 に示す。実画像系は、左右の TV カメラからのアナログ信号をデジタル化し、一旦画像メモリにサンプリングする。その後任意の倍率で拡大・縮小してフレームバッファに展開する [60]。複数の視点からのシーンが表示できるよう、4 台までのカメラ対が接続できる。画像のサンプリングは、一つのカメラチャンネルにつき毎秒 30 回発生する。TV カメラからの信号をデジタル化する回路は 1 組みだけ実装するので、4 対のカメラを同時に監視する場合は、一つのカメラ画像は毎秒 7.5 回サンプリングされる。



G

1.1±1.1 1.2±1.2 1.3±1.3 1.4±1.4 1.5±1.5 0.9±0.9 0.8±0.8 0.7±0.7 0.6±0.6 0.5±0.5

SPC 法

G G G G G G G G G G G

加重平均法

G G G G G G G G G G G

W

1.1±1.1 1.2±1.2 1.3±1.3 1.4±1.4 1.5±1.5 0.9±0.9 0.8±0.8 0.7±0.7 0.6±0.6 0.5±0.5

SPC 法

W W W W W W W W W W W

加重平均法

W W W W W W W W W W W

図 4.3: spc 法、加重平均法によるフォントの拡大縮小

拡大・縮小処理は、目的とする座標における濃淡値を決めるために原画像の近傍の4点の濃淡値を同時に読みだし、新たな座標から各々の近傍点までの距離によって重み付けられた平均を計算する[70]。縮小時には原画像データの読み出しが、拡大時にはフレームバッファへの書き込みが多重に起こるので、メモリアクセスだけで等倍表示の数倍の時間を要する。カラー画像では、データ量がモノクロームの場合の3倍になる。そこで、モノクローム画像だけを扱うこととし、4つの画素を並行して拡大・縮小処理するようにして、必要な速度を得ている。しかし、これではハードウェア量が増大するので、8ビットの入力濃淡値のうち上位6ビットだけを表示に使用している。ブラウン管上では、8ビット濃淡値と6ビット濃淡値の違いはほとんど観測できないのと、計算機による画像処理には8ビットの階調が利用できるので問題にならないと考えたからである。図4.5にウィンドウの大きさに対する表示速度を示す。原画像とはほぼ同じ大きさのとき、約20フレーム/秒の処理速度を得ている。200×150程度の小さいウィンドウを4つのカメラ対に対して表示する場合は、すべてのウィンドウをほぼ実時間で表示することができる。

サンプリングと表示には単発モードと自動モードがある。単発モードでは、ホストからのコマンドによって、サンプリングと表示が個別に起動される。ホストが画像処理を行なう場合、または画像を保存しておきたい場合に使用する。自動モードでは、各々の実画像チャネルについてサンプリングと表示が最適なサイクルで自動的に繰り返される。一般的な監視に用いる。

### グラフィクスパイプライン

グラフィクス系は、セグメントメモリからモデル情報を読み出し、多関節機構の肩から指先に至る階層的なモデリング変換、および視野・投影変換を加え、デプスバッファによる隠面、隠線消去を行なって図形を得る。モデルの構造については4.4.1節で述べる。高速グラフィクスを達成するため、グラフィクスパイプラインは座標変換、クリップ、描画のステージに分けられ、各々は並列動作する。パイプラインの構成を図4.6に示す

グラフィック制御プロセッサ グラフィック制御プロセッサは、管理プロセッサから表示を更新すべきウィンドウの集合を受け取ると、セグメントメモリにアクセスし、各ウィンドウに結合されたモデル構造をトラバースしてシリアル化し、左右のパイプラインに流す。

座標変換 座標変換ステージは、変換マトリックススタックと3個の乗算器からなる。ここで  
の主な作業は、3次元座標にカレントマトリクスを乗じて座標変換処理を施すことである。カレントマトリクスは、ウィンドウビューポート変換、透視変換、視野変換、ネストしたモデリング変換を表すマトリクスを一つに連結したもので、このためのマトリク



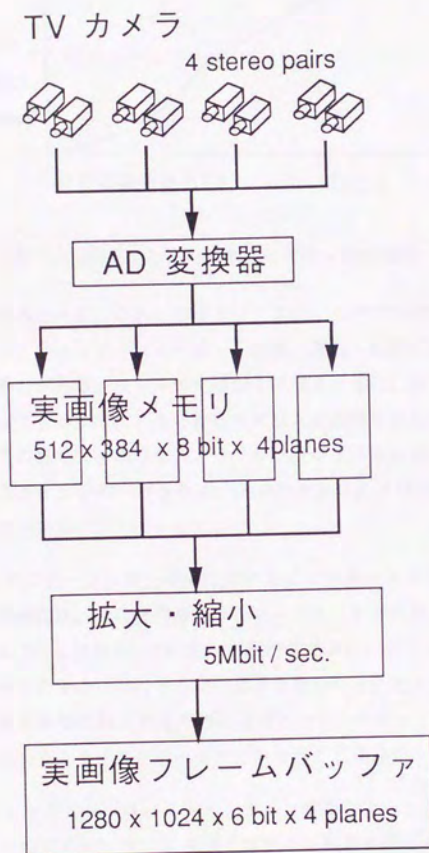


図 4.4: 実画像パイプラインの構成

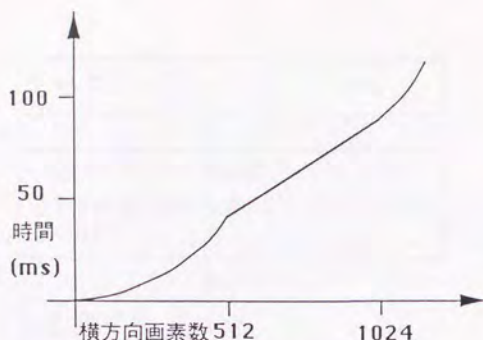


図 4.5: 縦横比 3/4 の実画像の拡大・縮小性能

ス乗算もここで処理される。これらのマトリクスは、モデリング変換のようにモデル毎に変化するものと、ウィンドウビューポート変換、透視・視野変換のようにウィンドウ単位で変化するものがある。モデルの階層が1段深まる毎に、カレントマトリクスをスタックにプッシュすると同時に、あらたなモデリング変換を乗じてカレントマトリクスとする。親モデルの処理に移る時はスタックが1段ポップされる。別のウィンドウの処理に移る時は、スタックがバージされる。一つのマトリクスは64バイトからなり、スタックは256レベルある。

クリッピング クリップステージは線分や面に対するビューポートクリッピングを行なう。

ウィンドウの重畳制御は、次に述べるようなハードウェアで行なっている、一つのビューボリューム (図 4.17 参照) に対する処理だけでよい。クリッピングは3次元で行う。これは、シーンの中から奥行き方向に必要な部分だけを抜きだし、次に述べるデブスバッファの精度を有効に利用するために必要となる。クリップは、x、y、zの各軸に対して専用のビットスライスプロセッサを割り当てて高速化を図っている。

エリアフィル 領域フィルステージは、スキャンライン変換によって隠面、隠線処理を行なう。ここでも三つのプロセッサによるパイプライン処理を行なっている。最初のプロセッサはYソートされたエッジリストを作成する。次のプロセッサは1ラスタ毎のx接片を求める。最後のプロセッサがx接片をソートして描画すべきラスタ片を求める。このとき同時に奥行きクリッピングによる切断面を計算する。モデルの面とデブスクリップ面との交差を計算することで、線分を得る。この切断面を接続することで切断面を合成するが、接続する線分がループを形成することを保証するために、モデルのストラクチャを物体定義の単位としている。



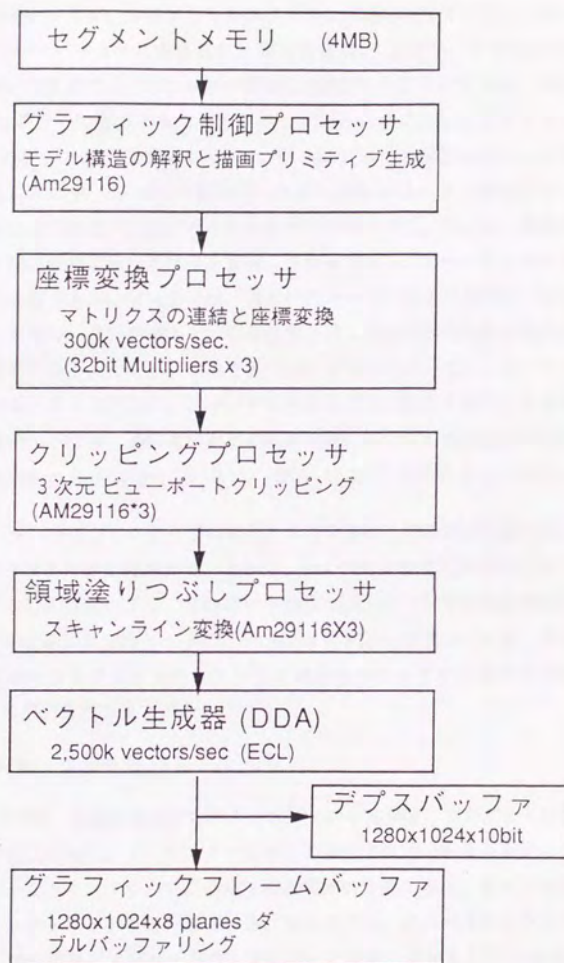


図 4.6: グラフィクスパイプラインの構成

DDA DDA ステージは、ワイヤフレーム描画モードの時は、実際に線分を発生する。また、隠面消去モードでは、エリアフィルステージから送られてくるラスタ片に色情報を付加してフレームバッファに書き込む。隠面消去は、10 ビットの精度を持つデプスバッファを用いて処理する。DDA の性能は描画速度に大きく影響する。MMD では、20 ピクセルを並行して処理することで対処している。各ピクセルは8 ビットの色情報と10 ビットのデプス値で構成されるので、計360 ビットの情報が800nsecで変更されることになる。この20 ピクセルの領域は、水平×垂直=10×2の構成を取っており、水平方向の線分は垂直方向の線分の5倍の速度で描画できる。これは、線分がラスタ方向に限られる隠面処理における性能を重視したためである。スーパーインポーズには隠線消去を行なう必要がある。MMDでは、通常の色コードに加えて透明色（後の実画像を隠さないような色コード）を導入し、隠面処理と同じ方法で面の内部を透明色で塗ることで隠線処理を行なう。ただし、このままではエッジ部において、エッジの方が面の内部より奥にあるように判定され、エッジが消失することが起こり得る。これを避けるため、隠線消去モードでは、通常の隠面消去処理の後、エッジを再描画する方法を取っている。anti-aliasing 処理は行っていない。DDA はECL素子によって構成している。

グラフィックパイプラインのステージは並行に処理を進め、理想的な状態では、約3マイクロ秒毎に一つのラスタ片が出力される。しかし、面の大きさや左右の視差により各ステージでの処理に要する時間は変化する。この違いを吸収し、ステージでの待ち時間が最小になるよう、ステージの連結には128ワードのハードウェアFIFOを用いている。総合的な性能として、約300面からなるロボットを512×512画素のウィンドウに表示する場合、毎秒10～20フレームの描画速度を得ている。

### 4.3.3 マルチウィンドウ機能

マルチウィンドウは、複数の視点からのグラフィクスや実画像、プログラムの情報を同時に表示するために必須である。ワークステーションに装備されているビットマップディスプレイでは重なりあったウィンドウの可視領域を矩形領域の合成と考え、各々の矩形領域によるクリッピングをソフトウェアで処理している。MMDでは、オーバラップウィンドウ制御をウィンドウマッピング表とよぶハードウェアによって処理している（図4.7参照）。まず、画面全体を128×128の小領域に分け、各々の小領域がどのウィンドウに属するかをマッピング表に記憶しておく。各パイプラインの出力をフレームバッファに書き込む段階で、各ピクセル毎にこのマッピング表が参照され、処理の対象になっているウィンドウの番号と、マッピング表に保持されたウィンドウ番号が一致する場合に限り、実際の書き込みが発生する。このようなハードウェアによるサポートによって、一部が隠されたウィンドウでのグラフィクス描画や実画像表示であっても効率が低下することがない。



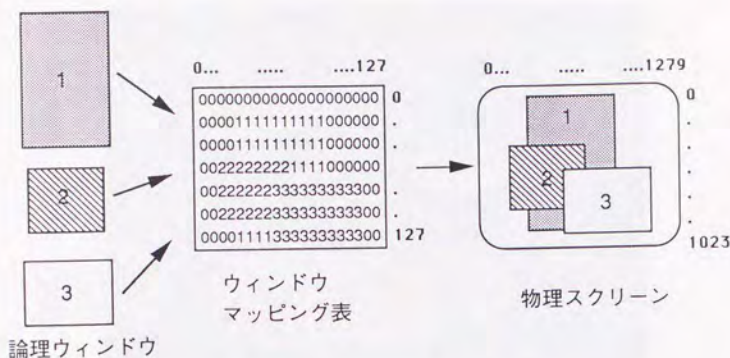


図 4.7: ウィンドウマッピング表を用いたマルチウィンドウ制御

ウィンドウの数は最大 60 であり、スクリーンの全領域を  $128 \times 128$  に量子化しているで、マッピング表の大きさは 16KB あれば十分である。スクリーンのすべての画素について同じ方法を取った場合、マッピング表の大きさは 1MB を越し、更新処理に時間がかかるようになるのに対し、この方法では十分な反応速度が実現できる。更新処理の詳細については 4.4.1 で述べる。

#### 4.3.4 浮動立体文字表示

グラフィック表示されたモデルの名前や座標を示すために、文字列を併用する。通常、これらの文字列は 3 次元空間中での線分の集まりとして表示される。これは、物体の表面に書かれた模様としての文字を描画するには十分である。しかし、仮想的なラベルとして用い、視点を変えたいいくつかのウィンドウに同時に表示する場合は、文字が裏返ったり隠れて見えなくなるなどの不都合を生ずる。このため、MMD では、浮動立体文字と呼ぶ 3 次元文字表示方式を取り入れている。

文字の表示には、文字が書かれる平面を定義し、その上で文字の位置、配置方向を指定する。この文字平面が、空間中で固定されるために前記のような不都合が生ずる。そこで、この文字平面が、ビューイングに応じて視線に垂直になるように動的に生成されるようにすることで、不自然な角度に文字が表示されることを防いでいる。同一の浮動立体文字のモデルを、異なった視点から見た様子を二つのウィンドウに表示した例が、図 4.8 に示されている。文字は、視点からの距離によってスケールリングされ、隠線消去可能な 3 次元の線分として描かれているが、文字平面は常に視点に向かっているため、共に容易に判読できるのがわかる。

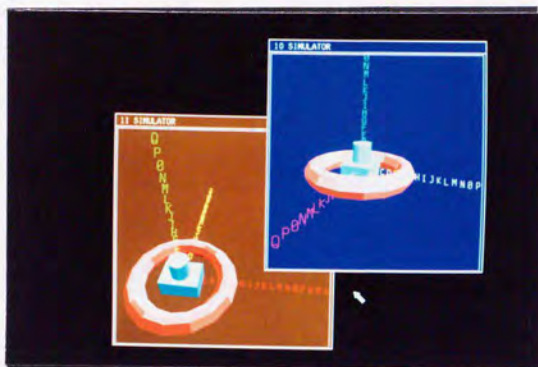


図 4.8: 二つのウィンドウに表示された浮動立体文字

#### 4.3.5 両眼立体視

立体視は実画像とグラフィクスにおいて実現している。TV 画像の立体視の方法は種々考案されているが、実時間の動体観察、広い観察範囲、表示輝度、実現の容易性などの観点から、左右の映像を1フレームずつ交互にブラウン管に表示し、フレーム周期に同期したシャッターレンズを通して観測する方法が最適である。MMD は、フレーム周期をノンインタレースの57Hzとしている。シャッターレンズにはPLZT デバイスを利用している。高電圧による駆動回路が必要となるが、液晶レンズに比べて高いコントラストが得られるという特長がある。グラフィクス、実画像生成のパイプラインを左右に2セット、フレームバッファも左右に2セットずつ備えているので、単眼視モードと立体視モードで、表示速度、解像度が変化することはない。図 4.9 に立体視表示されたモデルを示す。最も手前の物体は、前方クリップ面によって切断面が表示されている。

#### 4.3.6 ホストインタフェースと同期制御

MMD とホストとなるワークステーション (SunSparc/330) とはバスによって接続される。したがって、コマンドの送り出しや画像データの読み取りは効率良く行われる。

コマンドは、MMD のセグメントメモリ中に確保されるリングバッファ、およびハードウェア FIFO を通じて送られる。リングバッファは、32KB の容量を持ち、大量の情報を送るのに適しているが、多くの情報がバッファリングされるので即時性は低い。一方、FIFO は、128 バイトの容量しかないが、管理プロセッサによって優先的に処理され、高い即時性を発揮できる。そこで、リングバッファはモデルの定義や座標変換の更新に、FIFO バッ



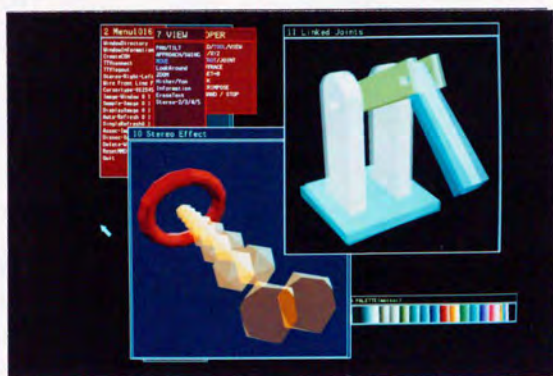


図 4.9: モデルの立体視表示

ファはカーソルの移動やテキスト系のコマンドにと使い分ける。

MMD が返すエラーやピッキングなどの情報は、小さなリングバッファに詰められ、割り込みによってホストに通知される。さらに、コマンドの終了を知り、同期を取るためにコマンドマークの機構を導入している。これは、ホストがあらかじめ id のついたマークコマンドを送っておくと、MMD がそこまでの処理を終了した時にホストに割り込みを送るものである。リング経由と FIFO 経由のコマンドの同期を取ったり、複数のホストプロセスが同期を取るために必要な機構である。

#### 4.4 MMD の動作と制御ファームウェア

この節では、MMD 管理プロセッサに搭載される制御用ファームウェアについて述べる。ファームウェアは次の 5 つのタスクからなるマルチタスク構成になっており、M68000 のアセンブリ言語で記述されている [73]。

##### 1. コマンドバッファ解析タスク

コマンドバッファからのコマンドの入力で起動され、コマンドを解釈・実行する。

##### 2. コマンド FIFO 解析タスク

FIFO バッファからのコマンドの入力で起動され、コマンドを優先的に解釈・実行する。

##### 3. グラフィックス更新タスク

ウィンドウマッピング表、ウィンドウの表示、消去、移動、およびグラフィックパイプラインの処理の起動を行う。

#### 4. テキスト更新タスク

一括処理されたテキスト部の表示、消去、移動のためにテキストパイプラインを起動する。

#### 5. 実画像更新タスク

コマンド、あるいはTVカメラ画像のサンプリング・表示の終了によって起動され、次の実画像のサンプリング・表示処理を起動する。

MMD は、表 4.2、4.3、4.4、4.5、4.6 に掲げるような、計 175 個のコマンド処理を実現している。これは、2 次元の GKS が 110、MMD のグラフィクスに近い機能を持つ PHIGS が 250、代表的なウィンドウシステムである Xwindow の Xlib ライブラリが 300 以上のコマンドを備えているのと比べると、非常にコンパクトなコマンドセットであると言える。その原因には、状態を問い合わせる関数は、MMD の資源にホストが直接アクセスできる機能を利用して、コマンドとしては実現しなかったこと、応用をロボットに限り、テキスト処理、陰影つけなどの機能を省略したことが上げられる。

### 4.4.1 モデルの表現と管理

#### データ構造

管理プロセッサは、ホストからのコマンドに従って、以下のような情報をセグメントメモリ中に構築する。データ構造の関係を図 4.10 に示す [68]。

ウィンドウ 表示する情報をまとめあげる単位であるとともに、表示場所を提供する。可視、強調(点滅)、重畳優先度、寸法、位置、背景色などの属性と、左右の実画像、グラフィクス、テキスト、タイトルなどに対応するレイヤ、ビューイングなどの情報を保持する。

タイトル、テキスト 文字列を格納するための複数の行バッファからなる。

実画像レイヤ TV カメラからの映像を記憶するメモリであり、カメラ識別子、動作状態、可視属性、イメージメモリアドレス、スケジューリング用のキューからなる。

グラフィックレイヤ グラフィクスモデルをグループ化する単位であり、可視、強調、ピッキング、描画法などの属性を、複数のモデル(ストラクチャ)に対して同時に設定することができる。

