筑波大学大学院博士課程

工学研究科修士論文

直観的な操作に基づく３次元モデリングツールと３次元ビジュアルプログラミングシステムの構築

電子・情報工学専攻

著者氏名　大芝　崇

指導教官　電子・情報工学系 田中 二郎

目次

図一覧

要旨

本論文では、次世代のコンピューティング環境の１つの例として３次元環境を取り上げ、その基礎技術の１つである３次元モデリングツールと３次元ビジュアルプログラミングシステムについて述べる。本研究では、３次元モデリングツールとして “Claymore” を開発し、３次元ビジュアルプログラミングシステムとして “3D-PP” を開発している。

我々は、既存のシステムについてサーベイした結果、様々な問題点があることを発見した。そこで、実際に我々が実装したシステムを具体例として取り上げ、その問題点が我々のシステムでどのように解決されるかについて論述する。

第 1 章

序論

近年、 VRML 等で記述されたウェブページの普及に伴ない、数多くの一般のコンピュータのユーザが３次元環境でのインタラクションを経験する機会が増えてきている。３次元グラフィックスはもはや特殊な技術ではなく、映画やゲーム、テレビなどで誰でもが日常的に接するものとなってきている。また、コンピュータのソフトウェア・ハードウェア技術の劇的な発達に伴ない、ユーザに対してこれまでとは異なる、新たなコンピューティング環境を提供することが可能になって来ている。我々は、そのような次世代のコンピューティング環境では、コンピューティング・パワーの多くはコンピュータ・ヒューマン・インタラクション (CHI) に注力されると考え、次世代コンピューティング環境の１つとして、３次元コンピューティング環境について注目している。

しかし、３次元環境とのインタフェースには、グラフィカル・ユーザ・インタフェース (GUI) に代表される従来の２次元環境におけるインタフェースにはなかった、数々の問題を解決しなければならず、一般のコンピュータのユーザには敷居の高いものとなっている。３次元環境におけるインタフェースとして、どのような形態が最も有効であるかは、未だ模索中であるのが現状である。

我々は、３次元環境が一般のコンピュータのユーザにとってより身近なものとなるようなインタフェースに着目しており、まず初めに、３次元環境を構築するために必要となる、３次元モデリングツールの開発を行った。従来使用されているような３次元モデリングツールでは、手軽に３次元物体を作成・編集することは困難であった。そのため、多くの一般ユーザにとって、３次元コンピュータ・グラフィックス (CG) は依然として専門家によって作成されたものを鑑賞するだけのものであり、３次元 CG を誰もが手軽に扱うことは出来ないのが現状である。そこで我々は、２次元のドローイングツールである MacDraw をあたかも３次元にしたような感覚で、容易に３次元物体の作成・編集が出来るような環境が必要不可欠であると考え、ユーザが直観的なモデリングを行えることを目標とする３次元モデリングツール “Claymore” を開発し

た。

次に我々は、３次元プログラミング環境について着目した。近年のコンピュータの低価格化・高性能化に伴い、頻繁に図形の描き換えを行うビジュアルプログラミングを、実際のアプリケーションとして使用することが現実的になってきている。我々は現在、特に３次元環境でのビジュアルプログラミングシステム (VPS) について興味があり、３次元プログラミング環境の１つとして、実際に３次元 VPS“3D-PP” を設計・開発している。

我々は、“3D-PP” を開発するに当たって、既存の VPS について調査し、以下のような問題点を解決しなければならないことに着目した。既存の VPS の多くは２次元表示によるプログラムの視覚化を行っている。「ノード」や「エッジ」などのビジュアルプログラムのプログラム要素は図形で表示されるため、プログラム要素の数が多くなるにつれて VPS の表示領域内がプログラム要素で埋め尽くされてしまい、それ以上ビジュアルプログラムにプログラム要素を追加出来なくなってしまう、という問題があった。２次元表示に基づく VPS では、このような貧弱なスケーラビリティが原因で、規模の大きい実用的なビジュアルプログラムは記述出来ず、結局はトイ・プログラムしか記述できなかった。

我々は３次元表示に基づく VPS こそ、 VPS のスケーラビリティ問題の解決への大きなブレイク・スルーになると考えている。単純な例で考えても、ある面積の表示領域を持つ２次元 VPS において、最大で $10 \times 10 = 100$ 個のノードを２次元空間内に配置可能な場合、同じ面積の表示領域を持つ３次元 VPS では奥行き方向をノードの配置に利用出来るため、 $10 \times 10 \times 10 = 1000$ 個ものノードを配置することが可能である。

また、既存の VPS では、比較的小さなビジュアルプログラムを記述する際にも、ユーザはマウスを用いて「ノードの生成」や「エッジによる結線」などの単純で低レベルな操作を何回も繰り返さなければならなくなっており、操作が煩雑になりがちであった。一方テキストプログラムの世界では、キーボードを用いて高速にプログラムを入力することができ、またプログラムの一部をコピー＆ペーストすることによって再利用することも簡単に出来る。我々は “3D-PP” を開発するに当たって、テキストプログラムの記述の際に用いられるような効率的な入力方法をヒントに、従来の VPS にはなかった効率的なビジュアルプログラムの入力方法を実装した。

既存の VPS では、プログラミング初心者のプログラミングの学習用として開発されたものもあり、小規模のビジュアルプログラムが記述出来れば十分有用とされている場合もあるが、我々は “3D-PP” がターゲットとするエンド・ユーザとしてプログラマも想定しているので、規模の大きなプログラムが記述出来るということは重要な

目標の１つである。

本論文では、我々が開発した３次元モデリングツール “Claymore” と３次元 VPS“3D-PP” において用いられている要素技術、及び対話的な操作方法について詳細に述べる。第 2章から、３次元モデリングツール “Claymore” の概要とその特徴について述べる。３次元モデリングツール環境を構築するに当たり、どのような操作方法・視覚化方法を用いればユーザが手軽に３次元物体を扱えるようになるかに焦点を当てる。また、ユーザが３次元モデルを作成・編集する際の手順についても言及する。第 5章からは、３次元 VPS“3D-PP” の概要について述べ、その次にユーザが直観的かつ効率的にビジュアルプログラムを記述できるようにするための要素技術について述べる。

第 2 章

３次元モデリングツール “Claymore”

３次元コンピューティング環境をより身近なものとして扱うには、ユーザが手軽に３次元コンピューティング環境を構成するコンテンツを作成出来ることが重要である。例えば、３次元 VPS 環境を構築するためには、３次元表現されたノード・エッジ・アイコン等の部品を、あらかじめ作成しておく必要がある。

３次元物体のコンテンツを作成するツールが「３次元モデリングツール」であり、ユーザは簡単な操作で３次元モデリングツールを使いこなせることが重要である。しかし、従来使用されているような３次元モデリングツールの多くものはプロ用に作られており、メニューや設定項目の数が覚え切れないほど多数ある等、初心者にとっては非常に敷居の高いものとなっていた。そのため、非専門家が手軽に３次元物体を作成・編集することは困難であった [6]。

我々は、２次元のドローイングツールである MacDraw をあたかも３次元にしたような感覚で、容易に３次元物体の作成・編集が出来るような環境が必要不可欠であると考える [15, 16, 26, 25]。

このような背景により、我々はユーザが直観的なモデリングを行えることを目標とする３次元モデリングツール “Claymore” を開発している。

2.1　三面図における問題点

従来の３次元モデリングツールは、物体を構成する頂点などの座標を、三面図や数値入力をもとに、直接入力していく方式を取っている [6]。

三面図とは、３次元空間を構成する３本の座標軸に垂直な視点から見た異なる３つのビューで構成されており、２次元の表示装置を用いて３次元の物体を表現するための一般的な手法である。しかし三面図を使用するシステムでは、ユーザは異なる３つのビューの間の整合性をとるメンタルモデルを再構成する認知的負荷を強いられる [29]。

このため、三面図を使用する際に、ユーザは三面図上において、何度も視点を移動させる必要がある。また、ユーザは自分がどのビューに対して操作を行えばタスクが正しく遂行されるかを常に意識し、確認しながら作業を行わなければならないので、三面図を用いた操作系はユーザにとって負担が大きな操作系であるということが言える。

このような理由から、三面図を使用するシステムでは、３次元物体を直観的に扱うことが困難であった。

2.2 従来の直接操作手法

直接操作手法とは、ユーザが扱っている対象の概念やユーザの認知的な負荷を減らすために、対象物に対して直接指示を出し、その結果がその対象物に反映される操作系のことを言う。ユーザが容易に３次元物体を操作するためには、直接操作手法 [28] が適用されたユーザインタフェースが非常に有効である。

しかし、本来 直接操作手法は２次元の操作系を持ったシステムに対して発展して来た手法であるので、２次元のシステムに対しては非常に親和性がある。しかし、ここで対象としているのは３次元空間に存在する物体への操作であり、従来の直接操作手法を単に３次元物体への操作系に適用するだけでは不十分である。

これは、入出力装置が２次元であるのに対し、操作の対象となる物体が３次元表示されていることに起因する。通常の３次元 CG では、２次元の平面であるコンピュータのディスプレイに投影表示を行なっている。また、入力デバイスとしてマウスを対象としているが、これは２次元の入力デバイスである。

2.3 強化された直接操作手法

そこで我々は、従来の直接操作手法を拡張した**強化された直接操作手法**を提案する [14, 24, 30]。これは、**付加情報**を用いることによって直接性・直観性が強化されたユーザインタフェースである。このユーザインタフェースを用いることにより、ユーザは２次元の入出力デバイスを用いていながら、１次元分のギャップを持った３次元の物体を容易に操作することが出来るようになっている。

2.3.1 付加情報

付加情報を物体と同時に表示することで、ユーザは**付加情報**を操作の補助として、効率的に直接操作を行なうことができるようになる。

付加情報として、具体的には以下のものをユーザに提示する (図 2.1)。

- **３次元マウスカーソル** — 操作の対象となっている位置を矢印のような形で表示することによって、ユーザは２次元の画面内部の３次元空間の、どの部分で作業を行っているのかという状況を把握できるようになる。なお、本来の表示になるべく支障のない形状であるべきであるので、実際には矢印ではなく、単純な形状である方向を持った線分として表現している。

- **地面** — もし３次元空間上に物体が浮かんでいるだけという表示の仕方をすると、ユーザは基準となる位置関係を把握することが困難になってしまう。そこで、地面を表示することによって、ユーザは操作の対象としている物体の、３次元空間上での絶対的位置を簡単に認識できるようになる。

- **物体の影** — 地面の上に物体の影を表示することで、ユーザは視点を動かさなくても、ある程度 物体の相対的位置関係を把握することが可能である。例えば、真っ暗な背景の上に３次元物体が２つ浮かんでいたとすると、ユーザは視点の移動等しない限り、２つの物体の相対的な位置関係を把握することは困難である。また、影の形状から、物体の形状の概形を把握することもできる。

- **３次元バウンディングボックス** — ３次元バウンディングボックスは、物体を囲む直方体である。実際のモデルの作成において、三面図で使用している３つの平面に平行な操作が要求される場合がある。このため、物体を囲む平行六面体であるバウンディングボックスの各面を基準面とすることで、三面図を用いなくてもそのような操作が可能である。

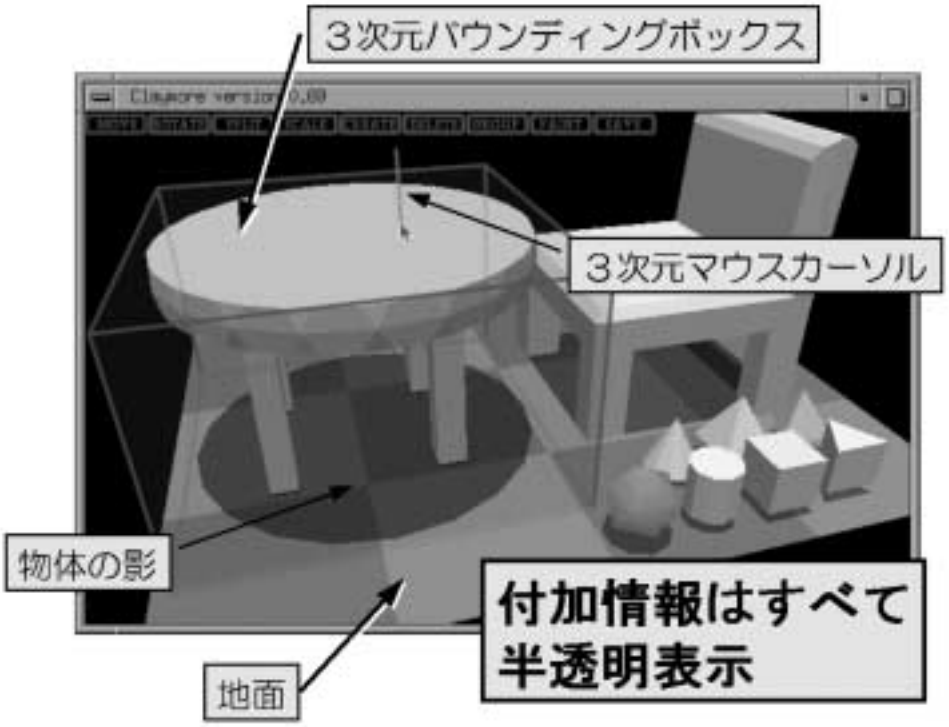


図 2.1: 付加情報を表示

2.3.2 半透明表示

付加情報の表示の際には、本来の物体の表示をなるべく阻害しないことが望ましい。本来の物体の上に**付加情報**を重ねて表示する際には、半透明表示が有効である [27]。そこで、**付加情報**の表示にはすべて半透明表示を用いた (図 2.1)。半透明表示により、ユーザは本来の操作対象である３次元物体に対する操作を行いながら、**付加情報**によって３次元物体に対する認識をより深くすることが出来る。

これらの**付加情報**により、１次元分の情報のギャップを補うことが出来るようになっているので、ユーザは３次元物体への直接操作をより自然に行なうことができるようになっている。

第 3 章

“Claymore” におけるモデルの作成・編集パラダイム

“Claymore” では、ユーザは以下の手順でモデルを作成する。ここでは例として、 “Claymore” を用いてテーブルを作成する手順を説明する (図 3.2, 3.4, 3.5, 3.6 参照)。

現段階で “Claymore” に実装されている機能には、物体の生成・移動・回転・切断・変形・削除・グループ化・属性の編集・セーブ等がある。ユーザはウィンドウ左上のモード選択ボタン (図 3.1) を用いることによって、それぞれの操作を選択する。

図 3.1: モード選択ボタン

“Claymore” は、正確で緻密な形状を時間をかけて作成することを目指したものではなく、比較的簡単な概形やモックアップを手軽に生成することを目的としている。このため、正確なモデルを生成する前段階のプロトタイピングを作成するための利用や、他の 3 次元アプリケーションで用いるコンテンツを手早く作成するなどの応用例が考えられる。

3.1 基本物体の生成

まず、 3 次元アイコンを用いて基本物体を生成する。 3 次元アイコンは常に地面の上に表示されている (図 2.1 右斜め下)。ユーザは 3 次元アイコンをクリックすることによって基本物体を生成することが出来る。

現段階では比較的単純な形状をした 3 次元アイコンが用意されているが、その形状データはそれぞれファイルとして保存されている。ユーザは、あらかじめ自分用に独

自の３次元アイコン用に用意し、３次元アイコンのファイルと書き換えればより複雑な形状をした３次元アイコンを使用することが出来る。

３次元アイコンには、地面の上に表示されることによって視点が移動しても常にユーザに３次元空間の上方向がどちら側なのかを容易に知らせる働きがある。ここではまず、立方体と球を生成する (図 3.2)。

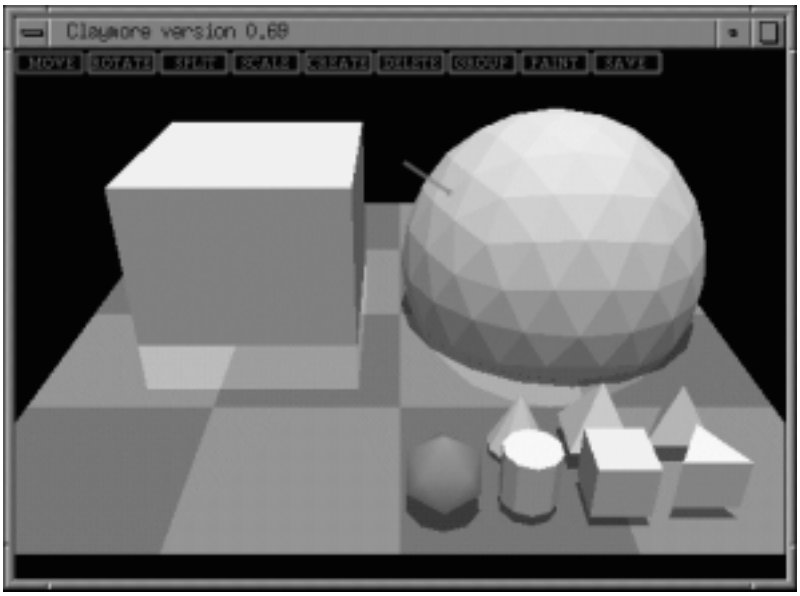


図 3.2: テーブルの作成・編集プロセス (基本物体の生成)

3.2 切断および変形

次に、切断および変形をすることによって、より複雑な形状をした部品を作成する。ここでは物体の切断機能について説明する。ユーザは切断用の平面をマウスによって操作し、切断用平面を境界として物体を２つに分割することができる。

切断用平面を半透明表示することで、切断後の形状をユーザに認識しやすいように工夫している。

ユーザは、３次元カーソルが３次元物体と接触している位置を変えることによって、以下の３通りの切断手法を使い分けることが出来る (図 3.3)。

図 3.3 a — ３次元カーソルが接触している３次元物体の平面の方向で切断

図 3.3 b — ３次元カーソルが接触している３次元物体の稜線の方向で切断

図 3.3 c — ３次元カーソルが接触している３次元物体の頂点の方向で切断

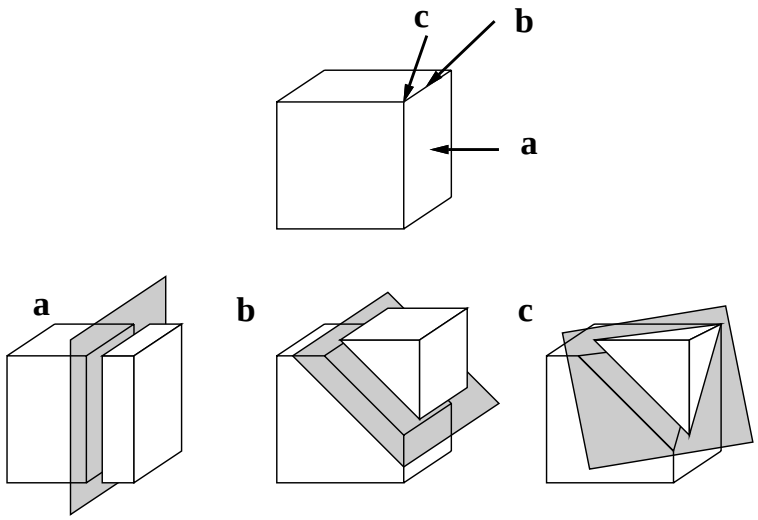


図 3.3: ３種類の切断用平面

ここでは球を２回切断することによってテーブル上部の円い板の部分を作り、立方体を切断することによってテーブルの４本の脚の部分を切り出す (図 3.4)。

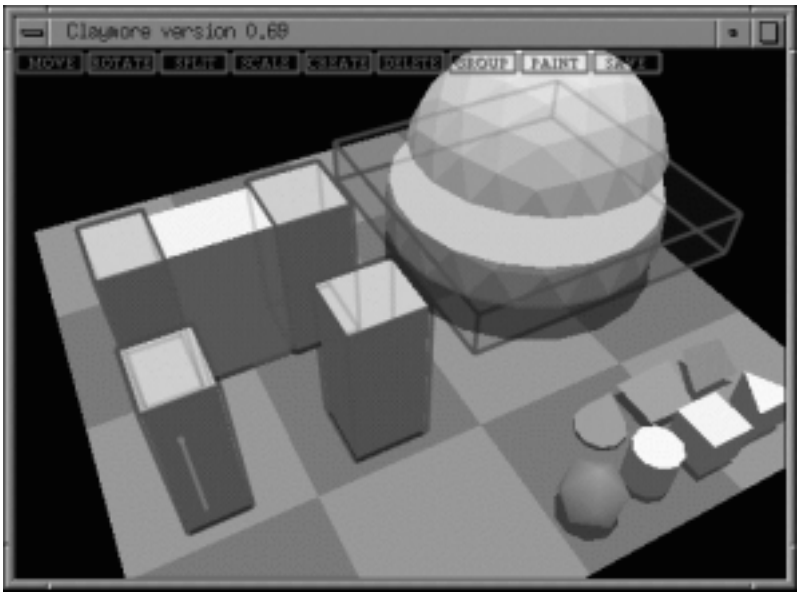


図 3.4: テーブルの作成・編集プロセス (切断および変形)

3.3 グループ化

個々の部品にグループ化を行って部品同士を組み合わせる。ユーザが選択した物体はバウンディングボックスで囲まれ、複数の物体を選択後グループ化のボタンを押すことによって、選択された物体は結合された１つの物体として扱われる。ここではテー

ブルの４本の脚をグループ化する (図 3.5)。

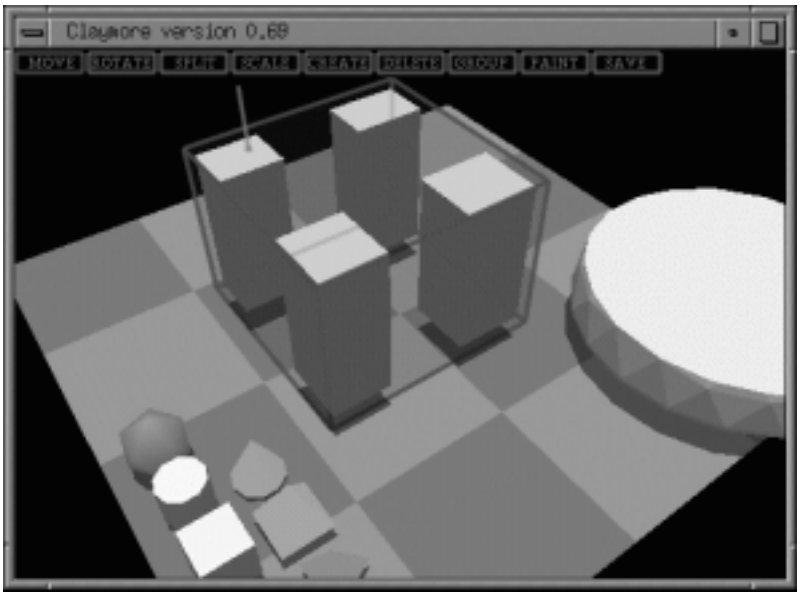


図 3.5: テーブルの作成・編集プロセス (グループ化)

3.4　移動・回転により配置

こうして作られた部品を、移動・回転によって配置する。ここではテーブルの円い板と４本の脚を適切な位置に配置する (図 3.6)。

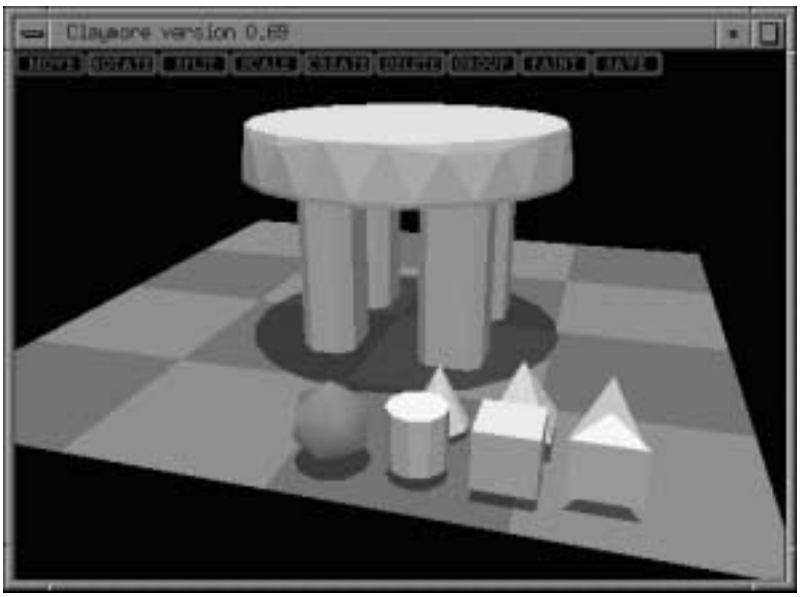


図 3.6: テーブルの作成・編集プロセス (移動・回転により配置)

3.5　セーブ

最後に、作成したモデルの形状データを VRML 形式または Wavefront OBJ 形式等のモデリングデータフォーマットでセーブする。

今後は、物体を粘土細工のような感覚で、引っ張ったり押しつぶしたり出来るような、より強力な変形機能を実装する予定である。

一旦セーブした形状データは、他の３次元アプリケーションで容易に再利用することが可能である。またその際には、まず一度 緻密な編集作業を目的とした他の３次元モデリングツールを用いて細部を編集してから他の３次元アプリケーションで用いる、というような利用の仕方も考えられる。

