〔非公開〕

ＴＲ－Ｃ－０１４７

臨場感通信会議における人物動作のリアルな実時間再現方式

成山　桂一　Keiichi NARIYAMA
カランシャー　シン　Karansher SINGH
大谷　淳　Jun OHYA
岸野　文郎　Fumio KISHINO

１９９６　３．１５

ＡＴＲ通信システム研究所

臨場感通信会議における人物動作のリアルな実時間再現方式

成山桂一　カランシャー・シン　大谷淳　岸野文郎

ＡＴＲ通信システム研究所

あらまし

臨場感通信会議における自然な人物全身動作の実時間再構成法を提案し、検討を行った。臨場感通信会議では、少数のセンサデータから全身像の動作を推定し、再現する必要がある。人物モデルを、骨格構造に対応するリンク機構としてモデル化すると共に、各人体パーツをワイヤーフレームモデル（WFM）だけでなく、Free Form Deformations（FFD）とImplicit Function Primitive（IFP）とを組み合わせてモデル化する。このモデルは、WFMの高速表示性とFFD及びIFPの自然な変形表現能力とを併せて有している。人物の上半身に装着された少数センサから、通信会議状況の限定動作を対象に、全身動作を推定する手法を提案した。実験により、FFDとIFPの変形表現能力の有効性を確認した。さらに、6自由度計測が可能な4個の磁気センサを上半身に装着し、着座から起立，お辞儀に至る遷移動作の再構成を8フレーム／秒で行えることを確認した。

1．まえがき

近年、人とコンピュータ，あるいは人と人との間の高度な情報伝達手段として、仮想現実感・仮想環境といった技術が注目され、ヒューマンインターフェースやマルチメディア，映像通信などの分野で急速に研究が進められている[1,2]。筆者らは、遠隔地にいる人々があたかも一堂に会するかのような感覚で会議や協調作業を行うことができる環境の提供を目的に、臨場感通信会議システム[3]を提案し、その実現に向けて研究を進めている。臨場感通信会議では、Computer Graphics（CG）技術により生成された3次元仮想空間において、実時間で検出された会議参加者の3次元的動き情報を、CG技術により生成された3次元人物モデルにおいて実時間で再現必要がある。

従来の筆者らのシステム[4]では、3次元人物モデルをワイヤーフレームモデル（WFM）により構築し、胴体，腕などの各人体パーツを剛体として扱っていたため、高速の3次元表示には適しているものの、胴体の屈曲や関節の回転等の動作の再現が不自然となる問題があった。また、従来のシステムでは人物の上半身のみを扱っていたため、下肢の運動を含めた全身動作を再現できないという問題があった。

一方、コンピュータグラフィックスの分野では、人物像の全身動作の表現に関して多くの研究が行われている。しかし、これらの研究では動きの生成処理のみを対象としたものが大半であり[5,6]、実時間で人物の全身の動きを検出し、再現する手法の検討例は非常に少ない[7,8]。

人物の全身像以外では、人物の胴体動作の再現に関して、Monheitら[9]が解剖学的な見地から脊椎をモデル化し、脊椎の運動方程式を導出，計算機にインプリメントすることにより、骨格レベルにおける胴体運動のシミュレーションを行った。しかし、実時間性およびデータの入

力方式に困難な問題があった。また、人物の動作に伴う筋肉の変形の再現に関して、ChadwickらはFree Form Deformation（FFD）手法[10]をCGモデルの形状変形に適用することにより、上腕における筋肉の盛り上がりをシミュレーションにより確認したが、実時間での動き検出，動作の再構成は行わなかった。一方、WyvilらはImplicit Function Primitive（IFP）[11]を用いることにより、柔軟物体の変形表現を実現したが、Ray tracing等によるレンダリングが必要であり、実時間性に問題があった。

本論文では臨場感通信会議への適用を目指し、人物の装着した少数のセンサから検出した動き情報に従い、リアルな動作再現を実時間で行うことが可能な手法の検討を行う。まず、従来用いていたWFMに、FFDとIFPとを組み合わせた３次元人物モデルを提案する。これにより、３次元WFMの特徴である高速表示能力を維持しつつ、FFDおよびIFPの有する特徴である自然な形状変形が実現できる。さらに、効率的な人体パーツと物体との衝突検出および衝突による変形表現が可能となる。

本論文では、２章で詳述するように、上半身に少数のセンサを装着し、全身動作の再構成を目指す。従って、下半身の運動を含めると、センサから得られる情報のみでは自由度が不足する。このため、本論文では、通信会議という状況における主要な人物動作として

１）頭および両腕の動きを含めた着座状態における上半身動作

２）着座から起立，お辞儀に至る遷移運動

を扱う。これにより、２）の拘束条件を用いて、不足した自由度を推定することにより、全身の姿勢を決定できる。

以下、２章では従来の臨場感通信会議における実時間動き検出方式と人物像表示，および、本論文の手法の考え方について述べる。３章では本論文で提案する人物の３次元モデルについて説明する。４章では少数センサ情報からの全身動作の再現方式について述べ、５章では提案方式を用いた実験の結果と考察について述べる。

２．臨場感通信会議における人物像表示

臨場感通信会議システムにおける３次元人物像処理は、人物像の３次元モデルの生成（モデリング），人物像の動きの検出（検出系），および人物像の生成（再現系）の３つのモジュールから構成されている[4]。会議参加者の３次元モデルは、通信会議に先立って作成しておいて受信側に配置し、会議中は送信側の参加者の動き情報を実時間で検出して受信側に送信し、受信側では検出された動きに応じて３次元人物モデルを変形，カラーテクスチャをマッピングし、受信側の参加者の視点に合わせて人物像を仮想の会議室の情景中に合成，３次元ディスプレイに表示する。

人物の３次元モデリングは以下のようにして行う。即ち、頭や胴体といった人体のパーツ毎に３次元WFMを作成し、これらのパーツを人体の構造に基づいて接続することにより、人物の全身の３次元モデルを作成する。人体パーツの３次元WFM作成のため、筆者らはCyberware社製カラー３次元ディジタイザ[4]を用いている。同ディジタイザは測定対象の周囲を回転しながら線状のレーザ光を照射し、その変形を計測することにより測定対象の３次元形状情報を入力すると共に、測定対象表面の色彩情報（カラーテクスチャ）も併せて獲得する。得られた３次元形状情報は、精度に応じた大小の三角形（三角パッチ）に変換され、３次元WFMが作成される。さらにカラーテクスチャを対応する場所の三角パッチにマッピングすることにより、

表１　動作検出方式の比較

	画像処理	磁気センサ（有線式）	磁気センサ（無線式）	ジャイロセンサ
検出精度	△	◎	◎	○
安定性	△～×	◎	◎	◎
検出速度	△	○	◎	○
検出可能距離	◎	○	○	◎
誤差の累積	○	◎	◎	△～×
センサ形状，重量	―	◎	×	○
システム規模	○	◎	△	△

各パーツの３次元モデルを作成する。

以上のようにWFMを用いて人物モデルを生成しているため、グラフィックス・ワークステーションのテクスチャマッピング用ハードウェアが利用ができる。従って、高速３次元表示が可能である反面、柔軟物体の変形表現は困難であるという問題があった。

一方、動き検出部では、実時間で人物の動きを検出する必要がある。人物の動きを検出する方式は、表１に示すように分類される。ユーザのセンサ装着等の負担を考慮すると、画像処理により非接触で人物動作を検出する方式が望ましいが、現状では精度や検出速度の面で適用は

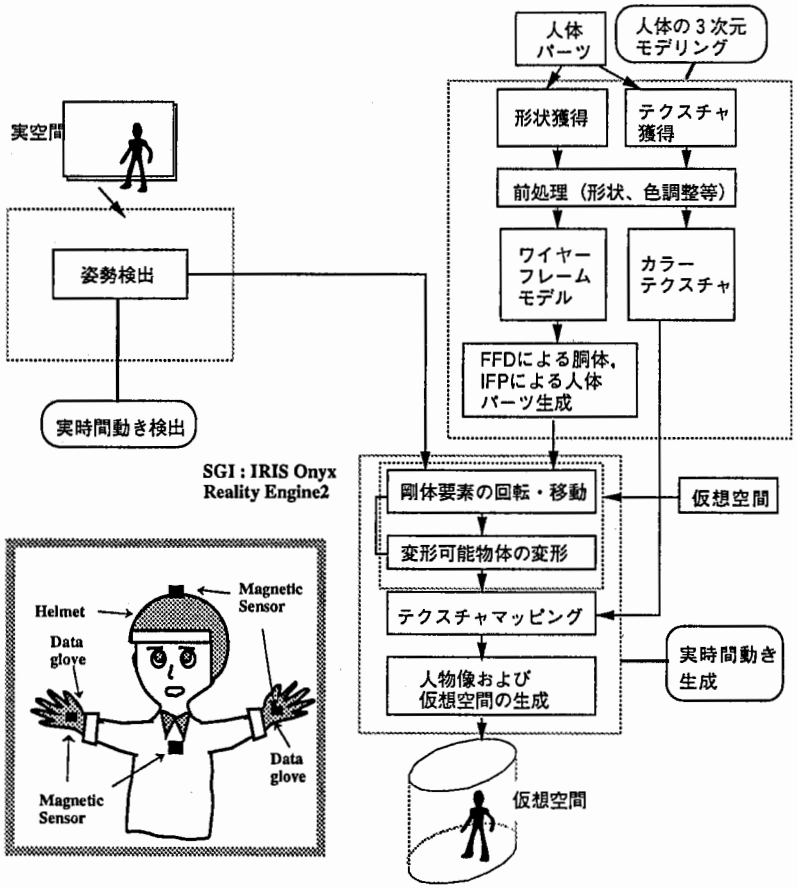


図１　臨場感通信会議における人物像処理の流れ

困難である。また、人体表面に適当な数の小円盤状等のマーカーを装着し、テレビカメラ画像中で追跡する方式も見られるが、マーカーの隠れへの対処法や計測精度に課題が残る。センサを装着する方式は、磁気センサとジャイロセンサの２方式に大別される。ジャイロセンサでは、検出される角加速度データを時間積分することにより３次元座標を計算するため、表１に示すように誤差が蓄積される問題がある。磁気センサに関しては有線および無線の２方式があるが、無線式ではセンサの形状および重量の面においてユーザの負担が問題となり、本論文の目的には適さない。有線式では検出速度の面で無線式に若干劣るが、30HZでの検出が可能である。以上の点を考慮し、本論文では動き検出方式として有線式の磁気センサを採用する。現在利用可能な有線式磁気センサ装置としては、４個の磁気センサからの情報を一括して検出可能なものがあるので、これを利用することにする。

本論文で提案する方式のブロック図を図１に示す。既に述べたように、人物のモデリングでは従来のWFMのみを利用する方式に代わるものとして、WFMにFFDとIFPを組み合わせる新しい３次元人物モデルを提案し、従来システムの問題の解決を目指す。また、動き検出部では、４個の磁気センサを人物に装着することにより、実時間で姿勢を検出する。この検出情報に基づき、４章で述べる手法により全身姿勢を推定するというアプローチをとる。

３．人物の３次元モデリング

３．１　３次元人物モデルの構成

本論文においては、３次元人物モデルを(1)リンクモデル，(2)WFM，(3)FFDモデルとIFPモデル、より構成された階層構造モデルとして定義する。

リンクモデルは図２に示すように、人物の骨格構造に対応するものであり、センサから得られる情報とリンクモデルの有するリンク長などのパラメータに基づいて、全身の姿勢推定に用いられる（４章参照）。

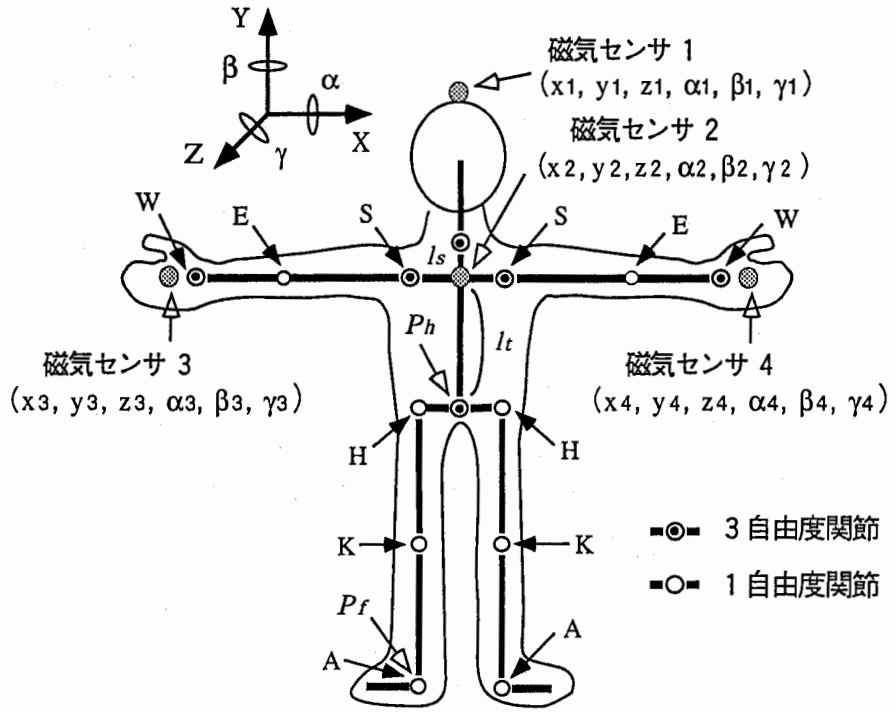


図２　人物骨格モデル

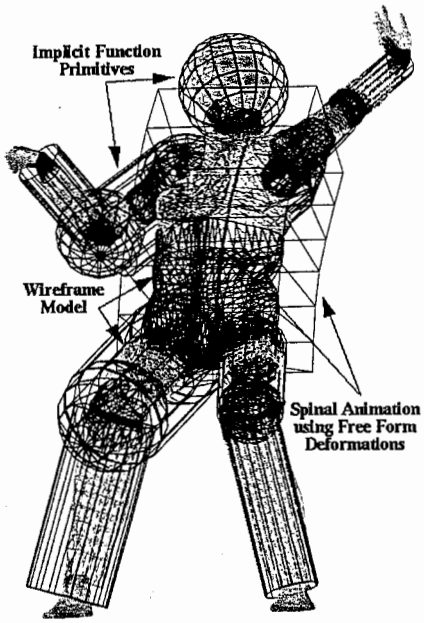


写真１　３次元人物モデル

３次元人物WFMは、２章で述べたように、皮膚や衣服の表面形状を近似するものである。各人体パーツの３次元WFMは、リンクモデルの対応する部分を覆う形で配置され、隣接したパーツ同士のWFMが接続される。そして、カラーテクスチャが各WFMを構成する三角パッチにマッピングされる。

FFDモデルおよびIFPモデル（図１）は、３次元人物WFMの３次元形状の変形計算に用いる概念的なモデルであり、写真１に示すようにFFDモデルは胴体パーツ，IFPモデルは胴体以外の人体パーツの記述に用いられる。ここで、FFDとIFPはWFMの各頂点とあらかじめ関連付けておく。動作の再現時には、FFDモデルおよびIFPモデルの３次元形状を変形し、その変形結果に基づいて３次元人物WFMを構成する各頂点を移動することにより、リアルな動作の再現を可能とする。本章では、以下FFDとIFPによるモデル化について説明する。

３．２　FFDに基づく胴体のモデル化

胴体パーツに関しては、脊椎が解剖学的に複雑な多関節構造を有し、回転方向や胴体の部位によって変形の比率が異なるのに対し、筆者らの従来システムにおいては剛体として扱っていたため、胴体パーツの形状変形が不自然となる問題が生じていた。

人間の胴体は、解剖学的には33個（頚椎：７，胸椎：12，腰椎：５，仙骨：５，尾骨：４）の脊椎骨によって構成されており、各脊椎骨を繋ぐ関節の回転により姿勢変更を行っている。これらの脊椎骨のうち、頚椎は首の動きに関係し、仙骨と尾骨は骨盤に含まれるため関節は回転せず、実際の胴体の動きに関しては、胸椎と腰椎が関係している。