ストラクチャ セグメントおよび他のストラクチャを複数含んだ構造的な物体を定義する単位である。可視、強調、ピッキング、カラー表、モデリング変換を属性とする。



INITMMD	MMDを初期化する
TESTMMD	MMDのテストプログラムの実行
MMDMODE	表示モード(立体視、右目、左目)の切替え
TXTPAL	テキストパレットへの書き込み
GRAPAL	グラフィクスパレットへの書き込み
IMGPAL	実画像パレットへの書き込み
SCRCRSRSHAPE	スクリーンカーソル形状の定義
CSRTYPE	スクリーンカーソル形状の選択
CSRATTR	スクリーンカーソルの色、点滅を選択
MOVSCRSR	スクリーンカーソルの移動
SCRCRSVB	スクリーンカーソルの可視性選択
TXTCR	テキストカーソルを表示するウィンドウ選択
MAKECURRENT	画像更新処理を起動する
NEWFRAME	ウィンドウ単位で画像更新処理を起動する
BEGINBATCH	一括更新モードに入る
ENDBATCH	一括更新モードを終る
COMMARK	コマンドマークを通知する
RESMARK	応答マークを通知する
ENDSLOT	コマンドブロックの終了
CREWINDOW	ウィンドウを生成する
DELWINDOW	ウィンドウを消去する
WINDOWPORT	ウィンドウの表示位置、寸法を指定する
TITLEPORT	タイトルを表示するウィンドウ中のビューポート
TEXTPORT	テキストを表示するウィンドウ中のビューポート
GRAPORT	グラフィクスを表示するウィンドウ中のビューポート
IMGPORT	実画像を表示するウィンドウ中のビューポート
WINDOWPRIORITY	ウィンドウの重なり優先度
WINDOWCOLOR	ウィンドウの背景色
WINDOWVVB	ウィンドウの可視性の on/off
WINDOWHL	ウィンドウの点滅表示の on/off
ASSOCVIEW	ウィンドウにビューイングを結合する
IMGWINDOW	実画像のウィンドウイング
REPWINDOW	ウィンドウを再定義する
CRELYR	テキスト、実画像、グラフィクスレイヤを生成する
DELLYR	テキスト、実画像、グラフィクスレイヤを消去する
ASSOCLYR	ウィンドウにレイヤを登録する
DISSOCLYR	ウィンドウからレイヤを切り離す
DISSOCALLLYR	ウィンドウからすべてのレイヤを切り離す
LAYERVB	レイヤの可視性を on/off する
LAYERHL	レイヤの点滅表示を on/off する
LAYERDT	レイヤの検出属性を on/off する
GRAMODE	グラフィクス描画モード(ワイヤ、隠線、隠面)
TEXTLYR	テキストレイヤの行数、桁数の指定
TEXTWINDOW	テキストレイヤのウィンドウイング

表 4.2: MMD のコマンド (システム、ウィンドウ、レイヤ)

CRESTR	ストラクチャを生成する
DELSTR	ストラクチャを消去する
CRESEG	静的セグメント定義を開始する
CLSSEG	静的セグメント定義を終了する
DELSEG	静的セグメントを消去する
ASSOCSEG	動的セグメントを生成し、静的セグメントをストラクチャに結合する
DISSOCSEG	ストラクチャからセグメントを切り離す
DISSOCALLSEG	ストラクチャからすべてのセグメントを切り離す
ASSOCSTRSTR	ストラクチャを他のストラクチャに結合する
DISSOCSTRSTR	ストラクチャからストラクチャを切り離す
DISSOCALLSTR	ストラクチャから全ての他のストラクチャを切り離す
ASSOCSTRLYR	ストラクチャをレイヤに登録する
DISSOCSTRLYR	レイヤからストラクチャを切り離す
DISSOCALLSTRLYR	レイヤから全てのストラクチャを切り離す
STRVB	ストラクチャの可視性の on/off
STRHL	ストラクチャの点滅表示の on/off
STRDT	ストラクチャの検出属性の on/off
STRTRANS	ストラクチャの座標系をスケール、並進、回転成分で指定
STRMATRIX	ストラクチャの座標系をマトリクスで指定
STRCOLOR	ストラクチャカラー表に書き込む
INITVIEW2	2次元ビューイングを初期化する
INITVIEW3	3次元ビューイングを初期化する
SELECTVIEW	ビューイングの変更を開始する
CLOSEVIEW	ビューイングの変更を終了する
VIEWUP2	2次元ビューイングの視平面上方向
VIEWMATRIX2	ワールドから2次元視野座標への変換マトリクス
VIEWREFER	3次元の視野基準点を定義する
VIEWNORMAL	視平面の法線を指定する
VIEWDIST	視点から視平面までの距離を指定する
VIEWDEPTH	前クリップ面と後クリップ面間の距離
PROJECTION	投影法と視点を指定する
VIEWWINDOW	視平面上のウィンドウを指定する
VIEWUP3	3次元視野座標系(uvw空間)の上方向(v軸)を指定する
FRONTCLIP	前面クリッピングを on/off する
BACKCLIP	後面クリッピングを on/off する
COORDSTYPE	ワールド座標系の右手系・左手系を指定する
VIEWMATRIX3	視野変換マトリクスを与える
PROJMATRIX	投影変換マトリクスを与える

表 4.3: MMD のコマンド (グラフィクス構造、ビューイング)



SEGV B	動的セグメントの可視性の on/off
SEGH L	動的セグメントの点滅表示の on/off
SEGDT	動的セグメントの検出属性の on/off
SEGTRANS	動的セグメントの座標系をスケール、並進、回転成分で指定
SEGMATRIX	動的セグメントの座標系をマトリクスで指定
SEGRE TENTION	静的セグメントの保存属性を on/off する
DEFAULTATTR	すべての静的属性をデフォルト値に設定する
STACOLOR	text,line,polyline を描画する色コードの指定
DYNCOLOR	text,line,polyline を描画する色をストラクチャカラー表で指定
STAEDGECOLOR	多角形のエッジの色コードを指定
DYNEDGECOLOR	多角形のエッジの色をストラクチャカラー表で指定
STAFILLCOLOR	多角形の中塗り色を色コードで指定
DYNFILLCOLOR	多角形の中塗り色をストラクチャカラー表で指定
LINESTYLE	実線、破線、一点鎖線等の線種を指定
EDGESTYLE	エッジを面の中塗り色で描画するか、edgecolor で描画するかを指定
PICKID	ピック動作で返される値の指定
CHARSIZE	グラフィック文字を描画する大きさ
CHARUP2	グラフィック文字の傾き
CHARUP3	3次元グラフィック文字の傾き
CHARPATH	グラフィック文字が並ぶ方向
CHARSPACE	グラフィック文字の間隔
CHARPLANE	3次元文字が描かれる平面の法線方向
CHARJUST	グラフィック文字の基準点の定義
CHARPREC	3次元浮動文字の指定
ANKTEXT	文字コード列を指定してグラフィック文字を描画
MOVEA3	カレント位置を3次元で絶対指定
MOVER3	カレント位置を3次元で相対移動
LINEA3	カレント位置から絶対指定された3次元位置まで線分を描画
LINER3	カレント位置から相対指定された3次元位置まで線分を描画
POLYLINEA3	絶対指定された3次元線分列の描画
POLYLINER3	相対指定された3次元線分列の描画
POLYGONA3	頂点を絶対指定された3次元多角形の描画
POLYGONR3	頂点を相対指定された3次元多角形の描画
MOVEA2	カレント位置を2次元で絶対指定
MOVER2	カレント位置を2次元で相対移動
LINEA2	カレント位置から絶対指定された2次元位置まで線分を描画
LINER2	カレント位置から相対指定された2次元位置まで線分を描画
POLYLINEA2	絶対指定された2次元線分列の描画
POLYLINER2	相対指定された2次元線分列の描画
POLYGONA2	頂点を絶対指定された2次元多角形の描画
POLYGONR2	頂点を相対指定された2次元多角形の描画
DEFINEFACE	3次元面の定義を開始する
CLOSEFACE	3次元面の定義を終了する
PROFILE	面の輪郭の頂点列を与える
HOLE	面内の穴の頂点列を与える
DEFINEGDP	GDP の定義を開始する
CLOSEGDP	GDP の定義を終了する
CALLGDP	GDP を呼び出す

WRT	テキスト文字列を表示する
SCA	テキスト文字の色、反転、点滅属性の指定
SSR	テキストレイヤ中のスクロール領域の設定
SCP	テキストカーソルの位置を退避記憶する
RCP	テキストカーソルを退避記憶した場所に戻す
GTA	テキストレイヤの属性を問い合わせる
CTX	テキストをコピーする
DFN	テキストフォントを登録する
CPR	テキストカーソルの位置を問い合わせる
CUB	テキストカーソルを左方向に移動
CUC	テキストカーソルの色指定
CUD	テキストカーソルを下方向に移動
CUF	テキストカーソルを右方向に移動
CUP	テキストカーソルを指定した位置に絶対移動
CUS	テキストカーソルの寸法指定
CUU	テキストカーソルを上方向に移動
DCH	テキストカーソルの文字を削除する
DL	テキストカーソルの行を削除する
EA	スクロール領域内部を消去する
ECH	テキストカーソルの文字を消去する
ED	テキストカーソルの上、下の領域を消去する
EL	テキストカーソルの行を消去する
FNT	フォントを選択する
ICH	文字を挿入する
IND	テキストカーソルを下方に移動させる
IL	行を挿入する
LNМ	復帰、改行コードでの動作を指定する
RI	テキストカーソルを上方に移動させる
RIS	テキストレイヤを初期化する
SCC	文字色を選択する
SCE	文字の強調(点滅、反転)を選択する
SD	スクロールダウン
SU	スクロールアップ
TBC	タブ位置クリア
HTS	タブ位置設定
SPA	スペースコードの属性

表 4.5: MMD のコマンド (テキスト)



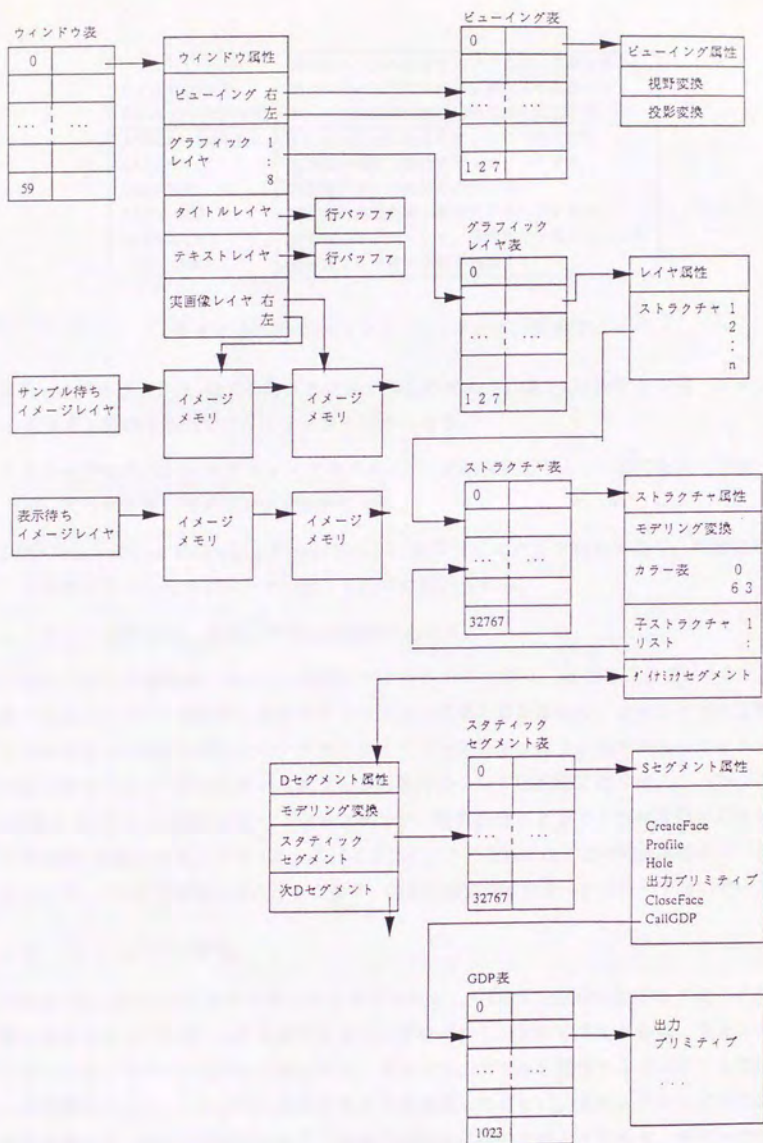


図 4.10: MMD グラフィクスモデルのデータ構造

PICKWINDOW	現在カーソルのあるウィンドウ名、位置を得る
PICKMODEL	グラフィクスモデルのピッキングを起動する
PICKAPERTURE	カーソル先端の感応領域の寸法を指定する
ASSOCCAMERA	TV カメラペアを実画像レイヤに結合する
SAMPIMG	実画像メモリに画像をサンプリングする
DISPIMG	実画像メモリの内容を表示する
AUTOREF	実画像の自動更新・単発更新モードを指定
QIMGSTAT	実画像のサンプリング、表示状況を問い合わせる
COPYIMG	実画像レイヤ間で画像を転送する

表 4.6: MMD のコマンド (ピッキング、実画像)

スタティクセグメント 静的な形状を定義するものであり、表 4.4 に列挙した面、エッジ、穴などを定義する出力プリミティブの列からなる。

ダイナミックセグメント スタティクセグメントに動的属性を与えるものであり、可視、強調、ピッキング、モデリング変換からなる。

GDP(Generalized Drawing Primitive) 一連のプリミティブ列からなり、頻繁に現れる表示パターンをサブルーチン化するために用いられる。

ビューイング 視野変換、透視・平行投影変換からなる。

これらのデータ構造は、決まった場所にディレクトリを持ち、id 番号 (16 ビット整数) で識別される。これらの表現が、従来のグラフィクス標準と異なるのは、ウィンドウの管理が増えている点と、モデルがストラクチャ、ダイナミックセグメント、スタティクセグメントのみつ組みによって表されている点である。前者については次節で述べる。

座標は 16 ビットの固定小数点で定義されるが、視野変換によって 4 つの成分からなる同次座標表現に変換される。グラフィクスパイプライン中の計算では、この同次座標の 4 つめの成分にスケーリングの情報を持たせることで、固定小数点演算のオーバフローを防いでいる。

#### 4.4.2 ウィンドウ管理

MMD は、Xwindow などのウィンドウシステム、CORE,PHIGS などのグラフィクス標準の機能を併せて実現しなければならない。プログラミングする立場からは、ウィンドウとグラフィクスモデルは分離して扱いたい。すなわち、モデルを操作するプログラムでは、それが実際にスクリーンのどこに表示されるかを意識したくない。また、ウィンドウの配置を変える操作は、中に何が表示されているかとは独立に行いたい。すなわち、モデルやウィンドウが変化した時、何を再描画 (ニューフレイム) すればよいかについてユーザのアプリケーションは関知すべきでなく、システムが自動的に管理しなければならない。



PHIGS は、複数のウィンドウに同一のモデルを表示する機能を持たないので、モデルが変化した時の自動更新は簡単な処理で実現できる。一方 Xwindow では、サーバが常時管理している表示情報は、ウィンドウ (window または pixmap) 中のビットマップデータだけであって<sup>4</sup>、グラフィクスモデルの管理はアプリケーションに任せている。このために、サーバからはウィンドウの配置が変わった旨のイベントが頻繁に送られ、それに応じてアプリケーションは大量の描画プリミティブを送出する必要がある。これは、ネットワークの通信量を増すだけでなく、ウィンドウとモデルの分離の原則に反する<sup>5</sup>。

ニューフレーム処理が起動される原因には、(1) モデルや視点の位置・姿勢、またモデルの色、接続関係、可視性など動的属性の変更、(2) ウィンドウの優先度、位置、寸法、可視性の変化、がある。(1) においては、モデルの変更が速やかに画面表示に反映されるように、(2) においては、必要最小限のウィンドウだけのニューフレームが発生するような制御を行う必要がある [71]。

#### モデル変更に伴うニューフレーム

第1番目のニューフレームは、モデルの動的属性、すなわち、ストラクチャおよびダイナミックセグメントのモデリング変換、ストラクチャカラー表、可視、検出、強調、階層の接続関係、が変更されることによって生ずるものである。この変更は、隠面処理における他のモデルとの位置関係を変えることもあるので、そのモデルだけを再描画するのでは不十分であり、ウィンドウ中の全モデルを再描画する必要がある。このためには、あるモデルが表示されている全ウィンドウを知ることができればよい。そのため、各モデル (ストラクチャ) はウィンドウ参照ベクタを保持している。これは 64 ビットのビットテーブルであり、ストラクチャがどのウィンドウに結合されているかを示している。したがって、モデルが動的変更を受けると、グラフィック更新タスクは、そのモデルが登録されているウィンドウに結合されたモデルをトラバースして、再描画処理を起動する。

#### ウィンドウ配置の変更に伴うニューフレーム

2番目のニューフレームは、ウィンドウの属性が変化することで起動される。このような操作には、ウィンドウの削除、可視属性の変化、ウィンドウの優先度の変化、ウィンドウの位置、大きさの変化がある。ところが、あるウィンドウの配置が変化すると、その下に隠れていたウィンドウの一部あるいは全部が可視になるので、そのウィンドウだけでなく、他

<sup>4</sup> レベルの低いサーバではビットマップデータすら記憶されておらず、ウィンドウの配置が変わった際の再表示は完全にユーザのアプリケーションプロセスに委ねられる。

<sup>5</sup> ウィンドウサーバにモデルの情報を持たせ、PHIGS の機能を実現させようとする試みが、PEX (Phigs EXtension) として進行中である。しかし、ウィンドウサーバの大幅な拡張を含むだけでなく、PHIGS の固定的なワークステーションの概念と X のウィンドウの不一致、モデルデータベースの集中管理、再表示の起動、ネットワーク環境などの整合性においてなお多くの問題がある。

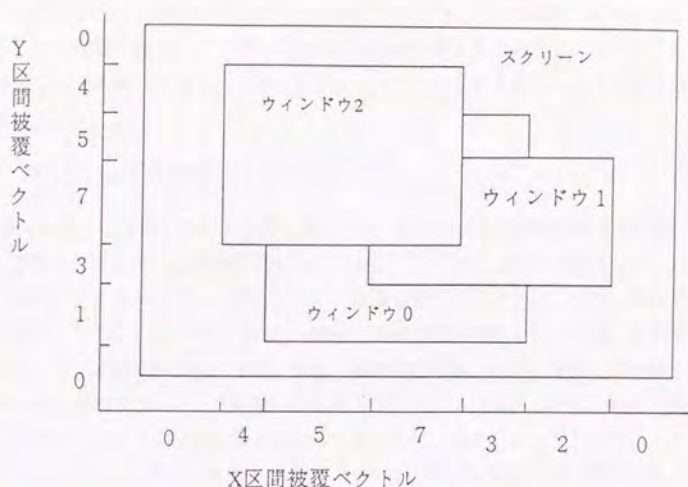


図 4.11: 区間被覆ベクトルの例

のウィンドウの再描画も必要になることがある。ただし、ウィンドウ  $A$  が  $B$  と  $C$  を隠していたからといって、 $B$  と  $C$  を両方ともニューフ্রেームする必要があるとは限らない。 $B$  と  $C$  の隠されていた部分が可視になる場合にだけ、ニューフ্রেームが必要である。また、ウィンドウ操作の前に可視であったウィンドウの集合  $S_1$ 、操作後に可視なウィンドウの集合  $S_2$  から、 $S_2 - S_1$  をニューフ্রেームすればよいというのも誤りである。

ウィンドウの優先度は、全順序をなすように定められる。ウィンドウ  $i$  がウィンドウ  $j$  より優先度が高いことを、 $i > j$  によって表すことにする。ウィンドウ  $i$  は、その表示位置を表す座標パラメータ  $(x_{i1}, y_{i1}, x_{i2}, y_{i2})$  を有する。 $x$  軸、 $y$  軸について、すべてのウィンドウの座標から、区間リスト  $(x_0, \dots, x_m)$  と  $(y_0, \dots, y_n)$  が得られる。各々の区間について、可視であるウィンドウのビットベクトル(区間被覆ベクトル)が決定できる。図 4.11 は、そのような区間被覆ベクトルの例である。

この区間被覆ベクトルをもとに、ある矩形領域  $(X_{\min}, Y_{\min}, X_{\max}, Y_{\max})$  に含まれるウィンドウは、

$$X_p \leq X_{\min} < X_{p+1}, X_{q-1} \leq X_{\max} < X_q$$

$$Y_r \leq Y_{\min} < Y_{r+1}, Y_{s-1} \leq Y_{\max} < Y_s$$

なる関係から定まる区間  $(X_p, X_q)$  および  $(Y_r, Y_s)$  に含まれるウィンドウとして求められる。さらに、このウィンドウの集合の中から、ある区間において最も優先度が高いウィンドウは、ビットベクトル中で 1 になっているウィンドウのうちで最も MSB に近いものである。



これは、ハードウェアによるプライオリティエンコーダによって高速に求めることができる。こうして、配置を変更しようとするウィンドウが被覆する他のウィンドウのうちで可視の区間を含むものをみつけることによってニューフレームを要するウィンドウを求めることができる。