第 4 章

新たに追加した機能・進捗状況

“Claymore” に、以下に述べるような新たな機能を追加した。また、他の 3 次元モデリングツールとの連携に関しても研究を進めた。

4.1 物体の属性の編集機能

先に述べた通り、我々は “Claymore” の応用例として、他の 3 次元アプリケーションへ 3 次元モデルを提供することを想定している。3 次元アプリケーションの 1 つのレイとして、3 次元ビジュアルプログラミングシステムがある。

一般に、ビジュアルプログラミングシステムではユーザのプログラムの視認性を支援するために [3]、プログラム要素に色情報等の属性を持たせることが多い。そこで我々は、編集している 3 次元物体のマテリアル情報を変更し、物体の色を変更する機能を新たに実装した。

ユーザがウィンドウ上部のモード選択ボタンから PAINT ボタンを選択すると、地面の上に 3 次元アイコンとして新たにカラーリング物体 (図 4.1 左斜め下) が表示される。カラーリング物体は普段は表示されず、 PAINT モードになった時だけに表示される。これは、不必要に多量の情報を表示させないことを意識した動作であり、また、論理的な情報隠蔽を物理的な情報隠蔽によって達成している [3]。

ユーザは、色情報を変更したい物体を選択し、変更したい色と同じ色を持つカラーリング物体をクリックすることによって、選択した物体の色をクリックしたカラーリング物体と同じ色に変更することが出来る (図 4.1)。

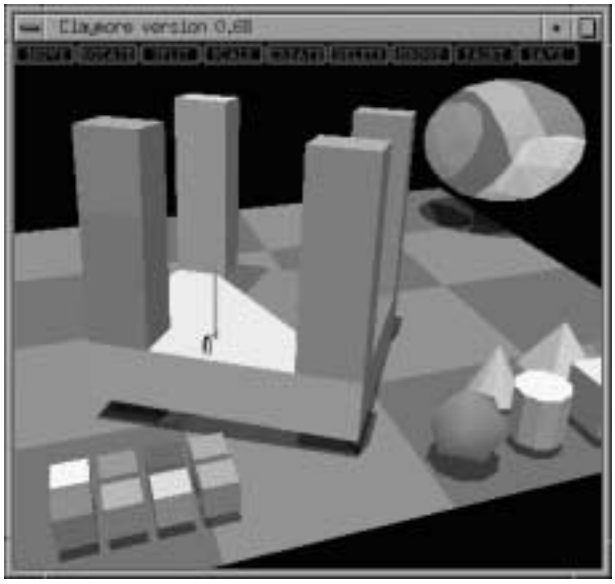


図 4.1: カラーリング機能

4.2 多関節物体のサポート

多関節物体等の、より複雑なモデルが扱えることが重要であると考え、多関節物体のサポートを行った。具体的には、これまでは “Claymore” を用いて VRML 形式と Wavefront OBJ 形式での形状データの出力が可能であったが、今回新たに LightWave3D 形式での出力もサポートすることが出来るようになった (図 4.2)。 LightWave3D 形式では物体間のグループ関係や親子関係を簡単に記述することが可能であるので、 “Claymore” 内で作成したコンテンツのグループ関係や関節情報、親子関係の状態を保ったままファイルに出力することが出来る。

LightWave3D 形式は、 LWO ファイルと LWS ファイルの 2 種類のファイル形式によって構成されている。 LWO ファイルには、各物体の形状データがバイナリの形で保存されている。 LWS ファイルには、シーンの光源・視点の配置や、シーンに登場する複数の物体の位置やグループ関係・親子関係を管理するシーンデータがテキストの形で保存されている。 LWS ファイル内で複数の LWO ファイル間の関係を記述することによって各物体間のグループ関係や親子関係を記述することが出来るようになっている。この機能を利用することにより、 “Claymore” 内で設定した関節構造やグループ化された状態を他の 3 次元ツールにエクスポートすることが出来る。

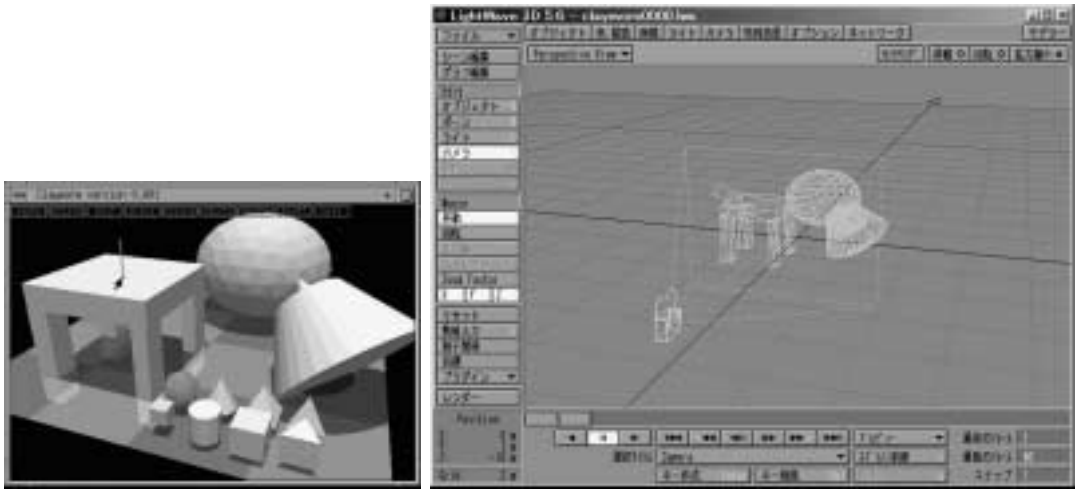


図 4.2: LightWave3D 形式への出力

実際に、神戸大学瀧研究室にて開発中の 3 次元動画生成システム $A\bullet\mathrm{T}$ [36] との間のコンテンツの受け渡しの共通フォーマットとして LightWave3D 形式を採用し、 $A\bullet\mathrm{T}$ 内に登場する人物や背景物体等の形状データを “Claymore” を用いて作成した (図 4.3)。また、今後は九州大学谷口研究室にて開発中の分散画像入力システム EZ に必要となるプリミティブに、 “Claymore” 内で作成した 3 次元モデルを使用してもらう予定である。

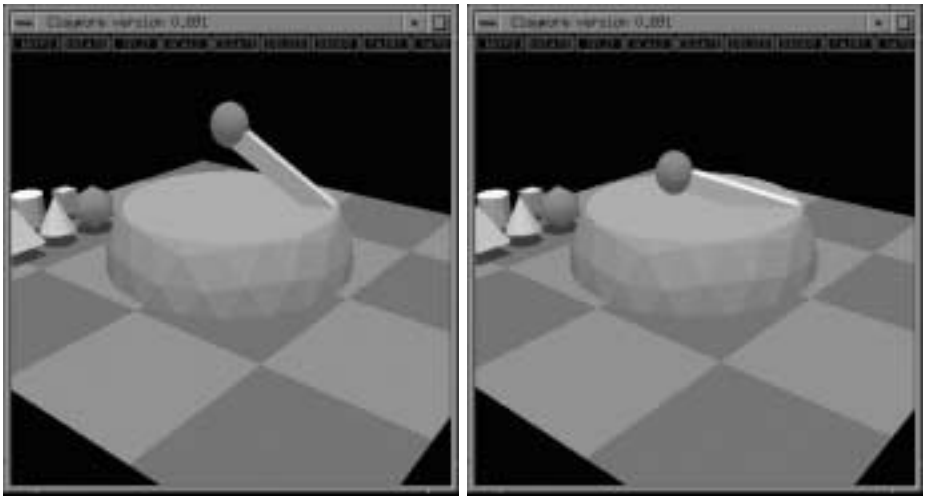


図 4.3: 関節を持った 3 次元モデル

4.3 公開ソフトウェア

多くのユーザに実際に使用してもらい、ユーザからの感想や要望を元に開発を進めて行くことも重要であると考え、“Claymore” を WWW 上に公開した。“Claymore” の web page の URL は

```
http://www.softlab.is.tsukuba.ac.jp/iplab/claymore/
```

であり、 web を閲覧する環境があれば誰もが “Claymore” を実際にダウンロードすることが出来るようになっている。“Claymore” はフリーソフトウェアとして公開されており、 Copyright は BSD Lisence に準じたものとなっている。実際に、現段階で海外のユーザから数件のメールによるフィードバックを受け取っている。

また、ユーザの各自のコンピュータ環境に合わせた “Claymore” のコンパイルの方法や、実際に “Claymore” を使用する際の操作マニュアル等の情報を入手することが出来るようになっている。

動作環境として、これまでは IRIX6.3, Solaris2.6, Windows9x であったが、新たに Linux でも動作させることが可能になった。

また、福岡大学鶴田研究室にて、 “Claymore” と九州大学谷口研究室で開発中のシステムとを結合する研究が進められており、この作業に関しての要望も “Claymore” の開発方針に取り入れて研究を行っている。

4.4 開発環境

“Claymore” は、汎用の３次元グラフィックスライブラリである OpenGL を使用し、開発言語としては C, C++ を使用して実装されている。また、ユーティリティツールキットとして GLUT(OpenGL Utility Toolkit) を利用している。開発用マシンとしては、主に Silison Graphics 社の O_2 を使用した。

4.5 評価

現段階で “Claymore” はモデリングツールとしての基本機能の実装が完了しているので、その有効性を評価するために市販の３次元モデリングツール等との比較実験を行う予定である。

具体的には、比較的簡単な３次元モデルを、何もない状態から作成するというタスクを設定し、双方のシステムでユーザのタスク完了までに要した操作のステップ数・

時間等を測定する。この結果について比較し、“Claymore” に実装されている手法の有効性を検討する。

4.6 “Claymore” に関連する研究

マウスを用いて物体を回転させる方法としては、Chen らによる研究 [4] がある。これは擬似的なトラックボールに対して操作を行なうことで、３軸すべての操作を行なうことができる手法である。

Chen の手法では操作の対象となる物体を、トラックボールを介して間接的にしか操作出来ないが、“Claymore” で用いられている手法ではユーザは操作の対象となる物体を直接的に操作出来る。また、本手法の特徴として、物体の回転量が常にドラッグを開始した点からのマウスの移動距離によって決まり、ドラッグ中のマウスの移動軌跡に影響を受けないという点において異なっている。

２次元の入力デバイスを用いて３次元形状を直接編集する試みとして、Robert らによる SKETCH [37] がある。これはペンを用いた入力を対象として、ジェスチャの概念により物体を簡単に生成・加工する手法である。物体の表示には平行投影を用いているため、完成時の物体と同一の映像を見ているわけではない。

本研究ではマウスを対象としており、また物体の表示には遠近法を用いた透視変換を用いているため、完成時の物体と同一の映像を見ながら編集を行なうことができるという点において異なっている。

SKETCH と似た手法を用いた研究として、五十嵐らによる手掛きスケッチによる３次元モデリングの研究 [8] がある。これもペンを用いた入力を対象とした研究であり、３次元モデリングシステム Teddy に実装されている。ユーザは手書きスケッチをする感覚で３次元物体を生成することができるようになっている。

Teddy は初心者のための娯楽用ソフトウェアとしての利用や、医師が患者に疾患部位を説明したり学校の先生が生体の構造を説明する場合など、電子黒板を利用した環境でのコミュニケーション支援への利用を想定して作成されているのに対し、“Claymore” は比較的簡単な概形やモックアップを手軽に生成することを目的としている。このため、正確なモデルを生成する前段階のプロトタイピングを作成するための利用や、他の３次元アプリケーションで用いるコンテンツを手早く作成するなどの利用を想定して開発されている点で異なっている。また、Teddy は動物やぬいぐるみのような物体の生成を対象としているが、建物や家具、機械部品などの人工的な物体の作成には向いていない。

物体の分割を用いてモデリングを行なう研究としては、米川らの空間分割モデルを

用いた形状モデラの研究 [35] および北川らの対話型手術シミュレーションの研究 [11] がある。

これらの研究では、モデルの表現にボクセルモデルである Octree を用いており、どんな形状でも簡単な計算で分割を行なえるが、データ量が大きいという特徴をもっていた。本研究では、サーフェースモデルを対象としており、 VRML 等のデータへ変換するといった作業を簡単に行なうことができるという点において異なっている。

マウスを入力デバイスとして用い、３次元物体を変形させる研究として、 Thomas の研究 [31] がある。これは３次元物体を非常に細かい三角形のポリゴンで表現し、マウスの軌跡に対して３次元物体をさらに細かい三角形で動的に分割することにより、物体を時間とともに連続的に変形させることが出来る変形手法を実現している。

Thomas の研究は、３次元物体のアニメーションを主眼に置き、アニメーションの生成に特化した手法であるが、本研究は３次元物体のモデリングを目的としている点において異なっている。

第 5 章

３次元ビジュアルプログラミングシステム “3D-PP”

我々の研究グループではビジュアルプログラミングについての研究を行っている。近年のコンピュータの低価格化・高性能化に伴い、頻繁に図形の描き換えを行うビジュアルプログラミングを、実際のアプリケーションとして使用することが現実的になってきている。また、オブジェクト指向方法論の発展に伴ない、アプリケーションの開発者は、自分でプログラムコードを書くのではなく、あらかじめ他人の手によって書かれた効率的で拡張性のあるプログラムを再利用して、上手く組み合わせることによってアプリケーションを開発して行く傾向が強くなってきている。我々はこの傾向の次の段階として、テキストコードを書く必要があまりないビジュアルプログラミングが今よりもさらに重要視されるのではないかと考えている。

我々は現在、特に３次元環境でのビジュアルプログラミングシステム (VPS) について興味があり、実際に３次元 VPS“3D-PP” を設計・開発している [17, 18, 20, 21]。

既存の VPS [5, 32] の多くは２次元表示によるプログラムの視覚化を行っている。「ノード」や「エッジ」などのビジュアルプログラムのプログラム要素は図形で表示されるため、プログラム要素の数が多くなるにつれて VPS の表示領域内がプログラム要素で埋め尽くされてしまい、それ以上ビジュアルプログラムにプログラム要素を追加出来なくなってしまう、という問題があった [7]。２次元表示に基づく VPS では、このような貧弱なスケーラビリティが原因で、規模の大きい実用的なビジュアルプログラムは記述出来ず、結局はトイ・プログラムしか記述できなかった　[3]。

5.1 VPS の３次元に伴なうメリットと、その目的

我々は３次元表示に基づく VPS こそ、 VPS のスケーラビリティ問題の解決への大きなブレイク・スルーになると考えている。単純な例で考えても、ある面積の表示領域を持つ２次元 VPS において、最大で $10 \times 10 = 100$ 個のノードを２次元空間内に配置可能な場合、同じ面積の表示領域を持つ３次元 VPS では奥行き方向をノードの配置に利用出来るため、 $10 \times 10 \times 10 = 1000$ 個ものノードを配置することが可能である。

このように、 VPS を３次元化するということは、それ自体が大変大きなパラダイム・シフトを引き起こすに十分な可能性を秘めていることが分かる。

既存の VPS では、プログラミング初心者を対象とした、プログラミングの学習用として開発されたものもあり、小規模のビジュアルプログラムが記述出来れば十分有用とされている場合 [9] もあるが、我々は “3D-PP” がターゲットとするエンド・ユーザとしてプログラマも想定しているので、規模の大きなプログラムが記述出来るということは重要な目標の１つである。

5.2 “3D-PP” で用いるコンテンツを “Claymore” を用いて作成

一般に、３次元ビジュアルプログラミング環境を新たに構築する際には、３次元表現された「プログラム要素」等のコンテンツをあらかじめ３次元モデリングツールで作成しておくことが必要不可欠である。

5.2.1 “Claymore” と “3D-PP” との連携

我々は “3D-PP” を開発するにあたって、 “3D-PP” に必要となるコンテンツである、数多くの「プログラム要素」を総て “Claymore” を使用して作成した。ここでは例として “3D-PP” の「プログラム要素」の１つである、ゴールの形状データを “Claymore” を使用して作成する時の方法を説明する。

“3D-PP” においてゴールは円柱の形状として表現される。また、プログラムの意味合いによって、ゴールには異なる色が付けられる。

実際に “Claymore” を使用して “3D-PP” のゴールの形状データを作成する際には、まず３次元アイコンによって円柱を生成し、切断機能によって円柱の上下部の角を削ることによって、ある程度の丸みを出す。次に、カラーリング機能によって適切な色を付ける。最後に、作成したゴールの形状データをセーブ機能によってファイルに保存する。 “3D-PP” では “Claymore” によってセーブされたファイルをロードすること

によって、ゴールの形状データを利用することが出来る。

“Claymore” を使用することによって、必要となるコンテンツである「プログラム要素」を容易に作成・編集することが可能になったため、“3D-PP” の 3 次元ビジュアルプログラミング環境の構築に要する開発期間等のコストを、大幅に軽減させることが出来た。

5.2.2 作成したコンテンツに意味を与える

一般に、ビジュアルプログラミングシステムには図形の形状・配置の解析を行う spatial parser のモジュールがある。これはテキスト言語の処理系における字句解析器・構文解析器にあたる。

現在、我々の研究室では 2 次元図形に意味や制約を与える空間パーサ (spatial parser) を持ったシステム “恵比寿” を開発している [1, 2]。

“Claymore” で作成・編集されたコンテンツは、単なる図形の集合であり そのままではプログラムとしての意味を持たない。現段階では “恵比寿” に入力される構文要素は 2 次元図形であるが、今後 “Claymore” を用いて作成した 3 次元図形に意味や制約を与えるようなシステムを開発する予定である。

5.3 “3D-PP” の概要

“3D-PP” は、並列論理型言語の 1 つである GHC [33] を対象として開発されている [18]。

“3D-PP” は従来の視覚化手法である 2 次元の視覚化 [9, 32] ではなく、3 次元による視覚化の新しい手法を取り入れた実用的なビジュアルプログラミングシステムであり、並列論理型言語 GHC の特徴であるゴールのリダクションの過程や論理変数の具体化に着目した表現手法を実装している。“3D-PP” において、ユーザは 3 次元表現されたノード・エッジ・アイコン等の「プログラム要素」を組み合わせていくだけでプログラミングをすることができる (図 5.1)。

5.4 “3D-PP” のシステム構成

5.4.1 言語のモデル化

“3D-PP” は、次のような理由から並列論理型言語の 1 つである GHC に基づいて開発されている。

- 『言語を構成する基本要素の数が少なく，規則がシンプルである』

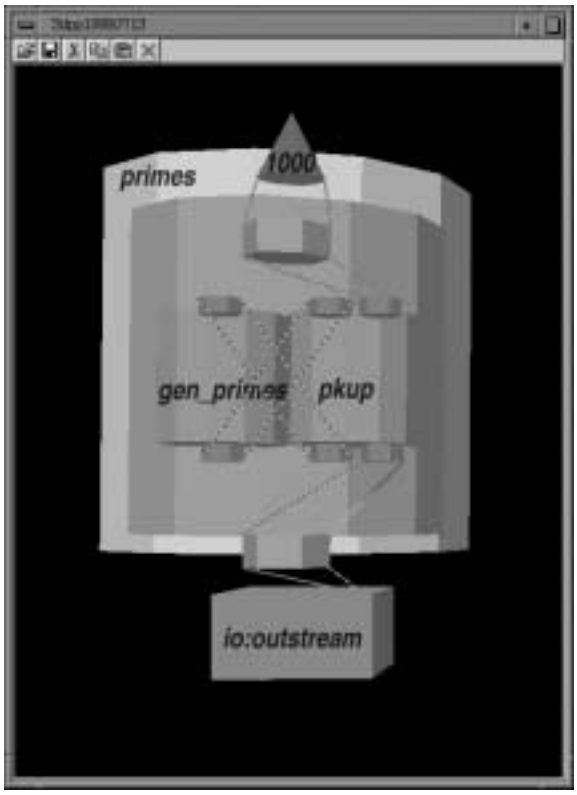


図 5.1: “3D-PP” の実行画面

GHC を含む宣言型の言語は、データ構造の要素となる意味単位 (アトム) や演算子、そしてそれらのデータ構造間の関係規則によって構成される。また実行規則も、ゴールをサブゴールに置換するというもののみで、非常に簡潔なものである。

このため、手続き呼び出し・変数・制御構造など多くの要素・規則を含む手続き型言語などに比べて視覚化が容易である。

- 『データ及びプログラムが単一の表現から成る』

  GHC のプログラムは、データ構造から新しいデータ構造への変換規則を定義するものであり、データ構造と同じ構成要素を用いて記述される。

  従ってビジュアルプログラム内のプログラム要素を操作する場合、操作対象による表現の違いを意識するような必要がなく、プログラマの負担も軽くて済む。

5.4.2 ユーザとのインタラクション

“3D-PP” は、ユーザがマウスを用いてビジュアルプログラムの作成や編集を行う「ビジュアル・エディタ部」と、 GHC に基づいて設計された並列論理型言語 KL1 [12] の

処理系である KLIC との通信を行う「KLIC エンジン部」の、2 つのサブシステムから構成されている。ビジュアルプログラムは内部表現を用いて表現され、2 つのサブシステム間は内部表現を介してやりとりを行う。

本研究では主に、ユーザ側から直接見える「ビジュアル・エディタ部」とユーザとのインタラクションに焦点を絞って論述する。ユーザが容易にビジュアルプログラムを記述できるようにするにはどうすれば良いかについて検討する。

第 6 章

Extended Drag and Drop — ３次元版 Drag and Drop 手法

プログラマはマウスを用いて「ビジュアル・エディタ部」を操作することによってビジュアルプログラムを記述する。

プログラマが容易に３次元表現されたプログラム要素を操作するためには、直接操作手法 [28] が適用されたユーザインタフェースが非常に有効である [14, 22, 23, 24, 25, 26, 30]。

しかし、本来 直接操作手法は２次元の操作系に親和性があるものであるので、従来の直接操作手法を単に３次元物体への操作系に適用するだけでは不十分である。これは、 “Claymore” の項で述べた、入出力デバイスが２次元であるのに対し、実際に扱っている物体が３次元であるという同じ理由による。

6.1 ３次元空間内での移動操作の問題点

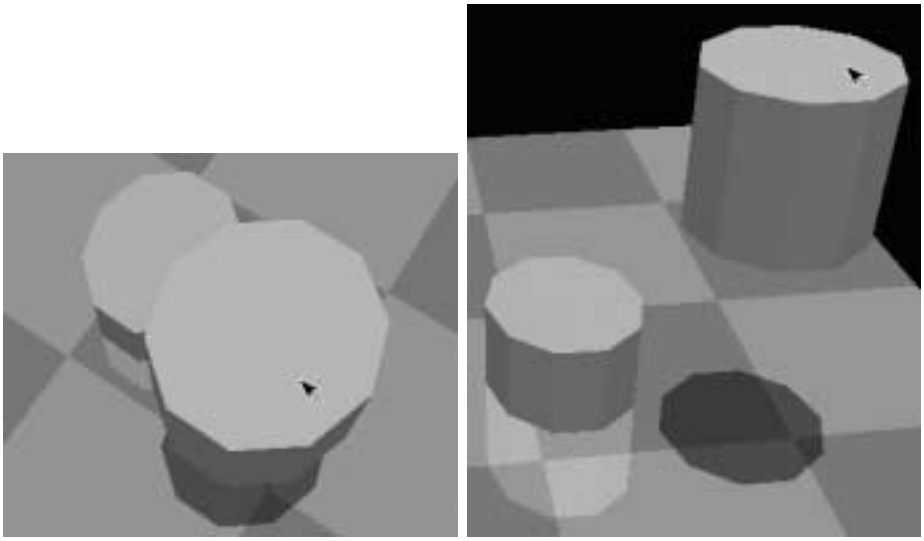
“3D-PP” では、プログラマがビジュアルプログラムを記述する際には、子ノードを親ノードへ Drag and Drop する必要がある。

もともと Drag and Drop 手法は直接操作手法に基づいて考案されたものであるので、２次元の操作系で用いられていた Drag and Drop 手法を、単純に３次元の操作系に適用するだけでは不十分であり、以下のような問題を解決する必要がある。

まず、図 6.1a では２つのノードは３次元空間内で物理的に重なっているように見えるが、実際には図 6.1b のように重なっていない場合があるという問題がある。これは、コンピュータのディスプレイが２次元であるために、３次元空間での奥行き情報が欠落してしまっていることに原因がある。

この場合、プログラマは自分では Drag and Drop 操作を完了したつもりでいても、

実際には Drop が行われていないことになってしまう。そのため、 Drag and Drop 操作をした後、プログラマは視点を移動する等して、常に Drop 操作が正しく行われたかどうかを確認しなければならない。



(a) プログラマからの視点　　(b) 別の視点からの画像

図 6.1: 奥行き情報の欠落

また、図 6.2 のように Drop しようとする対象物体が視点から見て奥にある場合、手前にある Drag 中の物体で隠れてしまうという問題もある。これは、視点から 3 次元空間を見た映像をディスプレイに投影射影する際に、同じ大きさの物体でも視点に近くに位置しているものは大きく、視点から遠い位置にあるものは小さく投影されるからである。

この場合、プログラマは常に Drag 中の物体 (子ノードに当たる) と Drop するべき物体 (親ノードに当たる) が両方見えるような位置に視点を移動してから Drag and Drop 操作を行わなければならない。