Monheitらは、胸椎と腰椎の計17個の脊椎骨から構成される脊椎の運動シミュレータを作成したが[9]、演算処理の増加と描画サイクル時間の低下が問題となり、臨場感通信会議への適用は困難である。一方、胸椎および腰椎を繋ぐ各関節の可動範囲は狭く、ほぼ一様な角度で回転するという特徴がある。よって本論文では、胴体の３次元WFMは、胸部と腰部の領域に応じて鉛直方向に２分割し、さらに各領域を３分割することにより、写真１に示すように計６個のセグメントに領域分けする。

自然な胴体の屈曲動作を再構成するためには、隣り合うセグメントの接合面での連続性が要求される。本論文では、胸部および腰部を構成する３次元WFMの各頂点に対して変形用の”重み”を設定し、Free Form Deformation（FFD）手法[10]を胴体パーツの変形に適用することにより、隣り合うセグメントとの連続性を保持しつつ、滑らかな３次元形状の変形を目指す。FFD手法を用いた胴体パーツの変形方法に関しては、４．２において詳述する。

３．３　IFPによるモデル化

従来システムでは、各人体パーツをWFMで記述し、剛体として扱っていたため、変形後の関節周辺の形状に”窪み”等が発生し、人体パーツの滑らかな接続が困難であった。また、関節の回転に伴う筋肉の盛り上がり等を再現することが困難であった。

リアルな関節の回転動作を再構成するためには、人体パーツを柔軟な物体として取り扱うと共に、高度な変形表現が要求される。よって本論文では、関節を含む人体パーツの表現において、従来の３次元WFMと併せて、Implicit Function Primitive（IFP）[11]を用いて記述する。IFPとしては、写真１に示すように球や円筒のような３次元形状が用いられる。

Implicit Function手法は柔軟物体の変形表現や衝突検出の計算に適しているものの、３次元表示のためにはRay tracingのような処理が必要であり、高速表示には適していない。そこで、人体パーツをIFPを用いて記述すると共に、IFPの表面上に３次元WFMの各頂点との対応点を設定し、形状の変形時においては、IFPの変形結果に基づいて３次元WFMの各頂点を移動させる。これにより、３次元WFMの特徴である表示の高速性を維持しつつ、筋肉の変形を含む”リアル”な関節および人体パーツの形状変形が可能となる。Implicit Function手法を用いた人体パーツの変形方式に関しては、４．３において詳述する。

４．少数センサ情報からの動作再現

４．１　リンクモデルによる上半身の姿勢推定

人物全身の骨格構造は、図２に示すように１または３自由度を有する回転関節と、実際の人物と同じ長さを有するリンクにより、基準点の３次元座標値および26自由度の回転関節を有する計29自由度のリンク機構として定義できる[12]。リンクモデルにおける各リンクに対して１つの座標系を割り当て、リンク座標系n-1からリンク座標系nへの x, y, z 軸の回転角度をそれぞれ α, β, γ とし、リンクの長さをdとすると、座標変換を表わす同次変換行列Rnは

$$Rn=\begin{pmatrix} C\beta C\gamma & S\alpha S\beta C\gamma - C\alpha S\gamma & C\alpha S\beta C\gamma + S\alpha S\gamma & 0 \\ C\beta S\gamma & S\alpha S\beta S\gamma + C\alpha C\gamma & C\alpha S\beta S\gamma - S\alpha C\gamma & d \\ -S\beta & S\alpha C\beta & C\alpha C\beta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad (1)$$

となる。ここで、SとCはsin関数とcos関数を表わす。

更に、基準座標系をX_0，リンク座標系nにおけるリンク先端の位置ベクトルをX_nとすると

$$X_0 = R_1 R_2 \cdots\cdots R_n X_n \qquad (2)$$

となり、行列Rnを用いることにより、リンクモデルの幾何学的構造を記述できる。

２章で述べたように、実時間で人物全身の全ての関節回転自由度を検出することは困難であるため、本論文では、３次元座標および３軸回転角度の計６自由度が検出可能な有線式磁気セ

ンサを装着し、限られたセンサ情報から全身姿勢の推定を行う。具体的には図２に示すように、胸，頭および両手甲に計４個の磁気センサを装着し、各センサより得られる３次元座標，３軸回転角度の計24個のデータから、後述のように拘束条件を用いることにより、前述の全身の姿勢に関する29自由度を推定する。

図２に示すリンクモデルにおいては、胸に装着した磁気センサ２の位置を基準座標原点としており、磁気センサより検出された情報により、胴体の６自由度，頭に装着した磁気センサより頭の３自由度が一意に求まる。

上肢の関節角度の推定のためには、肩Sの位置を基準座標としているので、センサ２の３自由度（$\alpha_2, \beta_2, \gamma_2$）と肩のリンク長$l_s$を用いることにより、肩Sの３次元座標を算出する。上肢（図２でS-E-Wのリンク）は７自由度を有するリンク機構としてモデル化できる。従って、手に装着した磁気センサ３から得られる６自由度データより、これら７自由度を推定することは、１自由度が不足しているため、解が無限に存在する不良設定問題となり、解が一意に求まらない。この問題を解決するため、拘束条件として上肢の運動に関して運動エネルギーを定義し、エネルギー最小化により関節角度を一意に決定する。詳細は文献[4]に譲るが、各関節を回転角度に比例して反発力を生じる回転バネ，リンクを質量を有する円柱としてモデル化し、上肢モデルの運動エネルギーを、重力によるポテンシャル，回転バネによるストレスおよび摩擦，モデルの移動に伴う運動エネルギーの計４エネルギーの総和として定義し、冗長な１自由度を運動エネルギー最小の条件で決定する。本手法により、誤差３cm以内で肘E（図２）の３次元座標を推定すると共に、自然で滑らかな上肢運動の再構成が確認されている[8]。

以上により、図２のリンクモデルにおける下肢の６自由度（尻(H)：２，膝(K)：２，足首(A)：２）を除く23自由度が推定できる。

４．２　胴体パーツの変形

胴体の屈曲動作の再構成に関しては、磁気センサ２（図２）から得られる胸の回りの３軸回転角度（$\alpha_2, \beta_2, \gamma_2$）を、胴体パーツの各セグメントの変形比率（５．１参照）に基づいて分割して各セグメントに分配し、各セグメントの形状を変形する。この変形には、Free Form Deformations（FFD）手法[6]を使用する。

FFDは与えられた変位量（座標，回転角度）に応じて３次元WFMを構成する各頂点の変位量を自動的に計算し、計算結果に基づいて頂点を移動することによりモデルの変形を行う。即ち、３．２で述べた胸部および腰部を構成する３次元WFMに対して、図３に示すように各セグメントの接合面の中心に制御点C_n（C_{nx}, C_{ny}, C_{nz}），（n=0～3）を設定する。３次元WFMの頂点P（p_x, p_y, p_z）の変形のための”重み”をt（t=0～1）とすると、FFDによる変換後の３次元座標P'は

$$P' = \begin{pmatrix} p_x - C_{0x} \\ y \\ p_z - C_{0z} \end{pmatrix} + \begin{pmatrix} C_{0x} & C_{1x} & C_{2x} & C_{3x} \\ C_{0y} & C_{1y} & C_{2y} & C_{3y} \\ C_{0z} & C_{1z} & C_{2z} & C_{3z} \end{pmatrix} \begin{pmatrix} (1-t)^3 \\ 3t(1-t)^2 \\ 3t^2(1-t) \\ t^3 \end{pmatrix} \qquad (3)$$

で表される。得られた座標値P'に式(1)の同次変換行列を乗算することにより、３次元空間中のWFMの頂点の座標が求められる。

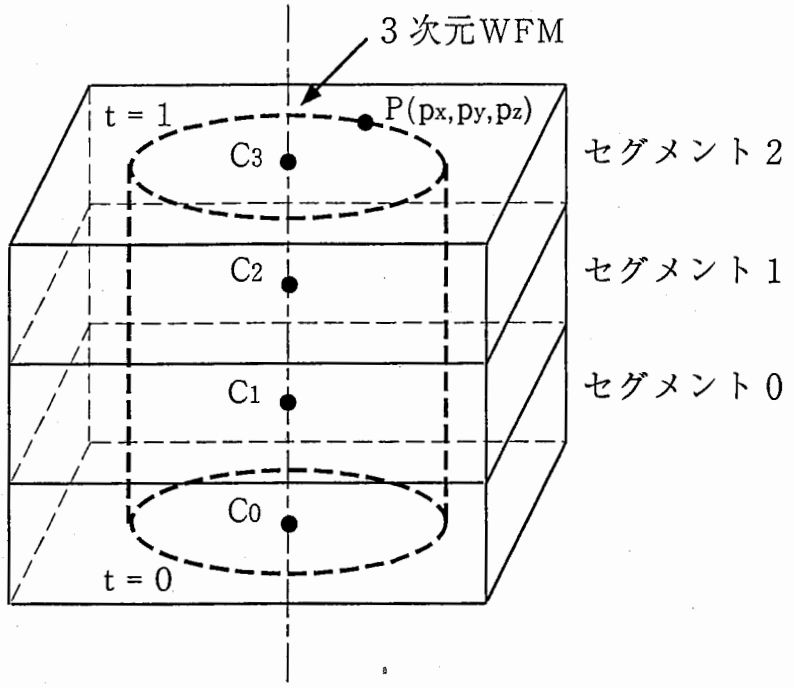


図3　胴体パーツへのＦＦＤ手法の適用

4．3　胴体以外の人体パーツの変形

肘や膝などの関節を含む部位の変形においては、人体パーツを従来の3次元WFMと共に、Implicit Function Primitive（IFP）を用いてモデル化する。

図4に示すようにIFP は球（sphere）や球面円柱（sphylinder）などの基本的な幾何学形状により表現され、体積V，骨格S（球では球の中心），および密度関数f(t)を有している。ここでtは距離率（distance ratio）を表し、同図でVの内部の点をp，S上におけるpとの最短距離を示す点をc，cからVの表面までの距離をrとすると、距離率tは次式で表される。

$$t = |p - c| / r \quad (0 \leqq t \leqq 1) \qquad (4)$$

密度関数f(t)は図4に示すように、tを0から1の範囲に写像する3次関数として表現される。さらに、同図に示すように

$$f(t) - T = 0 \qquad (5)$$

を満たす曲面はImplicit Surface（IS）とよばれ、ISが対象物の3次元形状を近似するように(5)式を設定する。(5)式において、TはImplicit Function の閾値とよばれる。

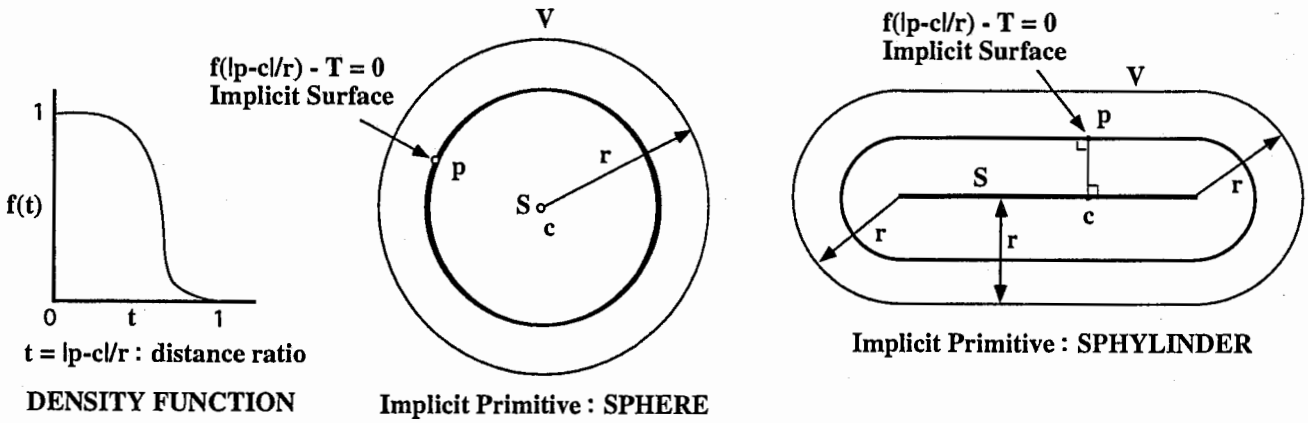


図4　Implicit Function Primitive の定義

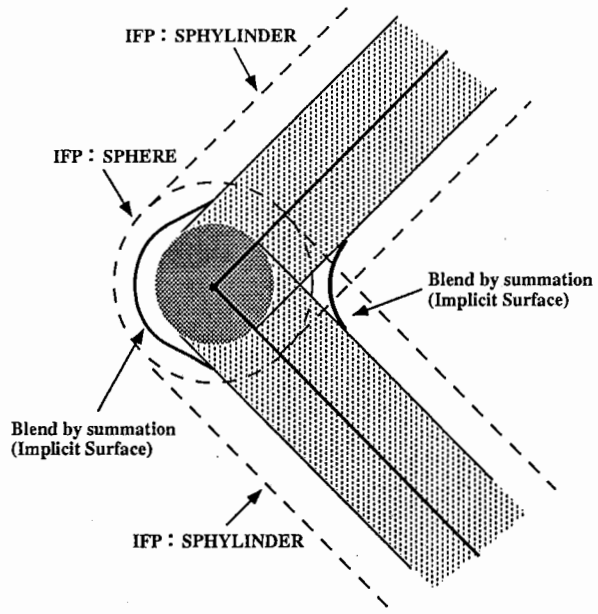


図５　**IFP** による関節のモデル化手法

本論文では、写真１に示すように関節部を球，人体パーツを球面円柱のIFPにより記述し、これらを適宜連結する。ここで、各IFPのISが対象パーツの３次元WFMの構造を極力近似するようにモデル化する。

パーツの変形は、モデルを構成する各IFPの密度関数fi(t)（iは各IFPに対応）を用いてBlending関数を定義し、この関数によりIFPモデルのISの３次元形状を変形する。即ち、パーツのBlending関数をF(t)とすると

$$F(t)=\sum_{i=1}^{N} f_i(t)-T \qquad (6)$$

で表される。ここでTは閾値を表し、０から１の範囲の値をとる。またNは図５に示すように点pが含まれるIFPの数を表す。(6)式において、閾値Tの値を最適化することにより、IFPモデルの各ISは滑らかに結合される（図５参照）。さらに各パーツの３次元WFMの各頂点を、対応するISの座標に駆動することにより、関節を含む部位における３次元WFMの滑らかな変形を実現する。

人体パーツの変形においては、筋肉の動作による変形表現が重要である。筋肉のモデル化の一例である上肢のモデル化においては、図６に示すように上腕に筋肉プリミティブとして楕円球のIFPを割り当てる。筋肉プリミティブは、肘の回転角度に応じて平行移動され、(6)式を適用することにより上腕プリミティブと滑らかに結合される（図６参照）。本手法を用いることにより、リアルな筋肉の盛り上がりを伴う肘関節の回転動作の再構成を実現する（５．２参照）。