#### 4.4.3 モデル属性の管理

モデルには多くの属性が定義される。属性には、定義時に定めた後は変更できない静的属性と、定義後も変更可能な動的属性がある。プログラミングの自由度を高めるためにはすべての属性を動的にするのが望ましいが、メモリ、描画の効率の点からは、属性は静的である方が都合が良い。GKSでは、可視、検出、強調<sup>6</sup>を動的属性、形状、色、位置・姿勢を静的属性と定めている。PHIGSでは、可視、検出、強調属性の他、位置・姿勢を動的属性とし、形状定義を含む描画プリミティブの動的な変更を可能にしている。しかし、後者の変更は、セグメント中のプリミティブの位置を指定して行う編集作業を必要とするので、アプリケーションプログラムは、属性を定めているプリミティブの位置を管理する作業が必要となる。

MMDは、位置・姿勢および色の動的な変更を可能にし、形状については、ストラクチャおよびセグメント間の接続関係の変更のみを可能にしている。形状の定義自身を動的に変更することはできない。これは、MMDをロボットに应用するに当たっては、CADのようにモデルをオンラインで作成することはまれであり、すでに記述済みの部品を組み合わせで複雑なモデルを構成することを重視したためである。

##### 色の動的指定

通常、色は、セグメントの定義の中で、SetLineColor等のプリミティブを発行することで指定する。これらの色指定プリミティブの引数はカラーパレットのエントリの番号である。したがって、カラーパレット中の色の定義を変更することで、動的な色の変更が可能になるが、カラーパレットはシステム全体で一つしかないので、他のモデルの色も変更してしまう結果となる。モデル間で色コードの干渉がないように、パレットをいくつかの領域に分けて、各々をモデルに割り当てる方法も考えられるが、色コードセットの数はそれほど多くないので、やはりモデルの色を独立に決めるのが難しくなる。

そこで、MMDでは、ストラクチャ毎に仮想的なパレット(ストラクチャカラー表)を持たせている。色指定プリミティブは、このストラクチャカラー表のエントリを指定することで実際のパレット中の色を間接的に選択する。ストラクチャカラー表のエントリを書き換えることで、他のモデルとは独立かつ動的にモデルの色を変更できる。

<sup>6</sup>可視はモデルが表示されるか否か、検出はピッキング操作によって選択されるか否か、強調は、高輝度、点滅などの方法で他と区別して表示するか否かを指定する属性である。

## モデルの階層構造の変更

MMD では、形状定義の要素として、ストラクチャ、ダイナミックセグメント、スタティックセグメント、の三つの構造化の単位を導入している [68]。これは、多くのグラフィクス標準がセグメントだけで構造化するのと対照的である。これは、構造の動的編集が簡単に行えるようにするため、および形状定義を複数のモデルで共有することでメモリの効率的利用を図るための措置である。

スタティックセグメントは、出力プリミティブの列によってモデルの静的な形状を定義する。出力プリミティブで最も重要なのは面の定義である。MMD の CreateFace コマンドは、中に Hole コマンドをネストさせることにより穴のあいた面を簡単に定義できるようにしている。これによって、穴のあいた面を分割によって穴のない面に再定義する手間を省くことができる。また、そのような分割によって生ずる境界線が描画されることがない。通常、スタティックセグメントは、その座標原点がワールドの原点に一致して置かれた状態で定義される。これは、部品要素を他の部品との関係を考慮することなく独立に定義し、形状モデルの再利用を促進するために重要である。

ダイナミックセグメントの基本目的は、スタティックセグメントの座標系を指定することである。ダイナミックセグメントはスタティックセグメントとペアになって、位置と姿勢の定まった形状を定義する。一つのスタティックセグメントに対して多くのダイナミックセグメントを生成することができる。これは、オブジェクト指向におけるクラスとインスタンスの関係としても捉えることができる。ダイナミックセグメントは小量のメモリで表現することができるが、スタティックセグメントのメモリ所要量は、最も単純な直方体で約 100 ワード、円柱では数百ワードになり形状の複雑さに応じて非常に大きなメモリを占有するようになる。たとえば、ネジのような部品は全く同じ形状をしたものが、一つのシーンに多く登場するので、一つ一つに対して同じ形状モデルをロードするのは得策でない。そこで、これらの規格化された部分物体に対しては単一のスタティックセグメントだけを定義しておく。多くのダイナミックセグメントで同一のスタティックセグメントを参照し、異なったモデリング変換を独立に与えることで、限られたセグメントメモリを効率良く利用することができるようになる。

マニピュレータや機械部品は、部品を階層的に組み合わせることで構成される。階層的と言うのは、末端の部品の位置・姿勢が上位レベルの部品に依存し、根本に近い部分の座標の変更が、多くの末端部品の座標に影響を及ぼすことを言っている。セグメントだけでこのような系を表現しようとする、あるダイナミックセグメントのモデリング変換を書き換えると、それに依存する他のセグメントの変換を、アプリケーション側で書き換えてやる必要が生ずる。ストラクチャは、このような階層構造を効率良く表現する単位として導入された。ストラクチャには、そのストラクチャ自身が表現するセグメントの列と、それに従属する他



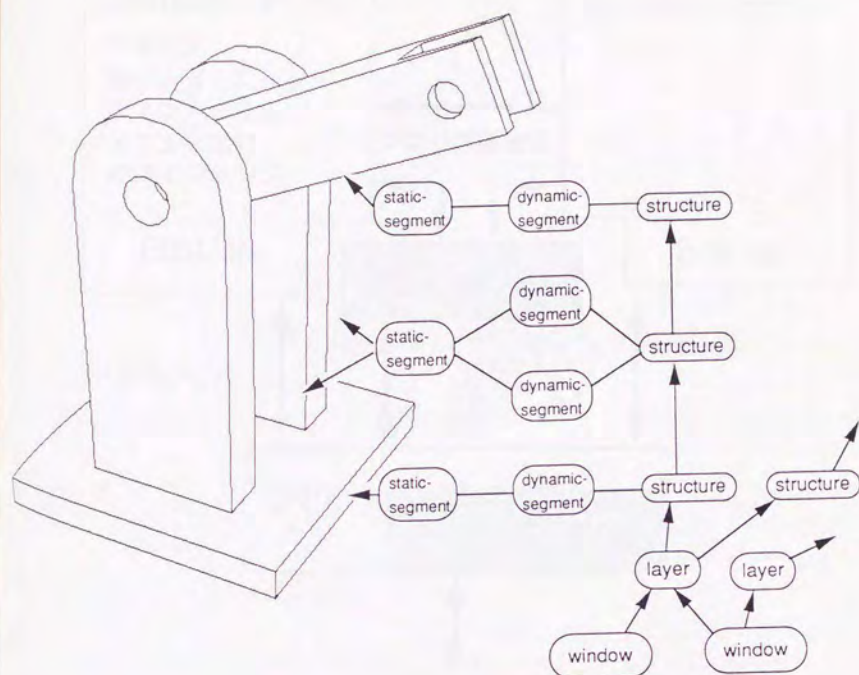


図 4.12: ストラクチャとセグメントによるリンク機構の表現

のストラクチャのリストが接続される。ストラクチャの位置、姿勢は、親ストラクチャに対する相対的な座標変換として与える。根本のストラクチャから末端のストラクチャまでの座標変換を連結し、ワールドでの座標表現を得る処理は、MMD が表示時に自動的に行う。さらに、これらの子構造は、動的に接続、分離できる。したがって、マニピュレータが機械から部品を取り外し、他の機械に取り付けるようなシミュレーションが、容易にプログラムできる。

図 4.12 に、簡単なリンク機構を定義した場合の、ウィンドウ、レイヤ、ストラクチャ、ダイナミックセグメント、スタティックセグメントの関係を示す。中央のストラクチャは、同じ形状をした二つの部品からなる。そこで、ストラクチャに対して二つのダイナミックセグメントを作成し、各々は一つのスタティックセグメントを共有するように定義されている。

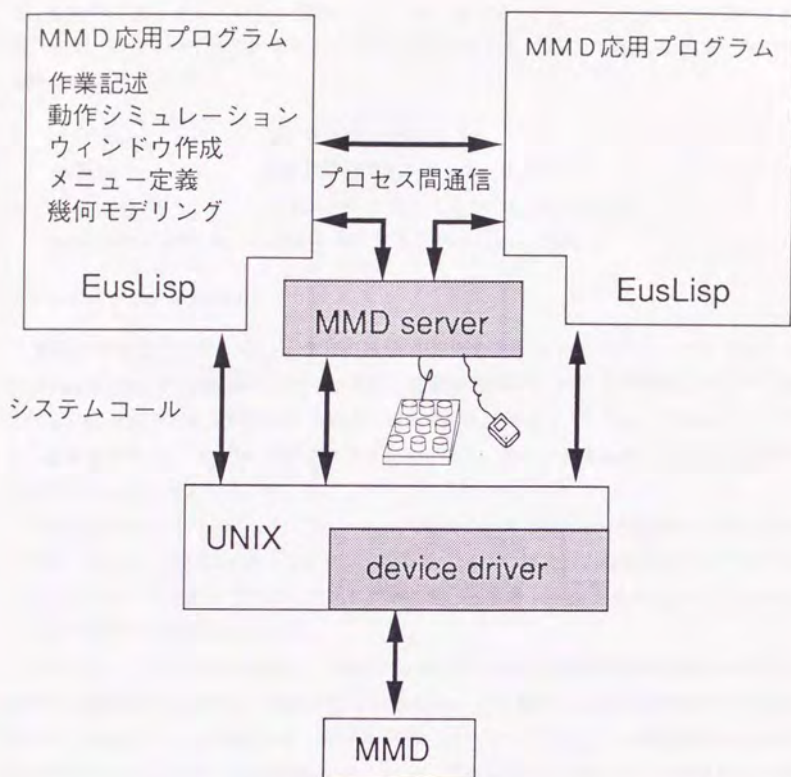


図 4.13: MMD の基本ソフトウェアの構成

## 4.5 MMD の基本ソフトウェア

MMD を利用するためのワークステーション上のソフトウェアの連携を図 4.13 に示す。基本ソフトウェアには、MMD とワークステーションの通信を管理するデバイスドライバと、システムウィンドウと入力デバイスの管理を行う MMD サーバがある [62]。

### 4.5.1 デバイスドライバ

ホストの UNIX カーネルには、MMD と交信するための 2 種類のデバイスドライバが組み込まれている。一つは MMD の機能呼び出すコマンドを発行するコマンドバッファドライバであり、他方は MMD のウィンドウを仮想端末化するためのウィンドウドライバであ



る。これらに関連するデバイス特殊ファイルは `/dev/mmd` ディレクトリの下に置かれている。また、これらのデバイスドライバをCで利用するための以下の include ファイルが用意されている。

<code>mmdcom.h</code>	コマンドコード定義
<code>mmddrv.h</code>	MMD資源のオフセット、応答の書式
<code>mmdioc1.h</code>	コマンドバッファドライバの <code>ioc1</code> 定義
<code>mmdwindowioc1.h</code>	ウィンドウドライバの <code>ioc1</code> 定義

### コマンドバッファドライバ

MMD との通信に用いるバッファの種類に応じて2つのコマンドバッファドライバ、`/dev/mmd/ring` と `/dev/mmd/fifo` がある。前者は MMD のメモリを利用した大きな容量 (32KByte) を持ち、大量のデータの転送に適している。後者はハードウェア的な FIFO であり、容量は小さい (128Byte) が前者より優先度が高く、カーソル移動のような高い即時性が要求されるコマンドの転送に用いる。

MMD にコマンドを送るためには、`/dev/mmd/ring` または `/dev/mmd/fifo` を出力モードでオープンし、配列に用意したコマンド列を `write` すればよい。複数のプロセスが同時にコマンドバッファドライバをオープンしていても、一つの `write` が他の `write` と混ざり合うことがないよう排他制御される。

コマンドバッファドライバには、`ioc1` を用いた表 4.7 に示す制御機能が実現されている。MMD に特徴的な `ioc1` は、同期を取るためのマークの機構と応答を受け取る方法である。マークは、プロセス id とシーケンス番号を表す二つの 16 ビット整数の組からなる。応用プログラムはまず、`MMDMARK ioc1` によってユニークなシーケンス番号を得た後、`commandmark`、`fifomark` コマンドによってそのマーク番号を MMD に送る。MMD は、コマンドバッファ中のコマンドを順次処理し、マークに出会うとホストへの割り込みを発生する。この割り込みによって、ホストはそのマーク以前のコマンドの処理が完了したことを知り、コマンド終了の待ちあわせが実現される。また、エラーが発生した場合、エラーの種類と同時にその時点で最も近いマークを報告することで、エラー原因の同定を容易にしている。`pickwindow`、`pickmodel` 等のホストに応答が返されるコマンドでも、応答マークによって応答を要求しているプロセスを識別している。したがって、MMD からの応答を受け取るには、`read` ではなく、`MMDGETRESPONSE ioc1` を用いる。

### ウィンドウドライバ

ウィンドウドライバは、MMD のウィンドウをテキスト出力用に仮想端末化する。ウィンドウは、`/dev/mmd/window1 ~ window60` をオープンすることで自動的に作成される。

MMDNSERR	MMD にエラーが発生したかどうか調べる
MMDGETERROR	エラーの報告を待つ
MMDMARK	コマンドマーク番号を得る
MMDRESMARK	応答マーク番号を得る
MMDWAITMARK	MMD からマーク終了の応答が到着するのを待つ
MMDGETRESPONSE	MMD からの応答を受け取る
MMDMAXSLOTS	コマンドバッファの大きさを指定する
MMDWINDOWID	ウィンドウ番号を得る
MMDLAYERID	グラフィクスレイヤ番号を得る
MMDSTRUCTUREID	グラフィクスストラクチャ番号を得る
MMDSEGMENTID	グラフィクスセグメント番号を得る
MMDVIEWINGID	グラフィクスビューイング番号を得る

表 4.7: コマンドバッファドライバの IOCTL 機能

ウィンドウは最大 60 個まで開けるが、window1 はシステムのウィンドウディレクトリとして利用することが規定されている。ウィンドウドライバに対する write 操作によってテキスト領域に文字を出力することができる。また、ANSI 標準の vt100 互換のエスケープシーケンスを解釈し、漢字コードの変換を行なう。したがって、UNIX の入出力リダイレクションの機能を使って次のようにファイルの内容やプロセスの状態をウィンドウに表示することができる。

```
% cat afile >/dev/mmd/window10
% ps >/dev/mmd/window11
```

ウィンドウの可視性、寸法、タイトルなどの属性の制御には表 4.8 のような ioctl を用いる。ただし、tty 固有の ioctl、stty 等を処理することはできない。デバイスをクローズしてもウィンドウは削除されない。ioctl を使う必要がある。

#### 4.5.2 MMD サーバ

システムの初期化とマウスを用いたウィンドウ操作、システムメニューの管理のために MMD サーバプロセスが常駐する。アプリケーションは、メッセージキューによるプロセス間通信によって MMD サーバからマウスの情報を取得する。

##### 初期化

MMD サーバは実行の開始に当たって次のセットアップを行なう。MMD サーバが起動すると、図 4.14 に示すウィンドウが現れる。



MWWINDOWVB	ウィンドウ可視性の変更
MWWINDOWHL	ウィンドウの強調表示の変更
MWMOVEWINDOW	ウィンドウの位置の変更
MWSCALEWINDOW	ウィンドウの相対的な拡大・縮小
MWWINDOWPRIORITY	ウィンドウの表示優先度の変更
MWTITLE	ウィンドウタイトル(名前)の変更
MWWINDOWDIRECTORY	ウィンドウディレクトリの更新
MWDELETEWINDOW	ウィンドウの削除
MWSELECTPORT	出力ポートに ring/fifo を指定
MWRESIZEWINDOW	ウィンドウの論理的寸法の指定
MWRESIZETEXT	ウィンドウ表示文字数の変更
MWSELECTCURSOR	テキストカーソルの表示
MWWINDOWCOLOR	ウィンドウの背景、枠の色指定
MWWAITMARK	ウィンドウ操作の終了を待つ
MWSETMSGQID	プロセス通信用のポート番号を登録する
MWGETMSGQID	プロセス通信用のポート番号を取り出す
MWSETTTYPID	仮想 tty 管理プロセスの pid を登録する
MWGETTTYPID	仮想 tty 管理プロセスの pid を登録する

表 4.8: ウィンドウデバイスの ioctl 機能

1. ウィンドウディレクトリを作成・表示する
2. デフォルトのカラーマップを設定する
3. エラー情報監視プロセスをフォークし、エラーウィンドウを開く
4. システムメニューを開く
5. マウス、9 軸ダイヤルの二つの入力デバイスを接続する

#### マウスによるウィンドウ操作

MMD サーバは、マウスの動きに合わせてカーソル移動コマンドを MMD に送り続ける。マウスには右、中、左の 3 つのボタンがあり、右ボタンは MMD サーバがウィンドウ操作のためにローカルに処理する。表 4.9 に掲げるウィンドウ操作が実現されている。

#### システムメニューの機能

システムメニュー (window2) の項目を中、あるいは左ボタンでクリックすることで、描画モードの切替え、実画像の表示、グラフィクスとのスーパーインポーズなど表 4.10 に示す機能が実行される。項目および文字の位置の両方が機能の選択に関与する。



図 4.14: MMD サーバが作成するウィンドウ

移動	ウィンドウ上で右ボタンを押したまま動かし、目的地でボタンを離す
ポップアップ	ウィンドウ上でマウスを動かさずに右ボタンをクリックする
消去	ウィンドウの id 番号の上で右ボタンをクリックする
呼び出し	ディレクトリの中のウィンドウ名をクリックする
拡大・縮小	ウィンドウ上で右、中ボタンを押し、そのままマウスを動かして離す、動かした距離が新しいウィンドウの対角線の長さになる
メニュー表示	右ボタンと左ボタンを同時に押す、メニューとディレクトリが現れる

表 4.9: マウスによるウィンドウ操作

Stereo/Right/Left/M	立体、右眼、左眼表示の選択
Window-Directory	ウィンドウディレクトリのリフレッシュ
Window-Information	ウィンドウ情報の表示
Create-Csh	新しいウィンドウに csh をフォーク
Connect-Tty	キーボードをウィンドウに接続
Wire/Front/Line/Face	ワイヤフレーム、前向き面、隠線、隠面表示切り換え
Cursor01234567	カーソルの形状の選択
ImageWindow 0123	実画像ウィンドウ (0 ~ 3) を作成
SampleImage0123	実画像をサンプリング
DisplayImage0123	実画像を表示
AutoRefresh0123	実画像を自動リフレッシュモードにする
DeleteWindow	ウィンドウを削除
ResetMMD	ユーザのウィンドウをすべて削除
Quit	MMDサーバの実行を終了

表 4.10: MMD サーバのメニュー



## アプリケーションでのマウスの利用法

マウスの中、左ボタンの情報はアプリケーションプログラムで取得することができる。そのためには、アプリケーションと MMD サーバの間に、UNIX のメッセージキューを用いた通信チャネルを設定しておく必要がある。複数のプロセスが MMD 上にそれぞれのウィンドウを開き、制御できるよう、メッセージキューは `MWSETMSGQID` の `ioctl` を用いてウィンドウ毎に設定する。

メッセージキューが設定されたウィンドウ上でマウスのボタンが押されると、ボタンが離されるまで、以下の書式でマウス情報が送られ続ける。ただし、メッセージキューバッファが溢れないよう、アプリケーションがメッセージキューに対して読み取り (`msgrcv`) を実行していない時はマウス情報は送られない<sup>7</sup>。

```
M button x y window-id line col image-x image-y
```

## 4.6 モデル定義と対話型モデル操作システム

3 章で述べた作業対象物やロボットのモデルを MMD に送り、対話的に操作するソフトウェアについて述べる。オペレータとの対話には、コマンドの選択のためのメニューと、動作量の指示のためのマウスおよび多軸ダイヤルを用いる。

### 4.6.1 モデル定義

マニピュレータと環境物体の定義は、基本的に 3.5 節、3.6 節で述べた方法と同一である。ただし、2.4.5 節で述べたように、Xwindow 等を用いた場合と異なる MMD に特有の処理が必要となる。それは、(1) MMD のモデル id を EusLisp のモデルと対応付ける、(2) 形状モデルをセグメントとして、動的属性をストラクチャとして生成する、(3) ストラクチャをレイヤおよびウィンドウに結合する、などの処理である。

これらの MMD に特有の処理は、`mmdsegment`、`mmdstructure` クラスにコーディングされている。これらの機能を参照するような関節、物体のクラスが `mmdjoint`、`mmdparts` として `joint`、`parts` クラスの下位に定義されている。したがって、MMD のためのモデルを定義するためには、

```
(setq *joint-class* mmdjoint)
(setq *part-class* mmdparts)
```

<sup>7</sup>UNIX では、プロセス間通信にはソケットを用いるのが一般的であるが、通信の効率がよいのと、通信バッファに貯えられたメッセージの数を調べ、MMD サーバ側で輻輳の制御を行うためにメッセージキューを用いている。