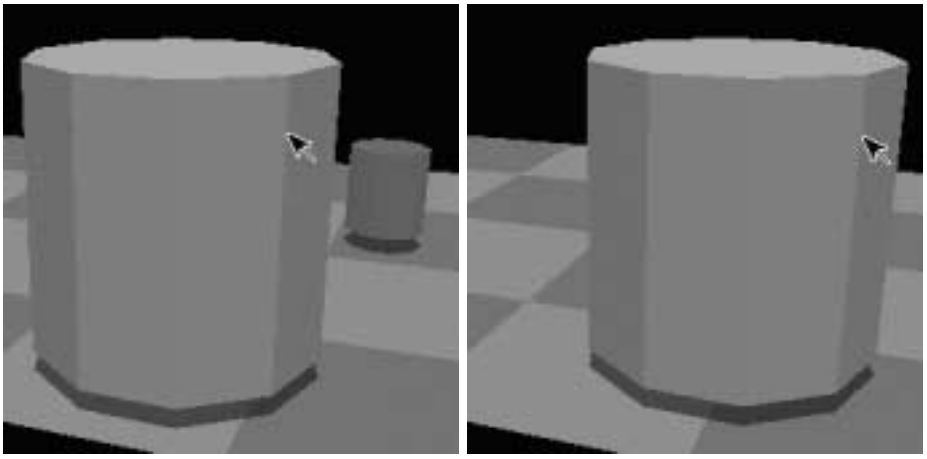


図 6.2: Drop する対象物体が Drag 中の物体で隠れてしまう

6.2 Extended Drag and Drop

このような問題を解決するために、我々は従来の Drag and Drop 手法を 3 次元環境に適応するように拡張した Drag and Drop 手法を提案する。この節では、この手法の仕組みを説明する。

まず、マウスの動きに合わせてノードを移動する際に、ノードの移動を 1 つの平面に拘束する拘束平面を用意する。拘束平面は実際には表示しないが、移動させるノードの位置の計算のために用いる。次に、プログラマが地面をドラッグすることによって視点を移動したとしても、拘束平面の法線を視線方向と常に平行にしておくことによって、拘束平面を常にディスプレイと平行な状態にしておく (図 6.3)。

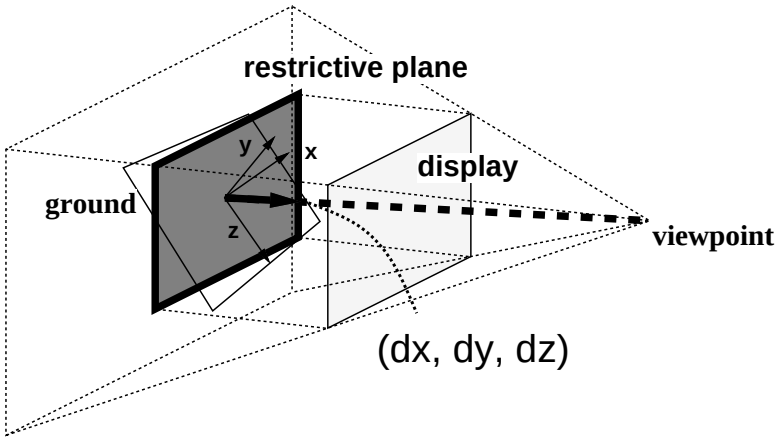


図 6.3: 拘束平面は常にディスプレイと平行

つまり、マウスを用いてノードを移動させると、その時の視点の位置に関わらず、そのノードを常にディスプレイと平行な平面の上を移動させることが出来る。さらに、

実際には図 6.1b のように 2 つのノードが 3 次元空間内で物理的に重なっていない場合でも、図 6.1a のように 2 次元画像で見て 2 つのノードの画像が重なっている場合には手前のノードを奥のノードに Drop することが出来るようにした。

また、図 6.4 のように手前にある Drag 中の物体を半透明表示することによって、Drop しようとする対象物体が視点から見て奥にある場合でも隠れないようにしている。

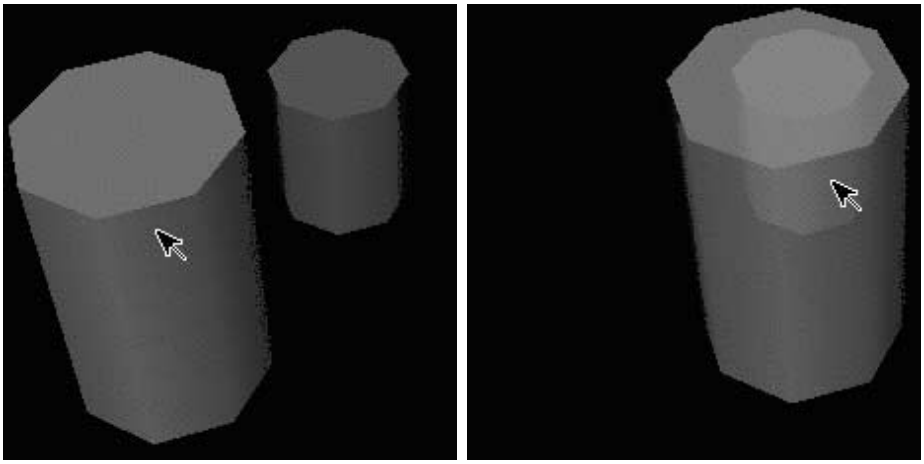


図 6.4: 半透明表示によって、重なってしまったノードも隠れない

これらの工夫によってプログラマは、実際には 3 次元の物体であるノードを、あたかも 2 次元の物体であるかのような感覚で Drag and Drop することが出来るようになる [10]。

我々はこの手法を **Extended Drag and Drop 手法**と呼んでいる。“3D-PP” でのノードの移動操作や Drag and Drop 操作は、全て Extended Drag and Drop 手法に基づいて行われるようになっている。このため、“3D-PP” では 3 次元空間についての知識が少ないユーザであっても簡単にビジュアルプログラムを記述することが出来るようになっている。

第 7 章

“3D-PP” におけるプログラムの記述方法と、ビジュアルプログラムの作成支援

“3D-PP” では３次元アイコンと呼ばれる、図 7.1 に示すような、ノードのミニチュア立体アイコンをマウスの左ボタンで Drag and Drop することによって、同じ形で少しサイズの大きなノードを生成することが出来る。生成したノード間をエッジによって結線する際には、ノードにマウスカーソルを合わせ、結線したいノードのところへマウスの右ボタンで Drag and Drop する。プログラマは、３次元アイコンから次々に必要なノードを生成し、ノード間をエッジで結線することによってビジュアルプログラムを記述して行くことが出来る。左ボタンと右ボタンによる Drag and Drop は、どちらも Extended Drag and Drop 手法が適用されているので、プログラマは実際には３次元の物体であるノードやエッジを、あたかも２次元の物体であるかのような感覚で操作出来る。

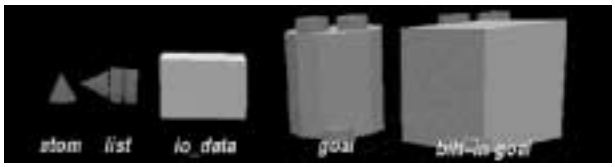


図 7.1: ３次元アイコン

7.1 テキスト言語での効率的なプログラム作成法の例

従来のテキストベースのプログラミング環境において、テキストエディタ等で規模の大きなプログラムを記述する際には、次のような方法を用いることによってプログラムの記述を効率化している。

1. プログラム内の、似たような構造を持つ部分を雛型とし、その複製を作ってから、変更が必要な部分だけを修正する。オブジェクト指向言語では、雛型となるような親クラスを 1 つ作り、そのクラスを継承させた子クラスを複数作って、必要な機能を追加する。

2. マクロ等の枠組みを利用することによって、複数の操作をまとめて一度に実行する。

3. 白紙の状態からプログラムを記述し始めるのではなく、似たようなプログラムが記述されたファイルを一括にロードし、必要な部分を流用する。

我々は、 VPS においても上記のような方法を適用して、入力の手間を軽減させることによってプログラマを支援することは非常に有用であると考えた。

7.2 “3D-PP” におけるビジュアルプログラムの作成支援

VPS 上でビジュアルプログラムを記述する際には、規模の小さいプログラムであれば、白紙の状態からでも [ノードの生成] → [ノード間をエッジで結線] → [ノードの生成] → [ノード間をエッジで結線] ··· という単純な作業の繰り返しでプログラムを完成させることは比較的容易である。

しかし、プログラムの規模が大きくなると、それに従って必要となるノードやエッジの総数が増加するため、白紙の状態からプログラマが全てのプログラムを入力するには低レベルな作業の繰り返しをしなければならず、ビジュアルプログラムの作成作業が煩雑になり過ぎてしまうという問題がある。

我々は 7.1 節で述べたテキスト言語でのプログラムの記述の効率化手法のアイデアを、 VPS に適応することによってこの問題の解決を試みた。実際には、以下に述べるような手法を “3D-PP” に適用することによって、プログラマが効率的にビジュアルプログラムを記述出来るようにした。

7.3 雛型を利用

7.1 節 (1) で述べたように、プログラム内の似たような構造を持つ部分を雛型として利用する。 GHC には「1 つのゴールは多くの場合複数の定義節によって構成されている」また「ヘッド部と同じ述語がボディ部に現われることが多い」という特徴がある。

“3D-PP” は GHC をベースとしているので、この特徴を利用した視覚化をすることは特に有効である。

具体的には、あらかじめ「ゴールノードが定義節ノードを数個包含している」雛型や、「ヘッド部と同じ述語がボディ部に既に配置されている」雛型を 3 次元アイコンとして用意する (図 7.2)。我々はこれらの 3 次元アイコンを特に **Readymade Icon** と呼んでいる。

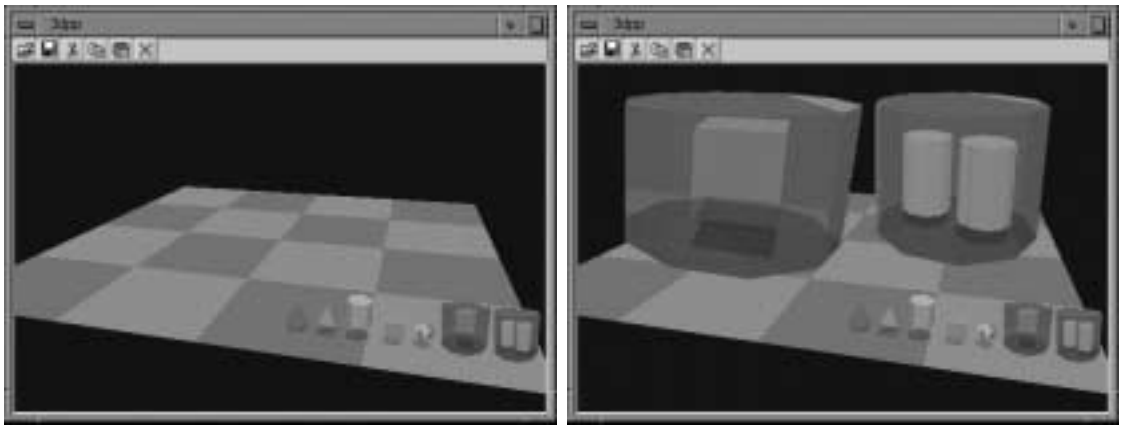


図 7.2: 雛型の 3 次元アイコン (右下の 2 つ)

プログラマはゴールを 1 つ作成する際には (図 7.2 の左側)、この雛型の 3 次元アイコンを Drag and Drop することによって、あらかじめ出来合いのゴールを一度の操作で生成することが出来る (図 7.2 の右側)。これによって、定義節用のノードをわざわざ数個作ったりボディ部にヘッド部と同じ述語を作ったりする手間を省略することが出来る。

雛型の 3 次元アイコンが 7.1 節 (1) の親クラスに当たり、生成された出来合いのゴールが子クラスに相当する。実際にプログラマが記述するプログラムは多種多様であるので、雛型の 3 次元アイコンとして用意されているもの以外の内部構造を持ったゴールを生成される場合がある。この時は、雛型の 3 次元アイコンから一度ゴールを生成しておき、必要な部分を修正するだけで要求に合った内部構造を持ったゴールを作成することが出来る。

Readymade Icon を用いることによって、ビジュアルプログラムの規模が大きくなり、必要となるノード数が増加しても、プログラマがビジュアルプログラムを記述する手間を軽減することが出来る。

7.4 一筆書き手法

“3D-PP” においてプログラマは図 7.1 に示したような 3 次元アイコンを用いてノードを次々に生成し、生成されたノード間をエッジで結線して行くことによってビジュアルプログラムを記述する。

この一連の複数の作業を 7.1 節 (2) に述べたように一度にまとめて実行出来れば、ビジュアルプログラムの記述の手間が大変軽減する。

我々は、2 次元のペイントツール上で一筆書きをするような感覚で、3 次元空間に浮かんでいる複数のノードの上をマウスの右ボタンでドラッグすることによって、ドラッグした軌跡と交わったノード間にエッジによる結線を行うことが出来る機能を実装した。さらに、一筆書き手法の対象を、3 次元空間に浮かんでいるノードだけでなく、3 次元アイコンも含めるようにした。これによって、アトムやゴール等の 3 次元アイコンや雛型の 3 次元アイコンの上をマウスの右ボタンでドラッグすることによって、ドラッグした軌跡と交わった 3 次元アイコンから、対応するノードを生成することが出来る。

一筆書き手法によって、プログラマはエッジの結線とノードの生成の 2 つの操作を複数回同時に実行することが出来るようになっている。例えば、通常、n 個のノードを直列にエッジで結線する際には、3 次元アイコンに対してマウスの左ボタンによる Drag and Drop 操作を n 回繰り返すことによってノードを n つ生成し、ノード間をマウスの右ボタンで $(n-1)$ 回 Drag and Drop しなければならないので、合計で $n+(n-1) = 2n-1$ 回 Drag and Drop 操作を繰り返さなければならない。しかし、一筆書き手法を用いるとノードの生成からエッジの結線の操作がたった 1 回の Drag and Drop 操作で実現出来る (図 7.3)。

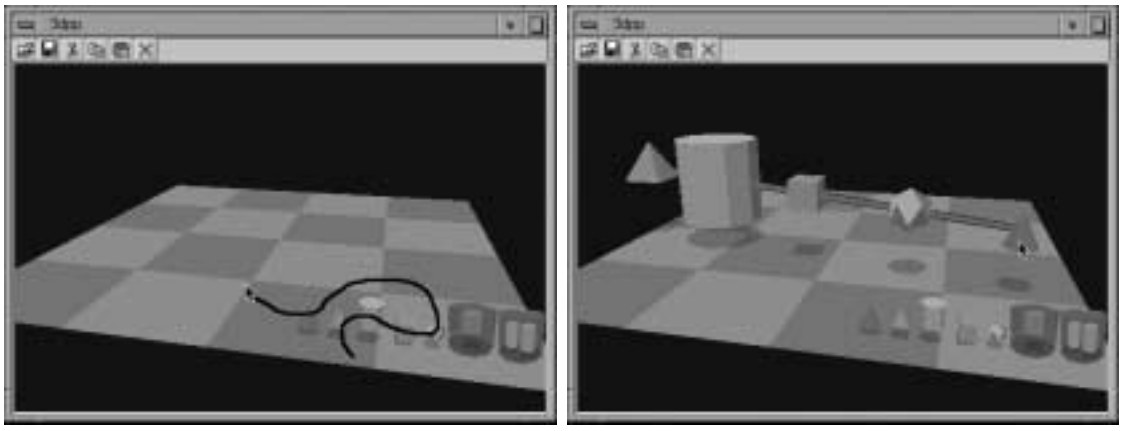


図 7.3: 一筆書き手法

図 7.3 の左側は、マウスの右ボタンを押しながらドラッグしている状態である。マウスカーソルが移動した軌跡が描画されている。図 7.3 の右側は、マウスの右ボタンをリリースした後の状態である。4 つのノード間が、一度の操作によってエッジによって結線されている。マウスの右ボタンはもともとエッジによる結線をする機能が割り当てられていたが、一筆書き手法と統合して、同時に使用することが出来るようになっている。つまり、単純に 2 つのノード間をエッジによって結線するという作業は、一筆書き手法の特殊な例として捉えることが出来る。

7.5 テキストを利用

7.1 節 (3) で述べたように、これから作成しようとしているプログラムとよく似ているプログラムが既に存在している場合、既存のプログラムが記述してあるファイルをロードするか、またはコピーして再利用することは、プログラムを白紙の状態から記述するよりも遥かに手間が少ない方法である。従来の VPS の多くで、テキスト形式で保存されたファイルをロードし、視覚表現に変換して再利用する機能が実装されている。

しかし、実際に既存のプログラムを再利用して新たなプログラムを作成する場合、必要となるのは既存のプログラムの一部分のみである場合が多い。にも関わらず不必要な部分も含めて一括にファイルからロードするのは効率が悪く、ロードした後もビジュアルプログラムの中から再利用したい必要な部分を新たに探し出すのにも苦労してしまう、という問題がある。

我々は、通常の (emacs 等の) テキストエディタで開かれた GHC のテキストのプログラムの、再利用したい部分をコピーし、“3D-PP” のメニューバーのペーストボタンを押すことによって直接 “3D-PP” のプログラム表示領域にビジュアルプログラムとして表示し、再利用することが出来る、部分ロードの機能を実装した (図 7.4)。

“3D-PP” の「ビジュアル・エディタ部」において、図 7.4 右側の図の上部の「Cut the Selection」「Copy the Selection」「Paste the Selection」の各ボタンには、ビジュアルプログラムに対してそれぞれ「選択されたノードの Cut」「選択されたノードの Copy」「Cut や Copy 操作によってクリップボードに転送されたノードを、今あるビジュアルプログラムに Paste」する機能が実装されている。

図 7.4 において、 GHC のテキストプログラムを “3D-PP” の「ビジュアル・エディタ部」へ部分ロードする際には、まずテキストエディタ上で部分ロードしたいテキストコードに対して通常のコピー操作を行う。次に、コピーされた GHC のテキストプログラムを “3D-PP” の「ビジュアル・エディタ部」上へペーストする際には、図 7.4

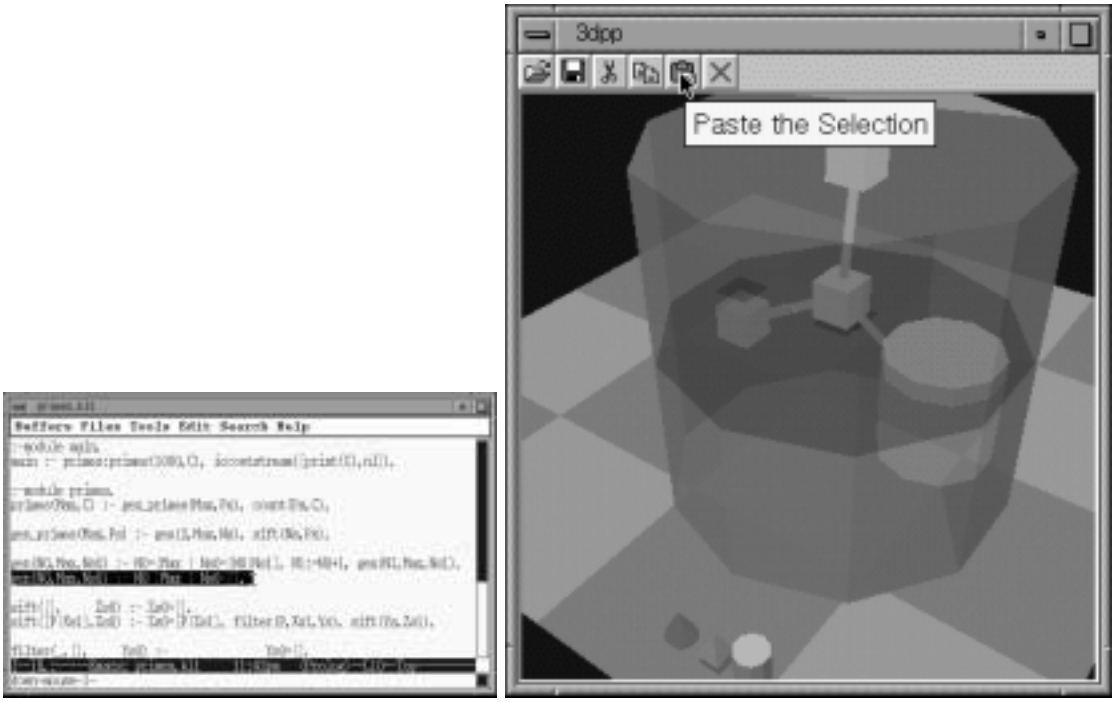


図 7.4: テキストプログラムをコピーし、ビジュアルプログラムをペーストする

右側の図の上部の「Paste the Selection」ボタンをクリックすることによって行われる。実際には、コピーされた部分の GHC のテキストコードを解析し、視覚表現に変換することによって “3D-PP” のプログラム表示領域にビジュアルプログラムとして表示している。

7.6 “3D-PP” におけるビジュアルプログラムの作成例

これまでに述べた手法を用いて、実際に “3D-PP” において append のプログラムを記述する例を示す。 append のプログラムを GHC で記述すると、

```
append([],In2,Out) :- Out=In2.
append([Msg|In1],In2,Out) :- Out=[Msg|OutTail],
        append(In1,In2,OutTail).
```

で表現される。これは、述語 append の、第 1 引数と第 2 引数に与えられた 2 つのリストを結合して、第 3 引数に格納するプログラムである。

まず、一筆書き手法を用いて、 In2, OutTail, append() をそれぞれ生成する (図 7.5)。図 7.5 の左側で、マウスの軌跡に重なった 3 次元アイコンから 3 つのノードが生成され、図 7.5 の右側のような状態になる。

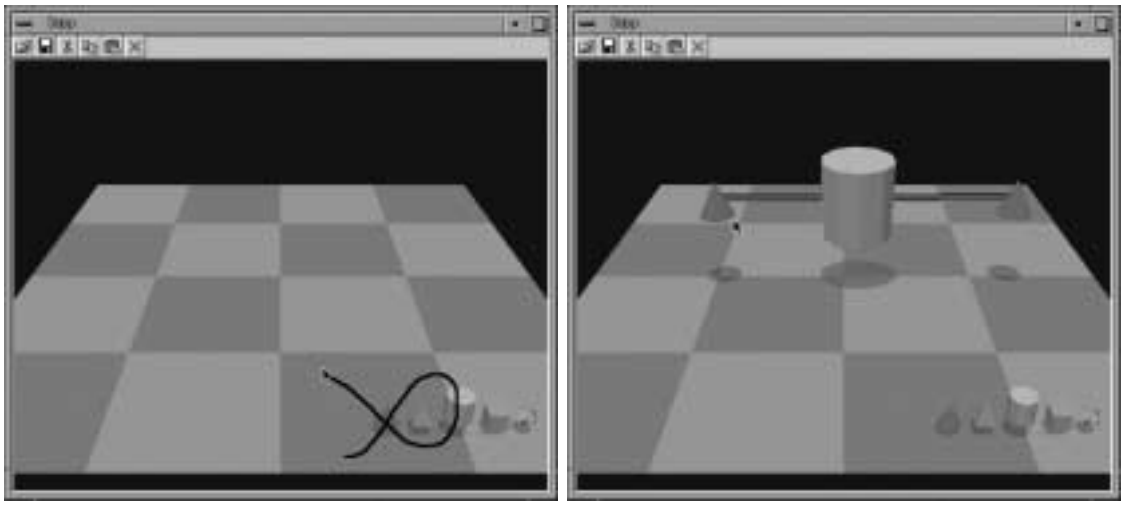


図 7.5: 一筆書き手法を用いて、 In2, OutTail, append() を生成

次に、雛型の 3 次元アイコンを用いてゴールを 1 つ生成し (図 7.6 左側)、 append と名前を付ける。そして、図 7.5 右側の要素を 1 つの定義節に Drag and Drop する (図 7.6 右側)。

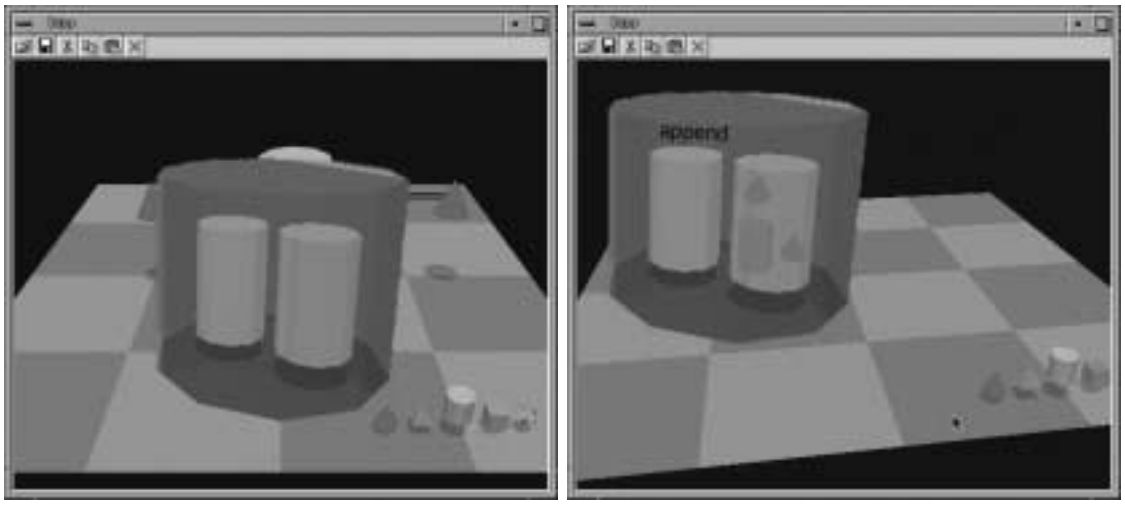


図 7.6: 雛型の 3 次元アイコンを用いてゴールを生成

次に、テキストエディタを用いて、他の GHC のテキストプログラムの中から

```
sift([],Zs0,Zs1) :- Zs1=Zs0.
```

という部分だけをコピーし、 “3D-PP” にペーストする。これは append プログラムの、

```
append([],In2,Out) :- Out=In2.
```