IFPは物体と効率的な衝突検出が可能である。図７に示すように物体（人物像の場合は人体パーツ）の数をm，物体PmのWFMの頂点数をnmとすると、全物体の頂点数の総和は

$$n=\sum_{i=1}^{m} n_i \qquad (7)$$

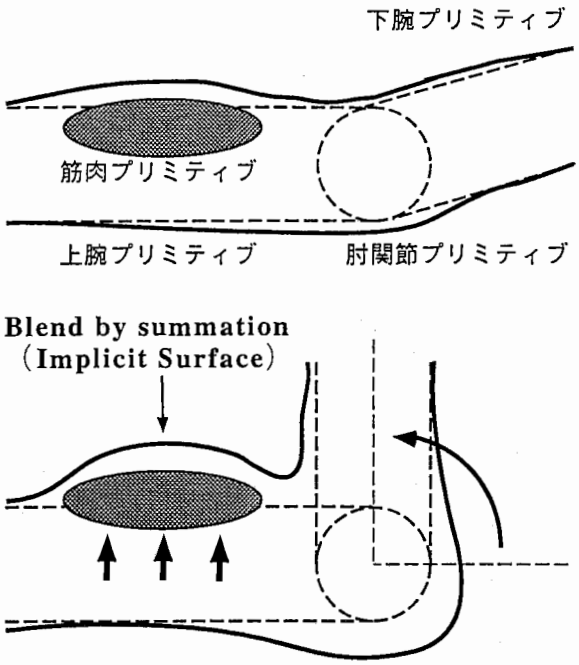


図6　筋肉の動き再現方式

で表され、IFPを用いていない場合には、全頂点総当たりの衝突チェックが必要となり、衝突チェックに要する回数はn^2のオーダーとなる。一方、IFPを用いる場合は、IFPの衝突の検出に要する演算回数はm^2のオーダーである。このチェックにより、衝突している可能性があると判断された場合には、PiとPjのWFMの頂点の衝突チェック及びPjとPiの頂点のチェック、即ちチェックの回数はnのオーダーである。よって、提案手法による衝突検出に要する演算回数のオーダーは

$$m^2 + mn \qquad (8)$$

となる。例えば、３次元人物WFMにおいてはm=20, n=13,000であり、

$$m^2 + mn\ (\fallingdotseq 10^4) \ll n^2 (\fallingdotseq 10^{10}) \qquad (9)$$

となる。このように、IFPモデルを用いることにより、衝突による形状変形表現と共に、効率的な衝突検出が可能である。

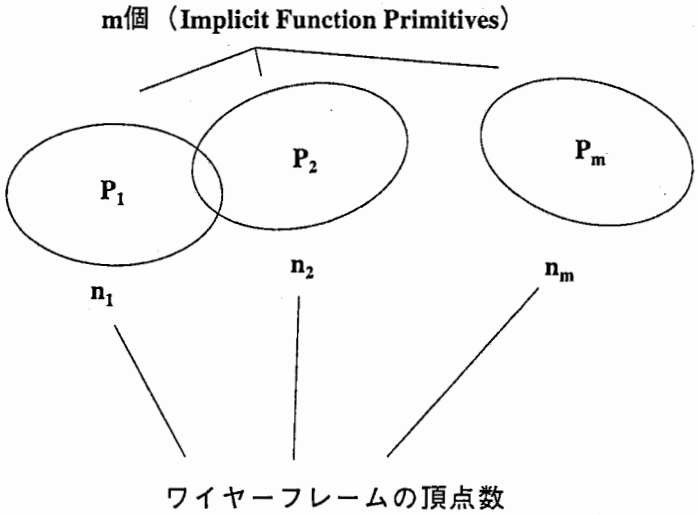


図7　効率的な衝突検出

4．4　下肢の姿勢推定と動作の再現方式

下肢の姿勢の推定においては、図2に示す胸の磁気センサ2から得られる角度データ（α_2, β_2,γ_2）と胴体のリンク長l_tを用い、腰の3次元座標P_hを求める。本論文で扱う下肢の運動を伴う全身動作は、2章で述べたように、着座から起立に至る遷移運動である。両脚は図8に示すように3自由度（尻，膝および足首）を有するリンク機構としてモデル化されているので、これらの3自由度は、算出された腰の3次元座標P_hと、以下に記す拘束条件を用いることにより推定される。

1）両足は床に固定されている

2）尻，膝および足首の関節回転角度は、それぞれ左右同一の値をとる

3）腰の軌跡はＹＺ平面上（ｘ＝0）を移動する

図8に示すように、胸の磁気センサ2より得られるＹＺ座標値（P_c），腰および足首のＹＺ座標値（P_h,P_f）を用いて長さL_1,L_2を計算し、モデルのリンク長（l_1～l_3）を用いることにより、関節角度α, β, γは幾何学的に計算できる。以上により、人物全身の姿勢に関する29個のパラメータが決定される。推定された姿勢パラメータに基づき、仮想空間において全身動作が以下の手順で再構成される。

4．1で述べたように、3次元人物モデルの構造は同次変換行列（式(1)）により記述できるので、推定された姿勢パラメータを用いて、人体パーツ毎に同次変換行列を計算する。頭などの変形を伴わない人体パーツに関しては、人体パーツに含まれる3次元WFMの頂点の3次元座標に同次変換行列を乗算することにより、WFMの三角パッチが変形，平行移動される。3次元WFMとFFD，またはIFPと組み合わせてモデル化された人体パーツに関しては、まずFFDとIFPの3次元形状を変形し、WFMを構成する各頂点をFFDまたはIFPの対応点の3次元座標に移動することにより、3次元人物WFMを変形する。そしてカラーテクスチャをマッピングし、人物モデルを仮想空間に合成することにより、人物の動きを再現する。

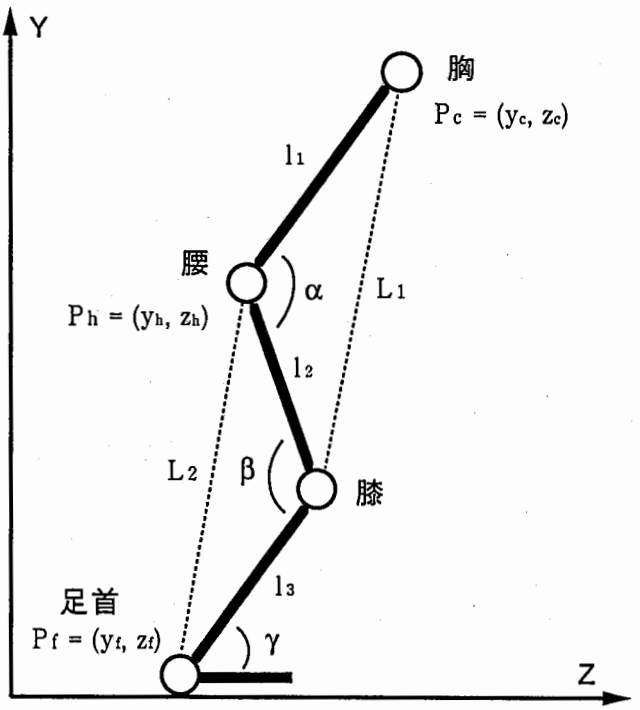


図8　下肢の骨格モデル

５．実験結果と考察

５．１　胴体動作の再構成

図１に示す人物像再現のための実験システムを構築した。人物動作の検出には、Polhemus社製の磁気センサ（FASTRAK）を用い、図２に示すように４個のセンサを頭，胸および両手甲に装着する。動作の再構成には、SGI社製のグラフィックス・ワークステーション（Onyx, Reality Engine2）を用いた。

４．２で提案した３次元人物モデルにおける自然な胴体の屈曲動作の再現を目的として、胴体パーツを構成する６個のセグメントにおける３軸回転方向の変形比率を求めるため実験を行った[14]。

実験方法としては、まず胴体の屈曲動作を被験者の真正面，真横に設置された２台のカメラで撮影，ビデオディスクに収録する。次に撮影された２方向の画像をグラフィックス・ワークステーション（GWS）に取り込み、GWS上で３次元人物 WFM と重ね合わせて表示する。さらに胴体パーツの各セグメントの変形比率を適宜変化させ、写真２に示すように実写画像と３次元WFMとの重なり具合を目視によって評価することにより、各セグメントにおける胴体の変形比率を求める。なお、各回転軸（X, Y, Z）に対する６個のセグメントの変形比率の総和は、それぞれ１と定義している。

実験結果を表２に示す。セグメントの番号については、写真１で示された胴体セグメントの最上部を１として、１から６の番号を割り当てている。表２より、胴体を前後方向に屈曲させるＸ軸回転に関しては、各セグメントとも均等な割合で変形しており、胴体の捻りを表すＹ軸回転，および左右方向への曲げを表すＺ軸回転に関しては、腰部での変形比率が高く、胸部に近づくにつれて比率が低くなっている事がわかる。

実験より得られた胴体の部位における変形比率の分布傾向に関しては、解剖学的な知見[5]とほぼ一致しており、これらの値を胴体パーツの形状変形に適用することにより、自然な胴体の動きが再現可能となる。

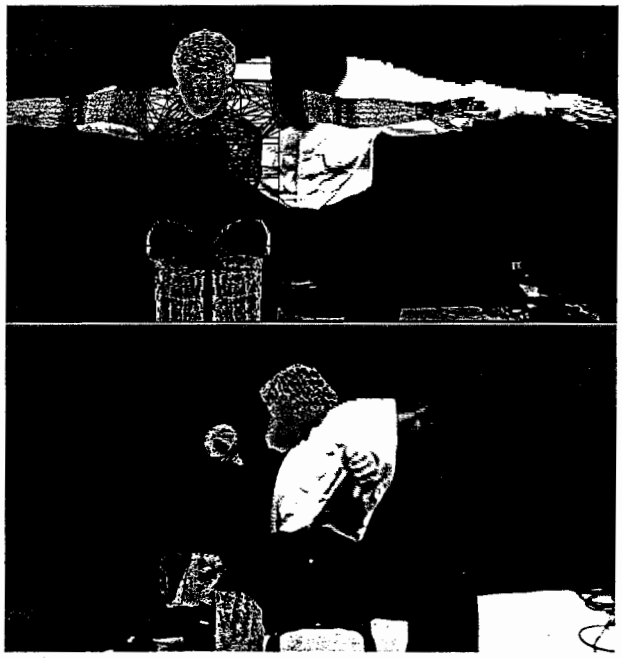


写真２　胴体の変形比率の検討

表２　胴体モデルの変形比率

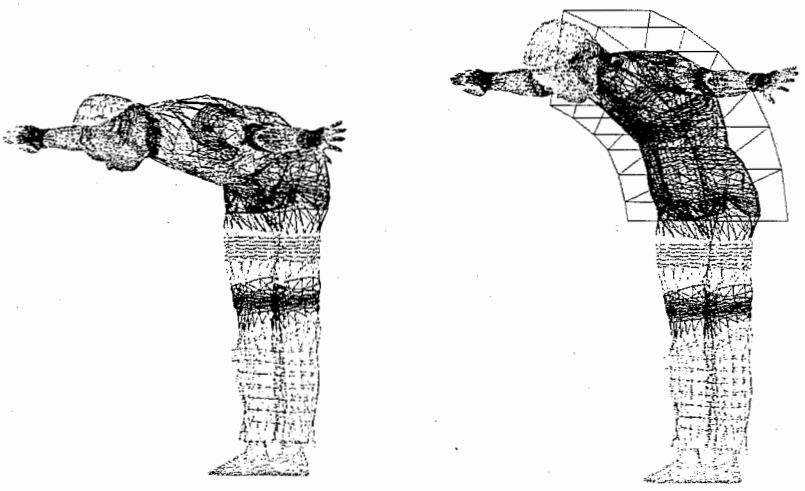
部　　位	X軸回転	Y軸回転	Z軸回転
セグメント１（胸上部）	0.166	0.075	0.100
セグメント２（胸中部）	0.166	0.075	0.100
セグメント３（胸下部）	0.167	0.150	0.150
セグメント４（腰上部）	0.167	0.200	0.200
セグメント５（腰中部）	0.167	0.250	0.200
セグメント６（腰下部）	0.167	0.250	0.250

得られた胴体の変形比率を胴体パーツの各セグメントの変形に適用した。シミュレーションにより胴体の回転角度（$\alpha_2, \beta_2, \gamma_2$）を変化させ、各セグメントの変形比率に基づいて回転角度データを分配し、胴体パーツの３次元形状を変形することにより胴体動作の再構成を行った。実験結果の一例を写真３に示す。同写真（ａ）は従来手法，（ｂ）は提案手法による胴体動作の再構成画像を表している。写真３より、提案手法を用いることにより、滑らかで自然な胴体の屈曲動作の再構成が可能であることが確認された。

５．２　**Implicit Function** による形状変形

４．３で提案した人体パーツの変形方式の有効性を確認するため、シミュレーションによりパーツの３次元形状の変形を行った。

関節周辺の人体パーツの形状変形結果の一例を写真４に示す。同写真において、右肘は従来手法，左肘は提案手法による変形結果を示している。従来手法においては肘関節の内側に”窪み”を生じ、上腕および下腕との接続が不自然となっているが、提案手法を用いることにより、自然で滑らかな形状変形が可能であることが確認できる。



（ａ）　　（ｂ）

写真３　胴体動作の再構成

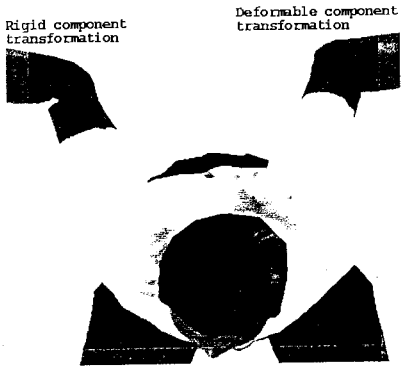


写真４　提案手法による関節の形状変形

次に衝突による形状変形結果の例を写真５に示す。同写真はボールとの衝突によるボールおよび人体パーツの形状変化のシミュレーション結果を示している。(5)式における閾値Tを変化させることにより、パーツの硬度が変更可能（T=0：軟，T=1：硬）となる。ボールおよび人体パーツの閾値をT_1,T_2とすると、写真５の左側１列目から３列目はそれぞれ、T_1を１から０，T_2を０から１に変化させた場合の再構成画像を表している。ワイヤーフレーム表示において、硬度に応じて両パーツが正しく変形していることが確認できる。閾値T_1とT_2の値を調整することにより、様々な硬度を有する物体間における、衝突による形状変形が再現可能となる。