として\*joint-class\*と\*part-class\*をセットした後、defpart、defjoint、def-manipulatorを行う。形状モデルは、図4.15に示すようなMMDのコマンド列に自動的に変換され、MMD内部にモデル構造が構築される。このとき同時に、モデルの座標系を表す軸が定義される。モデルをウィンドウに表示するには、graphic-layerオブジェクトにモデルを登録し、graphic-layerを今度はウィンドウ(次節参照)に登録する。ウィンドウにはまた、立体視を行うために二つのビューイングのペアを登録する。

```
(defpart cube :shape (make-cube 500 400 300)
              :color 6)

(setq glayer (instance graphic-layer :create))
(send glayer :add-structure cube)

(setq win (instance mmdwindow :open-window :title "example"))
(send win :layer glayer)
(setq svview
  (instance stereo-viewing :init
    mmd-perspective-viewing
    #f(1300 -400 450)           ;viewpoint
    40.0                       ;distance between eyes
    #f(0 0 100)))             ;target
(send win :stereo-viewing svview)
```

#### 4.6.2 ウィンドウ定義

ウィンドウは、クラス mmdwindow に定義されており、このインスタンスを作成することでウィンドウが開かれる。

```
(setq win1 (instance mmdwindow :open-window
                           :size 512 :x 100 :y 100
                           :title "robot"))
```

mmdwindow のサブクラスとして、simulator-window、menu-window がある。simulator-window は、マウスやダイヤルからのイベントを受け取って、モデルとビューイングに変換を与える機能を持つ。マウスの左ボタンはマニピュレータや対象物モデルを動かすために、中ボタンはビューイングの設定に用いる。menu-window は、表示された文字列をマウスでピックアップすることで、あらかじめ定義されたメソッドを起動する。メニューには図4.16に示す動作、編集、視野の3つがある。



```

(CRESTR 3)
(CRESEG 3)
  (EDGESTYLE 1) (DYNWEDGECOLOR 2) (PICKID 1)
  (CREFACE)
    (DYNFILLCOLOR 17)
    (PROFILE #f(250.0 200.0 -150.0) #f(250.0 -200.0 -150.0)
      #f(-250.0 -200.0 -150.0) #f(-250.0 200.0 -150.0))
    (CLOSEFACE)
  (CREFACE)
    (DYNFILLCOLOR 22)
    (PROFILE #f(-250.0 200.0 150.0) #f(-250.0 -200.0 150.0)
      #f(250.0 -200.0 150.0) #f(250.0 200.0 150.0))
    (CLOSEFACE)
  (CREFACE)
    (DYNFILLCOLOR 21)
    (PROFILE #f(250.0 200.0 -150.0) #f(-250.0 200.0 -150.0)
      #f(-250.0 200.0 150.0) #f(250.0 200.0 150.0))
    (CLOSEFACE)
  (CREFACE)
    (DYNFILLCOLOR 20)
    (PROFILE #f(250.0 -200.0 -150.0) #f(250.0 200.0 -150.0)
      #f(250.0 200.0 150.0) #f(250.0 -200.0 150.0))
    (CLOSEFACE)
  (CREFACE)
    (DYNFILLCOLOR 18)
    (PROFILE #f(-250.0 -200.0 -150.0) #f(250.0 -200.0 -150.0)
      #f(250.0 -200.0 150.0) #f(-250.0 -200.0 150.0))
    (CLOSEFACE)
  (CREFACE)
    (DYNFILLCOLOR 19)
    (PROFILE #f(-250.0 200.0 -150.0) #f(-250.0 -200.0 -150.0)
      #f(-250.0 -200.0 150.0) #f(-250.0 200.0 150.0))
    (CLOSEFACE)
  (CLOSESEG)
(ASSOCSEG 3 3 3)
(STRCOLOR 3 16 48) (STRCOLOR 3 17 49) (STRCOLOR 3 18 50) (STRCOLOR 3 19 51)
(STRCOLOR 3 20 52) (STRCOLOR 3 21 53) (STRCOLOR 3 22 54) (STRCOLOR 3 23 55)
(STRCOLOR 3 1 48)
(CRESEG 4)
  (CHARPREC 1) (CHARSIZE 0.6 0.6) (STACOLOR 121)
  (MOVEA3 #f(0.0 0.0 0.0))
  (LINEA3 #f(300.0 0.0 0.0))
  (LWIER3 #f(-15.0 -15.0 0.0))
  (MOVER3 #f(15.0 15.0 0.0))
  (LWIER3 #f(-15.0 15.0 0.0))
  (MOVEA3 #f(315.000 0.0 0.0))
  (ANXTEXT "I")
  (STACOLOR 122)
  (MOVEA3 #f(0.0 0.0 0.0))
  (LINEA3 #f(0.0 300.0 0.0))
  (LWIER3 #f(-15.0 -15.0 0.0))
  (MOVER3 #f(15.0 15.0 0.0))
  (LWIER3 #f(15.0 -15.0 0.0))
  (MOVEA3 #f(0.0 315.0 0.0))
  (ANXTEXT "Y")
  (STACOLOR 123)
  (MOVEA3 #f(0.0 0.0 0.0))
  (LINEA3 #f(0.0 0.0 300.0))
  (LWIER3 #f(15.0 0.0 -15.0))
  (MOVER3 #f(-15.0 0.0 15.0))
  (LWIER3 #f(0.0 15.0 -15.0))
  (MOVEA3 #f(0.0 0.0 315.000))
  (ANXTEXT "Z")
  (CLOSESEG)
(ASSOCSEG 4 3 4)

```

図 4.15: 形状定義の MMD コマンドへの展開

operation menu	edit menu	viewing menu
world/tool/view x/y/z x/y/z pos/rot/joint viewtrace target interference-chk superimpose command/stop	insert-after delete step backstep playback interpolate home	pan/tilt approach/swing move lookaround zoom hither/yon stereo-2/3/4/5
動作メニュー	編集メニュー	視野メニュー

図 4.16: 動作、編集、視点メニュー

### 4.6.3 メニューの機能

#### 動作メニュー

動作メニューは、マニピュレータや対象物の動作を指定するために用いる。物体の操作には6つの自由度を指定する必要がある。しかし、6自由度マスタームによる操作が常に最良であるとは限らない。それは、人間は必ずしも位置・姿勢を同時に指定するのに慣れていないこと、精密な作業には姿勢を一定に保ったり、直線的な動きに制限するなどの拘束を加える方が効率的であるためである[76]。そこで、本システムでは6自由度をマウスの2自由度に射影するか、6軸ダイヤルで各自由度を個別に操作する。拘束をうまく選択すれば動作入力の妨げとはならない。

動作メニューでは、world / tool / viewによって基準となる座標系を選択する。x/y/zによって移動平面または回転軸を、position / rotation / jointによって、位置、回転、関節角動作モードを指定する。

座標系にviewを選択することで指定される視平面・視線モードは、本システムに特有の座標系指定法である。このモードでは、ハンドの位置が視平面に平行な平面上に拘束される。マウスとハンドの動く方向が画面上で一致するので、直観的にわかりやすい操作が可能になる。また、視野座標系は自由に設定できるので、ハンドの位置・姿勢に独立な拘束を与えることができる。さらに、viewtraceモードを選択すると、マニピュレータの動きに合わせて視点が同じだけ移動する。こうして手先のクロースアップ表示中にマニピュレータを大きく動かした場合でも手先を常にウィンドウの中心に捕捉し続けることができる。

targetの上でマウスボタンを押し、変換を与えたいモデルの上でボタンを離すと、そのモデルが対象物体として選択される。interference-checkは、5.3.2節で述べる干渉検査



の対象を、やはりモデルのピックアップによって指定する。

このようにメニューで必要なモードと対象物を選択した後、動かしたいモデルが表示されているウィンドウにカーソルを移動させる。そこでマウスの左ボタンを押しながら動かす、または6軸ダイヤルを回転させることにより、モデルを自由に移動、回転させることができる。

#### 編集メニュー

入力された動作は編集メニューのinsertを選ぶことで履歴が記録される。編集されたプログラムはstep, backstepによって順方向、逆方向にたどることができ、deleteによって削除される。homeは、プログラムの最初の状態に復帰する。playbackによって、全体を通したシミュレーションが実施される。このとき、interpolateを選択しておけば、座標空間で補間した滑らかな動きが表示される。

各ポイントでは、3.6.3節で述べたようにワールドの状況がmanipulator-worldオブジェクトに保存されている。そのため、任意の時点での物体の結合関係、マニピュレータ動作の目的が再現される。

#### 視野メニュー

pan/tilt, close-up/swing, move, lookaround等のメニューは視野座標系を移動、回転させる。zoomは表示倍率を変更する。hither/yonは前方、後方クリップ面までの距離を指定するものであり(図4.17)、クリップ面では切断面表示が得られる。stereoは、両眼立体視の輻輳角を指定する。これらのモードを設定した後、目的のウィンドウの上にカーソルを移動させ、そこでマウスの中ボタンを押しながら動かすことで、ビューイングを変更することができる。

### 4.7 4章の結論

3次元モデルと実世界および人間が視覚的インタフェースを取るための装置であるマルチメディアディスプレイ(MMD)の機能と、ハードウェアの特徴、ソフトウェアの構成について論じた。

MMDは、(1)TVカメラ実画像とグラフィックスの重畳表示、(2)マルチウィンドウ、(3)高速グラフィックス、(4)立体視、を実現する表示デバイスである。

テキスト、TVカメラ実画像、グラフィックスを同時に扱うため、各々を処理するハードウェアをパイプラインとして構成し、並列動作させることで高い描画性能を得ている。テキスト以外は立体視を実現するために、左右2本のパイプラインとなっている。また、マルチウィンドウの管理を高速化するため、画面を小領域に量子化し、ウィンドウマッピングメモリを用いたハードウェアによる制御法を開発した。モデルの表現は、形状の定義と座標系、

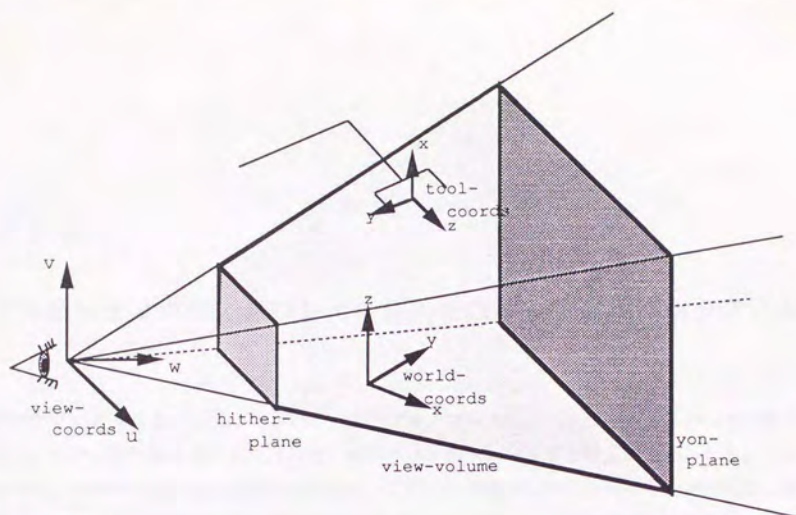


図 4.17: 視野座標系とクリップ面

色などの動的属性を分離し、マニピュレータのように連結された物体が適切に表現できるような階層的構造を取っている。また、複雑な形状定義(セグメント)がメモリ中で効率良く利用できるよう、セグメントを複数の構造体の中で共有できるような表現を実現している。

MMD を利用したプログラムはホストワークステーション上の EusLisp を用いて記述される。3 章で述べたモデルがそのまま MMD に送られ、表示される。メニューとマウス、ダイヤルを用いて、対話的に任意のモデルに任意の姿態を取らせることができる。また、モデルと TV カメラ画像を重ねさせることにより、その差異を際立たせることができる。MMD と EusLisp を組み合わせることにより、3 次元世界で動作するロボットのシミュレーション、動作の編集が容易に行えるようになる。



## 第 5 章

### マルチメディアディスプレイを用いたロボット遠隔作業システム

前章までに述べた EusLisp とマルチメディアディスプレイによって、ロボットのモデルを記述し、モデルを外界および人間操作者との間でインタフェースする枠組が定められた。この章では、バルブの組み立て作業を例に取り、これらの枠組を実際のロボットの遠隔作業に適用し、環境モデル作成、プログラミング、シミュレーション、実行監視の統合化を試みる [66, 27, 65]。

現段階では、ロボットの自律機能は極めて限られており、宇宙、海中、原子炉、災害現場などの危険な環境でロボットに作業をさせる場合は、人間操作者の知恵を借りた遠隔制御に頼らざるを得ない。従来、ロボットの遠隔制御はマスタスレーブ方式によって行われてきた。マスタスレーブ法は、広範囲の作業にも柔軟に対応できるが、人間の誤操作による事故発生の危険性をはらみ、操作に熟練を要し、繰り返し作業では操作者に負担を強いる等の問題がある。

そこで、単純な繰り返し作業はロボットに自動的に処理させ、人間はマクロな動作指令、状況の監視、ロボット単独では対処できない作業の援助を分担しようという知的遠隔制御の概念が提案されている [55, 44, 47, 75, 77, 41]。この概念を実現するためには、まずロボットと作業対象を幾何的に記述する環境モデルを作成する必要がある。次に、モデルに基づいて作業手順を記述し、動作の編集を行うステップが必要である。最後に実作業における動作の監視が要求される。これらのすべてを自律的に行なうことは困難であり、対話的な手法をとらざるを得ない。本章では、4 章までに述べた幾何モデル、作業記述、マルチメディアディスプレイの機能を応用して、統合的に遠隔作業を実施する方法について論ずる。

#### 5.1 環境モデリング

ロボットの動作教示、作業計画、表示等には環境の幾何モデルが不可欠である。幾何モデルは、多面体で近似された形状と、位置・姿勢を表す座標系に分けて表現される。ロボットが作業対象とする物体については、あらかじめ形状モデルを用意しておくことができる。この場合は、実画像に重なるようにモデルの線画を対話的に誘導することで、実物の位置・姿

勢を定められる。

この方法は、現場の監視用のTVカメラを利用できるのでレーザポインタのような特殊な装置を必要としない。また、形状モデルをあらかじめ作成しておけるので、その場で形状を定義する方法に比べて簡便な操作で済み、複雑な物体にも対処することができる。

### 5.1.1 カメラパラメータ

本手法では、物体の座標系はカメラの視点座標系に相対的に定まるので、まずカメラパラメータをキャリブレーションによって求めておく必要がある。

カメラパラメータ(カメラの位置、傾き、焦点距離)は、多くの点(十数個から数十個)の3次元座標と、それに対応する画面上での2次元座標とを計測することで決定することができる。3次元座標は、マニピュレータに参照点を設定し、マニピュレータを可視範囲内で動かしながら関節角を読み取ることで求める。2次元座標はMMDに表示された参照点にオペレータがカーソルを合わせることで求める。こうして得られた座標の組に最小2乗法を適用すると、視野変換、投影変換を合成した $4 \times 4$ マトリクスが得られる。このマトリクスに以下のようなGanapathy[10]の方法を適用すると、カメラの位置、回転、焦点距離、および投影中心パラメータが解析的に分離される。

実験で得られたサンプルデータに最小自乗を適用して得た変換マトリクスを $T$ とする。

$$T = \begin{pmatrix} t_{11} & t_{12} & t_{13} & t_{14} \\ t_{21} & t_{22} & t_{23} & t_{24} \\ t_{31} & t_{32} & t_{33} & t_{34} \end{pmatrix} \quad (5.1)$$

ワールド座標系から視野座標系への視野変換マトリクスを $V$ 、投影変換マトリクスを $P$ で表すと、3次元空間中の一点 $(x, y, z)^t$ から画面上の像 $(u, v)^t$ へは次の式で変換される。

$$\begin{bmatrix} wu \\ wv \\ w \end{bmatrix} = PV \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad (5.2)$$

$$= \begin{bmatrix} k_1 & 0 & u_0 \\ 0 & k_2 & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a & b & c & p \\ d & e & f & q \\ g & h & i & r \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad (5.3)$$

ここで、 $a, b, c, d, e, f, g, h, i$ はマトリクスの回転成分、 $p, q, r$ は視野座標系の原点位置を表す並進成分、 $k_1, k_2$ はカメラレンズの焦点距離と関係する投影のスケール比、 $u_0, v_0$ は投影中心のオフセットである。 $(wu, wv, w)^t$ は同次座標表現であり、 $(u, v)^t$ が投影面上の点位置を与える。視野座標系は、 $xy$ 平面が投影面である $uv$ 平面に平行で、 $z$ 軸が投影面から視



点に向かうような右手系で表す。回転をオイラー角  $(\phi, \theta, \psi)$  で表すと、 $V$  の回転成分  $a$  から  $i$  は次のように表される。

$$a = \cos \phi \cos \psi - \sin \phi \cos \theta \sin \psi$$

$$b = -\cos \phi \sin \psi - \sin \phi \cos \theta \cos \psi$$

$$c = \sin \phi \sin \theta$$

$$d = \sin \phi \cos \psi + \cos \phi \cos \theta \sin \psi$$

$$e = -\sin \phi \sin \psi + \cos \phi \cos \theta \cos \psi$$

$$f = -\cos \phi \sin \theta$$

$$g = \sin \theta \sin \psi$$

$$h = \sin \theta \cos \psi$$

$$i = \cos \theta$$

5.1式と5.3式において、 $T = PV$  とおくことで、マトリクスの各成分に対して次の関係式が得られる。

$$t_{11} = k_1 a + u_0 g \quad (5.4)$$

$$t_{12} = k_1 b + u_0 h \quad (5.5)$$

$$t_{13} = k_1 c + u_0 i \quad (5.6)$$

$$t_{14} = k_1 p + u_0 r \quad (5.7)$$

$$t_{21} = k_2 d + v_0 g \quad (5.8)$$

$$t_{22} = k_2 e + v_0 h \quad (5.9)$$

$$t_{23} = k_2 f + v_0 i \quad (5.10)$$

$$t_{24} = k_2 q + v_0 r \quad (5.11)$$

$$t_{31} = g \quad (5.12)$$

$$t_{32} = h \quad (5.13)$$

$$t_{33} = i \quad (5.14)$$

$$t_{34} = r \quad (5.15)$$

同次座標を用いて表現しているので、マトリクス  $T$  を定数でスケール倍しても投影面に現れる像は変わらない。したがって、5.4式から5.15式は独立な12の方程式ではなく、各々を  $r$  で割ることにより、16の未知数を含んだ11の方程式が得られることになる。残りの方程式には、回転マトリクスが正規直交であることを示す次の式を用いることができる。

$$a^2 + b^2 + c^2 = 1 \quad (5.16)$$

$$d^2 + e^2 + f^2 = 1 \quad (5.17)$$

$$g^2 + h^2 + i^2 = 1 \quad (5.18)$$

$$ae - bd = i \quad (5.19)$$

$$cd - af = h \quad (5.20)$$

$$bf - ec = g \quad (5.21)$$

このように、正規直交の拘束は六つあり、このままでは  $11+6=17$  の方程式が得られて過拘束になる。どれか一つを取り除くことも考えられるが、それではすべての回転成分に対する対称性が失われるので、最後の三つの拘束を次のように書き換える。

$$ad + be + cf = 0 \quad (5.22)$$

$$gd + he + if = 0 \quad (5.23)$$

これで、5.4式から5.14式、5.16式から5.18式、5.22、5.23の計16個の連立一次方程式が得られ、11個の  $T$  の成分から  $a, b, c, d, e, f, g, h, i$  の回転成分、 $p, q, r$  の並進成分、 $k_1, k_2$  のスケール比、 $u_0, v_0$  の投影中心の16個の未知数を定めることができる。さらに  $a, b, c, d, e, f, g, h, i$  からは、オイラー角  $(\phi, \theta, \psi)$  を定めることができる。

Ganapathy の方法は、マニピュレータの座標系とカメラの座標系が直交座標系であることを仮定しているが(5つの拘束条件式)、現実には両者ともわずかに斜行しているために誤差が生ずる。そこで、このマトリクスを初期値として、Lowe の反復法 [24] によって最終的なカメラパラメータを得る。Lowe の方法では、変換マトリクス  $T$  の各成分の微小変位に対する  $u, v$  の変化を  $x, y, z$  に対する偏微分として求めておき、これを用いて Newton-Raphson 法に基づく反復計算を繰り返すことで、斜行座標系を許容する座標変換を得る。これらから、グラフィクス表示のための視野座標変換と透視投影変換を再合成することができる。

両画像をスーパーインポーズすると図5.1のような一致が得られる。カメラには CCD 素子を用いているので原理的に図形歪みはない。また実測により、固定焦点レンズではレンズの歪みによる誤差は1画素未満であることが確認できた。

### 5.1.2 モデルフィッティングの手順

物体の位置・姿勢を決定するに当たっては、6自由度をマウスの2自由度に分解するのに適した座標系を選ぶ必要がある。ティーチングペンダントを用いた R.M.R.C (resolved motion rate control) 型の教示などでは、ワールドやハンド座標系で自由度を分解してい