の部分と似た形をしているので、 sift という名前を append に変えて、このノードを定義節として入れ替える (図 7.7)。これで append のプログラムが完成する。

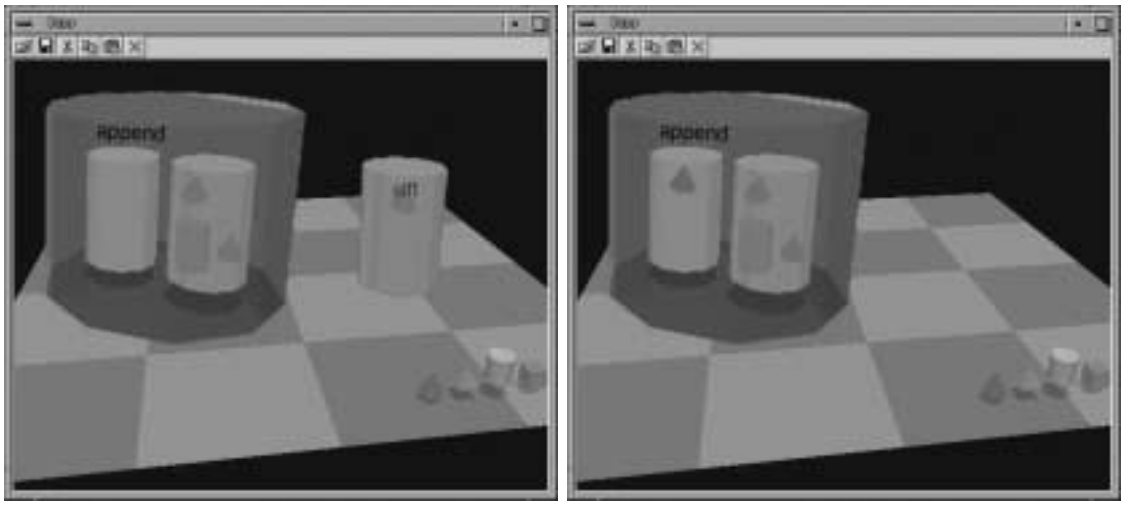


図 7.7: テキストプログラムの一部を再利用

この例で示したように、プログラマはいちいち全てのビジュアルプログラムを記述しなくても、上で述べたような手法を用いれば、効率的な入力を行うことが出来る。

今後は、ユーザ支援の手段として、 focus+context を考慮した fisheye views 等のズーミング・インタフェースを実装し、さらなるスケーラビリティの改善を図る予定である。

7.7 “3D-PP” に関連する研究

PrologSpace [34] や VisuaLinda [13], Cube [19] は 3 次元表示に基づく VPS である。 PrologSpace は論理型言語 Prolog をベースにしている。 VisuaLinda は並列言語 Linda をベースにしている。 Cube は並列論理型言語をベースにしている。

PrologSpace は VisualProlog を用いて実装されている VPS である。 VisualProlog は X ウィンドウ・ウィジェット・3 次元表示・アニメーション・オーディオ等がサポートされている Prolog の一種である。 PrologSpace は最低でも 4 つのサブウィンドウから構成されている。これに対し、 “3D-PP” は 1 つのウィンドウのみによって構成されている。

VisuaLinda は並列言語 Linda のプログラム実行の状態を視覚化する VPS であり、複数のプロセスの動作を時間軸に沿って視覚化されている。 VisuaLinda での視点の変更は 3 つのスクロールバーを操作することによって行われる。それぞれのスクロールバーは 3 次元空間の x 軸 y 軸 z 軸に対応している。スクロールバーを操作して視点を変更することは間接的な操作であるため、ユーザにとっては直観的ではない。これに対し、 “3D-PP” ではマウスで地面を操作することによって直接的に視点を変更す

ることが可能である。また、 VisuaLinda はプログラムの実行過程を視覚化するのみであり、プログラムの編集過程は視覚化しない。これに対し、 “3D-PP” ではプログラムの編集過程と実行過程の両方を視覚化することが出来る。

Cube はノードを直方体として視覚化し、データの移動をパイプとして視覚化している。また、 Cube はレンダリングの際に平行投影を用いている。 “3D-PP” では直接操作手法を用いて直観的にビジュアルプログラムを作成することが出来る。さらに、レンダリングの際には遠近法を用いた透視変換を用いており、地面や物体の影もプログラマに提示しているので、多くのノードが表示されていても、その間の位置関係を把握し易い。

第 8 章

まとめ

我々は、次世代のコンピューティング環境の 1 つとして 3 次元コンピューティング環境に着目し、3 次元モデリングツール “Claymore” と 3 次元ビジュアルプログラミングシステム “3D-PP” を開発した。

“Claymore” については、従来の直接操作手法を、付加情報を用いて強化した直接操作手法を提案し、実際に “Claymore” に実装した。また “Claymore” に新たに追加した機能についても述べた。

また、“3D-PP” については、従来の Drag and Drop 操作を拡張することによって、3 次元環境に親和性のある操作系を実現した。さらに、プログラマのビジュアルプログラム作成を支援するような機能を “3D-PP” に実装し、実際に例を示してプログラマがビジュアルプログラムを効率的に入力出来ることを示した。

謝辞

本研究を進めるにあたり終始丁寧に指導して下さいました田中二郎教授に心より感謝いたします。また筑波大学 電子・情報工学系 田中研究室のみなさんからはゼミ等を通して貴重なコメントを頂きました。特に Claymore に関するプロジェクトのメンバーの光延秀樹さん、神谷誠さん、青木裕伸さん、また、 3D-PP の研究に共同で取り組んだ 3D-PP グループのメンバーである宮下貴史さん、小川徹さん、中須正人さん、永田丈士さん、劉学軍さん、また昨年度の 3D-PP グループのメンバーである宮城幸司さん、遠藤浩通さん、神谷誠さんとはシステムの研究・開発にあたり有益な議論をしました。ここに感謝の意を表します。

参考文献

[1] Akihiro Baba, Jiro Tanaka, Eviss: a Visual System Having a Spatial Parser Generator, In *Proceedings of Asia Pacific Computer Human Interaction 1998 (APCHI98)*, pp.158–164, July, 1998.

[2] 馬場 昭宏, 田中 二郎: Spatial Parser Generator を持ったビジュアルシステム, 情報処理学会論文誌, Vol.39, No.5, pp.1385–1394, 1998.

[3] Margaret Burnett et al: Scaling Up Visual Programming Languages, *IEEE Computer*, Vol.28 No.3, pp.45–54, March 1995.

[4] Michael Chen, S. Joy Mountford, Abigail Sellen: A Study in Interactive 3-D Rotation Using 2-D Control Devices, In *ACM SIGGRAPH Computer Graphics*, pp.121–129, August, 1988.

[5] P. T. Cox, F. R. Giles and T. Pietrzykowski: Prograph: A Step towards Liberating Programming from Textual Conditioning, *1989 IEEE Workshop on Visual Languages*, Rome, pp.150–156, 1989.

[6] James M. Hebert: *LIGHTWAVE 3D USER GUIDE*, NewTek 社.

[7] D. A. Henderson and S. K. Card: Rooms: the Use of Multiple Virtual Workspaces to Reduce Space Contention in a Window-Based Graphical User Interface, *ACM Transactions on Graphics*, Vol.5, No.3, pp.211–243, July, 1986.

[8] Takeo Igarashi, Satoshi Matsuoka, Hidehiko Tanaka, Teddy: A Sketching Interface for 3D Freeform Design, In *ACM SIGGRAPH'99*, Los Angels, pp.409–416, 1999.

[9] Ken Kahn: ToonTalk — An Animated Programming Environment for Children, *Journal of Visual Languages and Computing*, pp.197–217, June, 1996.

[10] 神谷 誠: 3 次元ビジュアルプログラミングシステムにおけるドラッグ＆ドロップ手法の拡張, 平成 10 年度 筑波大学第三学群工学システム学類卒業研究論文, 1999.

[11] 北川 英志, 安田 孝美, 横井 茂樹, 鳥脇 純一郎: 仮想空間操作を利用した対話型手術シミュレーションシステムの基本機能の実現, 情報処理学会論文誌, Vol.37, No.6, 1996.

[12] KLIC 講習会テキスト -KL1 言語編 -, 財団法人 新世代コンピュータ技術開発機構 作成, 財団法人 日本情報処理開発協会開発研究室 改訂, 1995.

[13] Hideki Koike, Tetsuji Takada, Toshiyuki Masui: VisuaLinda: A Framework for Visualizing Parallel Linda Programs, In *Proceeding of 1997 IEEE Symposium on Visual Languages (VL'97)*, pp.174–180, 1997.

[14] Hideki Mitsunobu, Takashi Oshiba and Jiro Tanaka: Claymore: Augmented Direct Manipulation of Three-Dimensional Objects, In *Proceedings of Asia Pacific Computer Human Interaction 1998 (APCHI98)*, pp.210–216, July, 1998.

[15] 光延 秀樹: 直接操作に基づいた三次元モデラの構築, 平成 9 年度 筑波大学大学院修士課程理工学研究科修士論文, 1998.

[16] 光延 秀樹, 田中 二郎: 直接操作を用いた三次元モデリングツール "**Claymore**", 尾内理紀夫編, インタラクティブシステムとソフトウェア V, 日本ソフトウェア科学会 WISS'97, 近代科学社, pp.212, 1997.

[17] 宮城 幸司: 三次元ビジュアルプログラミング環境の構築, 平成 10 年度 筑波大学大学院修士課程理工学研究科修士論文, 1999.

[18] 宮城 幸司, 大芝 崇, 田中 二郎: 三次元ビジュアル・プログラミング・システム 3D-PP, 日本ソフトウェア科学会第 15 回大会論文集, pp.125–128, 1998.

[19] Marc A. Najork: Programming in Three Dimensions, *Journal of Visual Languages and Computing (1996) 7*, pp.219–242, 1996.

[20] Takashi Oshiba and Jiro Tanaka: "3D-PP": Visual Programming System with Three-Dimensional Representation, In *Proceeding of International Symposium on Future Software Technology (ISFST '99)*, pp.61–66, Nanjing, China, October 27th to 29th, 1999.

[21] Takashi Oshiba and Jiro Tanaka: "3D-PP": Three-Dimensional Visual Programming System, In *Proceeding of 1999 IEEE Symposium on Visual Languages (VL'99)*, pp.189–190, IEEE Computer Society Press, Tokyo, Japan, September 13th to 17th, 1999.

[22] Takashi Oshiba and Jiro Tanaka: Three-Dimensional Modeling Environment "Claymore" Based on Augmented Direct Manipulation Technique, In *Proceedings of The 8th International Conference on Human-Computer Interaction (HCI International '99)*, pp.1075–1079 (Volume 2), Munich, Germany, August 22th to 27th, 1999.

[23] 大芝 崇, 田中二郎: "Claymore": 付加情報によって強化された直接操作手法に基づく 3 次元モデリングツール, 日本ソフトウェア科学会 WISS'98, pp.195, 1998.

[24] 大芝 崇, 田中 二郎: 3 次元モデリングツール "**Claymore**": 付加情報によって強化された直接操作, 日本ソフトウェア科学会第 15 回大会論文集, pp.161–164, 1998.

[25] 大芝 崇: 3 次元物体の直接操作に関する研究, 平成 9 年度 筑波大学第三学群情報学類卒業研究論文, 1998.

[26] 大芝 崇, 光延 秀樹, 田中 二郎: 3 次元仮想空間への直接操作, 日本ソフトウェア科学会第 14 回大会論文集, pp.73–76, 1997.

[27] 暦本 純一: InformationCube: 半透明表示を用いた 3 次元情報視覚化技法, 竹内彰一編, インタラクティブシステムとソフトウェア I, 日本ソフトウェア科学会 WISS'93, 近代科学社, pp.1–8, 1993.

[28] Ben Shneiderman: Direct Manipulation: A Step Beyond Programming Languages, *IEEE Computer*, Vol.16, No.8, pp.57–69, 1983.

[29] 高田 哲司, 小池 英樹: VisuaLinda: 並列言語 Linda のプログラムの実行状態の 3 次元視覚化, 竹内彰一編, インタラクティブシステムとソフトウェア II, 日本ソフトウェア科学会 WISS'94, 近代科学社, pp.215–223, 1994.

[30] Jiro Tanaka, Hideki Mitsunobu and Takashi Oshiba: Claymore: Three-Dimensional Modeling Tool, In *Proceedings of The 20th International Conference on Software Engineering (ICSE98)*, pp.67–72 (Volume II), April, 1998.

[31] Bruce H. Thomas: Warping to Enhance 3D User Interface, In *Proceedings of Asia Pacific Computer Human Interaction 1998 (APCHI98)*, pp.169–174, July, 1998.

[32] Masashi Toyoda, Buntarou Shizuki, Shin Takahashi, Satoshi Matsuoka and Etsuya Shibayama: Supporting Design Patterns in a Visual Parallel Dataflow Programming Environment, In *Proc. 1997 IEEE Symposium on Visual Languages*, 1997.

[33] Kazunori Ueda: Guarded Horn Clauses, *ICOT Technical Report*, TR-103, Institute for New Generation Computer Technology, 1985.

[34] Masoud Yazdani and Lindsey Ford: Reducing the Cognitive Requirements of Visual Programming, In *Proceeding of 1996 IEEE Symposium on Visual Languages (VL'96)*, pp.225–262, 1996.

[35] 米川 和利, 小堀 研一, 久津 輪敏郎: 空間分割モデルを用いた形状モデラ, 情報処理学会論文誌, Vol.37, No.1, 1996.

[36] 吉川和宏, 正健太郎, 松嶋祥文, 瀧和男: 並列オブジェクトモデルを用いた実時間三次元アニメーション生成システムの実装, In *Proceedings of JSPP'98*, pp.119–126, June, 1998. l

[37] Robert C. Zeleznik, Kenneth P. Herndon and John F. Hughes: SKETCH: An Interface for Sketching 3D Scenes, In *ACM SIGGRAPH Computer Graphics*, pp.163–170, August, 1996.

付録 A

システムのソースコード

“Claymore” と “3D-PP” のソースコードを付録として添付する.

A.1 “Claymore”