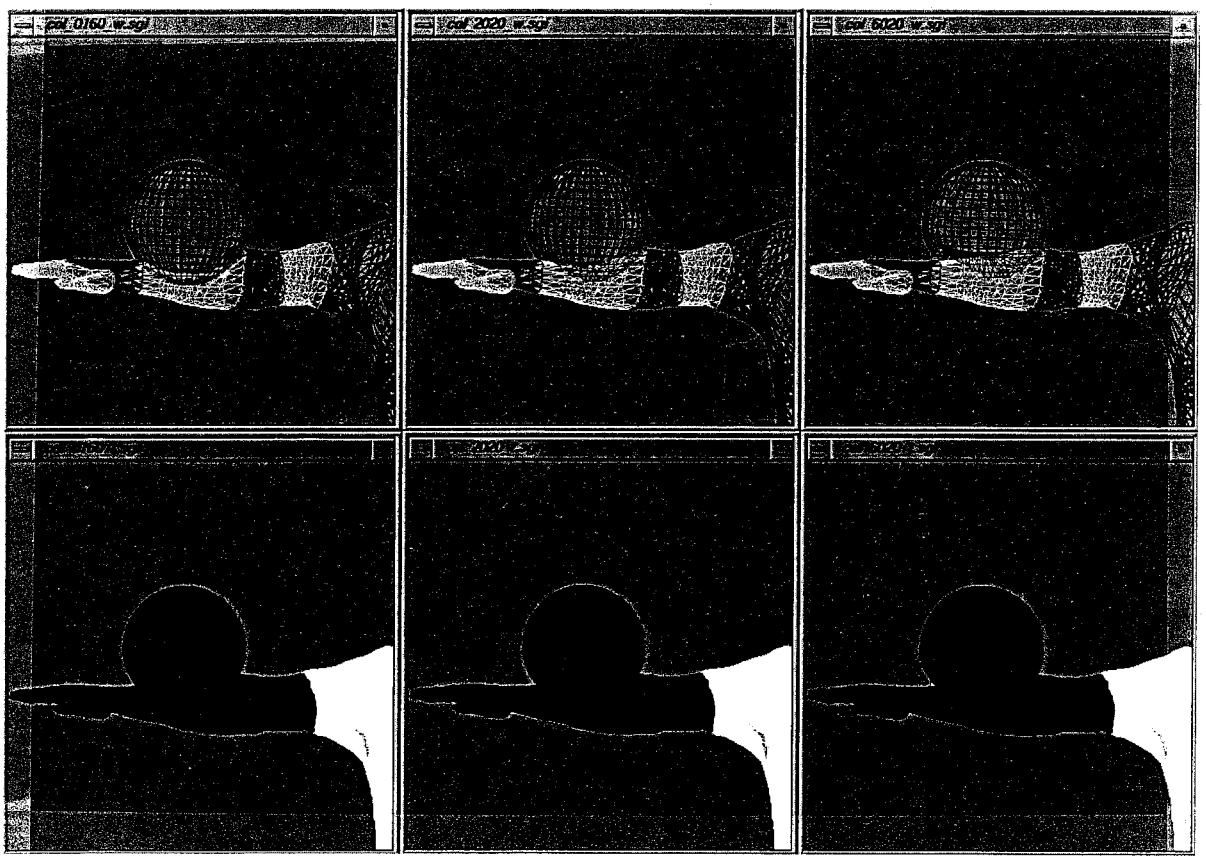


写真５　衝突による形状変形

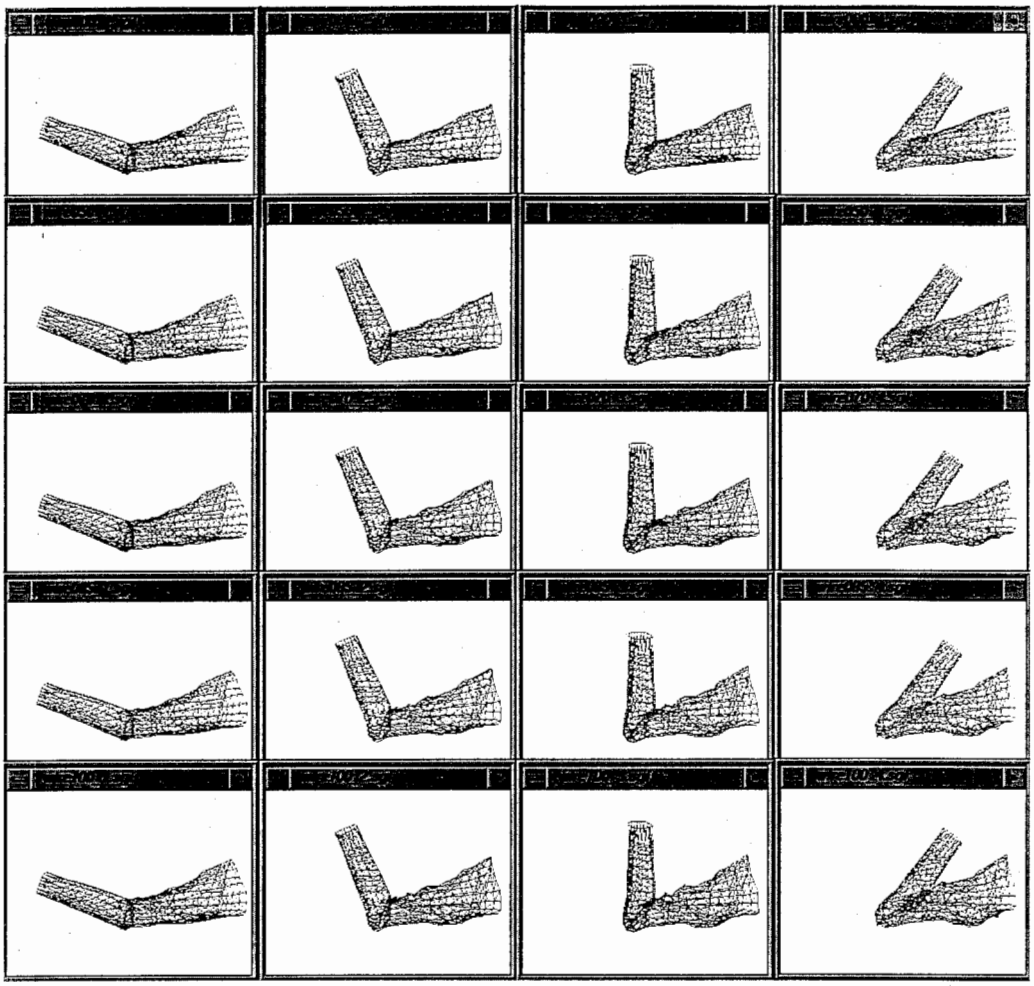


写真６　筋肉の動き再現

さらに上肢モデルにおける筋肉の盛り上がりの再現実験を行った。上腕の筋肉モデルにおけるImplicit Functionの閾値T（式(5)）の値を適宜変化させ、シミュレーションにより肘関節の回転動作の再構成を行った。実験結果を写真６に示す。同写真の１～５段目は、閾値T=0, 0.5, 0.7, 0.85, 1.0の場合の再構成画像を示している。写真６より、同一の回転角度において閾値Tの値を増加させる従い、上腕部の形状が盛り上がっていることが確認できる。人物の体型や性別等により、筋肉の変形状態は異なるが、人物毎に閾値Tの値を調整することにより、実態に近い筋肉の変形が再現可能となる。写真６より、提案手法を用いることにより、リアルな筋肉の盛り上がりを伴った肘関節の回転動作の再構成が可能であることが確認された。

５．３　全身動作の再構成

胸，頭および両手に装着した磁気センサから計測される計24個のデータに基づき、４章で述べた姿勢の推定方式を用いて全身の姿勢に関する29個の自由度を推定し、本論文で対象としている着座から起立，お辞儀に至る全身動作の再構成を行った。全身動作の再構成の結果の例を写真７に示す。提案方式を用いることにより、約８フレーム／秒の速度で自然な胴体の屈曲動作を含む全身動作の再構成が可能であることが確認された。

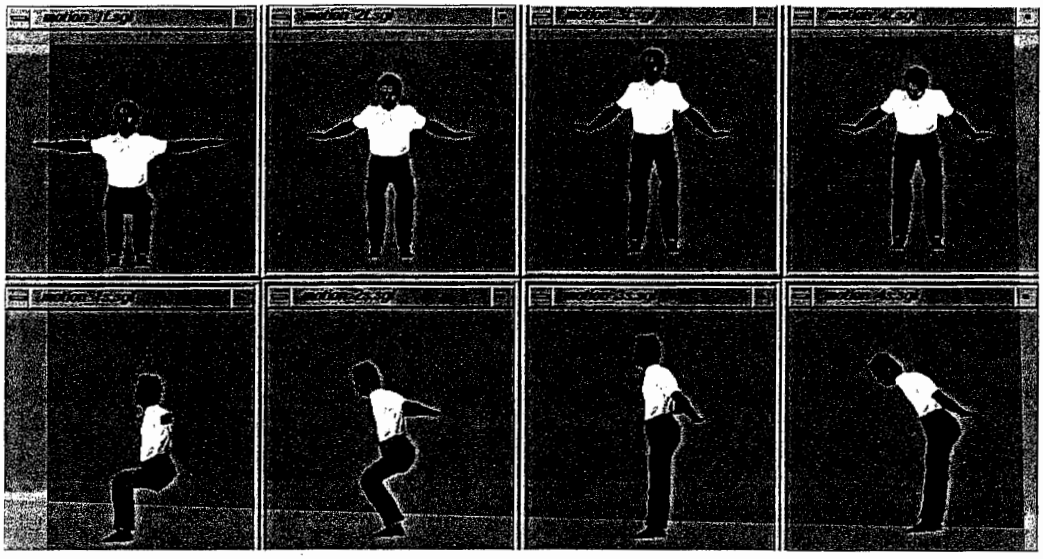


写真７　全身動作の再構成

６．むすび

本論文では、臨場感通信会議における”リアル”な人物動作の実時間再構成を目的として、人物像の３次元モデリング，実時間による動作の検出，実時間による姿勢推定および動作の再現方式について検討した。

自然な胴体の屈曲動作を再現するため、胴体を解剖学的な知見に基づき６個の領域に分割，３次元モデル化し、モデルの形状変形方式としてFree Form Deformations（FFD）手法の適用を提案した。さらに胴体の部位毎の変形比率を求めることにより、胴体の回転の３自由度の計測結果のみを用いて、自然な胴体の屈曲動作の再現結果が得られることを確認した。

関節の回転に伴う筋肉の盛り上がりや物体との衝突による変形など、リアルな人体の形状変形を再現するため、関節を含む人体パーツの表現において、従来の３次元ワイヤーフレームモデル（WFM）と併せて、Implicit Function（IF）手法を用いて記述し、IF手法の有する高度な変形表現と、３次元WFMの特徴である表示の高速性と組み合わせるモデル化方式を提案した。さらに形状変形パラメータを最適化することにより、リアルな筋肉の盛り上がりや衝突における人体および物体の形状変形が可能であることが確認された。

通信会議という状況における人物動作では、下肢の運動に拘束条件を設定できる場合が多いので、頭，胸および両手に装着した４個の磁気センサから得られる計24個の情報から、人物全身の姿勢に関する29自由度を推定する手法を検討し、約８フレーム／秒の速度で全身動作の再構成が可能であることが確認された。

今後は歩行動作を含む任意の全身動作の再構成への拡張を進める予定である。また、現状の磁気センサでは検出距離および使用個数に制限があるため、動作検出方式の検討が必要となる。

参考文献

[1] 廣瀬：”人工現実感の生成”，システム／制御／情報，33，11，pp. 590-597 (1989)

[2] 服部：”人工現実感の世界”，工業調査会， pp. 95-189 (1991)

[3] 岸野，山下：”臨場感通信会議のテレコンファレンス への適用”，信学技報，IE 89-35 (1989)

[4] J. Ohya et al.：”Virtual Space Teleconferencing : real-time reproduction of 3D human images”, Journal of Virtual Communication and Image Re-presentation, Vol. 6,No.1, March, pp. 1-25 (1995)

[5] C. B. Phillips, J. Zhao, N. I. Badler Interactive real-time articulated figure manipulation using multiple kinematic constraints”, Computer Graphics 24 (2), pp. 245-250, 271 (1990)

[6] 鵜沼，武内：”コンピュータアニメーションにおける感情を伴った人間の歩行動作の生成方法”，信学論 D-2，Vol. J76-D-2，pp. 1822-1831 (1993)

[7] 石井，望月，岸野：”人物像合成のためのステレオ画像からの動作認識法”，信学論 D-2，Vol. J76-D-2，pp. 1805-1812 (1993)

[8] K. Nariyama, K. Singh, J. Ohya, F. Kishino：”Realistic 3D Synthesis of Human Body Movements for Virtual Space Teleconferencing”, Proc. of the IASTED International Conference "MODELLING AND SIMULATION”, pp. 102-104 (1995)

[9] G. Monheit, N. I. Badler：”A Kinematic Model of the Human Spine and Torso”, IEEE Computer Graphics & Applications, Vol. 11, No.2, pp. 29-38 (1991)

[10] J. E. Chadwick, D. R. Haumann, R. E. Parent：”Layered Construction for Deformable Animated Characters”, Computer Graphics, Vol. 23, No. 3, pp. 243-252 (1989)

[11] G. Wyvill, C. McPheeters, B. Wyvill：Data structures for soft objects”, Visual Computer, Vol. 2, pp. 227-234 (1986)

[12] K. Singh, J. Ohya, R. Parent：”Human Figure Synthesis and Animation for Virtual Space Teleconferencing”, IEEE Virtual Reality Annual International Symposium, pp. 118-126 (1995)

[13] 伊藤(宏)，伊藤(正)：”生体とロボットにおける運動制御”，pp. 34-43，計測自動制御学会 (1991)

[14] 成山，大谷，岸野：”臨場感通信会議における人物動作の再構成に関する一検討”，信学技報，MVE 95-67 (1996)

（添付資料）

実験に用いたソース・プログラムを以下に記す。

ファイル一覧

実行プログラム作成ファイル

- Makefile : 実行プログラム"gattai"作成用ファイル

ヘッダファイル

- addon.h : FFD, IFP用ヘッダファイル
- body.h : 人物モデル用ヘッダファイル
- body_draw.h : 人物モデル描画用ヘッダファイル
- cyber_glove.h : サイバーグローブ用ヘッダファイル
- fastrak.h : 磁気センサ(Fastrak)用ヘッダファイル
- forms.h : 設定パネル作成用ヘッダファイル
- hirose.h : パラメータ設定パネル用ヘッダファイル
- newmodel.h : IFP形状変形関連ヘッダファイル
- rname_fc.h
- tool.h
- vector.h : ベクトル設定用ヘッダファイル

ソースファイル

- abc.c : 上肢の運動エネルギー計算用ファイル
- addon.c : FFD, IFPを用いた形状変形の計算用ファイル
- body_draw.c : 人体パーツの表示用ファイル
- calc_object.c : 全身の姿勢推定用ファイル
- calib.c : 磁気センサのキャリブレーション用ファイル
- colsp.c : 衝突検出，形状変形の計算用ファイル
- coord_tf.c : 座標変換用ファイル
- data.c
- draw3.c : 人物モデルの３次元描画用ファイル
- get_all_mat.c : 同時変換マトリックス作成用ファイル（磁気センサ用）
- init.c : システム初期化ファイル
- main.c : メインプログラム
- menu.c : ポップアップ・メニューの機能設定用ファイル
- mkrotm.c : 同時変換マトリックス作成用ファイル（人体パーツ用）
- newmodel.c : IFP形状変形関連ファイル
- primrtns.c : IFP表示関連ファイル
- readmap.c : イメージデータ読み込みファイル
- read_cyber.c : センサデータ変換用ファイル（サイバーグローブ）
- read_fastrak.c : センサデータ変換用ファイル（磁気センサ）
- scaling.c : スケール設定ファイル
- xdrcyber.c : データ通信用ファイル（サイバーグローブ）
- xdrfastrak.c : データ通信用ファイル（磁気センサ）

プログラムの実行

キーボードより「gattai」と入力することによりプログラムが実行される。

機能選択

・人物像の表示ウインドウ中で、マウスの右ボタンを押している間、ポップアップメニューが表示される。マウスの右ボタンを押しながら、選択したい機能に移動し、ボタンを離すことにより機能が選択される。

・マウスの操作により、磁気センサのデータを擬似的に与える機能を選択した場合、マウスの左ボタンが平行移動，中ボタンが回転移動に割り当てられており、ボタンを押しながらマウスを移動することにより、人物モデルに与えるデータが変化する（mune, atama, ）

・人物モデルの「移動・ズーム」機能を選択した場合、マウスの左ボタンが平行移動，中ボタンがズームに割り当てられており、ボタンを押しながらマウスを移動することにより、人物モデルが変化する。