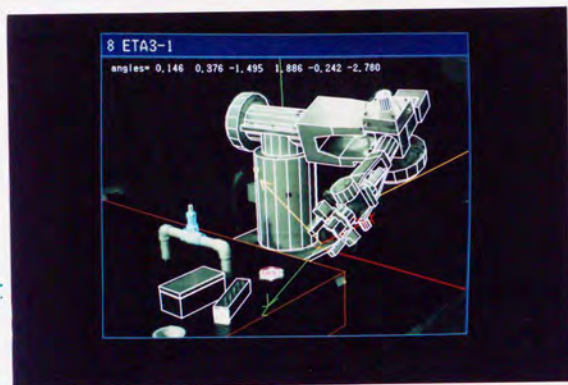


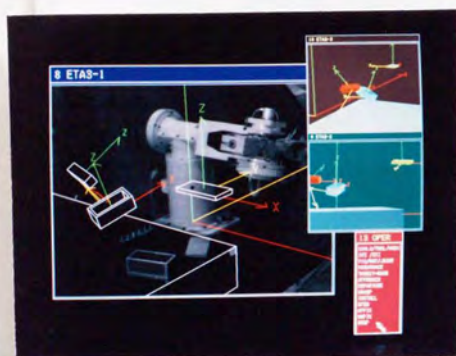
図 5.1: 作業環境とマニピュレータの重畳表示

る。しかしこの方法は、軸に沿った移動が像の表示位置、表示寸法、立体視の視差などの、3次元位置を読み取る手掛りとなるすべての量に同時に影響を与えるので、位置合わせが著しく困難になる。これらの量を独立に操作するためには、視野座標系を基準に取るのがよい。そこで本システムでは、4.6.3節で述べたような、視平面・視線移動モードを設けている。同様に、回転量を位置に干渉させないためには、モデル座標系の各軸を回転軸に取るのが最適である。

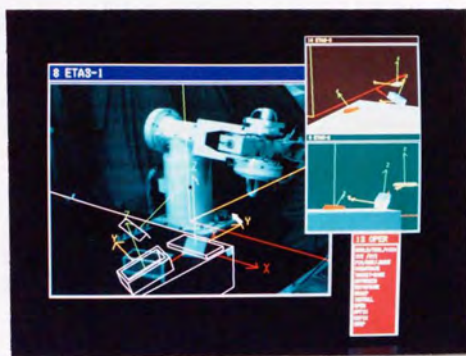
以上の考察により、基本的には次の手順に従って位置合わせを進める。図 5.2(a) は、モデルフィッティングを行う前の、モデルがランダムな位置・姿勢に置かれた初期状態を示している。

1. 2次元的な表示位置だけに着目し、視平面移動モードを用いてモデルの固有座標系原点を物体の原点に合わせる (図 5.2(b))。この段階では姿勢、奥行は問題にしない。
2. 原点位置は変えずに、モデルの  $x, y, z$  軸回りの回転を与え姿勢を合わせる。この操作が完了すると、モデルは物体の相似形として表示される (図 5.2(c))。
3. 視線モードを用い、モデルを視線に沿って前後に移動させて表示寸法を物体に合わせる。このとき立体視を併用する (図 5.2(d))。
4. 必要に応じて (1) に戻って調整を繰り返す。

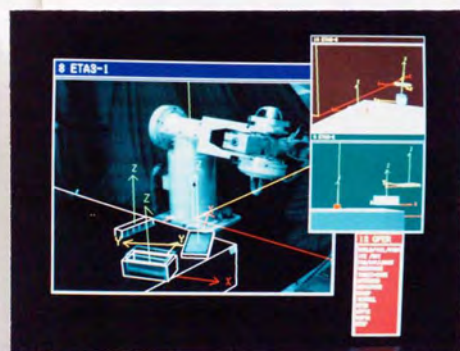
ここで、1 から 3 のステップは六つの自由度をそれぞれ、視野座標系における  $u, v$  座標、モデル座標系での回転、視野座標系での  $w$  座標 (図 4.17 参照) に分解して合わせる操作に相当している。



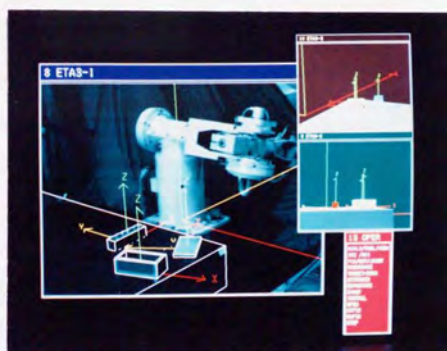
(a) 初期状態



(b) 視平面上での原点合わせ



(c) 姿勢(回転) 合わせ



(d) 視線方向に移動させて寸法を合わせる

図 5.2: モデルフィッティングの手順



	位置 (x,y,z)mm	回転 (オイラー角) deg.
box1 モデル	(389,-627, 136)	(24.0, 1.5, 37.6)
実測	(387,-631, 137)	(24.0, 0.0, 37.8)
box2 モデル	(431,-461, 111)	( 0.6, 0.7,150.7)
実測	(446,-470, 112)	( 0.0, 0.0,152.0)
box3 モデル	(171,-543, 205)	(-0.4, 0.1, 72.6)
実測	(182,-556, 212)	( 0.0, 0.0, 71.0)
valve モデル	(149,-566, 111)	( 0.8,-0.2, 93.2)
実測	(182,-556, 112)	( 0.0, 0.0, 90.5)

表 5.1: モデルフィッティングの精度

ステップ1の操作を簡単に行なうには、物体とモデルの座標原点の対応がとりやすいことが重要である。そこで通常は、座標原点を重心等の物体内部でなく、頂点にとる。さらにステップ2の操作を簡単に行なうため、モデルの形状と共に座標軸を表示する。これら一連の操作は、他の物体との関係がわかっていればさらに簡単になる。たとえば、「物体が既知のテーブルの上に置かれている」という拘束が与えられていれば、2つの位置自由度と1つの回転自由度だけになり、次のような簡単な手続きで位置合わせが完了する。

1. 物体の固有座標系原点をテーブル上に載せる
2. 物体の底面がテーブルに載るように回転を与える、
3. 視点をテーブルの真上に置き、視平面をテーブル面に平行に設定する、
4. 視平面モードで位置を、視線モードで回転を決定する

物体とテーブルとの接触は隠線消去によって容易に判別できる。ステップ2の姿勢合わせは、特徴的な図形の位置や大きさだけでなく、図形と図形の関係に注目する必要があるため、より困難であり誤差を生じやすい。他の物体との相互関係による拘束を積極的に利用することで、回転の自由度を減らす必要があると考えられる。

### 5.1.3 精度評価

図5.1における箱とその蓋、およびバルブについて、その線画モデルを操作し、座標系を決定する実験を試みた。位置合わせは1～数分で完了した。物体の座標系を実測し、モデルの座標系と比較して精度を求めたところ、単純な物体では位置の誤差は10～20mm程度に、回転角は1°～3°の範囲に収まった(表5.1)。

位置・姿勢の精度は、主にカメラキャリブレーション、カメラ画像の解像度、モデルの精密さに依存する。カメラキャリブレーションは正確であると仮定し、カメラと画像メモリの

解像度から決まる精度（分解できる最小の三次元領域）を見積もってみる。

スクリーンの水平方向、垂直方向をそれぞれ  $x, y$  軸、奥行方向を  $z$  軸にとる。カメラの焦点距離を  $f$ 、撮像面の寸法を  $X_{img} \times Y_{img}$  とし、画像メモリの画素数を  $X_{pix} \times Y_{pix}$  で表すと、視点からの距離  $z$  の撮像面に平行な面上では、次式で示される領域が 1 画素に写像される。

$$X_{res} = \frac{z X_{img}}{f X_{pix}} \quad (5.24)$$

$$Y_{res} = \frac{z Y_{img}}{f Y_{pix}} \quad (5.25)$$

$$(5.26)$$

次に、立体視における視差を求める。 $2\theta$  の幅角を持った 2 台のカメラの視線が距離  $L$  で交差する場合、次式のような視差を生ずる。

$$D = \frac{2 \sin \theta ((L-z)((L-z) \cos \theta - L) + x^2 \cos \theta)}{((L-z) \cos \theta - L)^2 - x^2 \sin^2 \theta} \quad (5.27)$$

$\arctan D$  は左右の視角の差である。実際には、視差の絶対量で奥行が知覚されるのではなく、物体が前後に移動することによって視差に偏位が生ずることで距離の違いが認識される。すなわち、物体までの距離  $z$  が偏位することで視差  $D$  が変化し、その変化分が  $x$  方向の 1 画素の視差を生む偏位量が  $Z_{res}$  である。

$$Z_{res} = \frac{X_{img}}{f X_{pix} \frac{\partial D}{\partial z}} \quad (5.28)$$

以上で得られた  $X_{res}, Y_{res}, Z_{res}$  から、 $X_{res} \times Y_{res} \times Z_{res}$  の体積が両眼立体視での 1 画素に写像されると言える。実験では、 $X_{img} = 6.6\text{mm}$ 、 $Y_{img} = 5.0\text{mm}$ 、 $X_{pix} = 512$ 、 $Y_{pix} = 384$  であり、カメラは  $f = 16\text{mm}$ 、 $L = 3.5\text{m}$ 、幅角  $2\theta = 3\text{deg}$  となっている。分解できる立方体の対角線の長さ  $\sqrt{X_{res}^2 + Y_{res}^2 + Z_{res}^2}$  の  $z$  軸上での変化を図 5.3 に示す。対象物のある  $z=3\text{m}$  付近では、水平  $\times$  垂直  $\times$  奥行  $= 25\text{mm} \times 25\text{mm} \times 40\text{mm}$  の領域が 1 画素に写像される。これが点を見つめる場合の両眼立体視の解像度である。

カメラキャリブレーションの誤差、画像のノイズ等が入ることを考えると、表 5.1 に示した実験の結果は、かなり良好な精度を示しているといえる。位置に関しては、両眼視差だけでなく物体の大きさによるマッチングが行なえるので、奥行方向に対して特に誤差が増えることはない。バルブのような複雑な物体は、モデル自体がすでに誤差を含んでいる。自動的なエッジマッチングの適用は困難であるが、本手法によれば操作者が多面体近似の誤差が少ない部分に注目したり、立体視を併用することで、よい精度が得られる。

### 5.1.4 立体視の効果

モデルフィッティングにおける奥行方向の位置合わせでは、物体の大きさの情報と両眼立体視の視差の情報の両方が活用されている。両眼立体視だけではどの程度の奥行の測定能力



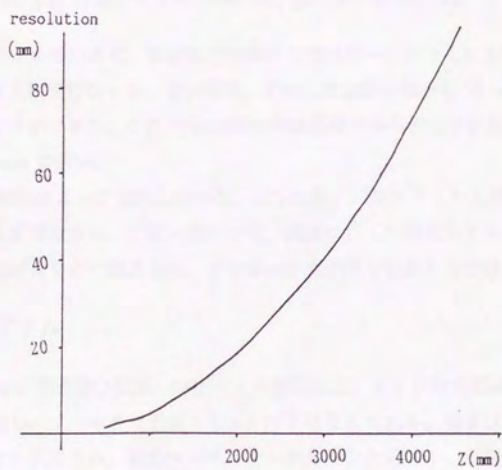
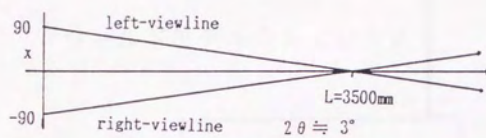


図 5.3: 両眼立体視の解像度

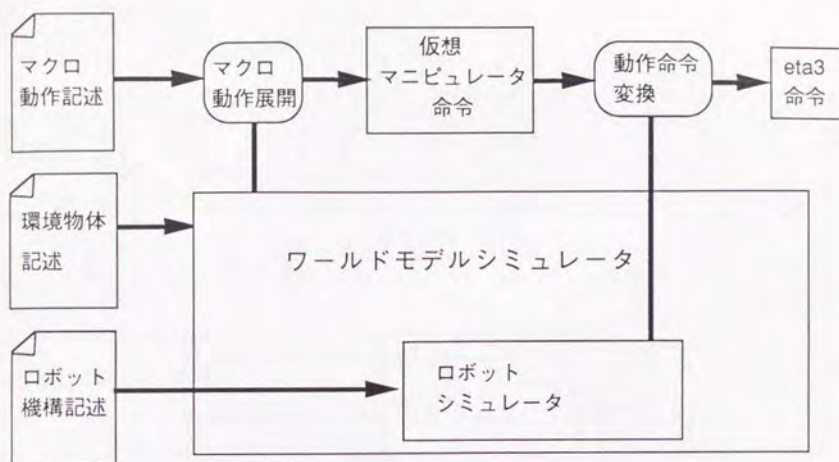


図 5.4: 作業プログラムの構成と動作命令生成の過程

が発揮されるかを確かめるために、小さな直方体を3次元カーソルとして用い、空間中の一点の座標を読み取る実験を行なった。その結果、平均して実測座標から50～100mmの距離の範囲を指し示すことができた。これは5.1.3節の精度見積りから予想される解像領域の奥行き方向の大きさ40mmに近い。

立体視は、運動視差によっても助長される。このため、グラフィックスが高速であることは立体視にとっても重要である。上記の実験では、視線に沿って物体モデルを前後に動かして見ることで距離感がよりよく認識され、よりよい精度が得られることが確認された。

## 5.2 作業プログラム

作業プログラムは、環境物体記述、ロボットの機構記述、マクロ動作記述からなる。前2者はワールドを構成し、ワールドモデルシミュレータに与えられる。後者は、ワールドモデルを参照しつつマクロ展開され、仮想マニピュレータ命令を生成する。仮想マニピュレータ命令は、さらに個別のマニピュレータに対応した変換を受け、実際のマニピュレータが実行可能な動作命令に落される。この過程を図5.4に示す。

作業の開始時点では、ワールドは、図5.5のような状況にある。図中には、プログラムで参照する物体の名称を併記した。以下、バルブ弁の交換作業を例に取り、各々の記述の実際について述べる。



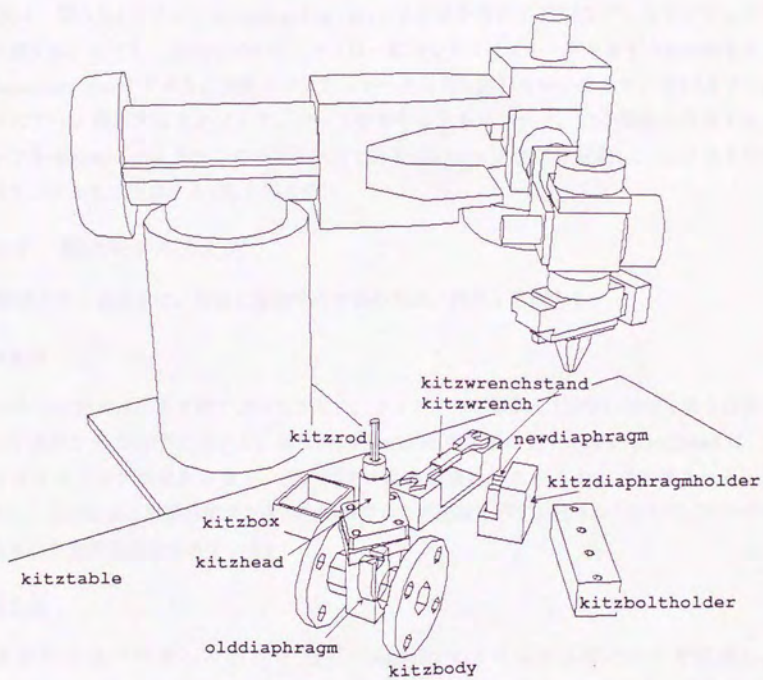


図 5.5: ワールドの初期状態と物体の名称

### 5.2.1 ロボットモデルの記述

作業に用いたロボットマニピュレータは、力制御の可能な6自由度ダイレクトドライブマニピュレータETA3である[45, 79]。ETA3は6つの関節と2本指のグリップからなる。各々の関節とハンド、グリップは、図5.6に示すようなキネマティックスを持つ。

まず、各々の関節の形状を幾何モデリングによって作成し、3.5.1節に示した`defjoint`マクロで、関節の軸、オフセット、可動範囲を定義する。そのプログラムを図5.7に示す。

次に、図5.8に示すように`defmanipulator`マクロを用いてETA3アームオブジェクトを作成する。ただし、`manipulator`クラスは一般的なマニピュレータを表すための抽象クラス(`abstract class`)であり、実際のマニピュレータとは対応しない。そこで、ETA3アーム固有のアーム解を求めるメソッド、パーク姿勢を与えるメソッド、右手姿勢を判別するメソッドを`manipulator`クラスのサブクラスである`eta3arm`クラスに定義し、`eta3`はそのインスタンスとして生成する(図3.16参照)。

### 5.2.2 環境モデルの記述

環境モデル記述では、作業に登場する物体の形状、属性を定義する。

#### 形状定義

パーツの形状は、第3章で述べたように、プリミティブ形状をCSGに基づく集合演算によって合成する方法で定義する。図5.9に`kitzhead`の定義の例を示す。`kitzhead`は、6個のプリミティブ形状からなり、その間で8回の合成操作をすることで定義される。図5.10に、`kitzhead`の形状要素と合成の結果得られる形状を示す。図5.5中のすべての物体がこのような方法で定義されている。

#### 属性定義

上記の方法で定義した形状を元に、`defpart`マクロによってパーツを定義し、`defgrasp`、`definstall`によって把握点、他のパーツへの装着点を付与する。最後に、各パーツに`:move-to`メッセージを送り、5.1節で決定された位置・姿勢に移す。図5.11は、`kitzbody`と`kitzhead`に関する属性、座標系定義を示している。

この結果、図5.12に示すような内容がモデルオブジェクトの内部に構築される。`plist`には、効率上あまり頻繁にはアクセスされない情報が記録されている。`color`はグラフィクスディスプレイに依存して定義される表示色であり、ここではMMDで利用可能な15の色セットの一つが指定されている。`mass`(質量)、`specific-gravity`(比重)、`material`(材質)は、物体の物理属性を表し、`centroid`(重心位置)と共に実際の作業の際に実行系に送られて、サーボパラメータの設定に用いられる。`rot`、`pos`から`evertedp`までは、幾何形状と座



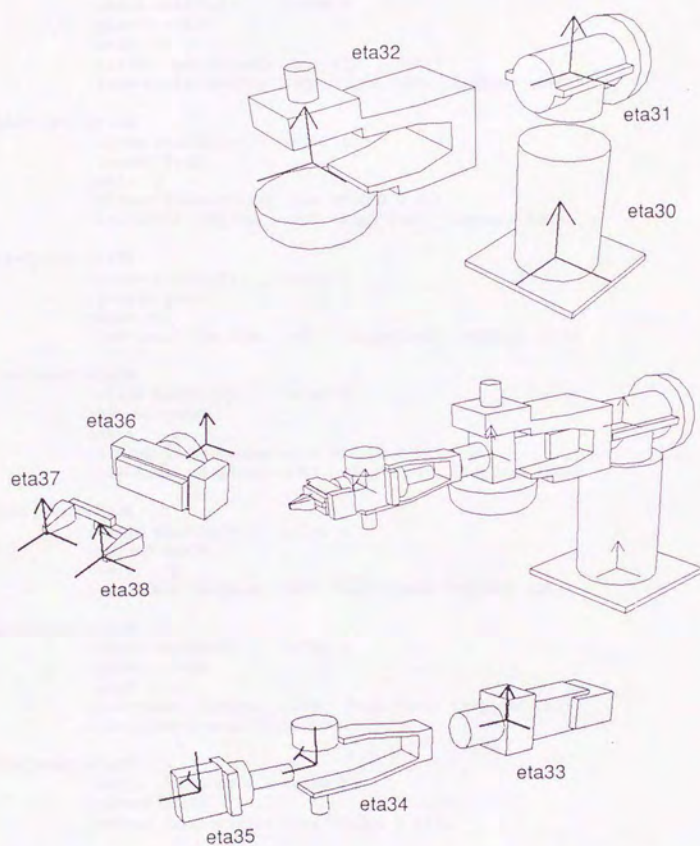


図 5.6: ETA3 アームのキネマティクスと関節の関係

```

(defjoint eta30                ;base has no rotational axis
:shape eta30body :color 3)

(defjoint eta31
:shape eta31body :color 3
:parent eta30
:axis :z
:offset (make-coords :pos #f(0 0 500))
:high-limit (deg2rad 160) :low-limit (deg2rad -160))

(defjoint eta32
:shape eta32body :color 4
:parent eta31
:axis :x
:offset (make-coords :pos #f(500 0 0))
:low-limit (deg2rad -160) :high-limit (deg2rad 160))

(defjoint eta33
:shape eta33body :color 4
:parent eta32
:axis :z
:low-limit (deg2rad -140) :high-limit (deg2rad 140))

(defjoint eta34
:shape eta34body :color 4
:parent eta33
:axis :x
:offset (make-coords :pos #f(400 0 0))
:low-limit (deg2rad -180) :high-limit (deg2rad 180))

(defjoint eta35
:shape eta35body :color 4
:parent eta34
:axis :y
:low-limit (deg2rad -120) :high-limit (deg2rad 120))

(defjoint eta36
:shape eta36body :color 1
:parent eta35
:axis :x
:low-limit (deg2rad -180) :high-limit (deg2rad 180)
:coordinates-axis 200)

(defjoint eta37
:shape eta37body :color 2
:parent eta36
:offset (make-coords :pos #f(200 0 0)))

(defjoint eta38
:shape eta38body :color 2
:parent eta36
:offset (make-coords :pos #f(200 0 0)))