glview.cc

```
// $Id: glview.cc,v 1.2 1998/06/30 06:18:43 ohshiba Exp $

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <math.h>
#include <GL/glut.h>

#include "rmd.h"
#include "rmdfile.h"
#include "objfile.h"
#include "vrmlfile.h"
#include "rmddraw.h"
#include "normcalc.h"
#include "polycalc.h"
#include "material.h"
#include "screen.h"
#include "glview.h"
#include "select.h"
#include "matrix.h"
#include "cmr.h"
#include "slice.h"
#include "drawPlane.h"
#include "paint.h"
#include "lwofile.h" //1999 7/16
#define Z_BUFFER

#define MODEID_MOVE 0
#define MODEID_ROTATE 1
```

```
#define MODEID_SPLIT 2
#define MODEID_SCALE 3
#define MODEID_CREATE 4
#define MODEID_DELETE 5
#define MODEID_GROUP 6
#define MODEID_PAINT 7//1998年6月18日(木)追加
#define MODEID_SAVE 8
#define MODEID_LOAD 9
//#define MODEID_SAVE 7
//#define MODEID_LOAD 8
//#define MODEID_PAINT 9

#define MODE_WIDTH 48
#define MODE_HEIGHT 13

#define LOADSCALE 350
#define SAVESCALE 350 // LWAVE


char* modename[]={
 "   MOVE",
 "ROTATE",
 "    SPLIT",
 "  SCALE",
 "CREATE",
 " DELETE",
 " GROUP",
 "  PAINT",//1998年6月18日(木)追加
 "   SAVE",
 "   LOAD",
};

#define MENUID_QUIT 0
#define MENUID_RESET 1
#define MENUID_SAVE 2

//#define MENUID_ROTATE1 0x10
//#define MENUID_ROTATE2 0x11
#define MENUID_ROTATE3 0x12
//#define MENUID_TOUCH 0xXX
//#define MENUID_SELECT 0xXX
//#define MENUID_ZOOM 0xXX
#define MENUID_MOVE1 0x20
#define MENUID_MOVE2 0x21
#define MENUID_SPLIT1 0x30
#define MENUID_DELETE 0x40
#define MENUID_CREATE 0x50
#define MENUID_GROUP 0x60
#define MENUID_SCALE 0x70
#define MENUID_PAINT 0x80
```

```
#define GROUP_MAX 256

static GLuint gltop,glbound;
stmode* modelsp; // モデル群の先頭
static stmode* rmp; // 現在選択中のモデル
static int polynum=-1; // 現在選択中のポリゴン
static vector3d tnormal; // 現在選択中の法線
//static vector3d dirst1,dirst2; // 現在選択中の平面の方角
static vector3d targetpos; // 現在選択中の座標
static vector3d targetpos0,targetpos1,tnormal0;
static vector3d toffset; // 変形量
static int taxis; // 変形軸

static int moemoeflag=0;
static int mousemode=MENUID_MOVE1,mmd=MENUID_MOVE1; // 左ボタンの動作種類
static int flagMouseL=0; // マウス左ボタンの状態 (NZ: 押されている)
static vector2i mousest; // マウス左クリックを開始した座標
static vector2i mousenw; // 現在のマウス座標 (flagMouseL!=0 時のみ有効)

static int flagDisplay=1; // 描画許可フラグ (NZ: 描画許可)
static int clickcount=0; // クリック判定用カウンタ

static vector3d posist; // 開始座標 (SPLIT1,CREATE)
//static vector3d splitpos; // 切断座標 (SPLIT1)
static matrix16d matrixst; // 開始回転行列 (ROTATE3,MOVE1)
static vector3d vectorst; // 開始モデル座標 (MOVE1)
static vector2f spinst; // 開始回転角度 (MOVE1/FLOOR)
static vector2f spin={0.0,0.0}; // 回転角 (MOVE1/FLOOR)
static matrix16d matrixc; // 物体生成行列 (CREATE,SCALE)
static vector3d vectorc; // 物体生成位置 (CREATE)
static double scalec; // 物体生成拡大率 (CREATE)

// ■文字表示
void StringDraw(int x, int y, char* p){
 int i;

 glRasterPos2i(x,y);
 i=strlen(p);
 while(i--) glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_10,*p++);
}

void StringDraw2(int x, int y, char* p){
 int i;

 glRasterPos2i(x,y);
 i=strlen(p);
 while(i--) glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24,*p++);
}

void SetModelView(stmode* mp){
 glMatrixMode(GL_MODELVIEW);
```

```
 glLoadIdentity();
 glRotatef(VROT,1.0,0.0,0.0);
 glTranslated(modelsp->vector.x,modelsp->vector.y,modelsp->vector.z);
 glMultMatrixd((GLdouble*)&modelsp->matrix);
 if(mp!=modelsp){
  glTranslated(mp->vector.x,mp->vector.y,mp->vector.z);
  glMultMatrixd((GLdouble*)&mp->matrix);
 }
}

void SetPureModelView(stmode* mp){
 glMatrixMode(GL_MODELVIEW);
 glLoadIdentity();
 if(mp!=modelsp){
  glTranslated(mp->vector.x,mp->vector.y,mp->vector.z);
  glMultMatrixd((GLdouble*)&mp->matrix);
 }
}

void SetModelShadowView(stmode* mp){
/*
 if(mp!=modelsp){
*/
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  glRotatef(VROT,1.0,0.0,0.0);
  glTranslated(-modelsp->vector.x,-1.0-modelsp->vector.y,-modelsp->vector.z);
  glMultMatrixd((GLdouble*)&modelsp->matrix);
  glScalef(1.0,0.0,1.0);
  glTranslated(mp->vector.x,0.0,mp->vector.z);
  glMultMatrixd((GLdouble*)&mp->matrix);
/*
 } else {
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  glRotatef(VROT,1.0,0.0,0.0);
// glTranslated(mp->vector.x,mp->vector.y,mp->vector.z);
  glMultMatrixd((GLdouble*)&mp->matrix);
 }
*/
}

//■モデル表示準備
void ModelPrepare(stmode* mp){
/*
 glNewList(modelsp-mp+gltop,GL_COMPILE);
 DrawRmd(mp,1);
 glEndList();
 glNewList(modelsp-mp+gltop+MODE_MAX,GL_COMPILE);
 DrawRmd(mp,0);
 glEndList();
```

```
*/
 glNewList(modelsp-mp+gltop,GL_COMPILE);
 DrawRmd(mp,0);
 glEndList();
}

//■モデル表示
void ModelDraw(stmode* mp){
 SetModelView(mp);
 glCallList(modelsp-mp+gltop);
}

//■影表示
void ShadowDraw(stmode* mp){
 SetModelShadowView(mp);
 glCallList(modelsp-mp+gltop);
// glCallList(modelsp-mp+gltop+MODE_MAX);
}

//■グループ化の情報を計算する
//・g: グループ番号 (不明の場合は 0 を指定する→全てのグループを計算する)
void SetGroup(int g){
 int i;
 stmode* mp;

 int gbuf[GROUP_MAX];
 for(i=1;i<GROUP_MAX;i++) gbuf[i]=FALSE;

 if(g){
  gbuf[g]=TRUE;
 } else {
   //for(mp=modelsp+10,i=10;i<MODE_MAX;i++,mp++)
  for(mp=modelsp+MODEL_OFFSET,i=MODEL_OFFSET;i<MODE_MAX;i++,mp++)
   if(mp->active)
    if(mp->group) gbuf[mp->group]=TRUE;
 }

 vector3d min,max;
 int j;
 for(j=1;j<GROUP_MAX;j++) if(gbuf[j]){
  int first=1;
  //for(mp=modelsp+10,i=10;i<MODE_MAX;i++,mp++) if(mp->active && mp->group==j){
  for(mp=modelsp+MODEL_OFFSET,i=MODEL_OFFSET;i<MODE_MAX;i++,mp++) if(mp->active && mp->group==j){
   if(first){
    min=mp->min;
    max=mp->max;
    first=0;
   } else {
    if(min.x>mp->min.x) min.x=mp->min.x;
    if(max.x<mp->max.x) max.x=mp->max.x;
    if(min.y>mp->min.y) min.y=mp->min.y;
```

```
    if(max.y<mp->max.y) max.y=mp->max.y;
    if(min.z>mp->min.z) min.z=mp->min.z;
    if(max.z<mp->max.z) max.z=mp->max.z;
   }
  }
#define OVER 15
  min.x-=OVER; min.y-=OVER; min.z-=OVER;
  max.x+=OVER; max.y+=OVER; max.z+=OVER;
#undef OVER
  //for(mp=modelsp+10,i=10;i<MODE_MAX;i++,mp++) if(mp->active && mp->group==j){
  for(mp=modelsp+MODEL_OFFSET,i=MODEL_OFFSET;i<MODE_MAX;i++,mp++) if(mp->active && mp->group==j){
   mp->center.x=(min.x+max.x)/2;
   mp->center.y=(min.y+max.y)/2;
   mp->center.z=(min.z+max.z)/2;
   mp->gmin=min;
   mp->gmax=max;
  }
 }
}

// ■バウンディングボックスの座標を計算する
void CalcBoundingBox(stmode* mp){
 vector3d min,max,v0,v1;
 matrix16d vmatrix;
 int i;

 SetPureModelView(mp);
 glGetDoublev(GL_MODELVIEW_MATRIX,(GLdouble*)&vmatrix);

 v0=*((vector3d*)&mp->vert[0]);
 CalcMatrixV((GLdouble*)&vmatrix,&v0,&v1);
 min=max=v1;
 for(i=1;i<mp->vertn;i++){
  v0=*((vector3d*)&mp->vert[i]);
  CalcMatrixV((GLdouble*)&vmatrix,&v0,&v1);
  if(min.x>v1.x) min.x=v1.x;
  if(max.x<v1.x) max.x=v1.x;
  if(min.y>v1.y) min.y=v1.y;
  if(max.y<v1.y) max.y=v1.y;
  if(min.z>v1.z) min.z=v1.z;
  if(max.z<v1.z) max.z=v1.z;
 }
#define OVER 6
 min.x-=OVER; min.y-=OVER; min.z-=OVER;
 max.x+=OVER; max.y+=OVER; max.z+=OVER;
#undef OVER

 mp->min=min;
 mp->max=max;

 if(mp->group) SetGroup(mp->group);
```

```
}

//■バウンディングボックスを表示する
void DrawBoundingBox(vector3d* min, vector3d* max){
 glPushMatrix();
 glTranslated((max->x+min->x)/2.0,(max->y+min->y)/2.0,(max->z+min->z)/2.0);
 glScaled((max->x-min->x)/2.0,(max->y-min->y)/2.0,(max->z-min->z)/2.0);

 glDepthMask(GL_FALSE);
 glDisable(GL_LIGHTING);
 glColor4ub(30,20,255,50);
 glCallList(glbound);

 glPolygonMode(GL_FRONT,GL_LINE);
 glPolygonMode(GL_BACK,GL_LINE);
 glDisable(GL_CULL_FACE);
 glColor4ub(60,40,255,150);
 glCallList(glbound);

 glDisable(GL_DEPTH_TEST);
 glColor4ub(60,40,255,30);
 glCallList(glbound);
 glEnable(GL_DEPTH_TEST);

 glEnable(GL_CULL_FACE);
 glPolygonMode(GL_FRONT,GL_FILL);
 glPolygonMode(GL_BACK,GL_FILL);
 glEnable(GL_LIGHTING);
 glDepthMask(GL_TRUE);

 glPopMatrix();
}

//■バウンディングボックスを表示する
void DrawBoundingBox2(vector3d* min, vector3d* max){
 glPushMatrix();
// glTranslated(toffset.x/2,toffset.y/2,toffset.z/2);
 glMultMatrixd((GLdouble*)&matrixc);
 glTranslated((max->x+min->x)/2.0,(max->y+min->y)/2.0,(max->z+min->z)/2.0);
 glScaled((max->x-min->x)/2.0,(max->y-min->y)/2.0,(max->z-min->z)/2.0);

 glDepthMask(GL_FALSE);
 glDisable(GL_LIGHTING);
 glColor4ub(255,20,30,50);
 glCallList(glbound);

 glPolygonMode(GL_FRONT,GL_LINE);
 glPolygonMode(GL_BACK,GL_LINE);
 glDisable(GL_CULL_FACE);
 glColor4ub(255,40,60,150);
 glCallList(glbound);
```

```
 glDisable(GL_DEPTH_TEST);
 glColor4ub(255,40,60,30);
 glCallList(glbound);
 glEnable(GL_DEPTH_TEST);

 glEnable(GL_CULL_FACE);
 glPolygonMode(GL_FRONT,GL_FILL);
 glPolygonMode(GL_BACK,GL_FILL);
 glEnable(GL_LIGHTING);
 glDepthMask(GL_TRUE);

 glPopMatrix();
}

//■情報表示
void InformationDraw(){
 char buf[128];

#ifdef Z_BUFFER
 glDisable(GL_DEPTH_TEST);
 glDepthMask(GL_FALSE);
#endif

 //移動メーター
 if(flagMouseL) if(mmd==MENUID_MOVE1){
  glMatrixMode(GL_MODELVIEW);
  glPushMatrix();//● InformationDraw,Push-Pop ラベル a ↓
  glLoadIdentity();
  glMatrixMode(GL_PROJECTION);
  glPushMatrix();//● InformationDraw,Push-Pop ラベル aa ↓
  glLoadIdentity();
  gluOrtho2D(0,scr.width,0,scr.height);
  glScalef(1,-1,1);
  glTranslatef(0,-scr.height,0);
  glDisable(GL_LIGHTING);

  if(rmp==modelsp){
   glBegin(GL_LINES);
   glColor4ub(80,60,255,100);
   glVertex3i(mousest.x-10000,mousest.y,0);
   glVertex3i(mousest.x+10000,mousest.y,0);
   glVertex3i(mousenw.x,mousest.y-10000,0);
   glVertex3i(mousenw.x,mousest.y+10000,0);
   glColor4ub(80,255,100,100);
   glVertex3i(mousest.x,mousest.y,0);
   glVertex3i(mousenw.x,mousest.y,0);
   glVertex3i(mousenw.x,mousest.y,0);
   glVertex3i(mousenw.x,mousenw.y,0);
   glEnd();
  } else {
```

```
 glColor4ub(80,60,255,160);
 glBegin(GL_LINES);
 glVertex3i(mousest.x,mousest.y,0); //glRasterPos2iv(&mousest);
 glVertex3i(mousenw.x,mousenw.y,0); //glRasterPos2iv(&mousenw);
 glEnd();
}

{
GLfloat r,t;
glColor4ub(255,60,100,160);
glBegin(GL_LINE_LOOP);
r=6;
for(t=0.0;t<PI*2.0;t+=PI*2.0/20.0)
 glVertex3i(mousenw.x+cos(t)*r,mousenw.y+sin(t)*r,0.0);
glEnd();
glBegin(GL_LINES);
r=10;
for(t=0.0;t<PI*2.0;t+=PI*2.0/16.0)
 glVertex3i(mousest.x+cos(t)*r,mousest.y+sin(t)*r,0.0);
glEnd();
}

glColor3ub(255,255,255);
glEnable(GL_LIGHTING);
glPopMatrix();// ● InformationDraw,Push-Pop ラベル aa ↑
glMatrixMode(GL_MODELVIEW);
glPopMatrix();// ● InformationDraw,Push-Pop ラベル a ↑
}

// 回転メーター
if(flagMouseL) if(mmd==MENUID_ROTATE3){
glMatrixMode(GL_MODELVIEW);
glPushMatrix();// ● InformationDraw,Push-Pop ラベル b ↓
glLoadIdentity();
glMatrixMode(GL_PROJECTION);
glPushMatrix();// ● InformationDraw,Push-Pop ラベル bb ↓
glLoadIdentity();
gluOrtho2D(0,scr.width,0,scr.height);
glScalef(1,-1,1);
glTranslatef(0,-scr.height,0);
glDisable(GL_LIGHTING);

{
GLfloat r,t;
glColor4ub(255,60,100,160);
glBegin(GL_LINE_LOOP);
r=6;
for(t=0.0;t<PI*2.0;t+=PI*2.0/20.0)
 glVertex3i(mousenw.x+cos(t)*r,mousenw.y+sin(t)*r,0.0);
glEnd();
glBegin(GL_LINES);
```

```
  r=10;
  for(t=0.0;t<PI*2.0;t+=PI*2.0/16.0)
   glVertex3i(mousest.x+cos(t)*r,mousest.y+sin(t)*r,0.0);
  glEnd();
  }

  /*
  {
  int wwx,wwy;
  int wwwx,wwwy;
  wwx=mousenw.x-mousest.x; wwy=mousenw.y-mousest.y;
  if(wwx || wwy){
  glColor4ub(80,60,255,160);
  glBegin(GL_LINES);
  glVertex3i(mousest.x,mousest.y,0); //glRasterPos2iv(&mousest);
  glVertex3i(mousenw.x,mousenw.y,0); //glRasterPos2iv(&mousenw);
  //  glVertex3i(mousest.x+wwx*1000,mousest.y+wwy*1000,0);
  //  glVertex3i(mousest.x-wwx*1000,mousest.y-wwy*1000,0);
  glEnd();
  wwwx=wwx*cos(PI/2.0)+wwy*sin(PI/2.0);
  wwwy=-wwx*sin(PI/2.0)+wwy*cos(PI/2.0);
  glColor4ub(255,60,100,100);
  glBegin(GL_LINES);
  glVertex3i(mousest.x-wwwx*1000,mousest.y-wwwy*1000,0);
  glVertex3i(mousest.x+wwwx*1000,mousest.y+wwwy*1000,0);
  glEnd();
  }
  }
  */
  {
  GLfloat r,t;
  int a;
  a=mousenw.x-mousest.x; r=a*a;
  a=mousenw.y-mousest.y; r+=a*a;
  r=sqrt(r);
  glColor4ub(40,160,60,128);
  glBegin(GL_LINES);
  for(t=0.0;t<PI*2.0;t+=PI*2.0/50.0)
   glVertex3i(mousest.x+cos(t)*r,mousest.y+sin(t)*r,0.0);
  glEnd();
  }

  glColor3ub(255,255,255);
  glEnable(GL_LIGHTING);
  glPopMatrix();// ● InformationDraw,Push-Pop ラベル bb ↑
  glMatrixMode(GL_MODELVIEW);
  glPopMatrix();// ● InformationDraw,Push-Pop ラベル b ↑
 }

#ifdef Z_BUFFER
 glEnable(GL_DEPTH_TEST);
```

```
 glDepthMask(GL_TRUE);
#endif

 //移動中の拘束面
 if(polynum>=0) if(flagMouseL) if(mmd==MENUID_MOVE1) if(moemoeflag){
  SetModelView(modelsp);
  glDisable(GL_LIGHTING);
  glDisable(GL_CULL_FACE);

  glDepthMask(GL_FALSE);
  glColor4ub(255,40,60,80);
  drawPlane(&tnormal0,&targetpos0,5000);
  glDepthMask(GL_TRUE);

#ifdef Z_BUFFER
  glDisable(GL_DEPTH_TEST);
#endif
  glColor4ub(255,40,60,20);
  drawPlane(&tnormal0,&targetpos0,5000);
#ifdef Z_BUFFER
  glEnable(GL_DEPTH_TEST);
#endif

  glEnable(GL_CULL_FACE);
  glColor3ub(255,255,255);
  glEnable(GL_LIGHTING);
 }

 //変形中の物体
 if(polynum>=0) if(flagMouseL) if(mmd==MENUID_SCALE){
  SetModelView(modelsp);
  DrawBoundingBox2(&rmp->min,&rmp->max);
 }

 //作成予定の物体
 if(polynum>=0) if(flagMouseL) if(mmd==MENUID_CREATE){
  glDepthMask(GL_FALSE);
  glDisable(GL_LIGHTING);
#ifdef Z_BUFFER
  glDisable(GL_DEPTH_TEST);
#endif


  glMatrixMode(GL_MODELVIEW);
  glPushMatrix();// ● InformationDraw,Push-Pop ラベル c ↓
  glLoadIdentity();
  glMatrixMode(GL_PROJECTION);
  glPushMatrix();// ● InformationDraw,Push-Pop ラベル cc ↓
  glLoadIdentity();
  gluOrtho2D(0,scr.width,0,scr.height);
  glScalef(1,-1,1);
```

```
  glTranslatef(0,-scr.height,0);

  glColor4ub(100,60,60,100);
  glBegin(GL_LINES);
  //  glVertex3dv((GLdouble*)&targetpos0);
  //  glVertex3dv((GLdouble*)&targetpos1);
  glVertex3i(mousest.x,mousest.y,0);
  glVertex3i(mousenw.x,mousenw.y,0);
  glEnd();

  glMatrixMode(GL_PROJECTION);
  glPopMatrix();// ● InformationDraw,Push-Pop ラベル cc ↓
  glMatrixMode(GL_MODELVIEW);
  glPopMatrix();// ● InformationDraw,Push-Pop ラベル c ↓


  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  glRotatef(VROT,1.0,0.0,0.0);
  glTranslated(modelsp->vector.x,modelsp->vector.y,modelsp->vector.z);
  glMultMatrixd((GLdouble*)&modelsp->matrix);
  glTranslated(vectorc.x,vectorc.y,vectorc.z);
  glScaled(scalec,scalec,scalec);
  glMultMatrixd((GLdouble*)&matrixc);
  glPolygonMode(GL_FRONT,GL_LINE);
  glPolygonMode(GL_BACK,GL_LINE);
  glDisable(GL_CULL_FACE);
  glColor4ub(60,40,255,150);
  glCallList(glbound);
  glEnable(GL_CULL_FACE);
  glPolygonMode(GL_FRONT,GL_FILL);
  glPolygonMode(GL_BACK,GL_FILL);
  glColor4ub(60,40,255,80);
  glCallList(glbound);

#ifdef Z_BUFFER
 glEnable(GL_DEPTH_TEST);
#endif

  glColor3ub(255,255,255);
  glEnable(GL_LIGHTING);
  glDepthMask(GL_TRUE);
 }

 //選択中の切断面
 if(polynum>=0) if(flagMouseL) if(mmd==MENUID_SPLIT1){
  SetModelView(modelsp);

  glDisable(GL_LIGHTING);
  glColor4ub(30,20,255,128);
  drawPlane(&tnormal,&targetpos,1000);
```

```
#ifdef Z_BUFFER
  glDisable(GL_DEPTH_TEST);
#endif
  glColor4ub(30,20,255,30);
  drawPlane(&tnormal,&targetpos,1000);
#ifdef Z_BUFFER
 glEnable(GL_DEPTH_TEST);
#endif

  glColor3ub(255,255,255);
  glEnable(GL_LIGHTING);
 }

#ifdef Z_BUFFER
 glDisable(GL_DEPTH_TEST);
 glDepthMask(GL_FALSE);
#endif

 /*
 //動作モード
 switch(mmd){
 case MENUID_MOVE1: strcpy(buf,"Move"); break;
 case MENUID_ROTATE3: strcpy(buf,"Rotate"); break;
 case MENUID_SPLIT1: strcpy(buf,"Split"); break;
 case MENUID_DELETE: strcpy(buf,"Delete"); break;
 case MENUID_CREATE: strcpy(buf,"Create"); break;
 case MENUID_GROUP: strcpy(buf,"Group"); break;
 case MENUID_SAVE: strcpy(buf,"Save as VRML"); break;
 }
 */

//printf("1:\n");
//DescribeStmode(rmp);

 //ボタン（四角の枠 ＋ 文字）の描画
 glDisable(GL_LIGHTING);
 glMatrixMode(GL_MODELVIEW);
 glLoadIdentity();
 glMatrixMode(GL_PROJECTION);
 glPushMatrix();//● InformationDraw,Push-Pop ラベル d ↓
 glLoadIdentity();
 gluOrtho2D(0,scr.width,0,scr.height);

 /*
 glColor4ub(0,40,0,200);
 StringDraw2(4,2,buf);
 glColor4ub(40,160,60,250);
 StringDraw2(3,3,buf);
 */
```

```
 int mode;
 switch(mousemode){
 case MENUID_MOVE1: mode=MODEID_MOVE; break;
 case MENUID_ROTATE3: mode=MODEID_ROTATE; break;
 case MENUID_SPLIT1: mode=MODEID_SPLIT; break;
 case MENUID_SCALE: mode=MODEID_SCALE; break;
 case MENUID_DELETE: mode=MODEID_DELETE; break;
 case MENUID_CREATE: mode=MODEID_CREATE; break;
 case MENUID_GROUP: mode=MODEID_GROUP; break;
 case MENUID_SAVE: mode=MODEID_SAVE; break;
 case MENUID_PAINT: mode=MODEID_PAINT; break;
 default: mode=MODEID_CREATE;
 }

 glTranslatef(0,scr.height,0);
 glScalef(1,-1,1);
 // for(int i=0;i<8;i++){
 for(int i=0;i<9;i++){
  int xx=i*MODE_WIDTH;

  glColor4ub(0,0,0,100);
  StringDraw(xx+4,8+5,modename[i]);
  glColor4ub(40,160,60,255);
  StringDraw(xx+3,8+4,modename[i]);

  if(i==mode)
   glColor4ub(160,40,60,200);
  else
   glColor4ub(40,160,60,200);
  glBegin(GL_LINE_LOOP);
  glVertex2i(xx+1,2);
  glVertex2i(xx+MODE_WIDTH-3,2);
  glVertex2i(xx+MODE_WIDTH-3,2+MODE_HEIGHT);
  glVertex2i(xx+1,2+MODE_HEIGHT);
  glEnd();
 }

 glColor3ub(255,255,255);
 glPopMatrix();
 glMatrixMode(GL_MODELVIEW);
 glEnable(GL_LIGHTING);

#ifdef Z_BUFFER
 glEnable(GL_DEPTH_TEST);
 glDepthMask(GL_TRUE);
#endif

 //選択中のポリゴン
 if(polynum>=0){
  vector3d v0,v1,v2;
  matrix16d vmatrix;
```

```
/*
  glDisable(GL_CULL_FACE);
  glPolygonMode(GL_FRONT,GL_LINE);
// glPolygonMode(GL_BACK,GL_LINE);
*/
  glDisable(GL_LIGHTING);

/*
  SetModelView(rmp);
  glGetDoublev(GL_MODELVIEW_MATRIX,(GLdouble*)&vmatrix);
  glLoadIdentity();
*/

  SetPureModelView(rmp);
  glGetDoublev(GL_MODELVIEW_MATRIX,(GLdouble*)&vmatrix);
  SetModelView(modelsp);

/*
  glColor4ub(30,20,255,128); //glColor3ub(0,160,0);
#ifdef Z_BUFFER
// glDisable(GL_DEPTH_TEST);
  glDepthMask(GL_FALSE);
#endif
  DrawPoly2(rmp,polynum);
#ifdef Z_BUFFER
// glEnable(GL_DEPTH_TEST);
 glDepthMask(GL_TRUE);
#endif
*/

#define NLENGTH 200.0
  v0=targetpos;
  v1=tnormal;
  v1.x*=NLENGTH; v1.x+=targetpos.x;
  v1.y*=NLENGTH; v1.y+=targetpos.y;
  v1.z*=NLENGTH; v1.z+=targetpos.z;

//  CalcMatrixV((GLdouble*)&vmatrix,&v1,&v2);
//  CalcMatrixV((GLdouble*)&vmatrix,&v0,&v1);
  v2=v1;
  v1=v0;

  glColor4ub(255,60,100,160);
  DrawVert(&v1);
// DrawVert(&v1);
  glBegin(GL_LINES);
  glVertex3dv((GLdouble*)&v1);
  glVertex3dv((GLdouble*)&v2);
  glEnd();
```

```
#ifdef Z_BUFFER
  glDisable(GL_DEPTH_TEST);
#endif
  glColor4ub(255,60,100,50);
  DrawVert(&v1);
  glBegin(GL_LINES);
  glVertex3dv((GLdouble*)&v1);
  glVertex3dv((GLdouble*)&v2);
  glEnd();
#ifdef Z_BUFFER
  glEnable(GL_DEPTH_TEST);
#endif

  /*
#ifdef Z_BUFFER
  glDisable(GL_DEPTH_TEST);
  glDepthMask(GL_FALSE);
#endif
  glLoadIdentity();

  sprintf(buf,"M%02d P%03d V(%+07.2f,%+07.2f,%+07.2f)",
  (rmp-modelsp),polynum,targetpos.x,targetpos.y,targetpos.z);
  glMatrixMode(GL_PROJECTION);
  glPushMatrix();
  glLoadIdentity();
  gluOrtho2D(0,scr.width,0,scr.height);
  // glScalef(1,-1,1); glTranslatef(0,-scr.height,0);
  glColor4ub(0,0,0,200);
  StringDraw(75,4,buf);
  glColor4ub(255,255,255,200);
  StringDraw(74,5,buf);
  glColor3ub(255,255,255);
  glPopMatrix();
  glMatrixMode(GL_MODELVIEW);
  */

  glEnable(GL_LIGHTING);
  //  glEnable(GL_CULL_FACE);
  //  glPolygonMode(GL_FRONT,GL_FILL);
  //// glPolygonMode(GL_BACK,GL_POINT);

#ifdef Z_BUFFER
 glEnable(GL_DEPTH_TEST);
 glDepthMask(GL_TRUE);
#endif
 }
}

//■ウインドウ内容の描画
void WindowDraw(){
 int i;
```

```
 stmode* mp;

//printf("WindowDraw0:\n");
//DescribeStmode(rmp);

#ifdef Z_BUFFER
 glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
#else
 glClear(GL_COLOR_BUFFER_BIT);
#endif
 for(mp=modelsp+1,i=1;i<MODE_MAX;i++,mp++) if(mp->active){
  ModelDraw(mp);
 }

 /*
 ModelDraw(modelsp+1); // ミニチュア板
 ModelDraw(modelsp+2); // ミニチュア球
 */

/*
 ModelDraw(modelsp+3);
 ModelDraw(modelsp+4);
 ModelDraw(modelsp+8);
 ModelDraw(modelsp+9);
 ModelDraw(modelsp+2); // 半透明
 ModelDraw(modelsp+1); // 半透明 // ミニチュア板
 ModelDraw(modelsp+7); // ミニチュア球
*/

 glDisable(GL_CULL_FACE);
 glPolygonMode(GL_FRONT,GL_FILL);
 glPolygonMode(GL_BACK,GL_FILL);
 ModelDraw(modelsp);
 glEnable(GL_CULL_FACE);

// SetModelView(modelsp);

{
 int gbuf[GROUP_MAX];
 for(i=1;i<GROUP_MAX;i++) gbuf[i]=TRUE;
 //for(mp=modelsp+10,i=10;i<MODE_MAX;i++,mp++) if(mp->active && mp->bound){