プログラムの修正

プログラムを変更する場合は、ソースファイルを修正した後、キーボードより「make」と入力すれば、Makefileにより実行プログラムが作成される。

```
IRISLIB = -lforms -lfm_s -lgl -limage
LIB     = -lm -lmalloc
CC_OPT  = -float -I. -I/home/nariyama/project6/Implicit//mc/include
CC_DEF  = -DIRIS -DIRIS4 -DIRISGT -DUNIX -DDIVPOL -DTANTAI
CFLAGS  = -g
HEADER  = tool.h body_draw.h

OBJS =  main.o init.o draw3.o scaling.o data.o menu.o body_draw.o \
        readmap.o coord_tf.o read_cyber.o read_fastrak.o \
        calc_object.o mkrotm.o get_all_mat.o abc.o \
        cyber/xdrcyber.o fastrak/xdrfastrak.o calib.o addon.o \
        newmodel.o primrtns.o colsp.o hirose.o

.c.o:
        $(CC) $(CFLAGS) $(CC_OPT) $(CC_DEF) -c $<


TAR   = gattai

$(TAR): $(OBJS)
        $(CC) $(CFLAGS) -L. -L/home/nariyama/project6/Implicit/mc/lib -o \
        $(TAR) $(OBJS) $(IRISLIB) $(LIB)

main.o: $(HEADER) main.c
        $(CC) -c $(CFLAGS) $(CC_OPT) $(CC_DEF) main.c

init.o: $(HEADER) init.c
        $(CC) -c $(CFLAGS) $(CC_OPT) $(CC_DEF) init.c

draw3.o: $(HEADER) draw3.c
        $(CC) -c $(CFLAGS) $(CC_OPT) $(CC_DEF) draw3.c

scaling.o: $(HEADER) scaling.c
        $(CC) -c $(CFLAGS) $(CC_OPT) $(CC_DEF) scaling.c

data.o: $(HEADER) data.c
        $(CC) -c $(CFLAGS) $(CC_OPT) $(CC_DEF) data.c

menu.o: $(HEADER) menu.c
        $(CC) -c $(CFLAGS) $(CC_OPT) $(CC_DEF) menu.c

body_draw.o: $(HEADER) body_draw.c
        $(CC) -c $(CFLAGS) $(CC_OPT) $(CC_DEF) body_draw.c

calc_object.o : body.h body_draw.h rname_fc.h calc_object.c
        $(CC) -c $(CFLAGS) calc_object.c $(CC_DEF)

mkrotm.o : body.h body_draw.h rname_fc.h mkrotm.c
        $(CC) -c $(CFLAGS) mkrotm.c $(CC_DEF)

get_all_mat.o : body.h body_draw.h rname_fc.h get_all_mat.c
        $(CC) -c $(CFLAGS) get_all_mat.c $(CC_DEF)

calib.o : calib.c
        $(CC) -c $(CFLAGS) calib.c $(CC_DEF)

abc.o : abc.c
        $(CC) -c $(CFLAGS) abc.c $(CC_DEF)

read_fastrak.o : fastrak/fastrak.h read_fastrak.c
        $(CC) -c $(CFLAGS) read_fastrak.c

read_cyber.o : cyber/cyber_glove.h read_cyber.c
        $(CC) -c $(CFLAGS) read_cyber.c

addon.o: $(HEADER) addon.c
```



```
        $(CC) -c $(CFLAGS) $(CC_OPT) $(CC_DEF) addon.c
```

```
/*** file : addon.h ***/
/*** author : singh ***/
/*** purpose :  ffd space,flexor type definintions ***/
/*** date : 29 june 94 ***/


typedef VERTEX          Axis[3];

typedef struct space_td
{
  VERTEX pt[4][4];      /* current 16 bez. control points */
  VERTEX orig[4][4];    /* orig. positions of points */
  Axis   normal[4];     /* 4 coord axes for each of the four planes */
  VERTEX center[4];     /* current centers of 4 planes */
  VERTEX corig[4];      /* original centers of 4 planes */
  float xang,yang,zang; /* angles of rotation of the space */
                        /* anchored at plane 0 */
  float na[2];
} Space;

typedef struct flexor_td
{
  Space lead;
  Space trail;
  int   joint;
  VERTEX center;
  float threshold;
] Flexor;


extern Space douspace,kosispace;
extern int ffdswitch,showimp,breatheswitch,collon;
extern VERTEX shcenter[];
extern float bb[3][2];
extern Matrix mune_mat;

extern float  S1,S2,PF1,PF2;
extern float SCFAC,POWF;
extern float BRFAC,BRINC,BRMAX;


extern float bellF();
extern float scAngle();
```

```
/*****************************************************************
*       ファイル名 :  body.h                                     *
*****************************************************************/

#include  <stdio.h>
#include  <math.h>
#include  <sys/time.h>
#include  <gl.h>
#include  <device.h>
#include  <gl/image.h>

#define         IN              /*  入力用引数   */
#define         OUT             /*  出力用引数   */
#define         INOUT           /*  入出力用引数   */

#define         X       0       /*  X座標   */
#define         Y       1       /*  Y座標   */
#define         Z       2       /*  Z座標   */
#define         XYZ     3       /*  次元数   */

/*  頭、首、肩、二の腕、腕、指の長さ   */
#define         ATAMA_LENG      20
#define         KUBI_LENG       10
#define         KATA_LENG       18
#define         NINOUDE_LENG    28
#define         UDE_LENG        28
#define         YUBI_LENG       3

/* トラッカーデータ、データグローブデータ */
static int  prt_bit[10] = {
    0x01, 0x02, 0x04, 0x08, 0x10,
    0x20, 0x40, 0x80, 0x100, 0x200
};

/* トラッカーデータ、データグローブデータ */
#define         AANGLE_BNO      0
#define         KANGLE_BNO      1
#define         HANGLE_BNO      2
#define         TANGLE_BNO      3
#define         DGLOVE_BNO      4
#define         TRKER5_BNO      5
#define         TRKER6_BNO      6
#define         TRKER7_BNO      7
#define         TRKER8_BNO      8

#define         TRKER_BIT       0xf0

#ifndef  EXTRN
#define  EXTRN  extern

EXTRN   Matrix  idmat;
EXTRN   float   texprops[];
EXTRN   float   tevprops[];

EXTRN   Matrix  atama_cal_mx;   /*  頭の校正マトリックス  (ワールド座標：キャリブレーション)  */
EXTRN   Matrix  mune_cal_mx;    /*  胸の校正マトリックス  (ワールド座標：キャリブレーション)  */
EXTRN   Matrix  rtekubi_cal_mx; /*  右手首の校正マトリックス  (ワールド座標：キャリブレーション)  */
EXTRN   Matrix  ltekubi_cal_mx; /*  左手首の校正マトリックス  (ワールド座標：キャリブレーション)  */

#else

EXTRN Matrix idmat = {
        1.0, 0.0, 0.0, 0.0,
        0.0, 1.0, 0.0, 0.0,
        0.0, 0.0, 1.0, 0.0,
```



```
        0.0, 0.0, 0.0, 1.0,
};

EXTRN float texprops[] = {TX_MINFILTER, TX_POINT,
                  TX_MAGFILTER, TX_BILINEAR,
                  TX_WRAP, TX_REPEAT, TX_NULL};

EXTRN float tevprops[] = { TV_MODULATE, TV_NULL };

EXTRN Matrix  atama_cal_mx = {
        1.0, 0.0, 0.0, 0.0,
        0.0, 1.0, 0.0, 0.0,
        0.0, 0.0, 1.0, 0.0,
        0.0, 0.0, 0.0, 1.0,
};

EXTRN Matrix  mune_cal_mx = {
        1.0, 0.0, 0.0, 0.0,
        0.0, 1.0, 0.0, 0.0,
        0.0, 0.0, 1.0, 0.0,
        0.0, 0.0, 0.0, 1.0,
};

EXTRN Matrix  rtekubi_cal_mx = {
        1.0, 0.0, 0.0, 0.0,
        0.0, 1.0, 0.0, 0.0,
        0.0, 0.0, 1.0, 0.0,
        0.0, 0.0, 0.0, 1.0,
};

EXTRN Matrix  ltekubi_cal_mx = {
        1.0, 0.0, 0.0, 0.0,
        0.0, 1.0, 0.0, 0.0,
        0.0, 0.0, 1.0, 0.0,
        0.0, 0.0, 0.0, 1.0,
};

#endif /* EXTRN */

EXTRN   int     mapdbindex;     /* テクスチュアのインデックス */

EXTRN char      tr_dname[30];   /* トラッカーのデータファイル名 */
EXTRN char      dg_dname[30];   /* データグローブのデータファイル名 */

/* トラッカーデータとデータグローブのデータをファイルから読み込む */
EXTRN FILE      *tr_fp;
EXTRN FILE      *dg_fp;

EXTRN int       rec_cnt;
EXTRN int       max_rec;
EXTRN int       rvse_flg;
EXTRN int       prt_flg;
EXTRN int       offset_bf[10000];

/* トラッカーデータ、データグローブデータ */
EXTRN float     tr_threshold;
EXTRN int       dg_threshold;

EXTRN float  dr_xmin, dr_xmax, dr_ymin, dr_ymax, dr_zmin, dr_zmax;
EXTRN float  vx, vy, vz;
EXTRN float  px, py, pz;
EXTRN float  twist;

EXTRN float  atama[6];          /*  頭のトラッカーデータ（X, Y, Z座標値、a, r, e回転角度
EXTRN float  mune[6];           /*  胸のトラッカーデータ（同上）  */
EXTRN float  rtekubi[6];        /*  右手首のトラッカーデータ（同上）  */
```

```
EXTRN float  ltekubi[6];          /*  左手首のトラッカーデータ（同上）  */

EXTRN float  atama_cal[6];        /*  頭の校正用トラッカーデータ（同上）  */
EXTRN float  mune_cal[6];         /*  胸の校正用トラッカーデータ（同上）  */
EXTRN float  rtekubi_cal[6];      /*  右手首の校正用トラッカーデータ（同上）  */
EXTRN float  ltekubi_cal[6];      /*  左手首の校正用トラッカーデータ（同上）  */

EXTRN float  mune_pos[3];
EXTRN float  rtekubi_pos[3];
EXTRN float  ltekubi_pos[3];

EXTRN float  xangle;              /*  肘のＸ軸まわりの回転角度 */

EXTRN float  rkata_a;             /*  右肩のＸ軸回転の角度 */
EXTRN float  lkata_a;             /*  左肩のＸ軸回転の角度 */

EXTRN float  rtekubi_a;           /*  右手首のＸ軸回転の角度 */
EXTRN float  rtekubi_y;           /*  右手首のＹ軸回転の角度 */
EXTRN float  rtekubi_z;           /*  右手首のＺ軸回転の角度 */
EXTRN float  ltekubi_a;           /*  左手首のＸ軸回転の角度 */

EXTRN Matrix  rot_m;              /*  作業用回転マトリックス  */

EXTRN Matrix  atama_omx;          /*  頭の回転マトリックス（生データ：ワールド座標）  */
EXTRN Matrix  mune_omx;           /*  胸の回転マトリックス（生データ：ワールド座標）  */
EXTRN Matrix  rtekubi_omx;        /*  右手首の回転マトリックス（生データ：ワールド座標）  */
EXTRN Matrix  ltekubi_omx;        /*  左手首の回転マトリックス（生データ：ワールド座標）  */

EXTRN Matrix  atama_om2;          /*  頭の回転マトリックス（ワールド座標：取り付け補正）  */
EXTRN Matrix  mune_om2;           /*  胸の回転マトリックス（ワールド座標：取り付け補正）  */
EXTRN Matrix  rtekubi_om2;        /*  右手首の回転マトリックス（ワールド座標：取り付け補正）  */
EXTRN Matrix  ltekubi_om2;        /*  左手首の回転マトリックス（ワールド座標：取り付け補正）  */

EXTRN Matrix  rkata_om2;          /*  右肩の回転マトリックス（ワールド座標：取り付け補正）  */
EXTRN Matrix  lkata_om2;          /*  左肩の回転マトリックス（ワールド座標：取り付け補正）  */
EXTRN Matrix  rhiji_om2;          /*  右肘の回転マトリックス（ワールド座標：取り付け補正）  */
EXTRN Matrix  lhiji_om2;          /*  左肘の回転マトリックス（ワールド座標：取り付け補正）  */

EXTRN Matrix  atama_m;            /*  頭の回転マトリックス（表示系）  */
EXTRN Matrix  mune_m;             /*  胸の回転マトリックス（表示系）  */
EXTRN Matrix  rtekubi_m;          /*  右手首の回転マトリックス（表示系）  */
EXTRN Matrix  ltekubi_m;          /*  左手首の回転マトリックス（表示系）  */

EXTRN Matrix  mune_rm;            /*  胸の逆回転マトリックス（表示系）  */

EXTRN Matrix  rkata_m;            /*  右肩の回転マトリックス（表示系）  */
EXTRN Matrix  rkata1_m;           /*  右肩の回転マトリックス（表示系：Ｘ軸回転）  */
EXTRN Matrix  rkata2_m;           /*  右肩の回転マトリックス（表示系：Ｙ，Ｚ軸回転）  */

EXTRN Matrix  lkata_m;            /*  左肩の回転マトリックス（表示系）  */
EXTRN Matrix  lkata1_m;           /*  左肩の回転マトリックス（表示系：Ｘ軸回転）  */
EXTRN Matrix  lkata2_m;           /*  左肩の回転マトリックス（表示系：Ｙ，Ｚ軸回転）  */

EXTRN Matrix  rhiji_m;            /*  右肘の回転マトリックス（表示系）  */
EXTRN Matrix  lhiji_m;            /*  左肘の回転マトリックス（表示系）  */

EXTRN Matrix  rtekubi1_m;         /*  右手首の回転マトリックス（表示系：Ｘ軸回転）  */
EXTRN Matrix  rtekubi2_m;         /*  右手首の回転マトリックス（表示系：Ｙ，Ｚ軸回転）  */
EXTRN Matrix  ltekubi1_m;         /*  左手首の回転マトリックス（表示系：Ｘ軸回転）  */
EXTRN Matrix  ltekubi2_m;         /*  左手首の回転マトリックス（表示系：Ｙ，Ｚ軸回転）  */

EXTRN Matrix  mkh_m;              /*  胸，肩，肘までの回転マトリックス（表示系）  */
EXTRN Matrix  mkh_rm;             /*  胸，肩，肘までの逆回転マトリックス（表示系）  */

EXTRN Matrix  royayubi_m[3];          /* 親指の回転マトリックス  */
EXTRN Matrix  rhitosasiyubi_m[3];     /* 人指し指の回転マトリックス  */
```



```
EXTRN Matrix  rnakayubi_m[3];          /* 中指の回転マトリックス  */
EXTRN Matrix  rkusuriyubi_m[3];        /* 薬指の回転マトリックス  */
EXTRN Matrix  rkoyubi_m[3];            /* 小指の回転マトリックス  */
EXTRN Matrix  loyayubi_m[3];           /* 親指の回転マトリックス  */
EXTRN Matrix  lhitosasiyubi_m[3];      /* 人指し指の回転マトリックス  */
EXTRN Matrix  lnakayubi_m[3];          /* 中指の回転マトリックス  */
EXTRN Matrix  lkusuriyubi_m[3];        /* 薬指の回転マトリックス  */
EXTRN Matrix  lkoyubi_m[3];            /* 小指の回転マトリックス  */

EXTRN float   rhiji_pnt[3];     /* 右肘の位置座標 */
EXTRN float   lhiji_pnt[3];     /* 左肘の位置座標 */

/************ kahansin start ************/
EXTRN   Matrix          kosi_m;
EXTRN   Matrix          rsiri_m;
EXTRN   Matrix          rhiza_m;
EXTRN   Matrix          rasikubi_m;
EXTRN   Matrix          lsiri_m;
EXTRN   Matrix          lhiza_m;
EXTRN   Matrix          lasikubi_m;
/************ kahansin end ************/

/***********************************
*           関数の宣言             *
***********************************/
/*  -----<<  body.c  >>-----  */
/*
void  coord_calc(float *d1, float *d2, Matrix mat);
void  cp_mat(Matrix m1, Matrix m2);
void  rev_mat(Matrix m1, Matrix m2);
void  set_trk_dg_data(float *trk, int *dg, float fang[5][3]);
void  set_coord(float *trk);

int   calc_model(float trk[24], int dg[10]);
*/
void  coord_calc();
void  cp_mat();
void  rev_mat();
void  set_trk_dg_data();
void  set_coord();

int   calc_model();

/*  -----<<  mkrotm.c  >>-----  */
void  get_mune_mat();
void  get_atama_mat();
void  get_tekubi_mat();
/*
void  get_rot_coord(float *pnt1, float *pnt2, float angle, char axis);

void  mk_hiji_rotm(int num, float kata_x, float hiji_x, float *tekubi,
                   Matrix km, Matrix m);

void  mk_kata_rotm(int num, float kata_x, float *tekubi, Matrix ka_m, Matrix hi_m, Ma
 trix hiza_m, Matrix momo_m);

void  mk_tekubi_rotm();

void  mk_yubi_rotm(float fang[3], char axis, float hosei[3], Matrix m );

float get_theta2(float c1, float c2, float ct);
*/
void  get_rot_coord();

void  mk_hiji_rotm();
```

```
void  mk_kata_rotm();

void  mk_tekubi_rotm();

void  mk_yubi_rotm();

float get_theta2();

double  delta_ms();
void    GetKataMatrix(int, float *, float, float, float, float, Matrix,
                      Matrix, float *);
void    GetTekubiMatrix(int, Matrix, Matrix, Matrix, Matrix, float *, float *,
                        float *, Matrix, Matrix);
void    GetTekubiMatrix2(int, Matrix, Matrix, Matrix, Matrix, float *, float *,
                        float *, Matrix, Matrix);
void    GetTekubiX(int, Matrix, float *, Matrix);
void    GetTekubiYZ(int, Matrix, float *, float *, Matrix);
void    GetKosiPosition(float, float, float, float, float, float,
                        float *, float *);
```

```
/*
 * ファイル名 : body_draw.h
 *
 * 機能       : 臨場感通信会議デモシステム組み込み用ヘッダー
 *
 * 履歴       : 1992年10月28日 ; 作成 ; 広瀬　正俊
 */
#include        <malloc.h>
#include        <fcntl.h>
#include        <stdio.h>
#include        <math.h>
#include        <gl.h>
#include        <gl/image.h>
#include        <device.h>

#define         max(a,b)        (((a)>(b)) ? (a) : (b))
#define         min(a,b)        (((a)<(b)) ? (a) : (b))

#define         ATAMA           0
#define         KUBI            1
#define         DOU             2
#define         KOSI            3
#define         KATA_D_R        4
#define         KATA_N_R        5
#define         NINOUDE_R       6
#define         HIJI_R          7
#define         UDE_R           8
#define         TE_R            9
#define         OYAYUBI_R_1     10
#define         OYAYUBI_R_2     11
#define         OYAYUBI_R_3     12
#define         HITOSASIYUBI_R_1  13
#define         HITOSASIYUBI_R_2  14
#define         HITOSASIYUBI_R_3  15
#define         NAKAYUBI_R_1    16
#define         NAKAYUBI_R_2    17
#define         NAKAYUBI_R_3    18
#define         KUSURIYUBI_R_1  19
#define         KUSURIYUBI_R_2  20
#define         KUSURIYUBI_R_3  21
#define         KOYUBI_R_1      22
#define         KOYUBI_R_2      23
#define         KOYUBI_R_3      24
#define         KATA_D_L        25
#define         KATA_N_L        26
#define         NINOUDE_L       27
#define         HIJI_L          28
#define         UDE_L           29
#define         TE_L            30
#define         OYAYUBI_L_1     31
#define         OYAYUBI_L_2     32
#define         OYAYUBI_L_3     33
#define         HITOSASIYUBI_L_1  34
#define         HITOSASIYUBI_L_2  35