```

図 5.7: ETA3 マニピュレータの関節定義



```
(defmanipulator eta3
  :class eta3arm
  :base eta30
  :joints (list eta31 eta32 eta33 eta34 eta35 eta36)
  :hand eta36
  :left-finger eta38
  :right-finger eta37
  :handcoords (make-coords :axis '(:y :z) :angle (list pi/2 pi))
  :toolcoords (make-coords :pos #f(0 0 200))
  :open-direction #f(0 -1 0)
  :max-span 120 )
```

図 5.8: ETA3 マニピュレータの定義

標系のデータである。grasps には把握姿勢がひとつ定義されている。installs には装着姿勢が定義される。kitzhead は kitzbody に装着されるが、その姿勢は kitzbody 側に記述されるので、kitzhead の installs は nil になっている。mmdagent は、structure-id、segment-id 等の MMD に関する情報を保持し、MMD と通信するオブジェクトである。

### 5.2.3 マクロ動作の記述と解釈

ここで取り上げたバルブ弁の交換作業は、次のようなステップからなる。

1. 弁台をつかんで作業し易い場所に移動する。
2. バルブヘッドをつかんでバルブの本体から取り出す。
3. 古い弁を箱の中に入れる。
4. 弁台に載せられた新しい弁をバルブのロッドに取り付ける。
5. バルブヘッドをバルブ本体に装着する。

この作業は図 5.13 のプログラムで記述され、動作の過程が図 5.16 に描かれている。最初の move-to は、ETA3 アームを初期位置に移動させる仮想マニピュレータ命令である。次の pick-up と place は、弁台を移動させるコマンドで、kitztable に記述された装着位置に移動する。

5 行目の pick-up によってバルブヘッドを持ち上げる。ここで、把握前の接点と把握後の離脱点が、図 5.16 の 6 と 8 に示すように異なっている。6 の把握前の接点とは、kitzhead の grasp-configuration に記述された把握接点であり、8 の離脱点は kitzhead の deproaches に記された姿勢である。また、物体を pick-up すると、デフォルトではマニピュレータの工具座標系がその物体の toolcoords になるが、6 行目で工具座標系を

```

(defun make-kitz-head ()
  (let (cutter base bolt-hole pipe pipe2 pipe-hole)
    ;make shape elements
    (setq base (make-prism (list #f(0.0 44.5 0.0) #f(54.0 0.0 0.0)
                                #f(0.0 -44.5 0.0) #f(-54.0 0.0 0.0))
                  #f(0.0 0.0 15.3)))
    (setq pipe (make-cylinder 13.0 88.1 10))
    (setq pipe2 (make-cylinder 16.5 70.4 12))
    (setq bolt-hole (make-cylinder 5.5 30.0 6))
    (send bolt-hole :translate #f(40.0 0.0 -1.0 ))
    (setq cutter (make-cube 101.0 83.0 40.0))
    (draw base pipe pipe2 pipe-hole bolt-hole cutter)
    ;
    (send pipe :translate #f(0 0 0.1))
    (setq pipe (body+ pipe pipe2))
    (send pipe :translate #f(0 0 0.1))
    ; drill four bolt holes
    (setq base (body- base bolt-hole))
    (send bolt-hole :translate #f(-80.0 0.0 0.0))
    (setq base (body- base bolt-hole))
    (send bolt-hole :translate #f(40.0 31.0 0.0))
    (setq base (body- base bolt-hole))
    (send bolt-hole :translate #f(0.0 -62.0 0.0))
    (setq base (body- base bolt-hole))
    ; cut four corners of the base
    (setq base (body* base cutter))
    (setq kitz-head (body+ base pipe))
    (setq pipe-hole (make-cylinder 5.5 90 6))
    (send pipe-hole :translate #f(0 0 -0.1))
    (setq kitz-head (body- kitz-head pipe-hole))
  ))

```

図 5.9: kitzhead の形状定義



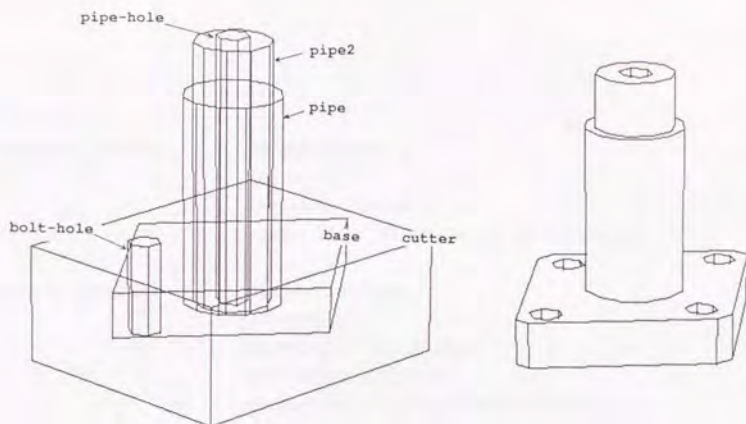


図 5.10: kithead の要素形状と合成結果

olddiaphragm の toolcoords にすることを陽に指定している。これには二つの理由がある。一つは、次の移動は 8、9 行目に示す中間通過点であるが、その位置を olddiaphragm の原点の位置で示したいこと、二つめは、実際の ETA3 マニピュレータは、作業座標系を直接に指定する方法で軌道を与えるが、olddiaphragm の位置誤差を減らすためには、そこに作業座標系を取る必要があるからである。

10、11 行目の place 動作では、olddiaphragm を kitzbox に装着することを指示している。ここでも、図 5.16 の 9 と 11 に現れるように、接近点と離脱点は異なった姿勢となっている。箱状の kitzbox に olddiaphragm を挿入する時はその上方から進入し、olddiaphragm を残したまま kitzrod を引き抜くには、横方向へ移動するようになっている。place 動作で、kitzrod に対する olddiaphragm の結合 (affixment) がはずれ、olddiaphragm は kitzbox に結合される。

次の 12、13 行目は pick-up 動作であるが、他の pick-up が指で新たな物体を把握するのに対し、この動作は kitzrod の先に newdiaphragm を装着する動作である。この区別は、(1)すでに指が何かを把握しており、(2)pick-up 命令の: on 節に把握中の物体またはそれに結合された物体が指定されている、ことから生じる。kitzrod には、newdiaphragm を subpart(子部品)として装着する姿勢が定義されている。この装着姿勢は、親部品(kitzrod)を固定しておき、それに相対的に子部品を位置決めするものとして与えられている。しかしこの pick-up 動作は、親となる kitzrod が移動可能であり、固定された子部品である newdiaphragm を拾い上げる操作なので、装着姿勢の逆変換を求め、そこに kitzrod

```

(defpart kitzbody      :shape kitz-body
                       :color 8
                       :material 'steel
                       :properties '(:eta3name valve-body)))

(defpart kitzhead      :shape kitz-head
                       :color 6
                       :departure #f(0 0 100)
                       :material 'aluminium
                       :properties '(:eta3name valve-head)))

(defgrasp kitzhead :name 2
          :pos #f(-8 0 40)
          :angle (list (* pi 0.22) (+ pi 0.01))
          :axis (list :y :z)
          :approach #f(0 0 60))

(definstall kitzhead :name kitzhead-on-kitzbody
                  :on kitzbody
                  :pos #f(0 0 0)
                  :approach #f(0 0 120) )

(send kitzhead :move-to
  (coords :4x4
    #2f((0.871942 -0.489555 -0.007203 -136.967392)
      (0.489601 0.871914 0.00749 -682.898987)
      (0.002614 -0.010058 0.999946 #,(+ 500.0 -241.284714))
      (0. 0. 0. 1.)))
  :world)

```

図 5.11: パーツの属性定義

```

plist=((:color . 6)
      (:mass . 309.524)
      (:specific-gravity . 2.69000)
      (:material . aluminium)
      (:eta3name . valve-head)
      (:name . kitzhead)
      (:volume . 115065.)
      (:centroid . #f(1.16708e-06 2.22501e-07 26.3240)))
rot=#2f((.871942 -.489555 -0.00720300)
        (.489601 .871914 0.00749000)
        (0.00261400 -0.0100580 .999946))
pos=#f(-136.967 -682.899 258.715)
parent=nil
descendants=(#<mmdparts #X38154c kitzrod -136.61 -683.27 208.72 / ...>
            #<grasp-configuration #X42c288 2 -144.23 -686.52 298.69 /...>)
worldcoords=#<coordinates #X3616b4 -136.97 -682.90 258.72 / ...>
manager=#<mmdparts #X361f84 kitzhead -136.97 -682.90 258.72 / ...>
changed=nil
faces=(#<face #X2d28bc> #<face #X2d2784> #<face #X2d19a4> ...)
edges=(#<edge #X2d2880> #<edge #X2d1908> #<edge #X2d18cc> ...)
vertices=(#f(50.5000 2.88426 0.0) #f(3.64045 41.5000 0.0) ...)
model-vertices=(#f(50.5000 2.88426 0.0) #f(3.64045 41.5000 0.0) ...)
box=#<surrounding-box #X3639a4>
convexp=nil
evertedp=nil
grasps=(#<grasp-configuration #X42c288 2 -144.23 -686.52 298.69 / ...>)
installs=nil
deproaches=(#f(0.0 0.0 100.000))
toolcoords=#<coordinates #X361324 0.00 0.00 0.00 / 0.00 -0.00 0.00>
mmdagent=#<mmdstructure #x361eac kitzhead>
do-update=nil

```

図 5.12: kitzhead オブジェクトに記述されたパーツモデル



```

(defplan kitz kitzw                               ..1
  (move-to :park nil *eta3park* :world)          ..2
  (pick-up kitzdiaphragmholder)                  ..3
  (place kitzdiaphragmholder :on kitztable)       ..4
  (pick-up kitzhead                               ..5
    :tool olddiaphragm                           ..6
    :velocity 500.0                               ..7
    :via (list                                    ..8
          (list #f(200 100 80) kitzbody)))        ..9
  (place olddiaphragm :on kitzbox :velocity 400.0 ..10
    :manual-move :departure)                     ..11
  (pick-up newdiaphragm :on kitzrod               ..12
    :velocity 300.0)                             ..13
  (place kitzhead :on kitzbody                    ..14
    :velocity 300.0                              ..15
    :manual-move '(:place :release))             ..16
  (move-to :park nil *eta3park* :world :velocity 1500.0) ..17

```

図 5.13: バルブ交換作業のマクロ記述

を移動させる。olddiaphragm と newdiaphragm は同一の形状を持った部品なので、その接近・離脱点も同一である。すなわち、図 5.16 の 11 と 12、9 と 14 は diaphragm から見て同じ姿勢になっているのがわかる。

14 から 16 行目は、kitzhead を kitzbody に装着する動作を記述している。:manual-move の指定があるので、kitzhead を kitzbody に突き当てる操作と、指を解放する操作の前にマスタスレーブモードに入る。13、14 行目ではゆっくりとした移動速度を指定しており、作業の正確を期するとともに万一の衝突の危険性を減じている。これに対して 17 行目の move-to では大きな移動速度を指定している。

#### 5.2.4 仮想マニピュレータ命令への展開

上記のマクロな動作記述は、3.7.1 節に述べた仮想マニピュレータの命令列に展開される。図 5.13 のマクロ記述は、わずか 8 つのステップであるが、45 の動作プリミティブのリストに変換される。全体は長くなるので示さないが、代表的な例として、10、11 行目の place 動作のマクロ展開の結果を図 5.14 に掲げる。アプローチ点、装着点への move-to 命令、olddiaphragm の kitzbox への affix、制御座標系の変更、離脱点への move-to に展開されていることがわかる。

個々の仮想マニピュレータ命令を実行すると、ワールドモデル中のロボットおよび物体の

位置・姿勢、また物体間の固着関係が変化する。したがって、このマクロ展開関数は、Lispのマクロを用いて簡便に定義することはできない。マクロ命令が仮想マニピュレータ命令の一つ生成するごとに、そのプリミティブがシミュレータによって実行され、ワールドモデルを更新する。次の命令は、新しいワールドモデルに記述された新たな座標系、固着関係を参照して生成される。

このように、マクロ展開はワールドモデルに副作用を及ぼすので、一度展開を行うとワールドモデルは初期状態とは全く異なった状態となり、再度マクロ展開を試みることはできない。これは、展開の途中で、アーム解が存在しない、固着関係が一貫しない、軌道の途中で干渉が発生するなどのエラーが発生したり、新たな動作を付け加えて対話的な編集を施す場合の障害となる。初期状態をロードし直して最初から展開を繰り返す方法では、動作列が長くなった場合の対話性に支障を来す。また、各仮想マニピュレータ命令について、その順効果と逆効果(もしその命令を取り去ったらワールドはどの状態に復帰するか)を記述して方法も考えられるが、順効果のマクロ動作のシミュレーション途中でエラーが発生した場合は取り扱いが難しくなる。

そこで、各々の仮想命令が生成される毎に、ワールドのスナップショットを取るようにしている。ワールドの状態は、3章の図3.23に示したようなmanipulator-worldオブジェクトに保持される。このオブジェクトは、ワールドを復元するのに必要な全論理的情報を含んでいる。したがって、仮想マニピュレータ命令を逆に辿ったり、任意の時点のワールドを再現することが可能になり、次節に述べる対話的な動作編集が容易に行えるようになる。また、Brepをコピーするのではなく、主に座標系だけを保持しているので、たとえ動作が長くなったとしてもメモリ消費はわずかである。

### 5.2.5 実マニピュレータ命令の生成

前節で仮想マニピュレータ命令列は、特定のマニピュレータを想定しない汎用的な表現である。特に、ワールドモデルシミュレータの作業対象物体の座標系管理、固着関係管理に関する部分は、全くマニピュレータ独立である。各々の仮想マニピュレータ命令を実際のマニピュレータ命令に変換する関数を機種毎に用意するだけで、広い範囲のマニピュレータに対応することができる。ここで実験に用いたETA3アームの主な基本命令は以下の通りである(ETA3の基本命令関数は、eta3mvパッケージに入れられている)。

**eta3mv:move** *coords velocity acceleration*

手先を、ワールド座標系で指定された *coords* 座標系に移動する。移動速度は、*velocity*、加速度は *acceleration* で指定する。

**eta3mv:compensate-position**

```

(move-to :approach
  #<install-configuration #X420510 kitzdiaphragm-on-kitzrod ...>
  #<coordinates #X2785f4 0.00 10.00 100.00 / 3.14 -0.00 0.00>
  #<mmdparts #X329a84 kitzbox ... >
  :velocity 400.000 :acceleration 400.000)
(move-to :place
  #<install-configuration #X4212cc diaphragm-in-box ...>
  #<install-configuration #X4212cc diaphragm-in-box ...>
  #<mmdparts #X329a84 kitzbox ... >
  :velocity 400.000 :acceleration 400.000)
(affix
  #<mmdparts #X3901b8 olddiaphragm ... >
  #<mmdparts #X329a84 kitzbox ... >
  )
(move-to :departure
  #<mmdparts #X3901b8 olddiaphragm ... >
  #<coordinates #X2784b0 ... >
  #<mmdparts #X3901b8 olddiaphragm ... >
  :velocity 400.000
  :acceleration 400.000)
(manual-move)
(set-tool
  #<coordinates #X2709e0 -63.53 -0.63 264.25 / -3.13 0.69 -3.14>
  #<coordinates #X2709e0 -63.53 -0.63 264.25 / -3.13 0.69 -3.14>)

```

図 5.14: place olddiaphragm のマクロ展開の結果



サーボに積分動作を入れ、位置誤差がなくなるように制御する。精密な位置合わせに用いる。

**eta3mv:touch** *direction velocity limit kv*

突き当て動作を行う。*direction*の方向に*velocity*の速度で手先を動かし、*limit*の反力を受けたところで停止する。*kv*はサーボパラメータである。

**eta3mv:set\_object** *mass centroid*

把握物体の物理属性*mass*(質量)、*centroid*(重心位置)を設定する。重力補償、ダイナミックス補償に用いる。

**eta3mv:set\_tool** *coords*

ハンドに対して手先(工具)座標系を*coords*に設定する。以後の**eta3mv:move**命令はこの座標系で指定する。

**eta3mv:f-width** *width*

指幅を*width*まで開くあるいは閉じる。

**eta3mv:grasp** *force1* [*force2*]

力を指定して指を閉じる。まず*force1*の力で把握した後、*force2*の力で把持を続ける。通常、*force1*>*force2*となるように指定する。

**eta3mv:assign-current-position-to** *object-name*

実マニピュレータの手先座標系の値を*object-name*の位置として記録する。

**eta3mv>manual-move**

マスタスレーブモードに入る。

図 5.14の仮想命令は、図 5.15のETA3命令に展開される。1行目の**set-object**は、把握物体、この場合はkitzhead、kitzrod、olddiaphragmの三つの集合体の質量、重心位置を指示するものであり、サーボの重力補償に用いられる。各々の物体の質量、重心が幾何モデルから取り出され、それらを合成した値を導出している。4から9行目はvia点への移動の展開であり、単純に**move**命令に変換されている。10から15行目の**move**は**approach**動作に対応している。**approach**は、次の装着動作を正確に行うために精密な位置決めが必要であり、**move**の後に**compensate-position**命令が生成されている。

17から19行目までは、**move-to :place ...**に対応するETA3命令であり、olddiaphragmをkitzboxに装着する操作である。**move**、**compensate-position**が一般的な位置制御指令なのに対し、17行目の**touch**命令は、ETA3マニピュレータに固有の力制御命令である。**move-to**の目的(3.7.1節の*purpose*参照)が**place**であることから突き

当て動作であると判断され、touch 命令が生成されている。手先の  $-z$  方向に 8mm/sec で移動し、200 グラムの反力が返ったところで kitzbox との接触を検出し、移動を停止する。touch によれば、たとえ kitzbox の位置が、実際の物体とモデルとでずれていたとしても、不十分な、あるいは過剰な力で押しつけることなく接触位置まで移動することができる。しかし、力制御指令を利用すると、モデルが期待する位置と実際に接触が発生した位置とに食い違いが生ずる。18 行目の assign-current-position-to 命令は、マニピュレータの現実の手先位置を olddiaphragm のモデルに登録し、モデルの座標を実際の位置に合わせて更新するものである。以後、たとえば olddiaphragm を把握に行く場合は、この更新された座標が参照される。このように、touch と assign-current-position-to を組み合わせることで物体位置の不確実性に対応できるようになる。

### 5.3 シミュレーションと動作編集

できあがった作業手順をただちに実機で動作させるのは危険であるため、グラフィクスによるシミュレーションを行なう [22, 13, 61]。

#### 5.3.1 グラフィクスシミュレーション

シミュレーションの過程を Xwindow に隠線消去表示したものを図 5.16 に示す。また、図 5.17 には、MMD による表示を示す。前者の描画には、Sun3/60 では 2 時間近くを要するが、後者は数秒で終了する。この高速性を利用して、補間動作の表示を行うことも可能である。この程度の複雑さのモデルであれば、MMD は 1 秒間に 10 回の表示更新が可能であるので、加速度台形則に従った軌道補間を行い、約 100msec 毎の状態を表示すると、MMD 上のモデルの動きは実際のマニピュレータの動きに近くなる。

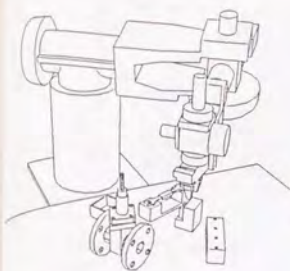
```

(eta3mv:set_object .154762 (float-vector 205.438 .148499 -14.8806)) ..1
(eta3mv:set_tool ..2
  #<coordinates #X2bd334 266.56 0.65 -65.44 / 3.13 0.88 -0.02>) ..3
(eta3mv:move ..4
  (eta3coords ..5
    (send #<coordinates #X279470 ... > ..6
      :in-world ..7
      #<mmdparts #X361f84 kitzhead ... >)) ..8
    300.000 300.000) ..9
  (eta3mv:move ..10
    (eta3coords ..11
      (send #<coordinates #X2785f4 ...> ..12
        :in-world ..13
        #<mmdparts #X329a84 kitzbox ... >)) ..14
      400.000 400.000) ..15
    (eta3mv:compensate-position) ..16
    (eta3mv:touch :z -8.0 2.0 .1) ..17
    (eta3mv:assign-current-position-to ..18
      #<mmdparts #X3901b8 olddiaphragm ... >) ..19
    (eta3mv:move ..20
      (eta3coords ..21
        (send #<coordinates #X2784b0 ... > ..22
          :in-world ..23
          #<mmdparts #X3901b8 olddiaphragm ... >)) ..24
        400.000 400.000) ..25
      (eta3mv>manual-move) ..26
      (eta3mv:set_tool ..27
        #<coordinates #X2b7c70 264.25 0.63 -63.53 / 3.13 0.88 -0.02>) ..29

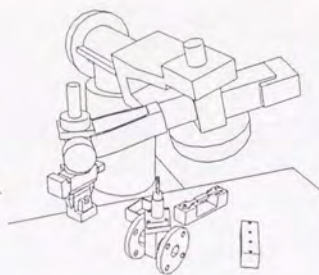
```