 for(mp=modelsp+MODEL_OFFSET,i=MODEL_OFFSET;i<MODE_MAX;i++,mp++) if(mp->active && mp->bound){
  if(mp->group){
   if(gbuf[mp->group]){
    gbuf[mp->group]=FALSE;
    DrawBoundingBox(&mp->gmin,&mp->gmax);
   }
  } else {
   DrawBoundingBox(&mp->min,&mp->max);
  }
 }
```

```
}

/*
{
 int gct=0;
 int gbuf[GROUP_MAX];
 for(i=1;i<GROUP_MAX;i++) gbuf[i]=0;
 for(mp=modelsp+10,i=10;i<MODE_MAX;i++,mp++) if(mp->active && mp->bound){
  if(mp->group){ gct++; gbuf[mp->group]++; }
  DrawBoundingBox(&mp->min,&mp->max);
 }
 if(gct){
  vector3d min,max;
  int j;
  for(j=1;j<GROUP_MAX;j++) if(gbuf[j]){
   int first=1;
   for(mp=modelsp+10,i=10;i<MODE_MAX;i++,mp++) if(mp->active && mp->group==j){
    if(first){
     min=mp->min;
     max=mp->max;
     first=0;
    } else {
     if(min.x>mp->min.x) min.x=mp->min.x;
     if(max.x<mp->max.x) max.x=mp->max.x;
     if(min.y>mp->min.y) min.y=mp->min.y;
     if(max.y<mp->max.y) max.y=mp->max.y;
     if(min.z>mp->min.z) min.z=mp->min.z;
     if(max.z<mp->max.z) max.z=mp->max.z;
    }
   }
#define OVER 15
   min.x-=OVER; min.y-=OVER; min.z-=OVER;
   max.x+=OVER; max.y+=OVER; max.z+=OVER;
#undef OVER
   DrawBoundingBox(&min,&max);
  }
 }
}
*/

 glDisable(GL_LIGHTING);
 glColor4ub(0,0,0,128);
 glDepthMask(GL_FALSE);
 for(mp=modelsp+1,i=1;i<MODE_MAX;i++,mp++) if(mp->active){
  ShadowDraw(mp);
 }
 glDepthMask(GL_TRUE);
 glColor3ub(255,255,255);
 glEnable(GL_LIGHTING);

 InformationDraw();
```

```
 glutSwapBuffers();
// glFlush();
}

//■ウインドウ変形時の処理
void WindowReshape(int x, int y){
if(opt_qv) printf("WindowReshape(%d,%d)\n",x,y);
 scr.width=x;
 scr.height=y;
 glViewport(0,0,x,y);
 glMatrixMode(GL_PROJECTION);
 glLoadIdentity();
 gluPerspective(scr.fov,(GLfloat)x/(GLfloat)y,ZNEAR,ZFAR);
 gluLookAt(0.0,0.0,-LOOKAT, 0.0,0.0,0.0, 0.0,-1.0,0.0);
}

//■一定時間おきに呼ばれる
void WindowIdle(){
}

//■アイコン化関連処理
void WindowVisible(int visible){
if(opt_qv) printf("WindowVisible(%d)\n",visible);
 glutIdleFunc(visible==GLUT_VISIBLE ? WindowIdle:NULL);
}

//■グループ化処理
int ScanFreeGroup(){
 static int groupno=0;
 groupno=(groupno%GROUP_MAX)+1;
 return groupno;
}

void ModelGroup(){
 stmode* mp;
 int i;
 int bct,gct;

 //グループ化の状況をチェック
 bct=0; gct=0;
 //for(mp=modelsp+10,i=10;i<MODE_MAX;mp++,i++){
 for(mp=modelsp+MODEL_OFFSET,i=MODEL_OFFSET;i<MODE_MAX;mp++,i++){
  if(mp->active && mp->bound){
   bct++;
   if(mp->group) gct++;
  }
 }
 if(bct==0) return; //該当グループが存在しなければ終了

 if(bct!=gct){
```

```
  //グループ化
  int g=ScanFreeGroup();
  //for(mp=modelsp+10,i=10;i<MODE_MAX;mp++,i++){
  for(mp=modelsp+MODEL_OFFSET,i=MODEL_OFFSET;i<MODE_MAX;mp++,i++){
   if(mp->active && mp->bound) mp->group=g;
  }
  SetGroup(g);
 } else {
  //グループ化解除
   //for(mp=modelsp+10,i=10;i<MODE_MAX;mp++,i++){
  for(mp=modelsp+MODEL_OFFSET,i=MODEL_OFFSET;i<MODE_MAX;mp++,i++){
   if(mp->active && mp->bound) mp->group=0;
  }
 }
}

void ModelDelete(){
 stmode* mp;
 int i;
 //for(mp=modelsp+10,i=10;i<MODE_MAX;i++,mp++){
 for(mp=modelsp+MODEL_OFFSET,i=MODEL_OFFSET;i<MODE_MAX;i++,mp++){
  if(mp->active && mp->bound){
   ModelFree(mp);
  }
 }
}

//■ウィンドウ内でマウスが動いた時の処理
void WindowPassiveMotion(short x, short y){
 polynum=CalcTouchPolygon(modelsp,x,y,&rmp);

 if(polynum>=0){
  vector3d v1;
  double ax,ay,az,bx,by,bz,n;
  int i,mx;

  mmd=mousemode;
  if(rmp==modelsp){
   mmd=MENUID_MOVE1;
  } else
  if(rmp==modelsp+1){
   mmd=MENUID_SPLIT1;
  } else
  if(rmp==modelsp+2){
   mmd=MENUID_GROUP;
  } else
  if(rmp==modelsp+3){
   mmd=MENUID_ROTATE3;
  } else
  if(rmp==modelsp+4){
   mmd=MENUID_DELETE;
```

```
} else
if(modelsp+7<=rmp && rmp<=modelsp+9){
 mmd=MENUID_CREATE;
}

// 論理的な情報隠蔽・物理的な情報隠蔽
// カラーリング物体を非表示
if (mousemode != MENUID_PAINT) {
  for (i=COLOR_OFFSET; i<=MODEL_OFFSET-1; i++)
    (modelsp+i)->active=0;
  glutPostRedisplay();
}
/*
if (mousemode != MENUID_PAINT)
  for (i=COLOR_OFFSET; i<=MODEL_OFFSET; i++) {
    (modelsp+i)->active=0;
    printf("%d ", i);
  }
printf("\n");
*/
/*
if (mousemode != MENUID_PAINT) {
  (modelsp+10)->active=0;
  (modelsp+11)->active=0;
  (modelsp+12)->active=0;
}
*/

// マウスカーソルが GUI ボタンの上に重なっている時にはマウスカーソルの形
// を「ゆび指してる手」に変える
//int cursor_shape = glutGet(GLUT_WINDOW_CURSOR);
if(y<2+MODE_HEIGHT+3)
  glutSetCursor(GLUT_CURSOR_INFO);
//else glutSetCursor(GLUT_CURSOR_LEFT_ARROW);
//else glutSetCursor(cursor_shape);
// 現在のモードによってマウスカーソルの形を変える
else
  switch (mousemode) {
  case MENUID_MOVE1:
    glutSetCursor(GLUT_CURSOR_INHERIT);
    break;
  case MENUID_ROTATE3:
    glutSetCursor(GLUT_CURSOR_CYCLE);
    break;
  case MENUID_SPLIT1:
    glutSetCursor(GLUT_CURSOR_LEFT_ARROW);
    break;
  case MENUID_SCALE:
    glutSetCursor(GLUT_CURSOR_CROSSHAIR);
    break;
  case MENUID_CREATE:
```

```
      glutSetCursor(GLUT_CURSOR_NONE);
      break;
    case MENUID_DELETE:
      glutSetCursor(GLUT_CURSOR_DESTROY);
      break;
    case MENUID_SAVE:
      glutSetCursor(GLUT_CURSOR_INFO);
      break;
    case MENUID_PAINT:
      glutSetCursor(GLUT_CURSOR_SPRAY);

      //論理的な情報隠蔽・物理的な情報隠蔽
      //カラーリング物体を表示
      for (i=COLOR_OFFSET; i<=MODEL_OFFSET-1; i++)
(modelsp+i)->active=1;
      glutPostRedisplay();
      /*
      for (i=COLOR_OFFSET; i<=MODEL_OFFSET; i++) {
(modelsp+i)->active=1;
printf("%d ", i);
      }
      printf("\n");
      */
      /*
      (modelsp+10)->active=1;
      (modelsp+11)->active=1;
      (modelsp+12)->active=1;
      */
      break;
    }

  CalcTouchPosition(rmp,polynum,x,y,&targetpos,&tnormal);

/*
  v1=*(vector3d*)(rmp->vert+rmp->poly[polynum].v[0]);
  dirst1=*(vector3d*)(rmp->vert+rmp->poly[polynum].v[1]);
  dirst1.x-=v1.x;
  dirst1.y-=v1.y;
  dirst1.z-=v1.z;
  n=sqrt(dirst1.x*dirst1.x+dirst1.y*dirst1.y+dirst1.z*dirst1.z);
//if(n!=0.0){
   dirst1.x/=n; dirst1.y/=n; dirst1.z/=n;
//}

  ax=dirst1.x; //vp[pp->v[2]].x-vp[pp->v[1]].x;
  ay=dirst1.y; //vp[pp->v[2]].y-vp[pp->v[1]].y;
  az=dirst1.z; //vp[pp->v[2]].z-vp[pp->v[1]].z;
  bx=-tnormal.x; //vp[pp->v[1]].x-vp[pp->v[0]].x;
  by=-tnormal.y; //vp[pp->v[1]].y-vp[pp->v[0]].y;
  bz=-tnormal.z; //vp[pp->v[1]].z-vp[pp->v[0]].z;
  dirst2.x=ay*bz-az*by;
```

```
  dirst2.y=az*bx-ax*bz;
  dirst2.z=ax*by-ay*bx;
*/

  //polynum=-1;
 }
if(opt_qv) printf("座標 (%f,%f,%f)\n",targetpos.x,targetpos.y,targetpos.z);
 if(flagDisplay) glutPostRedisplay();
}

//■ウィンドウ内でマウスがドラッグされた時の処理
void WindowMotion(short x, short y){
 clickcount++;
 mousenw.x=x; mousenw.y=y;
 if(flagMouseL) switch(mmd){
/*
 case MENUID_ROTATE1:{
  vector2f spn;
  spn.x=spinst.x-360.0*(x-mousest.x)/scr.width;
  if(spn.x>= 360.0) do spn.x-=360.0; while(spn.x>= 360.0); else
  if(spn.x<=-360.0) do spn.x+=360.0; while(spn.x<=-360.0);
  spn.y=spinst.y+360.0*(y-mousest.y)/scr.height;
  if(spn.y>= 360.0) do spn.y-=360.0; while(spn.y>= 360.0); else
  if(spn.y<=-360.0) do spn.y+=360.0; while(spn.y<=-360.0);
  if(spn.x!=spin.x || spn.y!=spin.y){
   spin=spn;
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
//glTranslated(rmp->vector.x,rmp->vector.y,rmp->vector.z);
   glRotatef(spin.x,0.0,1.0,0.0);
   glRotatef(spin.y,1.0,0.0,0.0);
   glGetDoublev(GL_MODELVIEW_MATRIX,(GLdouble*)&rmp->matrix);
   glutPostRedisplay();
  }
  break;
 }
*/
 case MENUID_ROTATE3:{
  int xx,yy;
  double t,r;
  xx=-(x-mousest.x); yy=-(y-mousest.y);
  t=atan2(yy,xx)*180.0/PI; r=sqrt(xx*xx+yy*yy);
if(opt_qv) printf("Range:%f Distance:%f\n",t,r);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();

//  if(rmp!=modelsp){
   matrix16d rrmatrix;
   CalcInvertMatrix((GLdouble*)&modelsp->matrix,(double*)&rrmatrix);
   glMultMatrixd((GLdouble*)&rrmatrix);
//// glTranslated(-modelsp->vector.x,-modelsp->vector.y,-modelsp->vector.z);
```

```
   glRotatef(-VROT,1.0,0.0,0.0);
//  } else {
//   glRotatef(-VROT,1.0,0.0,0.0);
//  }

  glRotatef(t,0.0,0.0,1.0);
  glRotatef(r,0.0,1.0,0.0);
  glRotatef(-t,0.0,0.0,1.0);

  /*
  glTranslated(
 -mp->vector.x+(mp->gmin.x+mp->gmax.x)/2,
 -mp->vector.y+(mp->gmin.y+mp->gmax.y)/2,
 -mp->vector.z+(mp->gmin.z+mp->gmax.z)/2);
  */

//if(rmp!=modelsp){
   glRotatef(VROT,1.0,0.0,0.0);
//// glTranslated(-modelsp->vector.x,-modelsp->vector.y,-modelsp->vector.z);
   glMultMatrixd((GLdouble*)&modelsp->matrix);
//} else {
// glRotatef(VROT,1.0,0.0,0.0);
//}

  glGetDoublev(GL_MODELVIEW_MATRIX,(GLdouble*)&rrmatrix);

{
 stmode* mp;
 int i;

 if(rmp->group==0){
  rmp->center=rmp->center0=rmp->vector;
  rmp->gmin=rmp->min;
  rmp->gmax=rmp->max;
 }

 //for(mp=modelsp+10,i=10;i<MODE_MAX;i++,mp++)
 if((rmp->group && mp->active && mp->group==rmp->group)
    || (rmp->group==0 && mp==rmp)){
 for(mp=modelsp+MODEL_OFFSET,i=MODEL_OFFSET;i<MODE_MAX;i++,mp++)
     if((rmp->group && mp->active && mp->group==rmp->group)
|| (rmp->group==0 && mp==rmp)){

  glPushMatrix();

  glMultMatrixd((GLdouble*)&matrixst);
//glGetDoublev(GL_MODELVIEW_MATRIX,(GLdouble*)&rmp->matrix);
//CalcBoundingBox(rmp);
  glGetDoublev(GL_MODELVIEW_MATRIX,(GLdouble*)&mp->matrix);
  CalcBoundingBox(mp);
```

```
  vector3d v0,v1;
  v0=mp->vector0;
  v0.x-=mp->center0.x;
  v0.y-=mp->center0.y;
  v0.z-=mp->center0.z;
  CalcMatrixN((GLdouble*)&rrmatrix,&v0,&v1);
  v1.x+=mp->center0.x;
  v1.y+=mp->center0.y;
  v1.z+=mp->center0.z;
  mp->vector=v1;

  glPopMatrix();
 }
}

  glutPostRedisplay();
  break;
 }

 case MENUID_SCALE:
 { //物体の拡大・変形
  vector3d v1;

  CalcCrossPosition(&tnormal0,&targetpos0,&matrixst,x,y,&v1);

  toffset.x=v1.x-targetpos1.x;
  toffset.y=v1.y-targetpos1.y;
  toffset.z=v1.z-targetpos1.z;

#define M11 0
#define M12 1
#define M13 2
#define M14 3
#define M21 4
#define M22 5
#define M23 6
#define M24 7
#define M31 8
#define M32 9
#define M33 10
#define M34 11
#define M41 12
#define M42 13
#define M43 14
#define M44 15

   matrixc.m[M11]=1;
   matrixc.m[M12]=0;
   matrixc.m[M13]=0;
   matrixc.m[M14]=0;
```

```
 matrixc.m[M21]=0;
 matrixc.m[M22]=1;
 matrixc.m[M23]=0;
 matrixc.m[M24]=0;

 matrixc.m[M31]=0;
 matrixc.m[M32]=0;
 matrixc.m[M33]=1;
 matrixc.m[M34]=0;

 matrixc.m[M41]=0;
 matrixc.m[M42]=0;
 matrixc.m[M43]=0;
 matrixc.m[M44]=1;

switch(taxis){
case 0:{
 double L=rmp->max.y-rmp->min.y;
 matrixc.m[M21]=toffset.x/L;
 matrixc.m[M22]=1+toffset.y/L;
 matrixc.m[M23]=toffset.z/L;

 matrixc.m[M41]=toffset.x/2;
 matrixc.m[M42]=toffset.y/2;
 matrixc.m[M43]=toffset.z/2;
 break;
}
case 1:{
 double L=rmp->min.y-rmp->max.y;
 matrixc.m[M21]=toffset.x/L;
 matrixc.m[M22]=1+toffset.y/L;
 matrixc.m[M23]=toffset.z/L;

 matrixc.m[M41]=toffset.x/2;
 matrixc.m[M42]=toffset.y/2;
 matrixc.m[M43]=toffset.z/2;
 break;
}
case 2:{
 double L=rmp->max.x-rmp->min.x;
 matrixc.m[M11]=1+toffset.x/L;
 matrixc.m[M12]=toffset.y/L;
 matrixc.m[M13]=toffset.z/L;

 matrixc.m[M41]=toffset.x/2;
 matrixc.m[M42]=toffset.y/2;
 matrixc.m[M43]=toffset.z/2;
 break;
}
case 3:{
 double L=rmp->min.x-rmp->max.x;
```

```
   matrixc.m[M11]=1+toffset.x/L;
   matrixc.m[M12]=toffset.y/L;
   matrixc.m[M13]=toffset.z/L;

   matrixc.m[M41]=toffset.x/2;
   matrixc.m[M42]=toffset.y/2;
   matrixc.m[M43]=toffset.z/2;
   break;
  }
  case 4:{
   double L=rmp->min.z-rmp->max.z;
   matrixc.m[M31]=toffset.x/L;
   matrixc.m[M32]=toffset.y/L;
   matrixc.m[M33]=1+toffset.z/L;

   matrixc.m[M41]=toffset.x/2;
   matrixc.m[M42]=toffset.y/2;
   matrixc.m[M43]=toffset.z/2;
   break;
  }
  case 5:{
   double L=rmp->max.z-rmp->min.z;
   matrixc.m[M31]=toffset.x/L;
   matrixc.m[M32]=toffset.y/L;
   matrixc.m[M33]=1+toffset.z/L;

   matrixc.m[M41]=toffset.x/2;
   matrixc.m[M42]=toffset.y/2;
   matrixc.m[M43]=toffset.z/2;
   break;
  }
  }

#undef M11
#undef M12
#undef M13
#undef M14
#undef M21
#undef M22
#undef M23
#undef M24
#undef M31
#undef M32
#undef M33
#undef M34
#undef M41
#undef M42
#undef M43
#undef M44

  glutPostRedisplay();
```

```
  break;
 }

 case MENUID_MOVE1:
 if(rmp==modelsp){ // 地面の移動 (視点の移動)
  vector2f spn;
  spn.x=spinst.x-360.0*(x-mousest.x)/scr.width;
/*
  if(spn.x>= 360.0) do spn.x-=360.0; while(spn.x>= 360.0); else
  if(spn.x<=-360.0) do spn.x+=360.0; while(spn.x<=-360.0);
*/
  spn.y=spinst.y+360.0*(y-mousest.y)/scr.height;
/*
  if(spn.y>= 360.0) do spn.y-=360.0; while(spn.y>= 360.0); else
  if(spn.y<=-360.0) do spn.y+=360.0; while(spn.y<=-360.0);
*/
  if(spn.x!=spin.x || spn.y!=spin.y){
   spin=spn;
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
   glTranslated(rmp->vector.x,rmp->vector.y,rmp->vector.z);
   glRotatef(spin.y,1.0,0.0,0.0);
   glRotatef(spin.x,0.0,1.0,0.0);
   glGetDoublev(GL_MODELVIEW_MATRIX,(GLdouble*)&rmp->matrix);
  } else return;
 } else { // 物体の移動
  vector3d v1;

  CalcCrossPosition(&tnormal0,&targetpos0,&matrixst,x,y,&v1);

  if(rmp->group==0){
   rmp->vector.x+=v1.x-targetpos1.x;
   rmp->vector.y+=v1.y-targetpos1.y;
   rmp->vector.z+=v1.z-targetpos1.z;
   CalcBoundingBox(rmp);
  } else {
   stmode* mp;
   int i;
   //for(mp=modelsp+10,i=10;i<MODE_MAX;i++,mp++){
   for(mp=modelsp+MODEL_OFFSET,i=MODEL_OFFSET;i<MODE_MAX;i++,mp++){
    if(mp->active && mp->group==rmp->group){
     mp->vector.x+=v1.x-targetpos1.x;
     mp->vector.y+=v1.y-targetpos1.y;
     mp->vector.z+=v1.z-targetpos1.z;
     CalcBoundingBox(mp);
    }
   }
  }

  targetpos1=v1;
 }
```

```
  glutPostRedisplay();
  break;

 case MENUID_SPLIT1:{
  int xx,yy;
  double r;

  xx=x-mousest.x; yy=y-mousest.y;

#define SPSCALE 5.0
#define SPOFFSET 10.0

  r=-sqrt(xx*xx+yy*yy)+SPOFFSET;
  if(r>0.0) r=0.0;
  r*=SPSCALE;

  targetpos=posist;
  targetpos.x+=tnormal.x*r;
  targetpos.y+=tnormal.y*r;
  targetpos.z+=tnormal.z*r;
  glutPostRedisplay();
  break;
 }

 case MENUID_CREATE:{
  CalcCrossPosition(&tnormal0,&targetpos0,&matrixst,x,y,&targetpos);
  targetpos1=targetpos;

  vector3d n0,n1;
  n0=targetpos1;
  n0.x-=targetpos0.x;
  n0.y-=targetpos0.y;
  n0.z-=targetpos0.z;
  double n=sqrt(n0.x*n0.x+n0.y*n0.y+n0.z*n0.z);
  scalec=n; // 2/sqrt(2.0);
  /*
  vectorc=targetpos0;
  vectorc.x+=targetpos1.x;
  vectorc.y+=targetpos1.y;
  vectorc.z+=targetpos1.z;
  vectorc.x/=2.0;
  vectorc.y/=2.0;
  vectorc.z/=2.0;
  */
  vectorc=targetpos0;
  vectorc.x+=tnormal0.x*scalec;
  vectorc.y+=tnormal0.y*scalec;
  vectorc.z+=tnormal0.z*scalec;
  if(n!=0.0){ n0.x/=n; n0.y/=n; n0.z/=n; }
  vector3d v0,v1;
  v0=targetpos0;
```

```
  v0.x+=tnormal0.x;
  v0.y+=tnormal0.y;
  v0.z+=tnormal0.z;
  CalcNormal(&targetpos1,&targetpos0,&v0,&n1);

  /*
  vector3d n2,n3;
  n2=n3=n0;
  n2.x+=n1.x;
  n2.y+=n1.y;
  n2.z+=n1.z;
  n3.x-=n1.x;
  n3.y-=n1.y;
  n3.z-=n1.z;

  n=sqrt(2);
  n2.x/=n;
  n2.y/=n;
  n2.z/=n;
  n3.x/=n;
  n3.y/=n;
  n3.z/=n;
  */

/*
#define M11 0
#define M12 4
#define M13 8
#define M14 12
#define M21 1
#define M22 5
#define M23 9
#define M24 13
#define M31 2
#define M32 6
#define M33 10
#define M34 14
#define M41 3
#define M42 7
#define M43 11
#define M44 15
*/

#define M11 0
#define M12 1
#define M13 2
#define M14 3
#define M21 4
#define M22 5
#define M23 6
#define M24 7
```

```
#define M31 8
#define M32 9
#define M33 10
#define M34 11
#define M41 12
#define M42 13
#define M43 14
#define M44 15

  /*
  matrixc.m[M31]=n3.x;
  matrixc.m[M32]=n3.y;
  matrixc.m[M33]=n3.z;
  matrixc.m[M34]=0.0;

  matrixc.m[M11]=n2.x;
  matrixc.m[M12]=n2.y;
  matrixc.m[M13]=n2.z;
  matrixc.m[M14]=0.0;
  */

  matrixc.m[M11]=n1.x;
  matrixc.m[M12]=n1.y;
  matrixc.m[M13]=n1.z;
  matrixc.m[M14]=0.0;

  matrixc.m[M31]=n0.x;
  matrixc.m[M32]=n0.y;
  matrixc.m[M33]=n0.z;
  matrixc.m[M34]=0.0;

  matrixc.m[M21]=tnormal0.x;
  matrixc.m[M22]=tnormal0.y;
  matrixc.m[M23]=tnormal0.z;
  matrixc.m[M24]=0.0;

  matrixc.m[M41]=0.0;
  matrixc.m[M42]=0.0;
  matrixc.m[M43]=0.0;
  matrixc.m[M44]=1.0;

  glutPostRedisplay();
  break;
 }

 case MENUID_DELETE:
 case MENUID_PAINT://1998年6月19日(金)追加
 case MENUID_SAVE:
  break;

/*
```

```
 case MENUID_TOUCH:{
  polynum=CalcTouchPolygon(modelsp,x,y,&rmp);
  if(polynum>=0) CalcTouchPosition(rmp,polynum,x,y,&targetpos);
if(opt_qv) printf("座標 (%f,%f,%f)\n",targetpos.x,targetpos.y,targetpos.z);
  glutPostRedisplay();
  break;
 }

 case MENUID_SELECT:{
  polynum=CalcTouchPolygon(modelsp,x,y,&rmp);
if(opt_qv) printf("ポリゴン:%d\n",polynum);
  glutPostRedisplay();
  break;
 }
*/

 }
}

//■マウスイベントの処理
void WindowMouseLeftDown(int x, int y){
  clickcount=0;
  flagDisplay=0;
  WindowPassiveMotion(x,y);
  flagDisplay=1;

  if(y<2+MODE_HEIGHT){
    clickcount=100;
    int mode=x/MODE_WIDTH;
    switch(mode){
    case MODEID_MOVE: mousemode=mmd=MENUID_MOVE1; break;
    case MODEID_ROTATE: mousemode=mmd=MENUID_ROTATE3; break;
    case MODEID_SPLIT: mousemode=mmd=MENUID_SPLIT1; break;
    case MODEID_SCALE: mousemode=mmd=MENUID_SCALE; break;
    case MODEID_CREATE: mousemode=mmd=MENUID_CREATE; break;
    case MODEID_DELETE: mousemode=mmd=MENUID_DELETE; break;
    case MODEID_GROUP: ModelGroup(); break;
    case MODEID_PAINT: mousemode=mmd=MENUID_PAINT; break;//1998年6月18日(木)追加
    case MODEID_SAVE: mousemode=mmd=MENUID_SAVE; break;
    case MODEID_LOAD: return;
    default: return;
    }
    glutPostRedisplay();
    return;
  }

  // mmd=mousemode;
  if(polynum<0) return;
  if(rmp==modelsp){
    //if(mmd!=MENUID_ROTATE3 && mouse!=MENUID_MOVE1) return;
    //mmd=MENUID_MOVE1;
```

```
}
// 3次元アイコンをクリックした時の処理
else if(rmp==modelsp+3){
  stmode* mp;
  polynum=-1;
  mp=ScanFreeModel();
  ModelAlloc(mp);
  if(rmdload("squcone.rmd",mp)) return;
  polyresize(mp,3.0);
  mp->vector.y=-300.0;
  CalcBoundingBox(mp);
  ModelPrepare(mp);
  glutPostRedisplay();
  return;
}
else if(rmp==modelsp+4){
  stmode* mp;
  polynum=-1;
  mp=ScanFreeModel();
  ModelAlloc(mp);
  if(rmdload("tube.rmd",mp)) return;
  polyresize(mp,1.8);
  mp->vector.y=-400.0;
  CalcBoundingBox(mp);
  ModelPrepare(mp);
  glutPostRedisplay();
  return;
}
else if(rmp==modelsp+5){
  stmode* mp;
  polynum=-1;
  mp=ScanFreeModel();
  ModelAlloc(mp);
  if(rmdload("tricone.rmd",mp)) return;
  polyresize(mp,1.8);
  mp->vector.y=-400.0;
  CalcBoundingBox(mp);
  ModelPrepare(mp);
  glutPostRedisplay();
  return;
}
else if(rmp==modelsp+6){
  stmode* mp;
  polynum=-1;
  mp=ScanFreeModel();
  ModelAlloc(mp);
  if(rmdload("tribox.rmd",mp)) return;
  polyresize(mp,1.8);
  mp->vector.y=-400.0;
  CalcBoundingBox(mp);
  ModelPrepare(mp);
```

```
  glutPostRedisplay();
  return;
  /*
    ModelDelete();
    glutPostRedisplay();
    return;
    */
}
else if(rmp==modelsp+7){
  stmode* mp;
  polynum=-1;
  mp=ScanFreeModel();
  ModelAlloc(mp);
  if(rmdload("sphere.rmd",mp)) return;
  mp->vector.y=-500.0;
  CalcBoundingBox(mp);
  ModelPrepare(mp);
  glutPostRedisplay();
  return;
}
else if(rmp==modelsp+8){
  stmode*mp;
  polynum=-1;
  mp=ScanFreeModel();
  ModelAlloc(mp);
  if(rmdload("cone.rmd",mp)) return;
  mp->vector.y=-350.0;
  CalcBoundingBox(mp);
  ModelPrepare(mp);
  glutPostRedisplay();
  return;
}
else if(rmp==modelsp+9){
  stmode*mp;
  polynum=-1;
  mp=ScanFreeModel();
  ModelAlloc(mp);
  if(rmdload("box.rmd",mp)) return;
  mp->vector.y=-400.0;
  CalcBoundingBox(mp);
  ModelPrepare(mp);
  glutPostRedisplay();
  return;
}

//カラー物体をクリックした時の処理
else if(rmp==modelsp+10){//10:カラー物体（赤）
  stmode* mp;
  int i;

  for(mp=modelsp+MODEL_OFFSET,i=MODEL_OFFSET;i<MODE_MAX;i++,mp++)
```

```
      if (mp->active && mp->bound) {
ChangeColor(mp, 0, color_red);// 色変更
ModelPrepare(mp);
glutPostRedisplay();
      }
    mousemode=mmd=MENUID_PAINT;
    return;
  }
  else if(rmp==modelsp+11){//11: カラー物体（橙）
    stmode* mp;
    int i;

    for(mp=modelsp+MODEL_OFFSET,i=MODEL_OFFSET;i<MODE_MAX;i++,mp++)
      if (mp->active && mp->bound) {
ChangeColor(mp, 0, color_orange);// 色変更
ModelPrepare(mp);
glutPostRedisplay();
      }
    mousemode=mmd=MENUID_PAINT;
    return;
  }
  else if(rmp==modelsp+12){//12: カラー物体（黄）
    stmode* mp;
    int i;

    for(mp=modelsp+MODEL_OFFSET,i=MODEL_OFFSET;i<MODE_MAX;i++,mp++)
      if (mp->active && mp->bound) {
ChangeColor(mp, 0, color_yellow);// 色変更
ModelPrepare(mp);
glutPostRedisplay();
      }
    mousemode=mmd=MENUID_PAINT;
    return;
  }
  else if(rmp==modelsp+13){//13: カラー物体（緑）
    stmode* mp;
    int i;