#define         HITOSASIYUBI_L_3  36
#define         NAKAYUBI_L_1    37
#define         NAKAYUBI_L_2    38
#define         NAKAYUBI_L_3    39
#define         KUSURIYUBI_L_1  40
#define         KUSURIYUBI_L_2  41
#define         KUSURIYUBI_L_3  42
#define         KOYUBI_L_1      43
#define         KOYUBI_L_2      44
#define         KOYUBI_L_3      45
#define         SIRI_R          46
#define         MOMO_R          47
```



```
#define         HIZA_R          48
#define         SUNE_R          49
#define         KUTU_R          50
#define         SIRI_L          51
#define         MOMO_L          52
#define         HIZA_L          53
#define         SUNE_L          54
#define         KUTU_L          55
#define         MAX_BUHIN       56

#define         OYAYUBI_HOSEI           40.0
#define         HITOSASIYUBI_HOSEI      10.0
#define         NAKAYUBI_HOSEI          (-10.0)
#define         KUSURIYUBI_HOSEI        (-20.0)
#define         KOYUBI_HOSEI            (-30.0)

#define         LINE_BUF        256
#define         MAX_NAME        256     /* ファイル名の最大長さ */

#define         DATA_DIR                "./"
#define         BODY_COLOR              1
#define         FACE_OBJ                2
#define         KUBI_OBJ                3
#define         DOU_OBJ                 4
#define         KOSI_OBJ                5
#define         KATA_D_R_OBJ            6
#define         KATA_N_R_OBJ            7
#define         NINOUDE_R_OBJ           8
#define         HIJI_R_OBJ              9
#define         UDE_R_OBJ               10
#define         TE_R_OBJ                11
#define         OYAYUBI_R_1_OBJ         12
#define         OYAYUBI_R_2_OBJ         13
#define         OYAYUBI_R_3_OBJ         14
#define         HITOSASIYUBI_R_1_OBJ    15
#define         HITOSASIYUBI_R_2_OBJ    16
#define         HITOSASIYUBI_R_3_OBJ    17
#define         NAKAYUBI_R_1_OBJ        18
#define         NAKAYUBI_R_2_OBJ        19
#define         NAKAYUBI_R_3_OBJ        20
#define         KUSURIYUBI_R_1_OBJ      21
#define         KUSURIYUBI_R_2_OBJ      22
#define         KUSURIYUBI_R_3_OBJ      23
#define         KOYUBI_R_1_OBJ          24
#define         KOYUBI_R_2_OBJ          25
#define         KOYUBI_R_3_OBJ          26
#define         KATA_D_L_OBJ            27
#define         KATA_N_L_OBJ            28
#define         NINOUDE_L_OBJ           29
#define         HIJI_L_OBJ              30
#define         UDE_L_OBJ               31
#define         TE_L_OBJ                32
#define         OYAYUBI_L_1_OBJ         33
#define         OYAYUBI_L_2_OBJ         34
#define         OYAYUBI_L_3_OBJ         35
#define         HITOSASIYUBI_L_1_OBJ    36
#define         HITOSASIYUBI_L_2_OBJ    37
#define         HITOSASIYUBI_L_3_OBJ    38
#define         NAKAYUBI_L_1_OBJ        39
#define         NAKAYUBI_L_2_OBJ        40
#define         NAKAYUBI_L_3_OBJ        41
#define         KUSURIYUBI_L_1_OBJ      42
#define         KUSURIYUBI_L_2_OBJ      43
#define         KUSURIYUBI_L_3_OBJ      44
#define         KOYUBI_L_1_OBJ          45
#define         KOYUBI_L_2_OBJ          46
```

```
#define         KOYUBI_L_3_OBJ          47
#define         SIRI_R_OBJ              48
#define         MOMO_R_OBJ              49
#define         HIZA_R_OBJ              50
#define         SUNE_R_OBJ              51
#define         KUTU_R_OBJ              52
#define         SIRI_L_OBJ              53
#define         MOMO_L_OBJ              54
#define         HIZA_L_OBJ              55
#define         SUNE_L_OBJ              56
#define         KUTU_L_OBJ              57
#define         MAX_FILE                58

#define         CALIB_BODY_FILE         "calib_body.dat"

#define         STANDING                0
#define         SITTING                 1
#define         SEIZA                   2

#define         TR_ATAMA        0
#define         TR_MUNE         1
#define         TR_MIGITE       2
#define         TR_HIDARITE     3

#define         ATAMA_TRAK_NO           0
#define         MUNE_TRAK_NO            1
#define         MIGITE_TRAK_NO          2
#define         HIDARITE_TRAK_NO        3
#define         ATAMA_TORITUKE_X        0.0
#define         ATAMA_TORITUKE_Y        15.0
#define         ATAMA_TORITUKE_Z        0.0
#define         ATAMA_TORITUKE_RX       0.0
#define         ATAMA_TORITUKE_RY       (-90.0)
#define         ATAMA_TORITUKE_RZ       (90.0)
#define         MUNE_TORITUKE_X         0.0
#define         MUNE_TORITUKE_Y         0.0
#define         MUNE_TORITUKE_Z         0.0
#define         MUNE_TORITUKE_RX        180.0
#define         MUNE_TORITUKE_RY        0.0
#define         MUNE_TORITUKE_RZ        90.0
#define         MIGITE_TORITUKE_X       (-7.0)
#define         MIGITE_TORITUKE_Y       1.0
#define         MIGITE_TORITUKE_Z       0.0
#define         MIGITE_TORITUKE_RX      90.0
#define         MIGITE_TORITUKE_RY      180.0
#define         MIGITE_TORITUKE_RZ      0.0
#define         HIDARITE_TORITUKE_X     (7.0)
#define         HIDARITE_TORITUKE_Y     1.0
#define         HIDARITE_TORITUKE_Z     0.0
#define         HIDARITE_TORITUKE_RX    90.0
#define         HIDARITE_TORITUKE_RY    0.0
#define         HIDARITE_TORITUKE_RZ    0.0

/*  Nariyama add  */
#define         TR_MIGIASHI     4
#define         TR_HIDARIASHI   5
/*  Nariyama add  */

#define         MAX_TR          6

#ifndef EXTRN
#define EXTRN   extern
#endif

EXTRN   float           G_trd[MAX_TR][6];
EXTRN   float           G_ang[7][6];
```



```
#ifdef CYBERGLOVE
EXTRN   int             G_dgd[32];
#else
EXTRN   int             G_dgd[28];
#endif

EXTRN   int             G_sisei;
EXTRN   int             trkswitch;

typedef float           VERTEX[3];
typedef float           VTEXTURE[2];
typedef int             POLYGON[6];
typedef float           OMOMI;

typedef struct _SKELTON {
        float           atama_len;
        float           kubi_len;
        float           kata_len;
        float           ninoude_len;
        float           ude_len;
        float           yubi_len;
        float           dou_len;
        float           momo_len;
        float           sune_len;
} SKELTON;

typedef struct _MATRIX {
        Matrix          atama;
        Matrix          mune;
        Matrix          rkata;
        Matrix          rhiji;
        Matrix          rtekubi;
        Matrix          lkata;
        Matrix          lhiji;
        Matrix          ltekubi;
        Matrix          kosi;
        Matrix          rsiri;
        Matrix          rhiza;
        Matrix          rasikubi;
        Matrix          lsiri;
        Matrix          lhiza;
        Matrix          lasikubi;
} MATRIX;


/*++++++++++++++++++++++*/
#include "addon.h"
/*++++++++++++++++++++++*/

typedef struct _JINBUTU {
        VERTEX          *vs[MAX_BUHIN];
        VERTEX          *va[MAX_BUHIN];
        VERTEX          center[MAX_BUHIN];
        VTEXTURE        *vt[MAX_BUHIN];
        POLYGON         *poly[MAX_BUHIN];
        OMOMI           *a[MAX_BUHIN];
        int             vmax[MAX_BUHIN];
        int             vtmax[MAX_BUHIN];
        int             polmax[MAX_BUHIN];
        int             tindex[MAX_BUHIN];
        Matrix          mat[MAX_BUHIN];
        SKELTON         ske;
        SKELTON         ske_real;
        SKELTON         ske_cg;
        float           rtekubi_a;
        float           ltekubi_a;
```

```
        float           mata;
        float           munex, muney, munez;
        float           munext, muneyt, munezt;
/*  Nariyama add  */
        float           kosix, kosiy, kosiz;
        float           rhizax, rhizay, rhizaz;
        float           lhizax, lhizay, lhizaz;
        Matrix          mune_hosei_w;
        Matrix          atama_calib;
        Matrix          atama_itm;
        Matrix          mune_calib;
        Matrix          mune_itm;
        Matrix          r_te_calib;
        Matrix          r_te_itm;
        Matrix          l_te_calib;
        Matrix          l_te_itm;
        int             stand;
        int             simple;
        MATRIX          mat1;
        Space           douspace,kosispace;
        Matrix          mune_mat;
        Matrix          kosi_mat;
        VERTEX          shcenter[2];
        float           tt[2][5];
        char            hname[MAX_NAME];
/*  Nariyama add  */
        float           *tval[MAX_BUHIN];
        float           *tt1[MAX_BUHIN];
        float           *tt2[MAX_BUHIN];
        float           *tt3[MAX_BUHIN];
        float           *tt4[MAX_BUHIN];
        float           *ipwt[MAX_BUHIN];
        int             bl[MAX_BUHIN];
        VERTEX          *map[MAX_BUHIN];
        VERTEX          *vn[MAX_BUHIN];
} JINBUTU;


/*****     <<< body_draw.c >>>     *****/
void            InitializeJinbutu(JINBUTU *);
void            GetFileName(char *, char [][MAX_NAME], float [][8]);
void            *ReadObjTexture(char *);
void            ReadYubiKansetuData(JINBUTU *, int, char *, float *, int);
void            KaradaZurasi(JINBUTU *);
void            GetCenter(JINBUTU *, int, float *, int, int);
void            CopyData(JINBUTU *, int, int);
void            HidariKataTunagi();
void            HidariUdeZurasi();
void            ReadWave(JINBUTU *, char *, int, float *);
void            Henkei3D(JINBUTU *);
void            HenkeiYubiKansetu(JINBUTU *, int);
void            DrawSimple3D(void *, float *);
void            Draw3D(void *, float *, int, int, float *);
void            Draw3DSeiza(void *, float *, int, int, float *);
void            Draw3DHand(void *, float *);
void            DrawYubiKansetu(JINBUTU *, int, float *);
void            DrawMap3(JINBUTU *, int, int, VERTEX *, float *);
void            DrawMap3Shade(JINBUTU *, int, VERTEX *, float *);
void            RotateVertex(JINBUTU *, int);
void            IshiiVertex(JINBUTU *, int);
void            IshiiVertex2(VERTEX *, OMOMI *, int, VERTEX *, float, float *);
void            CalcKahansin(JINBUTU *, float *, int);
void            CalcKansetuMatrix(JINBUTU *, int, Matrix);
void            c_calib_model(void *, float *);
void            CalcStandingKahansin(JINBUTU *, int, float *, float *,
                                     float *, float *, float *, float *);
```



```
void            calc_ang_geom(float, float, float, float, float,
                              float, float *, float *);
void            GetCalibMatrix(Matrix, Matrix, float *, float, float, float,
                               float, float, float);
void            FaceNormal(float [3][3], float [3]);
void            GetTukamiKakudo(JINBUTU *, float *, int);

/*++++++++++++++++++++++*/
#include "newmodel.h"
/*++++++++++++++++++++++*/
```

```
#include <stdio.h>
#include <malloc.h>
#include <ctype.h>
#include <math.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <rpc/rpc.h>
#include <utmp.h>
#include <netdb.h>
#include <sys/socket.h>
#include <sys/time.h>
#include <sys/errno.h>

/*#include <X11/Intrinsic.h>*/

#include "vt_types.h"
#include "vt_error.h"
#include "vt_glove.h"
#include "vt_serial_io.h"
#include "vt_glove-types.h"

#define CG_PROG_NUM     0x40000203
#define CG_PROG_VER     1
#define CG_PROG_PR1     1
#define CG_PROG_PR2     2
#define CG_PROG_PR3     3

#define CG_DEV  "/dev/ttyd1"

#ifndef FALSE
#define FALSE   0
#endif
#ifndef TRUE
#define TRUE    1
#endif

typedef struct {
  int  status;
  int  get_data[24];
} CG_SHM_STRUCT;

#define ARG_LENGTH      10

struct CYBERSTR {
  int  joint_angle[5][2];
  int  joint_space[4];
  int  wrist_angle[2];
};
```

```
/*********************************************
 **  Library for 3SpaceTracker Header File
 *********************************************/
#include <stdio.h>
#include <malloc.h>
#include <ctype.h>
#include <math.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <rpc/rpc.h>
#include <utmp.h>
#include <netdb.h>
#include <sys/socket.h>
#include <sys/time.h>
#include <sys/errno.h>

#define TR_PROG_NUM     0x40000202
#define TR_PROG_VER     1
#define TR_PROG_PR      1

#define ARRAYSIZE       4

typedef  struct {
  int     sens_num;
  double  x, y, z, a, e, r;
} PRCSTRUCT;

/**  Used Serial Port Device  **/
#define TR_DEV  "/dev/ttyd2"

/** Any mode define select **/
/*#define       ASCII_MODE              /* ASCII Mode */
/*#define       BIT32_BINARY_MODE       /* 32Bit BINARY Mode */
#define BIT16_BINARY_MODE       /* 16Bit BINARY Mode */

/**  Debug Mode Define  **/
/*#define       WARNING*/
/*#define       TIME_MEASURE*/

/**  Calibrate Data File Name  **/
#define CALIB_FILE      "3space.dat"

/**  Record Header Mask  **/
#define  LeadingMask    0x80

/**  Any Buffer Length  **/
#define ARG_LENGTH      10

#ifdef  ASCII_MODE
#define DBUF_SIZE       47
#endif
#ifdef  BIT32_BINARY_MODE
#define DBUF_SIZE       29
#endif
#ifdef  BIT16_BINARY_MODE
#define DBUF_SIZE       17
#define LIMIT_LENGTH    256
#endif

/** Shared Memory Use Variable  **/
typedef struct {
  int     status;
  char    binary_data[DBUF_SIZE];
} TR_SHM_STRUCT;
```

```
/*******************************************************************************
 *                                                                             *
 *  Forms is a library for building up dialogue and interaction boxes.  *
 *  It is based of the Silicon Graphics Graphical Library.             *
 *                                                                             *
 *  This software is public domain. It may not be resold.              *
 *                                                                             *
 *  Written by: Mark Overmars                                          *
 *              Department of Computer Science                         *
 *              University of Utrecht                                  *
 *              P.O. Box 80.089                                        *
 *              3508 TB Utrecht, the Netherlands                       *
 *              Email: markov@cs.ruu.nl                                *
 *                                                                             *
 *  Version 2.0 a                                                      *