図 5.15: ETA3 マニピュレータの動作命令への変換

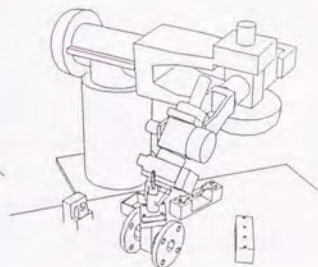




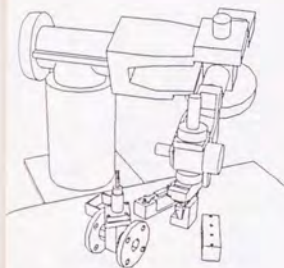
1. diaphragmholder へアプローチ



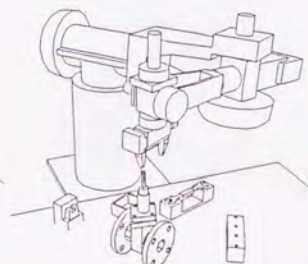
4. diaphragmholder を作業場所へ置く



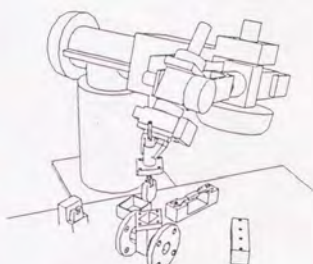
7. kitzhead を把握



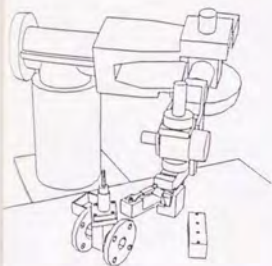
2. diaphragmholder を把握



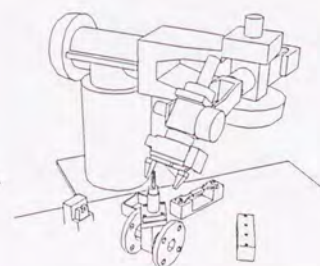
5. kitzhead を把握に行く中間通過点



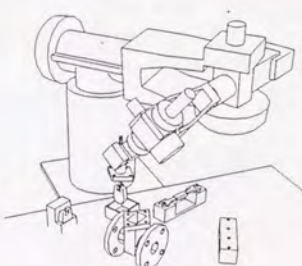
8. kitzhead の kitzbody に対する離脱点



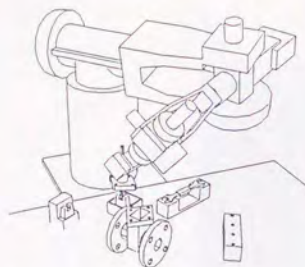
3. diaphragmholder から離脱



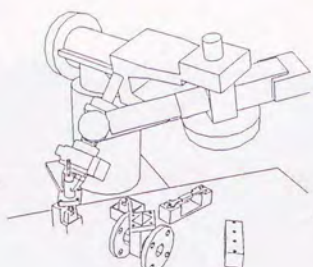
6. kitzhead へのアプローチ



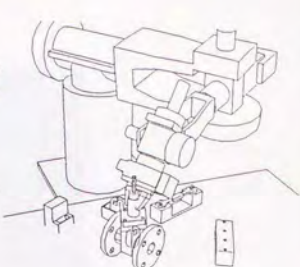
9. kitzbox への diaphragm のインストール点に対するアプローチ点



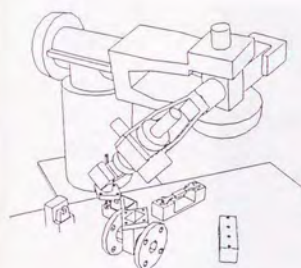
10. kitzbox への diaphragm のインストール点



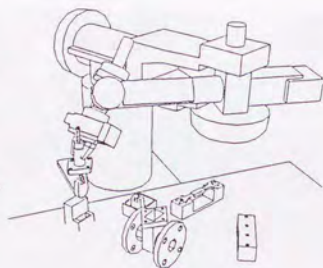
13. kitzrod に diaphragm をインストール



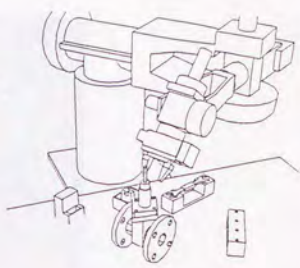
16. kitzbody に kitzhead をインストール



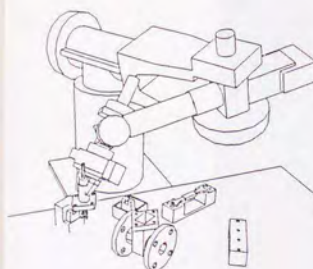
11. diaphragm と kitzrod のインストール点に対する離脱点



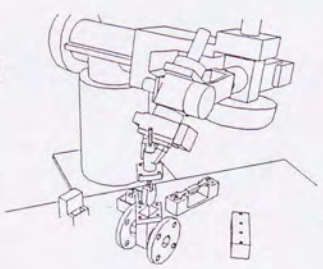
14. diaphragmholder に対する diaphragm の離脱点



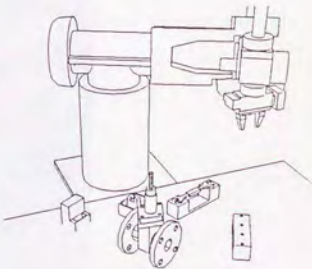
17. kitzhead の離脱点



12. 新しい diaphragm に対する kitzrod のアプローチ点



15. kitzbody に対する kitzhead のインストール点へのアプローチ



18. 作業が終了し eta3park へ復帰

図 5.11: バルブ弁交換作業の展開 (ソフトウェアによる隠線消去表示)

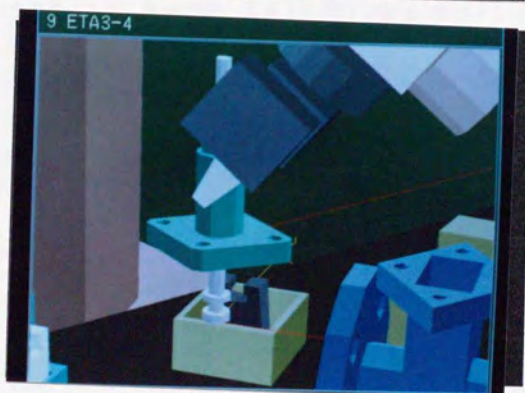
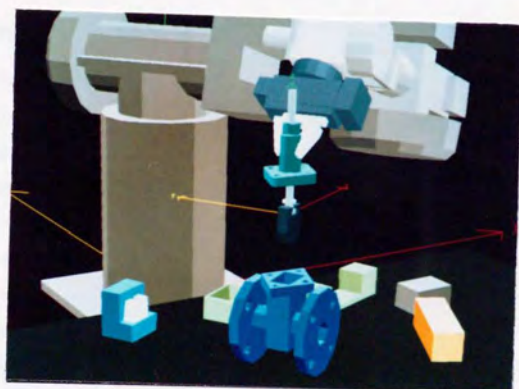
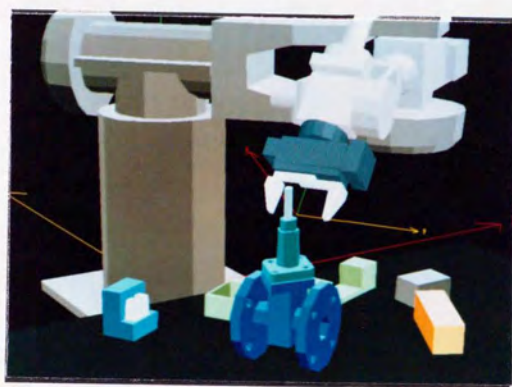


図 5.17: バルブ弁交換作業の展開 (MMD による高速表示)



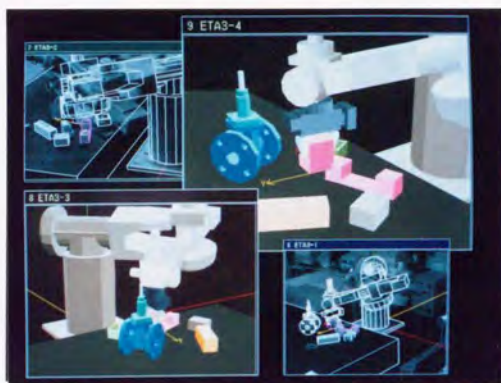


図 5.18: 幾何モデルを用いた干渉検査

### 5.3.2 干渉検査

シミュレーションでは物体同士の衝突の検査が重要である。移動物体の衝突検査については、岡野の方法 [51]、劉の方法 [80] などがあるが、本システムでは、各補間動作毎に、幾何モデルの面と面の交差を調べることで、ソフトウェアによる干渉検査を行っている。このとき、すべてのモデルの干渉を検査するのでは実時間性能が低下するので、干渉の可能性がある物体をオペレータが選択し (4.6.3節参照)、シミュレーションを実行する。図 5.13 のプログラムをそのまま実行すると、diaphragmholder を作業場所に運ぶ途中で、wrenchstand と衝突する。この状況は、図 5.18 に示すように、MMD のウィンドウ中で二つの物体を赤く表示することで明瞭に示される。

また、MMD の切断面表示機能を用いて簡単な干渉検査を行なうことも可能である。まず、ビューイングを操作して、干渉が起りそうな面に切断面 (前クリップ面) を設定する。次にシミュレーションによってマニピュレータや把握物体が切断面から飛び出して来ないかを監視する。図 5.19 の左上、右下のウィンドウはそのような表示であり、干渉のある部分が赤く塗られている。また、図 5.19 の左下のウィンドウのように物が邪魔になって手先の様子がよく見えない場合には、切断面によって手前側の物体を視野から除くこともできる。

ソフトウェアで自動的に干渉検査を行なうには、全環境物体についてのモデルが用意されていなければならない。ところが現実にはモデルの作成には手間がかかる。MMD ではスーパーバインポーズが可能なので、作業に重要でない環境の表示には TV カメラからの実画像を用い、マニピュレータモデルとの干渉は目視で検査する。こうして、複雑な環境をすべてモデリングしなくてもシミュレーションすることができる。これは緊急時の環境モデリングの時間の短縮、モデル化のミスに由来する誤動作の防止などに役立つ。

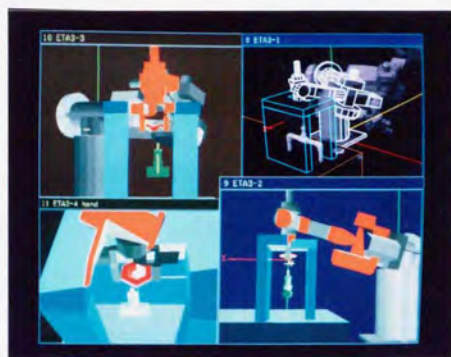


図 5.19: 切断面表示

### 5.3.3 動作編集

いずれかの方法によって、干渉が発見されると、軌道を変更しなければならない。現在、障害物回避軌道を自動的に生成する問題は各所で精力的に研究されているが、6自由度マニピュレータについて完全な軌道を短時間で発見する方法は未だに実用化されていない。そこで、オペレータによる対話的な動作の編集によって解決を図る。

オペレータが選択・付加する情報には、把握点の選択と中間通過点の挿入がある。対象物体によっては、把握姿勢が複数定義されている。オペレータは、順次提示される把握姿勢の中から、大局的状况を考慮して最も適当な姿勢を選ぶ。また、マウス、ダイヤルを操作してマニピュレータモデルを動かし、環境物体との干渉を避けられる点を中間通過点として挿入する。

## 5.4 作業の実行と監視

### 5.4.1 実行環境

シミュレーションによって作業が妥当なものであることが確認されると、実機を用いた作業が実行される。作業の実行には、図 5.20 に示すように、もう 1 台のワークステーション (ETA3 実行ステーション) を用いる。両ワークステーション上のプロセスは、イーサネットを通じた tcp/ip プロトコルを用いて通信する。図 5.15 に示された ETA3 命令は、さらに対応するコマンドコードに変換され、ネットワークを通じて ETA3 実行ステーション上の ETA3 サーバに送られる。コマンドを一つずつ送れば、シングルステップ形式で実行させることができる。

ETA3 サーバに問い合わせることで、実際の ETA3 の関節角を読み取れるので、その姿



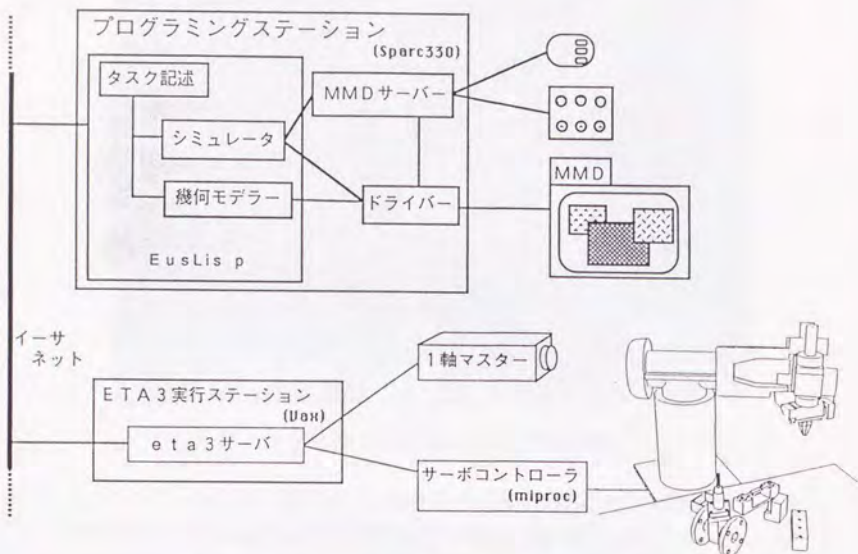


図 5.20: プログラミング、作業実行システムの構成

勢になるようにモデルを表示することができる。また、シミュレーションから予測される関節角度と比較することで、大きなエラーがないかを監視することができる。

一軸マスタは、eta3 サーバを介して eta3 マニピュレータをスレーブアームとして操作するためのデバイスである。eta3mv:manual 命令が送られると、この一軸マスタによるマスタスレーブモードに切り替わる。図 5.21 は、実際に ETA3 がバルブ交換作業をしている状況を示している。この作業では、シミュレーションした通りの動作が実現され、正しくバルブ交換作業を達成することができた。

#### 5.4.2 MMD を用いた実行監視

実作業を開始すると、各段階でエラーが発生しないかを監視しなければならない。エラーとは、システムが予測、管理しているモデルの状態と実際の状態の差異を意味する。この差異は実画像の上にモデルをスーパーインポーズすることで明瞭に示される。この方法によって、把握姿勢のずれや外乱による環境の変化などが容易に検出できる。

実行監視における MMD の利用法には、いくつか考えられる。一つは、グラフィックスによる動作の確認である。実マニピュレータの制御コマンドを一つ送る前に、その動作をシミュレータで実行し、スーパーインポーズウィンドウに表示する。これによって、システムが



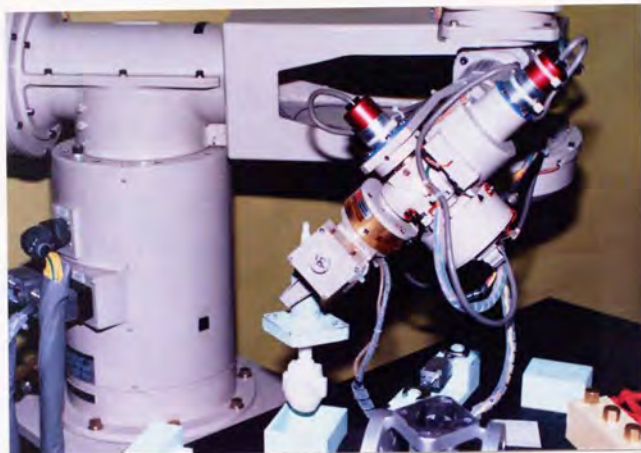


図 5.21: ETA3 による実際のバルブ交換作業

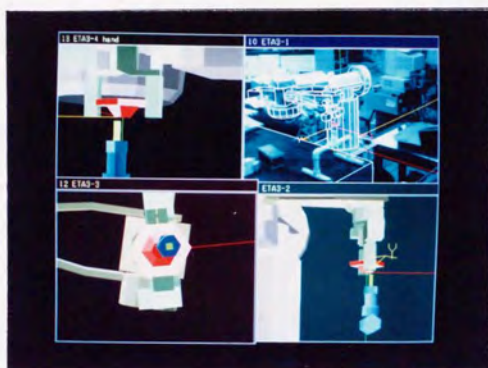
実行しようとしている動作を確認することができる。実行が正しく行われたかどうかは、TV カメラ画像とモデルの線画が正確に重なっていることを検査するだけでよい。

次に、作業精度の向上のためにグラフィックスを用いることが考えられる。実際の環境では任意の位置に TV カメラを配置することは難しいが、グラフィックスでは仮想的な視点を自由に設定可能である。したがって、たとえばはめあい作業の場合には、穴の中から指先を見上げるような表示によって作業の精度と効率を向上させることができる (図 5.22(a))。

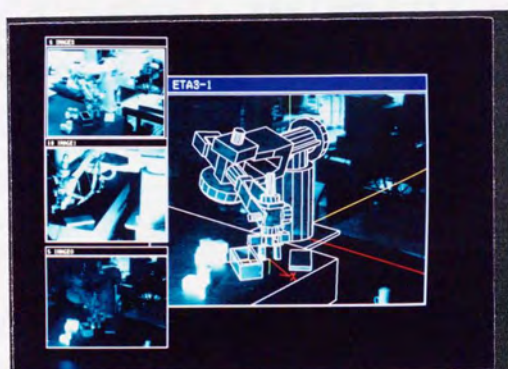
さらに、海底の作業では浮遊物などで視界が遮られたり、照明の問題などによって TV カメラによる監視が十分に機能しないことがある。このような場合は、TV カメラの実画像に代わる、グラフィックス表示が有効になる。すなわち、現場のロボットが送って来る関節角をモデルの関節角に反映させ、そのグラフィックス表示をコントラストの低下した実画像に重ねる (図 5.22(b))。操作者は実際のロボットの姿態を表す鮮明なグラフィックス像を見ながら作業を遂行できる。

また、MMD を海中や宇宙用ロボットの遠隔制御における伝送遅延を補うための予測表示に適用することも可能と考えられる。0.5 秒を越える映像フィードバックの遅れは、作業効率を大きく損なうと言われる [44]。海中での通信は超音波に頼るので、通信容量が限られている上に数百 m の海底ですら 1 秒の遅延が生じる。静止衛星では遅延は 2 秒に及ぶと言われる。Sheridan は、グラフィックスでロボットの動作を予測表示することで、地上と衛星を往復する長大なループを、操作者の手元の小さなループで置き換える手法を提案している。本システムでは 0.1 秒程度のループを形成することが可能である [64]。

もし、作業に不具合が見つかったり、または作業プログラムに指定された:manual-move に会すると、作業の実行はプログラムモードからマニュアルモードに移行する。このモードで



(a)



(b)

図 5.22: ロボット遠隔制御のグラフィクス支援

は、一軸マスタを操作して、マスタスレーブ形式でマニピュレータの動作を補助する。また、4.6節で述べたマウスとダイヤルを用いたモデルの対話的操作を、実際のマニピュレータに適用することも可能である。

## 5.5 5章の結論

マルチメディアディスプレイ (MMD) を用いて、ロボットの遠隔制御を統合的かつ対話的に進める手法およびそのシステム構成法について論じた。MMD は「TV カメラ画像にスーパーインポーズされたモデルの実時間・立体表示機能」を実現している。スーパーインポーズにより、実環境とモデルの差異が明瞭に提示され、実時間・立体表示によって3次元環境の理解が容易になる。この機能を用いて、環境モデリング、作業プログラミング、シミュレーション、実行監視の一連のステップが効率よく実施できることを示した。

環境モデリングにおいては、TV カメラ画像中の環境物体の上に線画が重なって表示されるようにグラフィクスモデルを操作することで、簡単に実物体の位置、姿勢を測定することができる。この操作は1から数分で完了し、注意深く操作すればカメラの解像度の限界に近い精度が得られることがわかった。

作業プログラミングは、4章で述べたような幾何モデルと記号的なマクロ記述を用いることにより、バルブの交換作業を短いプログラムで記述することができた。8つのマクロ動作は45の仮想マニピュレータ命令に分解され、さらに実際のETA3を制御する命令が生成される。生成された各々の動作はグラフィクスによってシミュレーションされる。シミュレーションによって物体の干渉が見つかった場合でも、オペレータが適当な中間通過点を対話的に挿入することで妥当な軌道を得ることができる。また、作業によって変化するワールドモデルをオブジェクトに保存しておくことにより、任意の時点のワールドを再現することができ、動作の対話的な編集を効率よく実行することができる。