    for(mp=modelsp+MODEL_OFFSET,i=MODEL_OFFSET;i<MODE_MAX;i++,mp++)
      if (mp->active && mp->bound) {
ChangeColor(mp, 0, color_green);// 色変更
ModelPrepare(mp);
glutPostRedisplay();
      }
    mousemode=mmd=MENUID_PAINT;
    return;
  }
  else if(rmp==modelsp+14){//14: カラー物体（白）
    stmode* mp;
    int i;
```

```
    for(mp=modelsp+MODEL_OFFSET,i=MODEL_OFFSET;i<MODE_MAX;i++,mp++)
      if (mp->active && mp->bound) {
ChangeColor(mp, 0, color_white);// 色変更
ModelPrepare(mp);
glutPostRedisplay();
      }
    mousemode=mmd=MENUID_PAINT;
    return;
  }
  else if(rmp==modelsp+15){//15: カラー物体（紫）
    stmode* mp;
    int i;

    for(mp=modelsp+MODEL_OFFSET,i=MODEL_OFFSET;i<MODE_MAX;i++,mp++)
      if (mp->active && mp->bound) {
ChangeColor(mp, 0, color_purple);// 色変更
ModelPrepare(mp);
glutPostRedisplay();
      }
    mousemode=mmd=MENUID_PAINT;
    return;
  }
  else if(rmp==modelsp+16){//16: カラー物体（青）
    stmode* mp;
    int i;

    for(mp=modelsp+MODEL_OFFSET,i=MODEL_OFFSET;i<MODE_MAX;i++,mp++)
      if (mp->active && mp->bound) {
ChangeColor(mp, 0, color_blue);// 色変更
ModelPrepare(mp);
glutPostRedisplay();
      }
    mousemode=mmd=MENUID_PAINT;
    return;
  }
  else if(rmp==modelsp+17){//17: カラー物体（水）
    stmode* mp;
    int i;

    for(mp=modelsp+MODEL_OFFSET,i=MODEL_OFFSET;i<MODE_MAX;i++,mp++)
      if (mp->active && mp->bound) {
ChangeColor(mp, 0, color_cyan);// 色変更
ModelPrepare(mp);
glutPostRedisplay();
      }
    mousemode=mmd=MENUID_PAINT;
    return;
  }

  //else if(rmp<modelsp+10){
  else if(rmp<modelsp+MODEL_OFFSET){
```

```
    return;
  }

  mousest.x=x;
  mousest.y=y;
  flagMouseL=1;

  switch(mmd){
    /*
      case MENUID_ROTATE1:
      spinst=spin;
      break;
      case MENUID_ROTATE2:
      break;
      */
  case MENUID_ROTATE3:
    // polynum=-1;
    matrixst=rmp->matrix;
    {
      stmode* mp;
      int i;
      //for(mp=modelsp+10,i=10;i<MODE_MAX;i++,mp++)
      if((rmp->group && mp->active && mp->group==rmp->group)
 || (rmp->group==0 && mp==rmp)){
      for(mp=modelsp+MODEL_OFFSET,i=MODEL_OFFSET;i<MODE_MAX;i++,mp++)
  if((rmp->group && mp->active && mp->group==rmp->group)
     || (rmp->group==0 && mp==rmp)){
mp->vector0=mp->vector;
mp->center0=mp->center;
      }
    }
    break;

    /*
      case MENUID_MOVE2:
      if(rmp==modelsp){
      spinst=spin;
      } else {
      vector3d v,v2;
      matrix16d rrmatrix;

      //polynum=-1;
      vectorst=*(vector3d*)(rmp->vector);

      //
      glMatrixMode(GL_MODELVIEW);
      glLoadIdentity();
      glRotatef(VROT,1.0,0.0,0.0);
      glMultMatrixd(modelsp->matrix);
      glTranslated(-modelsp->vector.x,-modelsp->vector.y,-modelsp->vector.z);
      glGetDoublev(GL_MODELVIEW_MATRIX,(GLdouble*)&matrixst);
```

```
v.x=0.0; v.y=-1.0; v.z=0.0;
CalcInvertMatrix((GLdouble*)&modelsp->matrix,(double*)&rrmatrix);
CalcMatrixN((GLdouble*)&rrmatrix,&v,&tnormal0);
//CalcMatrixN((GLdouble*)&matrixst,&v,&tnormal);

CalcMatrixV((GLdouble*)&rmp->matrix,&targetpos,&v2);
targetpos0.x=v2.x+rmp->vector.x;
targetpos0.y=v2.y+rmp->vector.y;
targetpos0.z=v2.z+rmp->vector.z;

CalcCrossPosition(&tnormal0,&targetpos0,&matrixst,x,y,&targetpos1);

//  glMatrixMode(GL_MODELVIEW);
//  glLoadIdentity();
//  glRotatef(VROT,1.0,0.0,0.0);
//  glMultMatrixd((GLdouble*)&modelsp->matrix);
//  glTranslated(-modelsp->vector.x,-modelsp->vector.y,-modelsp->vector.z);
//  glGetDoublev(GL_MODELVIEW_MATRIX,(GLdouble*)&matrixst);
//  v.x=0.0; v.y=-1.0; v.z=0.0;
//  CalcInvertMatrix((GLdouble*)&modelsp->matrix,(double*)&rrmatrix);
//  CalcMatrixN((GLdouble*)&rrmatrix,&v,&tnormal);
//  CalcCrossPosition(&tnormal,&targetpos,&matrixst,x,y,&v);
//  targetpos0=v;

//SetModelView(rmp);
//  glMatrixMode(GL_MODELVIEW);
//  glLoadIdentity();
//  glRotatef(VROT,1.0,0.0,0.0);
//  glMultMatrixd((GLdouble*)&modelsp->matrix);
//  glTranslated(-modelsp->vector.x,-modelsp->vector.y,-modelsp->vector.z);

//  glMatrixMode(GL_MODELVIEW);
//  glLoadIdentity();
//  glRotatef(-45.0,1.0,0.0,0.0);

//  SetModelView(modelsp);

//  glGetDoublev(GL_MODELVIEW_MATRIX,(GLdouble*)&matrixst);
//  v.x=0.0; v.y=-1.0; v.z=0.0;

//  tnormal.x=0.0;
//  tnormal.y=-1.0;
//  tnormal.z=0.0;

//  CalcInvertMatrix(&matrixst,&rrmatrix);
//  CalcMatrixN(&rrmatrix,&v,&tnormal);
// CalcMatrixN(&matrixst,&v,&tnormal);
//  CalcMatrixN((GLdouble*)&modelsp->matrix,&v,&tnormal);

//  CalcInvertMatrix((GLdouble*)&modelsp->matrix,(double*)&rrmatrix);
//  CalcMatrixN((GLdouble*)&rrmatrix,&v,&tnormal);
```

```
      //  CalcMatrixV((GLdouble*)&rmp->matrix,&targetpos,&v2);
      //printf("%f,%f,%f  %f,%f,%f\n",targetpos.x,targetpos.y,targetpos.z,v2.x,v2.y,v2.z);
      //  targetpos.x=v2.x;
      //  targetpos.y=v2.y;
      //  targetpos.z=v2.z;

      //  CalcCrossPosition(&tnormal,&targetpos,&matrixst,x,y,&v);
      //  targetpos0=v;
      }
      break;
      */

  case MENUID_SCALE:
    {
      taxis=polynum;
      toffset.x=toffset.y=toffset.z=0;

      SetModelView(modelsp);
      glGetDoublev(GL_MODELVIEW_MATRIX,(GLdouble*)&matrixst);
      tnormal0.x=0.0;
      tnormal0.y=-1.0;
      tnormal0.z=0.0;

      targetpos0=targetpos;
      CalcCrossPosition(&tnormal0,&targetpos0,&matrixst,x,y,&targetpos1);
      break;
    }

  case MENUID_MOVE1:
    if(rmp==modelsp){
      spinst=spin;
      // mousemode=mmd=MENUID_MOVE1;
    } else {
      //   vector3d v,v2;
      //   matrix16d rrmatrix;

      // vectorst=rmp->vector;
      SetModelView(modelsp);
      glGetDoublev(GL_MODELVIEW_MATRIX,(GLdouble*)&matrixst);
      tnormal0.x=0.0;
      tnormal0.y=-1.0;
      tnormal0.z=0.0;

      /*
CalcMatrixV((GLdouble*)&rmp->matrix,&targetpos,&v2);
targetpos0.x=v2.x+rmp->vector.x;
targetpos0.y=v2.y+rmp->vector.y;
targetpos0.z=v2.z+rmp->vector.z;
*/
      targetpos0=targetpos;
```

```
    CalcCrossPosition(&tnormal0,&targetpos0,&matrixst,x,y,&targetpos1);
  }
  break;

case MENUID_SPLIT1:{
  /*
    vector3d v1;
    double ax,ay,az,bx,by,bz,n;
    */
  //  splitpos=
  posist=targetpos;
  /*
    //  normst=tnormal;
    v1=*(vector3d*)(rmp->vert+rmp->poly[polynum].v[0]);
    dirst1=*(vector3d*)(rmp->vert+rmp->poly[polynum].v[1]);
    dirst1.x-=v1.x;
    dirst1.y-=v1.y;
    dirst1.z-=v1.z;
    n=sqrt(dirst1.x*dirst1.x+dirst1.y*dirst1.y+dirst1.z*dirst1.z);
    if(n!=0.0){ dirst1.x/=n; dirst1.y/=n; dirst1.z/=n; }

    ax=dirst1.x; //vp[pp->v[2]].x-vp[pp->v[1]].x;
    ay=dirst1.y; //vp[pp->v[2]].y-vp[pp->v[1]].y;
    az=dirst1.z; //vp[pp->v[2]].z-vp[pp->v[1]].z;
    bx=-tnormal.x; //vp[pp->v[1]].x-vp[pp->v[0]].x;
    by=-tnormal.y; //vp[pp->v[1]].y-vp[pp->v[0]].y;
    bz=-tnormal.z; //vp[pp->v[1]].z-vp[pp->v[0]].z;
    dirst2.x=ay*bz-az*by;
    dirst2.y=az*bx-ax*bz;
    dirst2.z=ax*by-ay*bx;

    //  polynum=-1;
    */
  break;
}

case MENUID_CREATE:{
  SetModelView(modelsp);
  glGetDoublev(GL_MODELVIEW_MATRIX,(GLdouble*)&matrixst);
  tnormal0=tnormal;
  targetpos0=targetpos;
  break;
}

case MENUID_DELETE:{
  if(rmp->group){
    stmode* mp;
    int i;
    //for(mp=modelsp+10,i=10;i<MODE_MAX;i++,mp++){
    for(mp=modelsp+MODEL_OFFSET,i=MODEL_OFFSET;i<MODE_MAX;i++,mp++){
```

```
if(mp->active && mp->group==rmp->group){
  if(mp!=rmp) ModelFree(mp);
}
      }
    }
    ModelFree(rmp);
    polynum=-1;
    glutPostRedisplay();
    return;
  }

  case MENUID_PAINT:{
    /*
    stmode* mp;
    int i;

    for(mp=modelsp+MODEL_OFFSET,i=MODEL_OFFSET;i<MODE_MAX;i++,mp++)
      if((rmp->group && mp->active && mp->group==rmp->group)
 || (rmp->group==0 && mp==rmp)) {
printf("PAINT\n");
printf("%i %i %i\n",mp->mate[0].data[0],mp->mate[0].data[1],mp->mate[0].data[2]);
ChangeColor(mp, 0, color_red);// 色変更
ModelPrepare(mp);
glutPostRedisplay();
printf("%i %i %i\n",mp->mate[0].data[0],mp->mate[0].data[1],mp->mate[0].data[2]);
      }
    */
    return;
  }

  case MENUID_SAVE:{
    /*
      vrmlsave("claymore.wrl",rmp);
      */

    int i;
    stmode* mp;

    FILE* fp;
    FILE* sfp;
    int ct;
    char fbase[256];
    char fname[256];

    //////////////lwo。lws 形式のセーブ
    for(ct=0;;ct++){
sprintf(fbase,"claymore%04d",ct);
sprintf(fname,"%s.lws",fbase);
sfp=fopen(fname,"w");
if(sfp) break;
    }
```

```
    fprintf(sfp,"LWSC\n1\n\n");
    //キーフレーム
    fprintf(sfp,"FirstFrame 1\n");
    fprintf(sfp,"LastFrame 60\n");
    fprintf(sfp,"FrameStep 1\n");
    fprintf(sfp,"FramePerSecond 30.000000\n\n");

    ct=0;
    //オブジェクト
    for(mp=modelsp+MODEL_OFFSET,i=MODEL_OFFSET;i<MODE_MAX;i++,mp++){
if(mp->active){
    for(;;ct++){
sprintf(fbase,"claymore%04d",ct);
sprintf(fname,"%s.lwo",fbase);
fp=fopen(fname,"w");
if(fp) break;
    }
    //
    printf("Filename: %s,",fname);
    lwosave(fp,mp);ct++;
    fprintf(sfp,"LoadObject %s\n",fname);
    fprintf(sfp,"ShowObject 4 7\n");
    fprintf(sfp,"ObjectMotion (unnamed)\n");
    fprintf(sfp,"  9\n  1\n");
    fprintf(sfp,"  %6f %6f %6f 0 0 0 1 1 1\n",
    (float)mp->vector.x/SAVESCALE,
    -(float)mp->vector.y/SAVESCALE,
    (float)mp->vector.z/SAVESCALE);
    fprintf(sfp,"  0 0 0 0 0\n");
    fprintf(sfp,"EndBehavior 1\n");
    fprintf(sfp,"ShadowOptions 7\n\n");
    //
    fclose(fp);
}
    }
    lwssave(sfp);
    fclose(sfp);

    ////////////


    for(ct=0;;ct++){
      sprintf(fbase,"claymore%04d",ct);
      sprintf(fname,"%s.wrl",fbase);
      fp=fopen(fname,"w");
      if(fp) break;
    }
    printf(",%s",fname);
    // fp=fopen("claymore.wrl","w");
    fprintf(fp,
```

```
    "#VRML V1.0 ascii\n"
    "\n"
    "Separator {\n"
    " ShapeHints {\n"
    "  vertexOrdering CLOCKWISE\n"
    " }\n"
    " NormalBinding {\n"
    "  value PER_VERTEX_INDEXED\n"
    " }\n"
    );
    //for(mp=modelsp+10,i=10;i<MODE_MAX;i++,mp++)
    for(mp=modelsp+MODEL_OFFSET,i=MODEL_OFFSET;i<MODE_MAX;i++,mp++)
      if(mp->active) vrmlsave2(fp,mp);
    fprintf(fp,
    "}\n"
    );
    fclose(fp);


    // FILE* fp=fopen("claymore.obj","w");
    sprintf(fname,"%s.obj",fbase);
    fp=fopen(fname,"w");
    if(fp){
      printf(", %s",fname);

      fprintf(fp,"# Claymore\n");
      fprintf(fp,"g object1 object0\n");
      int vo=0,no=0,po=0;

      //for(mp=modelsp+10,i=10;i<MODE_MAX;i++,mp++){
      for(mp=modelsp+MODEL_OFFSET,i=MODEL_OFFSET;i<MODE_MAX;i++,mp++){
if(mp->active){
  mp->verto=vo; vo+=mp->vertn;
  mp->normo=no; no+=mp->normn;
  po+=mp->polyn;
  overtsave(fp,mp);
}
      }
      fprintf(fp,"# %d vertices\n",vo);

      fprintf(fp,"g\n");

      //for(mp=modelsp+10,i=10;i<MODE_MAX;i++,mp++){
      for(mp=modelsp+MODEL_OFFSET,i=MODEL_OFFSET;i<MODE_MAX;i++,mp++){
if(mp->active){
  onormsave(fp,mp);
}
      }
      fprintf(fp,"# %d normals\n",no);

      fprintf(fp,"usemtl object0\n");
```

```
      fprintf(fp,"g object1 object0 object2\n");

      //for(mp=modelsp+10,i=10;i<MODE_MAX;i++,mp++){
      for(mp=modelsp+MODEL_OFFSET,i=MODEL_OFFSET;i<MODE_MAX;i++,mp++){
if(mp->active){
  opolysave(fp,mp);
}
      }
      fprintf(fp,"# %d elements\n",po);
      fprintf(fp,"g\n");
      fclose(fp);
    }
    printf("\n");

    clickcount+=100;
    glutPostRedisplay();
    return;
  }

  }
  WindowMotion(x,y);
}

void WindowMouseLeftUp(){
 flagMouseL=0;

 if(clickcount<3){
  if(rmp==modelsp){
   mousemode=mmd=MENUID_MOVE1;
   glutPostRedisplay();
   return;
  }
  //if(rmp<modelsp+10) return;
  if(rmp<modelsp+MODEL_OFFSET) return;

  rmp->bound^=1;
  if(rmp->group){
   stmode* mp;
   int i;
   //for(mp=modelsp+10,i=10;i<MODE_MAX;i++,mp++){
   for(mp=modelsp+MODEL_OFFSET,i=MODEL_OFFSET;i<MODE_MAX;i++,mp++){
    if(mp->active && mp->group==rmp->group) mp->bound=rmp->bound;
   }
  }
  glutPostRedisplay();
  return;
 }

 //if(rmp<modelsp+10) return;
 if(rmp<modelsp+MODEL_OFFSET) return;
```

```
 switch(mmd){
 case MENUID_CREATE:{
  stmode* mp;
  polynum=-1;
  mp=ScanFreeModel();
  ModelAlloc(mp);
  if(rmdload("bound2.rmd",mp)) return;
  mp->vector=vectorc;
  mp->matrix=matrixc;
  polyresize(mp,scalec);
  CalcBoundingBox(mp);
  ModelPrepare(mp);
  break;
 }

 case MENUID_SCALE:{
  vector3d v0;
  for(int i=0;i<rmp->vertn;i++){
   CalcMatrixV((GLdouble*)&matrixc,&rmp->vert[i],&v0);
   rmp->vert[i]=v0;
  }
  CalcBoundingBox(rmp);
  ModelPrepare(rmp);
  break;
 }

 case MENUID_SPLIT1:{
  double pl[4];
  stmode* splitmp[16];
  stmode* mp;
  int size,i,j;
  int g=rmp->group,g1=0,g2=0;
  int b=rmp->bound;

/*
  pl[0]=tnormal.x;
  pl[1]=tnormal.y;
  pl[2]=tnormal.z;
  pl[3]=-(targetpos.x*tnormal.x+targetpos.y*tnormal.y+targetpos.z*tnormal.z);
*/
/*
  vector3d v0,v1;
  matrix16d vmatrix;
  SetPureModelView(rmp);
  glGetDoublev(GL_MODELVIEW_MATRIX,(GLdouble*)&vmatrix);
  CalcMatrixNR((GLdouble*)&vmatrix,&tnormal,&v0);
  CalcMatrixVR((GLdouble*)&vmatrix,&targetpos,&v1);
  pl[0]=v0.x;
  pl[1]=v0.y;
  pl[2]=v0.z;
  pl[3]=-(v1.x*v0.x+v1.y*v0.y+v1.z*v0.z);
```

```
*/

  if(g){ g1=ScanFreeGroup(); g2=ScanFreeGroup(); }
  //for(mp=modelsp+10,j=10;j<MODE_MAX;j++,mp++){
  for(mp=modelsp+MODEL_OFFSET,j=MODEL_OFFSET;j<MODE_MAX;j++,mp++){
   if((g==0 && mp==rmp) || (g && mp->active && mp->group==g)){
    vector3d v0,v1;
    matrix16d vmatrix;
    SetPureModelView(mp);
    glGetDoublev(GL_MODELVIEW_MATRIX,(GLdouble*)&vmatrix);
    CalcMatrixNR((GLdouble*)&vmatrix,&tnormal,&v0);
    CalcMatrixVR((GLdouble*)&vmatrix,&targetpos,&v1);
    pl[0]=v0.x;
    pl[1]=v0.y;
    pl[2]=v0.z;
    pl[3]=-(v1.x*v0.x+v1.y*v0.y+v1.z*v0.z);

    size=sliceModel(mp,splitmp,pl);
    for(i=0;i<size;i++){
     stmode* mmp=ScanFreeModel();
     *mmp=*splitmp[i];
     CalcBoundingBox(mmp);
     ModelPrepare(mmp);
     if(b) mmp->bound=1;
     mmp->group=(i==0)? g1:g2;
     free(splitmp[i]);
    }
    ModelFree(mp);
   }
  }
  SetGroup(0);

  polynum=-1;
 }
 break;
 }
 glutPostRedisplay();
}

// ■マウスイベントの処理２
void WindowMouseRightDown(int x, int y){
 clickcount=0;
 //if(rmp<modelsp+10){
 if(rmp<modelsp+MODEL_OFFSET){
  WindowMouseLeftDown(x,y);
  return;
 }

 if(polynum<0) return;

 flagDisplay=0;
```

```
 WindowPassiveMotion(x,y);
 flagDisplay=1;

 mousest.x=x;
 mousest.y=y;
 flagMouseL=1;

 switch(mmd){
 case MENUID_SCALE:
 {
  taxis=polynum;
  toffset.x=toffset.y=toffset.z=0;

  SetModelView(modelsp);
  glGetDoublev(GL_MODELVIEW_MATRIX,(GLdouble*)&matrixst);
  tnormal0.x=sin(-spin.x*PI*2.0/360.0);
  tnormal0.y=0.0;
  tnormal0.z=cos(-spin.x*PI*2.0/360.0);

  targetpos0=targetpos;
  CalcCrossPosition(&tnormal0,&targetpos0,&matrixst,x,y,&targetpos1);
  WindowMotion(x,y);
  return;
 }

 case MENUID_MOVE1:
  if(rmp==modelsp){
   spinst=spin;
  } else {
//   vector3d v,v2;
//   matrix16d rrmatrix;

// vectorst=rmp->vector;
   SetModelView(modelsp);
   glGetDoublev(GL_MODELVIEW_MATRIX,(GLdouble*)&matrixst);
   tnormal0.x=sin(-spin.x*PI*2.0/360.0);
   tnormal0.y=0.0;
   tnormal0.z=cos(-spin.x*PI*2.0/360.0);

/*
   CalcMatrixV((GLdouble*)&rmp->matrix,&targetpos,&v2);
   targetpos0.x=v2.x+rmp->vector.x;
   targetpos0.y=v2.y+rmp->vector.y;
   targetpos0.z=v2.z+rmp->vector.z;
*/
   targetpos0=targetpos;

   CalcCrossPosition(&tnormal0,&targetpos0,&matrixst,x,y,&targetpos1);
  }
  WindowMotion(x,y);
  return;
```

```
 }

 WindowMouseLeftDown(x,y);
 return;
}

void WindowMouse(int button, int state, int x, int y){
 if(button==GLUT_LEFT_BUTTON){
  switch(state){
  case GLUT_DOWN:
   WindowMouseLeftDown(x,y);
   break;
  case GLUT_UP:
   WindowMouseLeftUp();
   break;
  }
 } else
#ifdef _WIN32
 if(button==GLUT_RIGHT_BUTTON){
#else
 if(button==GLUT_MIDDLE_BUTTON){
#endif
  switch(state){
  case GLUT_DOWN:
   WindowMouseRightDown(x,y);
   break;
  case GLUT_UP:
   WindowMouseLeftUp();
   break;
  }
 }
}

//■メニュー処理
void WindowMenu(int ID){
 polynum=-1;
 switch(ID){
 case MENUID_QUIT: exit(0);
 case MENUID_RESET:{
  int i;
  stmode* mp;
  spin.x=spin.y=0.0;
  glMatrixMode(GL_MODELVIEW);
  glPushMatrix();
  glLoadIdentity();
  for(mp=modelsp,i=0;i<MODE_MAX;i++,mp++) if(mp->active){
   glGetDoublev(GL_MODELVIEW_MATRIX,(GLdouble*)&mp->matrix);
  }
  glPopMatrix();
// glutPostRedisplay();
  break;
```

```
 }

/*
 case MENUID_ROTATE1:
 case MENUID_ROTATE2:
*/
 case MENUID_ROTATE3:
/*
 case MENUID_TOUCH:
 case MENUID_SELECT:
*/
 case MENUID_MOVE1:
 case MENUID_SPLIT1:
 case MENUID_CREATE:
 case MENUID_DELETE:
 case MENUID_SAVE:
 case MENUID_PAINT:
  mousemode=mmd=ID;
  break;
 }
 glutPostRedisplay();
}

//■キーボード処理
void WindowKeyboard(char key, int x, int y){
 switch(toupper(key)){
 case 3:
 case 0x1B:
 case 'Q': exit(0);
 case 'I':
  ModelInfo(rmp);
  DescribeStmode(rmp);
  break;
// case '?':
//  polycalc(rmp);
//  glutPostRedisplay();
//  break;
 case ' ':
 case 'M': WindowMenu(MENUID_MOVE1); break;
 case 'R': WindowMenu(MENUID_ROTATE3); break;
 case 'S': WindowMenu(MENUID_SPLIT1); break;
 case 'V': WindowMenu(MENUID_SAVE); break;
// case 'T': WindowMenu(MENUID_TOUCH); break;
// case 'S': WindowMenu(MENUID_SELECT); break;
 case 'C': WindowMenu(MENUID_CREATE); break;
 case 'O': WindowMenu(MENUID_RESET); break;
 case 'D': WindowMenu(MENUID_DELETE); break;
 case 'G': ModelGroup(); glutPostRedisplay(); break;
// case 'D': ModelDelete(); glutPostRedisplay(); break;
   // case 'P': moemoeflag^=1; glutPostRedisplay(); break;
 case 'P': WindowMenu(MENUID_PAINT); break;
```

```
 }
}

//■モデルデータの表示
void glview(char* window_name, stmode* mp){
 // int MenuModeID;

 modelsp=rmp=mp;
 InitScreen();

 //OpenGL 使用開始
#ifdef Z_BUFFER
 glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGBA|GLUT_DEPTH);
#else
 glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGBA);
#endif
 glutInitWindowPosition(scr.startx,scr.starty);
 glutInitWindowSize(scr.width,scr.height);
 glutCreateWindow(window_name);
 glClearColor(scr.bcr,scr.bcg,scr.bcb,scr.bca);

 SetLight_MaterialParameter();
 SetLightModel();
 SetLight();
 SetMaterial();

 glEnable(GL_LIGHTING);
 glEnable(GL_LIGHT0);

#ifdef Z_BUFFER
 glEnable(GL_DEPTH_TEST);
#endif
 glEnable(GL_NORMALIZE);
// glEnable(GL_AUTO_NORMAL);

 glPolygonMode(GL_FRONT,GL_FILL);
 glPolygonMode(GL_BACK,GL_FILL);
 glCullFace(GL_BACK);
 glEnable(GL_CULL_FACE);

 glBlendFunc(GL_SRC_ALPHA,GL_ONE_MINUS_SRC_ALPHA);
 glEnable(GL_BLEND);

// glHint(GL_POLYGON_SMOOTH_HINT,GL_FASTEST);
// glEnable(GL_POLYGON_SMOOTH);
 // glLineWidth(1.5);
 glLineWidth(3);
 glHint(GL_LINE_SMOOTH_HINT,GL_FASTEST);
 glEnable(GL_LINE_SMOOTH);
 glPointSize(5.0);
 glHint(GL_POINT_SMOOTH_HINT,GL_FASTEST);
```

```
 glEnable(GL_POINT_SMOOTH);

 //光源行列を初期化
 glMatrixMode(GL_MODELVIEW);
 glLoadIdentity();
 glRotatef(-45.0,1.0,0.0,0.0);
 glLightfv(GL_LIGHT0,GL_POSITION,Light0.posision);

 //メニュー項目の設定
// MenuModeID=glutCreateMenu(WindowMenu);
// glutAddMenuEntry("Rotate1",MENUID_ROTATE1);
// glutAddMenuEntry("Rotate2",MENUID_ROTATE2);
// glutAddMenuEntry("Rotate3",MENUID_ROTATE3);
// glutAddMenuEntry("Touch",MENUID_TOUCH);
// glutAddMenuEntry("Select",MENUID_SELECT);
// glutAddMenuEntry("Move1",MENUID_MOVE1);

 glutCreateMenu(WindowMenu);
// glutAddSubMenu("Mode",MenuModeID);
 glutAddMenuEntry("Move",MENUID_MOVE1);
 glutAddMenuEntry("Rotate",MENUID_ROTATE3);
 glutAddMenuEntry("Split",MENUID_SPLIT1);
 glutAddMenuEntry("Group",MENUID_GROUP);
 glutAddMenuEntry("Delete",MENUID_DELETE);
 glutAddMenuEntry("Save",MENUID_SAVE);
 glutAddMenuEntry("Reset",MENUID_RESET);
 glutAddMenuEntry("Quit",MENUID_QUIT);
#ifdef _WIN32
 glutAttachMenu(GLUT_MIDDLE_BUTTON);
#else
 glutAttachMenu(GLUT_RIGHT_BUTTON);
#endif

 gltop=glGenLists(1024);
 glbound=glGenLists(1);
 MakeDisplayList(glbound,&bound2,0);
 for(int i=0;i<MODE_MAX;i++,mp++) if(mp->active){
  if(mp==modelsp){
   Material.diffuse[3]=0.7;
   ModelPrepare(mp);
   Material.diffuse[3]=1.0;
  } else {
   ModelPrepare(mp);
  }
 }

 //動作開始
 glutDisplayFunc(WindowDraw);
 glutReshapeFunc(WindowReshape);
 glutVisibilityFunc(WindowVisible);
 glutKeyboardFunc((void(*)(unsigned char,int,int))WindowKeyboard);
```

```
 glutMouseFunc(WindowMouse);
 glutMotionFunc((void(*)(int,int))WindowMotion);
 glutPassiveMotionFunc((void(*)(int,int))WindowPassiveMotion);
 glutIdleFunc(WindowIdle);

 glutMainLoop();
}
```