 *  Date:  Nov 26, 1991                                                *
 *******************************************************************************/

#ifndef __FORMS_H__
#define __FORMS_H__

#if defined(_LANGUAGE_C_PLUS_PLUS) || defined(__cplusplus)
extern "C" {
#endif

#include <stddef.h>
#include <gl/gl.h>

/************   The type OBJECT            ************/

#define FL_LABEL_SIZE           64
#define FL_SHORTCUT_SIZE        32

struct forms;

typedef struct objs {
   int objclass;        /* What type of object */
   int type;            /* The type within the class */
   int boxtype;         /* The type of the bounding box */
   float x,y,w,h;       /* The bounding box */
   int col1,col2;       /* Two possible colors */

   char label[FL_LABEL_SIZE];   /* The label */
   int align;           /* Label or text alignment */
   int lcol;            /* Color of the label */
   float lsize;         /* Size of the label */
   int lstyle;          /* Style of the text label */

   char shortcut[FL_SHORTCUT_SIZE];     /* The list of shortcuts */

   int (*handle)(struct objs *, int, float, float, char);
                        /* Handling procedure. */

   int *spec;           /* pointer to special stuff for object */

   int pushed;          /* wheter pushed */
   int focus;           /* wheter focussed */
   int belowmouse;      /* Whether currently below the mouse */

   int frozen;          /* Whether it will be redrawn */
   int active;          /* Whether active */
   int input;           /* Whether receiving input */
   int visible;         /* Whether being displayed */
   int radio;           /* Whether a radio object */
   int automatic;       /* Whether this object gets timer events */
   void (*object_call_back)(struct objs *, long);
```



```
                        /* The call-back routine */
   long argument;       /* Its argument */

   struct objs *next;   /* Next object in the form */
   struct objs *prev;   /* Previous object in the form */
   struct forms *form;  /* Form to which object belong */
} FL_OBJECT;

/************   The type FORM              ************/

typedef struct forms {
   long window;         /* Window of the form */
   float w,h;           /* The size of the form */
   long x,y;            /* Position of form on screen */

   int deactivated;     /* Whether deactivated */
   int visible;         /* Whether being displayed */
   int frozen;          /* When true no redraws are performed */

   int doublebuf;       /* Whether in double buffer mode */

   void (*form_call_back)(struct objs *);
                        /* The call-back routine */

   struct objs *first;  /* First object in the form */
   struct objs *last;   /* Last object in the form */

   struct objs *focusobj; /* Object to which input is directed */
} FL_FORM;

/************   General Constants          ************/

#ifndef NULL
#define NULL                    0
#endif

#ifndef FALSE
#define FALSE                   0
#endif

#ifndef TRUE
#define TRUE                    1
#endif

/***** Placement *****/

#define FL_PLACE_FREE           0
#define FL_PLACE_SIZE           1
#define FL_PLACE_ASPECT         2
#define FL_PLACE_MOUSE          3
#define FL_PLACE_CENTER         4
#define FL_PLACE_POSITION       5

/***** Finds *****/

#define FL_FIND_INPUT           0
#define FL_FIND_AUTOMATIC       1
#define FL_FIND_MOUSE           2

/***** Special Objects  *****/

#define FL_BEGIN_GROUP          10000
#define FL_END_GROUP            20000

/***** Alignments *****/
```

```
#define FL_ALIGN_TOP            0
#define FL_ALIGN_BOTTOM         1
#define FL_ALIGN_LEFT           2
#define FL_ALIGN_RIGHT          3
#define FL_ALIGN_CENTER         4

/***** Bounding boxes *****/

#define FL_NO_BOX               0
#define FL_UP_BOX               1
#define FL_DOWN_BOX             2
#define FL_FLAT_BOX             3
#define FL_BORDER_BOX           4
#define FL_SHADOW_BOX           5
#define FL_FRAME_BOX            6
#define FL_ROUNDED_BOX          7
#define FL_RFLAT_BOX            8
#define FL_RSHADOW_BOX          9

/***** Boundary Colors *****/

#define FL_TOP_BOUND_COL        51
#define FL_LEFT_BOUND_COL       55
#define FL_BOT_BOUND_COL        40
#define FL_RIGHT_BOUND_COL      35

#define FL_COL1                 47
#define FL_MCOL                 49
#define FL_LCOL                 0

#define FL_BOUND_WIDTH          3.0

/***** Events *****/

#define FL_DRAW                 0
#define FL_PUSH                 1
#define FL_RELEASE              2
#define FL_ENTER                3
#define FL_LEAVE                4
#define FL_MOUSE                5
#define FL_FOCUS                6
#define FL_UNFOCUS              7
#define FL_KEYBOARD             8
#define FL_STEP                 9
#define FL_MOVE                 10

/***** Font *****/

#define FL_FONT_NAME            "Helvetica"
#define FL_FONT_BOLDNAME        "Helvetica-Bold"
#define FL_FONT_ITALICNAME      "Helvetica-Oblique"
#define FL_FONT_FIXEDNAME       "Courier"
#define FL_FONT_ICONNAME        "Icon"

#define FL_SMALL_FONT           8.0
#define FL_NORMAL_FONT          11.0
#define FL_LARGE_FONT           20.0

#define FL_NORMAL_STYLE         0
#define FL_BOLD_STYLE           1
#define FL_ITALIC_STYLE         2
#define FL_FIXED_STYLE          3
#define FL_ENGRAVED_STYLE       4
#define FL_ICON_STYLE           5
```



```
/************   General Routines           ************/

/***** In objects.c *****/

typedef int (*FL_HANDLEPTR)(FL_OBJECT *, int , float, float, char);

FL_FORM         *fl_make_form(float,float);
FL_OBJECT       *fl_make_object(int,int,float,float,float,float,char [], FL_HANDLEPTR
 );

void    fl_free_object(FL_OBJECT *);
void    fl_free_form(FL_FORM *);

void    fl_add_object(FL_FORM *, FL_OBJECT*);
void    fl_insert_object(FL_OBJECT *, FL_OBJECT*);
void    fl_delete_object(FL_OBJECT *);

void    fl_set_object_align(FL_OBJECT *, int);
void    fl_set_object_boxtype(FL_OBJECT *, int);
void    fl_set_object_color(FL_OBJECT *, int, int);
void    fl_set_object_label(FL_OBJECT *, char []);
void    fl_set_object_lcol(FL_OBJECT *, int);
void    fl_set_object_lsize(FL_OBJECT *, float );
void    fl_set_object_lstyle(FL_OBJECT *, int);

void    fl_show_object(FL_OBJECT *);
void    fl_hide_object(FL_OBJECT *);

void    fl_set_object_focus(FL_FORM *, FL_OBJECT *);

void    fl_set_object_shortcut(FL_OBJECT *, char []);

FL_OBJECT       *fl_find_object(FL_OBJECT *, int, float, float);
FL_OBJECT       *fl_find_first(FL_FORM *, int, float, float);
FL_OBJECT       *fl_find_last(FL_FORM *, int, float, float);

void    fl_redraw_object(FL_OBJECT *);
void    fl_redraw_form(FL_FORM *);

void    fl_freeze_object(FL_OBJECT *);
void    fl_unfreeze_object(FL_OBJECT *);
void    fl_freeze_form(FL_FORM *);
void    fl_unfreeze_form(FL_FORM *);

void    fl_handle_object(FL_OBJECT *, int, float, float, char);
int     fl_handle_object_direct(FL_OBJECT *, int, float, float, char);

/***** In forms.c *****/

extern FL_FORM *fl_current_form;

FL_FORM         *fl_bgn_form(int, float, float);
void            fl_end_form(void);
void            fl_addto_form(FL_FORM *);

FL_OBJECT       *fl_bgn_group(void);
FL_OBJECT       *fl_end_group(void);

void            fl_set_form_position(FL_FORM *, long, long);
long            fl_show_form(FL_FORM *, int, int, char *);
void            fl_hide_form(FL_FORM *);

FL_OBJECT       *fl_do_forms(void);
FL_OBJECT       *fl_check_forms(void);
FL_OBJECT       *fl_do_only_forms(void);
FL_OBJECT       *fl_check_only_forms(void);
```

```
void            fl_activate_form(FL_FORM *);
void            fl_deactivate_form(FL_FORM *);
void            fl_activate_all_forms();
void            fl_deactivate_all_forms();

/***** In events.c *****/

extern FL_OBJECT *FL_EVENT;

typedef void (*FL_CALLBACKPTR)(FL_OBJECT *, long);
typedef void (*FL_FORMCALLBACKPTR)(FL_OBJECT *);
typedef void (*FL_EVENTCALLBACKPTR)(short, short);

void    fl_init_events();

void    fl_set_call_back(FL_OBJECT *, FL_CALLBACKPTR, long);
void    fl_set_event_call_back(FL_EVENTCALLBACKPTR);
void    fl_set_form_call_back(FL_FORM *, FL_FORMCALLBACKPTR);

void    fl_qdevice(Device);
void    fl_unqdevice(Device);
int     fl_isqueued(Device);
long    fl_qtest(void);
long    fl_qread(short *);
long    fl_blkqread(short *, short);
void    fl_qreset(void);
void    fl_qenter(short,short);
void    fl_tie(Device, Device, Device);

void    fl_treat_user_event(void);

FL_OBJECT       *fl_object_qtest(void);
FL_OBJECT       *fl_object_qread(void);
void            fl_object_qenter(FL_OBJECT *);

/***** In goodies.c *****/

long    fl_show_buttonbox(void);
void    fl_hide_buttonbox(void);
void    fl_set_buttonbox_label(int, char []);

void    fl_show_message(char [], char[], char []);
int     fl_show_question(char [], char[], char []);
int     fl_show_choice(char [], char[], char [], int,char [], char[], char []);
char    *fl_show_input(char[], char[]);

int     fl_show_colormap(int);

/***** In fselect.c *****/

char    *fl_show_file_selector(char [], char [], char [], char []);

char    *fl_get_directory();
char    *fl_get_pattern();
char    *fl_get_filename();

/***** In draw.c *****/

void    fl_get_mouse(float *, float *);

void    fl_set_clipping(float, float, float, float);
void    fl_unset_clipping(void);

void    fl_init_fonts();
void    fl_set_font(char [], char [], char [], char[]);
```



```
float   fl_get_char_height(float, int);
float   fl_get_char_width(float, int, char);
float   fl_get_string_width(float, int, char []);

void    fl_init_colormap();
void    fl_color(int);
void    fl_mapcolor(int, short, short, short);
void    fl_getmcolor(int, short *, short *, short *);

void    fl_line(float, float, float, float, int);
void    fl_rect(float, float, float, float, int);
void    fl_bound(float, float, float, float, int);
void    fl_rectbound(float, float, float, float, int);

void    fl_drw_box(int ,float, float, float, float, int,float);
void    fl_drw_text(int, float, float, float, float, int, float, int, char[]);
void    fl_drw_text_beside(int, float, float, float, float, int, float, int, char[]);
void    fl_drw_text_cursor(int, float, float, float, float, int, float, int, char[],
 int, int);

/***** In symbols.c *****/

typedef void (*FL_DRAWPTR)(int);

void    fl_init_symbols();

int     fl_add_symbol(char [], FL_DRAWPTR, int);
int     fl_draw_symbol(char [], float, float, float,float, int);

/***** In support.c *****/

extern int fl_rgbmode;
extern int fl_doublebuf;

void    fl_init();
void    fl_set_graphics_mode(int, int);

void    fl_show_errors(int);
void    fl_error(char [], char []);

void    *fl_malloc(size_t);

/************************ The Classes ***********************/

/************   Object Class: Bitmap       ************/

/***** Class    *****/

#define FL_BITMAP               3

/***** Types    *****/

#define FL_NORMAL_BITMAP        0

/***** Defaults *****/

#define FL_BITMAP_BOXTYPE       FL_NO_BOX
#define FL_BITMAP_COL1          0
#define FL_BITMAP_COL2          FL_COL1
#define FL_BITMAP_LCOL          FL_LCOL
#define FL_BITMAP_ALIGN         FL_ALIGN_BOTTOM

/***** Others   *****/

#define FL_BITMAP_MAXSIZE       128*128
#define FL_BITMAP_BW            FL_BOUND_WIDTH
```

```
/***** Routines *****/

FL_OBJECT       *fl_create_bitmap(int, float, float, float, float, char []);
FL_OBJECT       *fl_add_bitmap(int, float, float, float, float, char []);

void            fl_set_bitmap(FL_OBJECT *, int, int, char *);


/************   Object Class: Box          ************/

/***** Class    *****/

#define FL_BOX                  1

/***** Types    *****/

/* See the bouding boxes */

/***** Defaults *****/

#define FL_BOX_BOXTYPE          FL_UP_BOX
#define FL_BOX_COL1             FL_COL1
#define FL_BOX_LCOL             FL_LCOL
#define FL_BOX_ALIGN            FL_ALIGN_CENTER

/***** Others   *****/

#define FL_BOX_BW               FL_BOUND_WIDTH

/***** Routines *****/

FL_OBJECT       *fl_create_box(int, float, float, float, float, char []);
FL_OBJECT       *fl_add_box(int, float, float, float, float, char []);

/************   Object Class: Browser      ************/

/***** Class    *****/

#define FL_BROWSER              71

/***** Types    *****/

#define FL_NORMAL_BROWSER       0
#define FL_SELECT_BROWSER       1
#define FL_HOLD_BROWSER         2
#define FL_MULTI_BROWSER        3

/***** Defaults *****/

#define FL_BROWSER_BOXTYPE      FL_DOWN_BOX
#define FL_BROWSER_COL1         FL_COL1
#define FL_BROWSER_COL2         3
#define FL_BROWSER_LCOL         FL_LCOL
#define FL_BROWSER_ALIGN        FL_ALIGN_BOTTOM

/***** Others   *****/

#define FL_BROWSER_SLCOL        FL_COL1
#define FL_BROWSER_BW           FL_BOUND_WIDTH
#define FL_BROWSER_LINELENGTH   128
#define FL_BROWSER_MAXLINE      512

/***** Routines *****/

FL_OBJECT       *fl_create_browser(int, float, float, float, float, char []);
```



```
FL_OBJECT       *fl_add_browser(int, float, float, float, float, char []);

void    fl_set_browser_topline(FL_OBJECT *, int);
void    fl_clear_browser(FL_OBJECT *);
void    fl_add_browser_line(FL_OBJECT *, char []);
void    fl_addto_browser(FL_OBJECT *, char []);
void    fl_insert_browser_line(FL_OBJECT *, int , char []);
void    fl_delete_browser_line(FL_OBJECT *, int );
void    fl_replace_browser_line(FL_OBJECT *, int , char []);
char    *fl_get_browser_line(FL_OBJECT *, int);
int     fl_load_browser(FL_OBJECT *, char []);
int     fl_get_browser_maxline(FL_OBJECT *);
void    fl_select_browser_line(FL_OBJECT *, int );
void    fl_deselect_browser_line(FL_OBJECT *, int );
void    fl_deselect_browser(FL_OBJECT *);
int     fl_isselected_browser_line(FL_OBJECT *, int );
int     fl_get_browser(FL_OBJECT *);
void    fl_set_browser_fontsize(FL_OBJECT *, float);
void    fl_set_browser_fontstyle(FL_OBJECT *, int);
void    fl_set_browser_specialkey(FL_OBJECT *, char);
/************   Object Class: Button       ************/

/***** Class    *****/

#define FL_BUTTON                       11

/***** Types    *****/

#define FL_NORMAL_BUTTON        0