実行監視においては、スーパーインポーズ表示を用いて、モデルが予測する動作と、実際の作業進行を明瞭に対比させることができる。TV カメラからの実画像に加えて、高速、立体視グラフィクスを援用することにより、カメラにとって劣悪な環境であっても、高い精度と高い応答性を確保することができるようになる。



## 第 6 章

### 結 論

オブジェクト指向型モデルの上に統合化されたロボットプログラミングシステムの構成について述べてきた。知能ロボットの実現には、複雑で大規模なソフトウェアが必要になる。環境・状況の認識、タスク記述と行動計画、器用な動作スキル等、未だに研究を必要とする要素技術は山積しているが、多くのソフトウェアを一つのシステムに統合化する技術は、それらの個々の研究に劣らず重要である。本研究では、統合化のための基盤技術が幾何モデルに基づくシステム記述言語にあると捉え、オブジェクト指向に基づく幾何モデリング機能を備えた「ロボットプログラミングの核言語 EusLisp」を提案した。また、3次元世界で動作するロボットのプログラミングを効率良く進め、モデルと現実の世界を対応付けるための視覚的インタフェース機能として「マルチメディアディスプレイ」を開発し、両者を用いた「統合的なロボット遠隔制御法」を提案した。

まず、ロボットのためのモデルを記述する枠組としては、オブジェクト指向プログラミングが、構造を柔軟に組み立てられる点、多くのロボットに汎用的に拡張できる点から最も適していると考え、システム記述の基盤をオブジェクト指向に置くこととした。EusLisp は、CommonLisp のほとんどの組み込みデータ型をクラスによって記述しており、これらの組み込み型を基にして、システムの柔軟かつ大幅な拡張を可能にしている。また、これにより、Lisp 自体の実現が統一的かつ容易になると言う利点も生まれる。オブジェクト指向言語で問題となるスロットアクセス、メッセージ探索、インスタンス生成の効率に対しては、単一継承を基本としメッセージ転送の機能を付加する、メッセージ探索を型の包含テストに置き換える、フィボナッチパディによるメモリ管理を実施する、などにより解決を図った。結果として、オブジェクトの上に効率の良い Common Lisp 処理系が少ない記述量で実現できることが示された。

幾何モデルの振舞いは、クラスによって定義される。座標系クラスを上位クラスとし、稜線、面、物体クラスを階層的に定義することにより、従来 Fortran 等で実現されることの多かった幾何モデルを、はるかに少ない量で整然と記述することができた。EusLisp 上に幾何モデリング機能と記号処理機能を統合することにより、作業のマクロな記述を可能とし、シ

ミュレーションの段階では干渉検査などの幾何計算機能を利用し、実マニピュレータの制御命令への展開では、重心、質量などのマスプロパティ情報を取り出すことが容易に行えるようになった。

記号的な作業の記述だけでは、3次元空間中にパターンとして広がる現実の環境を入力・表現することは困難であり、人間（プログラマー、オペレータ）と3次元情報を交換するのにも問題がある。モデル、実環境、人間の三つの世界間で相互に3次元情報を交換するための視覚的インタフェース装置としてマルチメディアディスプレイ（MMD）を開発した。マルチメディアディスプレイは、文字・グラフィックス・TVカメラ実画像を重畳させるマルチメディア機能、マルチウィンドウ、高速3次元グラフィックス、両眼立体視などの機能を備えている。これらの機能を並列動作するハードウェアによって実現することにより、実環境とモデルとの比較、奥行きを持った3次元情報の認識、3次元動作の対話的編集が効率よく処理できるようになった。

EusLisp、幾何モデル、マルチメディアディスプレイを用いて、従来はマスタスレーブに頼らざるを得なかったロボットの遠隔制御に、環境モデリング、言語によるプログラミング、シミュレーション、実行監視を統合した遠隔制御を実施する方法を提案した。バルブ部品の交換作業を例に取り、環境物体の位置、姿勢の決定、モデルを用いた作業プログラミングとそのシミュレーションによる動作の検証および対話的編集、スーパインポーズディスプレイとグラフィックスを用いた実行監視などが一つのプログラミングステーションで実施できることを示した。

本論文で論じてきたオブジェクト指向型モデルに基づくロボットプログラミングシステムは、アドバンスドテレオペレーションの実現に一步近付いてはいるが、未だに人間が判断を下す場面も多く残されている。たとえば、環境モデリングでは、画像理解の研究の成果を活かし、線画の自動的なマッチングが行えれば、モデルフィッティングの作業はより容易になるであろう。また、動作シミュレーションの結果、中間通過点を挿入しなければならないとき、障害物回避軌道生成の研究成果の一部でも導入できれば、オペレータとの対話の量を減らすことができるであろう。作業記述では、たとえば物体の持つ機能や意味からその組み立て手順が生成できるとしたら、作業はより高水準かつ簡潔に表現できると期待される。画像理解、障害物回避、作業記述等の研究においても、その基礎となるデータベースは物体の幾何形状である。実際、EusLispは音田らによる障害物回避軌道計画[34]、小俣による多指による物体把握計画[33]、比留川らによる物体の運動の拘束条件の導出[74]、坂根らによる視覚認識のための光源設定計画[31]、平井らの遠隔制御システムMEISTER[12]等の研究に応用され、扱い易い幾何モデリング機能を提供することで各方面で着実な成果をおさめつつある。本研究で論じてきた、幾何モデルを統合したプログラミングシステムをさらに発展させることで、今後より一層のロボットの知能化の研究が促進されると信ずる。



## 謝辞

本研究は、通商産業省「極限作業ロボット」大型プロジェクトのもとで進めた「オブジェクト指向に基づくロボットプログラミングシステム EusLisp」、「マルチメディアディスプレイを用いたロボットの知的遠隔制御」の研究をまとめたものである。論文をまとめるに当たっては、東京大学工学部機械工学科、井上博允教授のご指導を仰いだ。本研究を遂行するに当たって、若松清司元制御部長（現電総研所付主任研究官）、白井良明前制御部長（現大阪大学教授）、弓場敏嗣知能システム部長、佐藤知正自律システム研究室長、柿倉正義前自律システム研究室長（現電機大学教授）、赤堀寛元システム制御研究室長（現神奈川工科大学教授）、高瀬国克前行動知能研究室長（現電総研企画室長）には、終始暖かい励ましと論文の執筆方針について適切な御教示を頂いた。深甚の謝意を表するものである。

マルチメディアディスプレイの開発は、情報アーキテクチャ部分散システム研究室の塚本亨治氏との共同研究になるもので、仕様の検討からソフトウェアの作成方針まで指導を仰いだ。仕様の決定に当たっては、連日深夜までの議論が続いたが、氏の飽くことのない探究心と完璧な仕様を求める姿には驚くばかりであり、入所してまもない筆者は、研究に対する姿勢について多くを学ぶことができた。暖かい御指導に心から感謝致します。MMD ハードウェアの開発は、三菱電機コンピュータ製作所をお願いした。研究者の乱暴な要求の相談に乗り、立派なハードウェアを完成させて下さった三菱電機、西出政司氏に感謝致します。

当研究所ロボットグループの方々には、多くの面で御理解と御協力を頂いた。平井成興氏、長谷川勉氏には、論文の執筆方法について御教示を頂いた。カメラのキャリブレーションでは坂根茂幸氏および音田弘氏の、ETA3 アームの操作では末広尚士氏の、マニピュレータモデル作成には小笠原司氏の御援助を頂いた。また、小俣透氏には補間軌道計算法について、比留川博久氏には幾何計算法に関して御教示を頂いた。厚くお礼申し上げます。EusLisp の開発に当たっては、東京大学工学部情報工学科、稲葉雅幸助教授にご支援を頂いた。氏が多くのアプリケーションに EusLisp を実際に適用し、多くの改善の提案を頂いたお蔭で、ここまで機能を充実させることができた。EusLisp、MMD 共に、電子技術総合研究所のロボットグループの諸氏による熱心な議論と批判を浴びることでここまで成長できたものである。すべての EusLisp ユーザの方々、および自律システム研究室、行動知能研究室の皆様へ感謝致します。



## 参考文献

- [1] ANSI. American national standard for the functional specification of the programmer's hierarchical interactive graphics standard (phigs), 1985.
- [2] ANSI. American national standard graphical kernel system (gks), 1985.
- [3] B. Baumgart. *Geometric Modelling for Computer Vision*. PhD thesis, Stanford University, 1974.
- [4] D.G. Bobrow, L.G. DeMichiel, R.P. Gabriel, S.E. Keene, G. Kiczales, and D.A. Moon. Common lisp object system specification. Technical Report X3J13-Documnt 88-002R, ANSI, 1988.
- [5] I. C. Braid. *Designing with Volumes*. PhD thesis, Univ. Cambridge, 1973.
- [6] I. C. Braid and C. A. Lang. Computer-aided design of mechanical components with volume building blocks. In *PROLAMAT'73*, 1973.
- [7] H. Bromley and R. Lamson. *LISP LORE: A Guide to Programming the Lisp Machine 2nd Ed*. Kluwer Academic Publishers, 1987.
- [8] B. Cranson and R. Thomas. A simplified recombination scheme for the fibonacci buddy system. *CACM*, 18(6), 1975.
- [9] R.P. Gabriel. *Performance and Evaluation of Lisp Systems*. The MIT Press, 1985.
- [10] S. Ganapathy. Decomposition of transformation matrices for robot vision. In *International Conference on Robotics and Automation*, 1984.
- [11] D. D. Grossman. Procedural representation of three-dimensional objects. *IBM Journal of Research and Development*, 20(1):582-596, 1976.
- [12] S. Hirai, T. Sato, T. Matsui, and M. Kakikura. Integration of a task knowledge base and a cooperative maneuvering system for the telerobot meister. In *Proceedings of IEEE International Workshop on Intelligent Robots and Systems IROS'90*, 1990.
- [13] M. L. Hornick and B. Ravani. Computer-aided off-line planning and programming of robot motion. *Int. Journal of Robotics Research*, 4(4):18-31, 1986.

- [14] M. Hosaka and F. Kimura. An interactive geometrical design system with hand-writing input. In B. Gilchrist, editor, *Information Processing '77*, pages 167-172. North-Holland, 1977.
- [15] M. Hosaka, F. Kimura, and N. Kakishita. A unified method for processing polyhedra. *Information Processing '74*, pages 768-772, 1974.
- [16] G. L. Steel Jr. *Common Lisp the Language*. Digital-Press, 1984.
- [17] G. L. Steele Jr. Data representations in pdp-10 maclisp. In *Proceedings of 1977 MACSYMA Users Conference*. NASA Scientific and Technical Information Office Washington, D. C., July 1977.
- [18] S.E. Keene. *Object-Oriented Programming in Common Lisp*. Addison-Wesley, 1988.
- [19] Alfons Kemper and Mechtild Wallrath. An analysis of geometric modeling in database systems. *Computing Surveys*, 19(1):47-91, March 1987.
- [20] J. Kempf. Design for high performance dynamic generic dispatch in the common lisp object system. In *Proc. of CLOS users and Implementor Workshop*, 1988.
- [21] T Koshikawa and Y Shirai. A 3-d modeler for vision research. In *Proc. of Int. Conf. Advanced Robots*, 1985.
- [22] G. Larson, M. Donath, and D. Riley. Interactive interference checking for robot work cell emulation. In *ASME, Computer Engineering Conference*, pages 107-110, 1984.
- [23] L. I. Lieberman and M.A. Wesley. Autopass: A very high level programming language for mechanical assembler system. *IBM Research Report*, 1975.
- [24] D. G. Lowe. Solving for the parameters of object models from image descriptions. In *Image Understanding*, pages 121-127, 1984.
- [25] T. Lozano-Perez and P. H. Winston. Lama: A language for automatic mechanical assembly. In *Proceedings of 5th IJCAI*, pages 710-716, 1977.
- [26] T. Matsui and M. Inaba. Euslisp: An object-based implementation of lisp. *Journal of Information Processing*, 13(2), 1990.
- [27] T. Matsui and M. Tsukamoto. An integrated robot teleoperation method using multimedia display. In *Proceedings of 5th International Symposium on Robotics Research (ISRR)*, pages 156-163, 1989.



- [28] David J. Miller and R. Charleene Lennox. An object-oriented environment for robot system architecture. In *IEEE International Conference on Robotics and Automation*, pages 352-361, May 1990.
- [29] M. Minsky. A framework for representing knowledge. In P. H. Winston, editor, *The Psychology of Computer Vision*. McGraw-Hill, 1975.
- [30] S. Mujtaba and R. Goldman. *AL User's Manual*, January 1979.
- [31] 坂根茂幸、R. Niepold、佐藤知正、白井良明. 環境モデルに基づくハンドアイシステムの照明設定プランニング. *日本ロボット学会誌*, 7(3):130-141, 1989.
- [32] N. Okino, Y. Kakazu, and H. Kubo. Tips-1, technical information processing system for computer aided design, drawing and manufacturing. In *PROLAMAT'73*, 1973.
- [33] T. Omata. Fingertip positions of a multifingered hand. In *Proceedings of IEEE international conference on Robotics and Automation*, pages 1562-1567, 1990.
- [34] H. Onda, T. Hasegawa, and T. Matsui. Collision avoidance for a 6-dof manipulator based on empty space analysis of the 3-d real world. In *Proceedings of IEEE International Workshop on Intelligent Robots and Systems IROS'90*, 1990.
- [35] Richard P. Paul. *Robot Manipulators: Mathematics, Programming and Control*. MIT Press, 1981.
- [36] J.L. Peterson and T.A. Norman. Buddy systems. *CACM*, 20(6), 1977.
- [37] R.J. Popplestone, A.P. Ambler, and I.M. Bellos. An interpreter for a language for describing assemblies. *Artificial Intelligence*, 14:79-106, 1980.
- [38] Aristides A. G. Requicha. Representations for rigid solids: Theory, methods, and systems. *Computing Surveys*, 12(4):437-464, December 1980.
- [39] R. Roberts and I. Goldstein. *The FRL Manual*, 1977.
- [40] J Rooney. A survey of representations of spatial rotation about a fixed point. *Environment and Planning B*, 4:185-210, 1977.
- [41] T. Sato and S. Hirai. Meister: A model enhanced intelligent and skillful teleoperational robot system. In *Proceedings of 4th International Symposium on Robotics Research (ISRR)*, pages 155-162, 1987.
- [42] R. W. Scheifler and Jim Gettys. The x window system. *ACM Transactions on Graphics*, 5(2):79-109, 1986.



- [43] M.I. Shamos and F.Preparata. *Computational Geometry*. Springer Verlag, 1985.
- [44] T.B. Sheridan. Human supervisory control of robot systems. In *Proceedings of 1986 IEEE International Conference on Robotics and Automation*, pages 808-812, 1986.
- [45] T. Suehiro and K. Takase. Development of a direct drive manipulator: Eta-3 and enhancement of servo stiffness by a second-order digital filter. In *15th ISIR*, 1985.
- [46] R. Taylor, P. Summers, and J. Meyer. Aml: A manufacturing language. *The International Journal of Robotics Research*, 1(3):19-41, 1982.
- [47] Michiharu Tsukamoto and Toshihiro Matsui. Conceptual design of a distributed operating system for intelligent robots. In *Proceedings of 83ICAR*, pages 399-406, 1983.
- [48] M. A. Wesley, T. Lozano-Perez, L. I. Lieberman, M. A. Lavin, and D. D. Grossman. A geometric modelling system for automated mechanical assembly. *IBM Journal of Research and Development*, 24(1):64-74, 1980.
- [49] T. Yuasa and M. Hagiya. *Kyoto Common Lisp Report*. Teikoku Insatsu, 1985.
- [50] 越川和忠、大川和隆. かげを伴う立体の幾何学的モデリングの一手法. 電子技術総合研究所彙報, 52(6):845-888, 1988.
- [51] 岡野彰、川辺真嗣、嶋田憲司. シミュレーションによる移動物体間の衝突チェック. 日本ロボット学会誌, 6(1):35-41, 1988.
- [52] 沖野, 嘉数, and 久保. 自動設計プロセサー tips-1 のシステム設計. 精密機械, 44(3), 1976.
- [53] 沖野教朗. 自動設計の方法論. 養賢堂, 1982.
- [54] 沖野教郎. ロボットプログラミングとシミュレーション. 日本ロボット学会誌, 3(2):118-123, 1985.
- [55] 高瀬国克, 若松清司. 知的オペレーションシステムの構成法とその要素技術. 日本ロボット学会誌, 第2巻(2号):566-575, 1984.
- [56] 小笠原、松井. ロボットプログラミング機能を持つ lisp 処理系の開発. In 日本ロボット学会第2回学術講演会, 1984.
- [57] 小笠原司、井上博允. 知能ロボットプログラミングシステム cosmos. 日本ロボット学会誌, 2(6):507-525, 1984.

- [58] 松井俊浩、稲葉雅幸. Euslisp: オブジェクト指向に基づく lisp の実現と 幾何モデラへの応用. In 情報処理学会記号処理研究会, number 2 in 50, 1989.
- [59] 松井俊浩、塚本亨治. ロボット用マルチメディアディスプレイ—テキスト機能—. In 情報処理学会第 30 回全国大会, 1985.
- [60] 松井俊浩、塚本亨治. ロボット用マルチメディアディスプレイ—実画像機能—. In 日本ロボット学会第 3 回学術講演会, 1985.
- [61] 松井俊浩、塚本亨治. ロボット用マルチメディアディスプレイ—マウスを用いた教示と操縦への応用—. In 日本ロボット学会第 4 回学術講演会, 1986.
- [62] 松井俊浩、塚本亨治. ロボット用マルチメディアディスプレイ—管理ソフトウェア—. In 情報処理学会第 32 回全国大会, 1986.
- [63] 松井俊浩、塚本亨治. Euslisp: 対象指向による型拡張性を有する lisp の実現. In 情報処理学会第 35 回全国大会, 1987.
- [64] 松井俊浩、塚本亨治. ロボット用マルチメディアディスプレイを用いた 遠隔ロボット操作システム. In 宇宙用人工知能とロボットシンポジウム SAIRAS-87, pages 23-26, 1987.
- [65] 松井俊浩、塚本亨治. マルチメディアディスプレイを用いた遠隔ロボット制御システム. 電子技術総合研究所彙報, 52(12):1826-1850, 1988.
- [66] 松井俊浩、塚本亨治. マルチメディアディスプレイを用いた遠隔ロボット制御法. 日本ロボット学会誌, 6(4):301-310, 1988.
- [67] 松井俊浩、塚本亨治、小笠原司. Lisp 上の対象指向型プログラミング機能 leo. 電子技術総合研究所彙報, 49(7), 1985.
- [68] 松井俊浩、塚本亨治、西出政司. ロボット用マルチメディアディスプレイ—階層化モデルと実時間グラフィクス—. In 第 2 回 NICOGRAPH 論文コンテスト, pages 11-18, 1986.
- [69] 松原仁、岡野彰、井上博允. 作業目標レベルのロボット言語の設計と試作. 日本ロボット学会誌, 3(3):220-227, 1985.
- [70] 森田秀樹、小町祐史, and 安田靖彦. 投影法に基づく高速画素密度変換方式. 画像電子学会誌, 11(2):72-83, 1982.
- [71] 塚本亨治、松井俊浩. ロボット用マルチメディアディスプレイ—ニューフレームアルゴリズム—. In 情報処理学会第 30 回全国大会, 1985.

- [72] 塚本亨治、松井俊浩、西出政司. ロボット用マルチメディアディスプレイ—グラフィックス機能—. In 情報処理学会第 31 回全国大会, 1985.
- [73] 塚本亨治、松井俊浩、大野次彦、佐々木雅之. ロボット用マルチメディアディスプレイ—制御ソフトウェア—. In 情報処理学会第 32 回全国大会, 1986.
- [74] 比留川博久、松井俊浩、高瀬国克. 多面体間の接触による拘束条件を満たす微小変位を幾何モデルから求める方法. In ロボット学会第 7 回学術講演会, 1989.
- [75] 平井成興、佐藤知正. Larts/t を用いた言語主導型マスタスレーブマニピュレーション法. 日本ロボット学会誌, 第 2 巻 (6 号):526-534, 1984.
- [76] 平井成興、佐藤知正. 言語介在型マスタスレーブ・マニピュレータシステム. 計測自動制御学会論文誌, 第 20 巻 (6 号), 1984.
- [77] 平井成興、佐藤知正. 極限環境とテレロボット. 日本ロボット学会誌, 7(6):739-744, 1989.
- [78] 米澤明憲. オブジェクト指向計算の現状と展望. 情報処理, 29(4):290-294, 1988.
- [79] 末広 and 高瀬. 直接計算方式による作業座標サーボに基づくマニピュレーションシステム. 日本ロボット学会誌, 3(2):11-21, 1984.
- [80] 劉雲輝、登尾啓史、有本卓. 移動物体間の干渉が効率的にチェックできるソリッドモデル hsm の提案. 日本ロボット学会誌, 7(5):426-434, 1988.