A.2 “3D-PP”

models-gui.cpp

```
#include <qgl.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "models.h"
#include "models-gui.h"
#include "rmddraw.h"
#include "polycalc.h"
#include "glbox.h"
#include "rmdfile.h"
#include "screen.h"

int GUINodeCore::ID_max = 0;

GUINodeCore::GUINodeCore()
{
    //    ID= ++ID_max; // 地面の ID が 2 になる。
    ID= ID_max++; // 地面の ID が 1 になる。
  Node = new NodeCore;

  ModelAlloc(&shape);

  displaylist1 = glGenLists(1);
  displaylist2 = glGenLists(1);
  displaylist3 = glGenLists(1);

  surface = TRUE;
  transparent = FALSE;
  wireframe = FALSE;

  focus = FALSE;
  invisible = FALSE;

  NodePrev = NULL;
  NodeNext = NULL;

  EdgeStartID = 0;
  EdgeEndID = 0;
```

```
  for(int i=0;i<8;i++){
    child[i] = NULL;
  }
  parent = NULL;
}

GUINodeCore::~GUINodeCore(){
}

GUINodeCore* GUINodeCore::NodeNextTo(GUINodeCore *guinode, int times){
  GUINodeCore *guinodenext;
  guinodenext = guinode;
  if(guinode->NodeNext != NULL && times > 0)
    guinodenext = NodeNextTo(guinode->NodeNext, times - 1);
  return guinodenext;
}

void GUINodeCore::DrawMode(bool mode1, bool mode2, bool mode3){
  surface = mode1;
  transparent = mode2;
  wireframe = mode3;
}

void GUINodeCore::FocusMode(bool mode){
  if((focus&&mode)||(!focus && !mode)){
  }
  else {
    if(focus && !mode){
      focus = mode;
      polyresize(&shape,1.0);
      //polyresize(&shape,0.3125);
    }
    else {
      focus = mode;
      polyresize(&shape,1.0);
      //polyresize(&shape,3.2);
    }
    glDeleteLists(displaylist1,(GLsizei)1);
    displaylist1 = glGenLists(1);
    MakeDisplayList(displaylist1, &shape, DRAW_MODE_NORMAL);

    glDeleteLists(displaylist2,(GLsizei)1);
    displaylist2 = glGenLists(1);
    MakeDisplayList(displaylist2, &shape, DRAW_MODE_TRANSPARENT);

    glDeleteLists(displaylist3,(GLsizei)1);
    displaylist3 = glGenLists(1);
    MakeDisplayList(displaylist3, &shape, DRAW_MODE_WIREFRAME);
  }
}
```

```
void GUINodeCore::ChildUpdate(){
  if(focus){
    for(int i=0;i<8;i++){
      if(child[i]!=NULL){
child[i]->invisible = FALSE;
child[i]->shape.vector = shape.vector;
      }
    }
  }
 else{
    for(int i=0;i<8;i++){
      if(child[i]!=NULL){
child[i]->invisible = TRUE;
child[i]->shape.vector = shape.vector;
      }
    }
  }
}
void  GUINodeCore::EdgeDraw(GUINodeCore* guinode,GUINodeCore* gui_world,
    GUINodeCore* gui_list_start,matrix16d matrixst){
  if( guinode->EdgeStartID != NULL & guinode->EdgeEndID != NULL ){
    //     printf("EdgeDrawing!!!\n");
     int StartID = guinode->EdgeStartID;
     int EndID = guinode->EdgeEndID;
     vector3d first_axis, second_axis;

      for (guinode = gui_list_start->NodeNext; //始点座標抽出
   guinode->ID < StartID;
   guinode = guinode->NodeNext) {
      }

      first_axis.x = guinode->shape.vector.x;
      first_axis.y = guinode->shape.vector.y;
      first_axis.z = guinode->shape.vector.z;

      for (guinode = gui_list_start->NodeNext; //終点座標抽出
   guinode->ID < EndID;
   guinode = guinode->NodeNext) {
      }
second_axis.x = guinode->shape.vector.x;
second_axis.y = guinode->shape.vector.y;
second_axis.z = guinode->shape.vector.z;

//線の描画
glMatrixMode(GL_MODELVIEW);
matrix16d vmatrix;
glGetDoublev(GL_MODELVIEW_MATRIX,(GLdouble*)&vmatrix);
glPushMatrix();
glLoadIdentity();

glRotatef(VROT, 1.0, 0.0, 0.0);
```

```
glTranslated(gui_list_start->shape.vector.x, gui_list_start->shape.vector.y, gui_list_start->shape.vector.z);

SetModelView(&(gui_world->shape), &(gui_world->shape));
glGetDoublev(GL_MODELVIEW_MATRIX,(GLdouble*)&matrixst);

glBegin(GL_LINE_LOOP);
glVertex3d(first_axis.x, first_axis.y, first_axis.z);
glVertex3d(second_axis.x, second_axis.y, second_axis.z);
glEnd();

glPopMatrix();
  }
}

// 地面の上に３次元アイコンを生成（起動時に数回だけ呼ばれる）
int Mini3DIconNode::create3DIcon(GUINodeCore* guinode,
  int whichicon, double resize,
  double movex,
  double movey,
  double movez)
{
/*
    stmode* mp = &(guinode->shape);

    switch (whichicon) {
      case ICON_ATOM:
if (rmdload("/rmd/cone.rmd", mp)) return 2;
ChangeColor(mp, 0, color_blue);
polyresize(mp, resize);
mp->vector.x = movex;
mp->vector.y = movey;
mp->vector.z = movez;
MakeDisplayList(guinode->displaylist1, mp, DRAW_MODE_NORMAL);
MakeDisplayList(guinode->displaylist2, mp, DRAW_MODE_TRANSPARENT);
MakeDisplayList(guinode->displaylist3, mp, DRAW_MODE_WIREFRAME);
break;

      case ICON_LIST:
if (rmdload("/rmd/tube.rmd", mp)) return 2;
ChangeColor(mp, 0, color_green);
polyresize(mp, resize);
mp->vector.x = movex;
mp->vector.y = movey;
mp->vector.z = movez;
MakeDisplayList(guinode->displaylist1, mp, DRAW_MODE_NORMAL);
MakeDisplayList(guinode->displaylist2, mp, DRAW_MODE_TRANSPARENT);
MakeDisplayList(guinode->displaylist3, mp, DRAW_MODE_WIREFRAME);
break;

      case ICON_GOAL:
```

```
if(rmdload("/rmd/tube.rmd", mp)) return 2;
ChangeColor(mp,0,color_green);
polyresize(mp, resize);
mp->vector.x = movex;
mp->vector.y = movey;
mp->vector.z = movez;
MakeDisplayList(guinode->displaylist1, mp, DRAW_MODE_NORMAL);
MakeDisplayList(guinode->displaylist2, mp, DRAW_MODE_TRANSPARENT);
MakeDisplayList(guinode->displaylist3, mp, DRAW_MODE_WIREFRAME);
break;

      case ICON_BLTIN:
if(rmdload("/rmd/bound.rmd", mp)) return 2;
ChangeColor(mp,0,color_purple);
polyresize(mp, resize);
mp->vector.x = movex;
mp->vector.y = movey;
mp->vector.z = movez;
MakeDisplayList(guinode->displaylist1, mp, DRAW_MODE_NORMAL);
MakeDisplayList(guinode->displaylist2, mp, DRAW_MODE_TRANSPARENT);
MakeDisplayList(guinode->displaylist3, mp, DRAW_MODE_WIREFRAME);
break;

      case ICON_VAR:
if(rmdload("/rmd/bound.rmd", mp)) return 2;
ChangeColor(mp,0,color_orange);
polyresize(mp, resize);
mp->vector.x = movex;
mp->vector.y = movey;
mp->vector.z = movez;
MakeDisplayList(guinode->displaylist1, mp, DRAW_MODE_NORMAL);
MakeDisplayList(guinode->displaylist2, mp, DRAW_MODE_TRANSPARENT);
MakeDisplayList(guinode->displaylist3, mp, DRAW_MODE_WIREFRAME);
break;
    }
*/
}

// クリックされた３次元アイコンと同じ形のものを GUINodeCore のノード
// リストに加える
int Mini3DIconNode::AddNodeToGUINodeCore(GUINodeCore* guinode,
 int whichicon, double resize,
 double movex,
 double movey,
 double movez)
{
    stmode* mp = &(guinode->shape);

    switch (whichicon) {
      case ICON_ATOM:
if (rmdload("/rmd/minicone.rmd", mp)) return 2;
```

```
ChangeColor(mp, 0, color_blue);
polyresize(mp, resize);
mp->vector.x = movex;
mp->vector.y = movey;
mp->vector.z = movez;
MakeDisplayList(guinode->displaylist1, mp, DRAW_MODE_NORMAL);
MakeDisplayList(guinode->displaylist2, mp, DRAW_MODE_TRANSPARENT);
MakeDisplayList(guinode->displaylist3, mp, DRAW_MODE_WIREFRAME);
break;

      case ICON_LIST:
if (rmdload("/rmd/squcone.rmd", mp)) return 2;
ChangeColor(mp, 0, color_red);
polyresize(mp, resize);
mp->vector.x = movex;
mp->vector.y = movey;
mp->vector.z = movez;
MakeDisplayList(guinode->displaylist1, mp, DRAW_MODE_NORMAL);
MakeDisplayList(guinode->displaylist2, mp, DRAW_MODE_TRANSPARENT);
MakeDisplayList(guinode->displaylist3, mp, DRAW_MODE_WIREFRAME);
break;

      case ICON_GOAL:
if(rmdload("/rmd/tube.rmd", mp)) return 2;
ChangeColor(mp,0,color_green);
polyresize(mp, resize);
mp->vector.x = movex;
mp->vector.y = movey;
mp->vector.z = movez;
MakeDisplayList(guinode->displaylist1, mp, DRAW_MODE_NORMAL);
MakeDisplayList(guinode->displaylist2, mp, DRAW_MODE_TRANSPARENT);
MakeDisplayList(guinode->displaylist3, mp, DRAW_MODE_WIREFRAME);
break;

      case ICON_BLTIN:
if(rmdload("/rmd/minibox.rmd", mp)) return 2;
ChangeColor(mp,0,color_purple);
polyresize(mp, resize);
mp->vector.x = movex;
mp->vector.y = movey;
mp->vector.z = movez;
MakeDisplayList(guinode->displaylist1, mp, DRAW_MODE_NORMAL);
MakeDisplayList(guinode->displaylist2, mp, DRAW_MODE_TRANSPARENT);
MakeDisplayList(guinode->displaylist3, mp, DRAW_MODE_WIREFRAME);
break;

      case ICON_VAR:
if(rmdload("/rmd/minisp.rmd", mp)) return 2;
ChangeColor(mp,0,color_orange);
polyresize(mp, resize);
mp->vector.x = movex;
```

```
mp->vector.y = movey;
mp->vector.z = movez;
MakeDisplayList(guinode->displaylist1, mp, DRAW_MODE_NORMAL);
MakeDisplayList(guinode->displaylist2, mp, DRAW_MODE_TRANSPARENT);
MakeDisplayList(guinode->displaylist3, mp, DRAW_MODE_WIREFRAME);
break;
    }
}


//■■一筆書き手法・コンストラクタ
OneStroke::OneStroke() {

    /*
    vector2f temp;
    temp.x = -1; temp.y = -1;
    locus[0]->x = temp.x;
    locus[0]->y = temp.y;
    */
    for (int i = 0; i < LOCUSMAX; i++) {
locusX[i] = -1;
locusY[i] = -1;
    }
    indexmax = 0;

    for (int j = 0; j < CONNECT_NODE_MAX; j++)
connestlist[j] = NULL;
    connectmax = 0;
    lasttouch = -1;
}

//■■一筆書き手法で、マウスの軌跡を「格納する」メソッド（マウスボタン
//が押されている時に何度も呼ばれる）
void OneStroke::StoreMouseLocus(OneStroke* onestroke,
double nowX, double nowY) {

    int i = 0;
    while (i < LOCUSMAX) {
if (onestroke->locusX[i] != -1 && onestroke->locusY[i] != -1)
    i++;
else {
    onestroke->indexmax = i;
    //マウスがちょっとでも動いただけで locusX,locusY にその時の
    //マウスの位置を格納していては、 LOCUSMAX を大きく取らないと
    //いけなくなってしまうので、前回の locusX,locusY の値と比べ
    //てある程度 (LOCUSDIFF) マウスが動いた後だったら格納する。
    if (onestroke->indexmax < 1) {
onestroke->locusX[onestroke->indexmax] = nowX;
onestroke->locusY[onestroke->indexmax] = nowY;
    }
    else if (fabs(onestroke->locusX[onestroke->indexmax-1]-nowX)
```

```
     > LOCUSDIFF ||
     fabs(onestroke->locusY[onestroke->indexmax-1]-nowY)
     > LOCUSDIFF) {
onestroke->locusX[onestroke->indexmax] = nowX;
onestroke->locusY[onestroke->indexmax] = nowY;
    }
    break;
}
    }
}

// ■■一筆書き手法で、マウスの軌跡を「描画する」メソッド（マウスボタン
// が押されている時に何度も呼ばれる）
void OneStroke::DrawMouseLocus(OneStroke* onestroke) {

    /*
      #ifdef Z_BUFFER
      glDisable(GL_DEPTH_TEST);
      glDepthMask(GL_FALSE);
      #endif
      */

    glMatrixMode(GL_MODELVIEW);
    glPushMatrix();// ● InformationDraw,Push-Pop ラベル a ↓
    glLoadIdentity();
    glMatrixMode(GL_PROJECTION);
    glPushMatrix();// ● InformationDraw,Push-Pop ラベル aa ↓
    glLoadIdentity();
    gluOrtho2D(0,scr.width,0,scr.height);
    glScalef(1,-1,1);
    glTranslatef(0,-scr.height,0);
    glDisable(GL_LIGHTING);

    // マウスの軌跡を描画
    //    glColor4ub(255,60,100,160);
    if (onestroke->indexmax == LOCUSMAX - 1)
glColor3ub(255,60,100);
    else
glColor3ub(255,60,10);
    for (int i = 1; i < onestroke->indexmax; i++) {
//     printf("\ti=%d, x=%f, y=%f, max=%d\n", i, locusX[i], locusY[i], onestroke->indexmax);
glBegin(GL_LINE_STRIP);
glVertex2i((int)onestroke->locusX[i-1],
   (int)onestroke->locusY[i-1]);
glVertex2i((int)onestroke->locusX[i],
   (int)onestroke->locusY[i]);
glEnd();
    }

    /*
    // マウスの軌跡の先端を描画
```

```
    if (onestroke->indexmax > 1) {
glColor3ub(255,60,100);
glBegin(GL_LINE_LOOP);
glVertex2i((int)onestroke->locusX[onestroke->indexmax - 2],
   (int)onestroke->locusY[onestroke->indexmax - 2]);
glVertex2i((int)onestroke->musnow.x,
   (int)onestroke->musnow.y);
glEnd();
    }
    */

    glColor3ub(255,255,255);
    glEnable(GL_LIGHTING);
    glPopMatrix();// ● InformationDraw,Push-Pop ラベル aa ↑
    glMatrixMode(GL_MODELVIEW);
    glPopMatrix();// ● InformationDraw,Push-Pop ラベル a ↑

    /*
      #ifdef Z_BUFFER
      glEnable(GL_DEPTH_TEST);
      glDepthMask(GL_TRUE);
      #endif
      */
}

// ■■一筆書き手法で、マウスの軌跡と交差したノードの ID を格納し、リスト
// にするメソッド（マウスボタンが押されている時に何度も呼ばれる）
void OneStroke::DetectTouchNode(OneStroke* onestroke, GUINodeCore* node) {

    // ノードにマウスカーソルが触れていたら・・・
    if (node != NULL) {
// 初めて触ったノード
if (onestroke->lasttouch != node->ID) {
    if (onestroke->connestlist[0] == NULL) {
// 結線するノードのリストに加える
onestroke->connestlist[connectmax] = node;
onestroke->connectmax++;
onestroke->lasttouch = node->ID;
    }
    else if (node->ID < ICON_MAX + 2) {
// 結線するノードのリストに加える
onestroke->connestlist[connectmax] = node;
onestroke->connectmax++;
onestroke->lasttouch = node->ID;
    }
    else if (onestroke->connestlist[connectmax-1]->ID != node->ID) {
// 結線するノードのリストに加える
onestroke->connestlist[connectmax] = node;
onestroke->connectmax++;
onestroke->lasttouch = node->ID;
    }
```

```
}
    }
    // マウスカーソルがノードから外れたら・・・
    else
onestroke->lasttouch = -1;
}

// ■■一筆書き手法で、マウスの軌跡を「リセットする」メソッド（マウスボ
// タンを離した時にだけ呼ばれる）
void OneStroke::ResetMouseLocus(OneStroke* onestroke) {

    for (int i = 0; i < LOCUSMAX; i++) {
onestroke->locusX[i] = -1;
onestroke->locusY[i] = -1;
    }
    onestroke->indexmax = 0;

    for (int j = 0; j < CONNECT_NODE_MAX; j++)
onestroke->connestlist[j] = NULL;
    onestroke->connectmax = 0;
    onestroke->lasttouch = -1;
}




//**************//

int ModelAlloc(stmode *mp){
  mp->text=(sttext *)malloc(sizeof(sttext)*TEXT_MAX);
  mp->vert=(stvert *)malloc(sizeof(stvert)*VERT_MAX);
  mp->norm=(stnorm *)malloc(sizeof(stnorm)*NORM_MAX);
  mp->poly=(stpoly *)malloc(sizeof(stpoly)*POLY_MAX);
  mp->mate=(stmate *)malloc(sizeof(stmate)*MATE_MAX);
  mp->grou=(stgrou *)malloc(sizeof(stgrou)*GROU_MAX);
  mp->active=1;
  mp->bound=0;
  mp->group=0;

  mp->vector.x=mp->vector.y=mp->vector.z=0.0;

  mp->matrix.m[ 0]=1.0;mp->matrix.m[ 1]=0.0;mp->matrix.m[ 2]=0.0;mp->matrix.m[ 3]=0.0;
  mp->matrix.m[ 4]=0.0;mp->matrix.m[ 5]=1.0;mp->matrix.m[ 6]=0.0;mp->matrix.m[ 7]=0.0;
  mp->matrix.m[ 8]=0.0;mp->matrix.m[ 9]=0.0;mp->matrix.m[10]=1.0;mp->matrix.m[11]=0.0;
  mp->matrix.m[12]=0.0;mp->matrix.m[13]=0.0;mp->matrix.m[14]=0.0;mp->matrix.m[15]=1.0;
  return 0;
}

// glview.cc から。そのうち使うと思う
```

```
void SetModelView(stmode* mp, stmode* modelsp){
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  glRotatef(VROT,1.0,0.0,0.0);
  glTranslated(modelsp->vector.x,modelsp->vector.y,modelsp->vector.z);
  glMultMatrixd((GLdouble*)&modelsp->matrix);
  if(mp!=modelsp){
    glTranslated(mp->vector.x,mp->vector.y,mp->vector.z);
    glMultMatrixd((GLdouble*)&mp->matrix);
  }
}


void SetPureModelView(stmode* mp, stmode* modelsp){
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  if(mp!=modelsp){
    glTranslated(mp->vector.x,mp->vector.y,mp->vector.z);
    glMultMatrixd((GLdouble*)&mp->matrix);
  }
}

/*
void ModelPrepare(stmode* mp){
  glNewList(modelsp-mp+gltop,GL_COMPILE);
  DrawRmd(mp,0);
  glEndList();
}
*/

void ModelDraw(GUINodeCore* guinode, GUINodeCore* world){
  SetModelView(&(guinode->shape), &(world->shape));

  if(!guinode->invisible){ // 表示して良ければ
    if(guinode->surface) // 表面
      glCallList(guinode->displaylist1);
    if(guinode->transparent) // 透明表示
      glCallList(guinode->displaylist2);
    if(guinode->wireframe) // ワイヤフレーム表示
      glCallList(guinode->displaylist3);
  }
}

void SetModelShadowView(stmode* mp, stmode* modelsp){
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  glRotatef(VROT,1.0,0.0,0.0);
  glTranslated(-modelsp->vector.x,-1.0-modelsp->vector.y,-modelsp->vector.z);
  glMultMatrixd((GLdouble*)&modelsp->matrix);
  glScalef(1.0,0.0,1.0);
  glTranslated(mp->vector.x,0.0,mp->vector.z);
```

```
  glMultMatrixd((GLdouble*)&mp->matrix);
}

void ShadowDraw(GUINodeCore* guinode, GUINodeCore* world){
  SetModelShadowView(&(guinode->shape), &(world->shape));
  glCallList(guinode->displaylist1);
}


/**** paint.cc から ***/
short color_white[] = {(short)255, (short)255, (short)255};
short color_black[] = {(short)0, (short)0, (short)0};
short color_red[] = {(short)255, (short)0, (short)0};
short color_orange[] = {(short)255, (short)128, (short)0};
short color_yellow[] = {(short)255, (short)255, (short)0};
short color_green[] = {(short)64, (short)255, (short)0};
short color_blue[] = {(short)0, (short)0, (short)208};
short color_purple[] = {(short)208, (short)0, (short)208};
short color_cyan[] = {(short)0, (short)208, (short)208};

void ChangeColor(stmode* mp, int layer, short color[3])
{
  //色変更
  mp->mate[layer].data[0] = *color++;
  mp->mate[layer].data[1] = *color++;
  mp->mate[layer].data[2] = *color;
}
```