#define FL_PUSH_BUTTON          1
#define FL_RADIO_BUTTON         2
#define FL_HIDDEN_BUTTON        3
#define FL_TOUCH_BUTTON         4
#define FL_INOUT_BUTTON         5
#define FL_RETURN_BUTTON        6
#define FL_HIDDEN_RET_BUTTON    7

/***** Defaults *****/

#define FL_BUTTON_BOXTYPE       FL_UP_BOX
#define FL_BUTTON_COL1          FL_COL1
#define FL_BUTTON_COL2          FL_COL1
#define FL_BUTTON_LCOL          FL_LCOL
#define FL_BUTTON_ALIGN         FL_ALIGN_CENTER

/***** Others   *****/

#define FL_BUTTON_MCOL1         FL_MCOL
#define FL_BUTTON_MCOL2         FL_MCOL
#define FL_BUTTON_BW            FL_BOUND_WIDTH

/***** Routines *****/

FL_OBJECT       *fl_create_button(int, float, float, float, float, char []);
FL_OBJECT       *fl_add_button(int, float, float, float, float, char []);

int     fl_get_button(FL_OBJECT *);
void    fl_set_button(FL_OBJECT *, int);

void    fl_set_button_shortcut(FL_OBJECT *, char []);

/************   Object Class: Chart        ************/

/***** Class    *****/

#define FL_CHART                4
```

```
/***** Types    *****/

#define FL_BAR_CHART            0
#define FL_HORBAR_CHART         1
#define FL_LINE_CHART           2
#define FL_FILLED_CHART         3
#define FL_SPIKE_CHART          4
#define FL_PIE_CHART            5
#define FL_SPECIALPIE_CHART     6

/***** Defaults *****/

#define FL_CHART_BOXTYPE        FL_BORDER_BOX
#define FL_CHART_COL1           FL_COL1
#define FL_CHART_LCOL           FL_LCOL
#define FL_CHART_ALIGN          FL_ALIGN_BOTTOM

/***** Others   *****/

#define FL_CHART_BW             FL_BOUND_WIDTH
#define FL_CHART_MAX            128

/***** Routines *****/

FL_OBJECT       *fl_create_chart(int, float, float, float, float, char []);
FL_OBJECT       *fl_add_chart(int, float, float, float, float, char []);

void    fl_clear_chart(FL_OBJECT *);
void    fl_add_chart_value(FL_OBJECT *, float, char [], int);
void    fl_insert_chart_value(FL_OBJECT *, int, float, char [], int);
void    fl_replace_chart_value(FL_OBJECT *, int, float, char [], int);
void    fl_set_chart_bounds(FL_OBJECT *, float, float);
void    fl_set_chart_maxnumb(FL_OBJECT *, int);
void    fl_set_chart_autosize(FL_OBJECT *, int);
/************   Object Class: Choice       ************/

/***** Class    *****/

#define FL_CHOICE               42

/***** Types    *****/

#define FL_NORMAL_CHOICE        0

/***** Defaults *****/

#define FL_CHOICE_BOXTYPE       FL_DOWN_BOX
#define FL_CHOICE_COL1          FL_COL1
#define FL_CHOICE_COL2          FL_LCOL
#define FL_CHOICE_LCOL          FL_LCOL
#define FL_CHOICE_ALIGN         FL_ALIGN_LEFT

/***** Others   *****/

#define FL_CHOICE_BW            FL_BOUND_WIDTH
#define FL_CHOICE_MCOL          FL_MCOL
#define FL_CHOICE_MAXITEMS      128
#define FL_CHOICE_MAXSTR        64

/***** Routines *****/

FL_OBJECT       *fl_create_choice(int ,float ,float ,float ,float ,char []);
FL_OBJECT       *fl_add_choice(int ,float ,float ,float ,float ,char []);

void    fl_clear_choice(FL_OBJECT *);
```



```
void    fl_addto_choice(FL_OBJECT *, char []);
void    fl_replace_choice(FL_OBJECT *, int , char []);
void    fl_delete_choice(FL_OBJECT *, int );
void    fl_set_choice(FL_OBJECT *, int );
int     fl_get_choice(FL_OBJECT *);
char    *fl_get_choice_text(FL_OBJECT *);
void    fl_set_choice_fontsize(FL_OBJECT *, float);
void    fl_set_choice_fontstyle(FL_OBJECT *, int);


/************   Object Class: Clock        ************/

/***** Class    *****/

#define FL_CLOCK                61

/***** Types    *****/

#define FL_SQUARE_CLOCK         0
#define FL_ROUND_CLOCK          1

/***** Defaults *****/

#define FL_CLOCK_BOXTYPE        FL_UP_BOX
#define FL_CLOCK_COL1           37
#define FL_CLOCK_COL2           42
#define FL_CLOCK_LCOL           FL_LCOL
#define FL_CLOCK_ALIGN          FL_ALIGN_BOTTOM

/***** Others   *****/

#define FL_CLOCK_TOPCOL         FL_COL1
#define FL_CLOCK_BW             FL_BOUND_WIDTH

/***** Routines *****/

FL_OBJECT       *fl_create_clock(int, float, float, float, float, char []);
FL_OBJECT       *fl_add_clock(int, float, float, float, float, char []);

void    fl_get_clock(FL_OBJECT *,int * ,int *, int *);

/************   Object Class: Counter      ************/

/***** Class    *****/

#define FL_COUNTER              25

/***** Types    *****/

#define FL_NORMAL_COUNTER       0
#define FL_SIMPLE_COUNTER       1

/***** Defaults *****/

#define FL_COUNTER_BOXTYPE      FL_UP_BOX
#define FL_COUNTER_COL1         FL_COL1
#define FL_COUNTER_COL2         4
#define FL_COUNTER_LCOL         FL_LCOL
#define FL_COUNTER_ALIGN        FL_ALIGN_BOTTOM

/***** Others   *****/

#define FL_COUNTER_BW           FL_BOUND_WIDTH

/***** Routines *****/
```

```
FL_OBJECT       *fl_create_counter(int, float, float, float, float, char []);
FL_OBJECT       *fl_add_counter(int, float, float, float, float, char []);

void    fl_set_counter_value(FL_OBJECT *, float);
void    fl_set_counter_bounds(FL_OBJECT *, float, float);
void    fl_set_counter_step(FL_OBJECT *, float, float);
void    fl_set_counter_precision(FL_OBJECT *, int);
float   fl_get_counter_value(FL_OBJECT *);

void    fl_set_counter_return(FL_OBJECT *, int);
/************   Object Class: Dial         ************/

/***** Class    *****/

#define FL_DIAL                 22

/***** Types    *****/

#define FL_NORMAL_DIAL          0
#define FL_LINE_DIAL            1

/***** Defaults *****/

#define FL_DIAL_BOXTYPE         FL_NO_BOX
#define FL_DIAL_COL1            FL_COL1
#define FL_DIAL_COL2            37
#define FL_DIAL_LCOL            FL_LCOL
#define FL_DIAL_ALIGN           FL_ALIGN_BOTTOM

/***** Others   *****/

#define FL_DIAL_TOPCOL          FL_COL1
#define FL_DIAL_BW              FL_BOUND_WIDTH

/***** Routines *****/

FL_OBJECT       *fl_create_dial(int ,float, float, float, float, char []);
FL_OBJECT       *fl_add_dial(int ,float, float, float, float, char []);

void    fl_set_dial_value(FL_OBJECT *, float);
float   fl_get_dial_value(FL_OBJECT *);
void    fl_set_dial_bounds(FL_OBJECT *, float, float);
void    fl_get_dial_bounds(FL_OBJECT *, float *, float*);

void    fl_set_dial_step(FL_OBJECT *, float);
void    fl_set_dial_return(FL_OBJECT *, int);
/************   Object Class: Free         ************/

/***** Class    *****/

#define FL_FREE                 101

/***** Types    *****/

#define FL_NORMAL_FREE          1
#define FL_SLEEPING_FREE        2
#define FL_INPUT_FREE           3
#define FL_CONTINUOUS_FREE      4
#define FL_ALL_FREE             5

/***** Defaults *****/

/***** Others   *****/

/***** Routines *****/
```



```
FL_OBJECT       *fl_create_free(int, float, float, float, float, char [], FL_HANDLEPT
 R);
FL_OBJECT       *fl_add_free(int, float, float, float, float, char [], FL_HANDLEPTR);

/************   Object Class: Input        ************/

/***** Class    *****/

#define FL_INPUT                31

/***** Types    *****/

#define FL_NORMAL_INPUT         0
#define FL_FLOAT_INPUT          1
#define FL_INT_INPUT            2
#define FL_HIDDEN_INPUT         3

/***** Defaults *****/

#define FL_INPUT_BOXTYPE        FL_DOWN_BOX
#define FL_INPUT_COL1           13
#define FL_INPUT_COL2           5
#define FL_INPUT_LCOL           FL_LCOL
#define FL_INPUT_ALIGN          FL_ALIGN_LEFT

/***** Others   *****/

#define FL_INPUT_TCOL           FL_LCOL
#define FL_INPUT_CCOL           4
#define FL_INPUT_BW             FL_BOUND_WIDTH
#define FL_INPUT_MAX            128

/***** Routines *****/

FL_OBJECT       *fl_create_input(int, float, float, float, float, char []);
FL_OBJECT       *fl_add_input(int, float, float, float, float, char []);

void    fl_set_input(FL_OBJECT *, char []);
void    fl_set_input_color(FL_OBJECT *, int, int);
char    *fl_get_input(FL_OBJECT *);
void    fl_set_input_return(FL_OBJECT *, int);

/************   Object Class: Lightbutton  ************/

/***** Class    *****/

#define FL_LIGHTBUTTON          12

/***** Types    *****/

    /* Same types as for buttons */

/***** Defaults *****/

#define FL_LIGHTBUTTON_BOXTYPE  FL_UP_BOX
#define FL_LIGHTBUTTON_COL1     39
#define FL_LIGHTBUTTON_COL2     3
#define FL_LIGHTBUTTON_LCOL     FL_LCOL
#define FL_LIGHTBUTTON_ALIGN    FL_ALIGN_CENTER

/***** Others   *****/

#define FL_LIGHTBUTTON_TOPCOL   FL_COL1
#define FL_LIGHTBUTTON_MCOL     FL_MCOL
#define FL_LIGHTBUTTON_BW1      FL_BOUND_WIDTH
#define FL_LIGHTBUTTON_BW2      FL_BOUND_WIDTH/2.0
```

```
#define FL_LIGHTBUTTON_MINSIZE  12.0

/***** Routines *****/

FL_OBJECT  *fl_create_lightbutton(int, float, float, float, float, char []);
FL_OBJECT  *fl_add_lightbutton(int, float, float, float, float, char []);

/************   Object Class: Menu         ************/

/***** Class    *****/

#define FL_MENU                 41

/***** Types    *****/

#define FL_TOUCH_MENU           0
#define FL_PUSH_MENU            1

/***** Defaults *****/

#define FL_MENU_BOXTYPE         FL_BORDER_BOX
#define FL_MENU_COL1            55
#define FL_MENU_COL2            37
#define FL_MENU_LCOL            FL_LCOL
#define FL_MENU_ALIGN           FL_ALIGN_CENTER

/***** Others   *****/

#define FL_MENU_BW              FL_BOUND_WIDTH
#define FL_MENU_MAX             300

/***** Routines *****/

FL_OBJECT       *fl_create_menu(int, float, float, float, float, char []);
FL_OBJECT       *fl_add_menu(int, float, float, float, float, char []);

void    fl_set_menu(FL_OBJECT *, char []);
void    fl_addto_menu(FL_OBJECT *, char []);
long    fl_get_menu(FL_OBJECT *);

/************   Object Class: Positioner   ************/

/***** Class    *****/

#define FL_POSITIONER           23

/***** Types    *****/

#define FL_NORMAL_POSITIONER    0

/***** Defaults *****/

#define FL_POSITIONER_BOXTYPE   FL_DOWN_BOX
#define FL_POSITIONER_COL1      FL_COL1
#define FL_POSITIONER_COL2      1
#define FL_POSITIONER_LCOL      FL_LCOL
#define FL_POSITIONER_ALIGN     FL_ALIGN_BOTTOM

/***** Others   *****/

#define FL_POSITIONER_BW        FL_BOUND_WIDTH

/***** Routines *****/

FL_OBJECT       *fl_create_positioner(int, float, float, float, float, char []);
FL_OBJECT       *fl_add_positioner(int, float, float, float, float, char []);
```



```
void    fl_set_positioner_xvalue(FL_OBJECT *, float);
float   fl_get_positioner_xvalue(FL_OBJECT *);
void    fl_set_positioner_xbounds(FL_OBJECT *, float, float);
void    fl_get_positioner_xbounds(FL_OBJECT *, float *, float *);
void    fl_set_positioner_yvalue(FL_OBJECT *, float);
float   fl_get_positioner_yvalue(FL_OBJECT *);
void    fl_set_positioner_ybounds(FL_OBJECT *, float, float);
void    fl_get_positioner_ybounds(FL_OBJECT *, float *, float *);
void    fl_set_positioner_xstep(FL_OBJECT *, float);
void    fl_set_positioner_ystep(FL_OBJECT *, float);


void    fl_set_positioner_return(FL_OBJECT *, int);
/************   Object Class: Roundbutton  ************/

/***** Class    *****/

#define FL_ROUNDBUTTON          13

/***** Types    *****/

    /* Same types as for buttons */

/***** Defaults *****/

#define FL_ROUNDBUTTON_BOXTYPE  FL_NO_BOX
#define FL_ROUNDBUTTON_COL1     7
#define FL_ROUNDBUTTON_COL2     3
#define FL_ROUNDBUTTON_LCOL     FL_LCOL
#define FL_ROUNDBUTTON_ALIGN    FL_ALIGN_CENTER

/***** Others   *****/

#define FL_ROUNDBUTTON_TOPCOL   FL_COL1
#define FL_ROUNDBUTTON_MCOL     FL_MCOL
#define FL_ROUNDBUTTON_BW       FL_BOUND_WIDTH

/***** Routines *****/

FL_OBJECT  *fl_create_roundbutton(int, float, float, float, float, char []);
FL_OBJECT  *fl_add_roundbutton(int, float, float, float, float, char []);


/************   Object Class: Slider       ************/

/***** Class    *****/

#define FL_SLIDER               21
#define FL_VALSLIDER            24

/***** Types    *****/

#define FL_VERT_SLIDER          0
#define FL_HOR_SLIDER           1
#define FL_VERT_FILL_SLIDER     2
#define FL_HOR_FILL_SLIDER      3
#define FL_VERT_NICE_SLIDER     4
#define FL_HOR_NICE_SLIDER      5

/***** Defaults *****/

#define FL_SLIDER_BOXTYPE       FL_DOWN_BOX
#define FL_SLIDER_COL1          FL_COL1
#define FL_SLIDER_COL2          FL_COL1
#define FL_SLIDER_LCOL          FL_LCOL
```

```
#define FL_SLIDER_ALIGN         FL_ALIGN_BOTTOM

/***** Others   *****/

#define FL_SLIDER_BW1           FL_BOUND_WIDTH
#define FL_SLIDER_BW2           FL_BOUND_WIDTH*0.75

#define FL_SLIDER_FINE          0.05
#define FL_SLIDER_WIDTH         0.08

/***** Routines *****/

FL_OBJECT       *fl_create_slider(int, float, float, float, float, char []);
FL_OBJECT       *fl_add_slider(int, float, float, float, float, char []);

FL_OBJECT       *fl_create_valslider(int, float, float, float, float, char []);
FL_OBJECT       *fl_add_valslider(int, float, float, float, float, char []);

void    fl_set_slider_value(FL_OBJECT *, float);
float   fl_get_slider_value(FL_OBJECT *);
void    fl_set_slider_bounds(FL_OBJECT *, float, float);
void    fl_get_slider_bounds(FL_OBJECT *, float *, float *);

void    fl_set_slider_return(FL_OBJECT *, int);

void    fl_set_slider_step(FL_OBJECT *, float);
void    fl_set_slider_size(FL_OBJECT *, float);
void    fl_set_slider_precision(FL_OBJECT *, int);
/************   Object Class: Text         ************/

/***** Class    *****/

#define FL_TEXT                 2

/***** Types    *****/

#define FL_NORMAL_TEXT          0

/***** Defaults *****/

#define FL_TEXT_BOXTYPE         FL_NO_BOX
#define FL_TEXT_COL1            FL_COL1
#define FL_TEXT_LCOL            FL_LCOL
#define FL_TEXT_ALIGN           FL_ALIGN_LEFT

/***** Others   *****/

#define FL_TEXT_BW              FL_BOUND_WIDTH

/***** Routines *****/

FL_OBJECT       *fl_create_text(int, float, float, float, float, char []);
FL_OBJECT       *fl_add_text(int, float, float, float, float, char []);

/************   Object Class: Timer        ************/

/***** Class    *****/

#define FL_TIMER                62

/***** Types    *****/

#define FL_NORMAL_TIMER         0
#define FL_VALUE_TIMER          1
#define FL_HIDDEN_TIMER         2
```



```
/***** Defaults *****/

#define FL_TIMER_BOXTYPE        FL_DOWN_BOX
#define FL_TIMER_COL1           FL_COL1
#define FL_TIMER_COL2           1
#define FL_TIMER_LCOL           FL_LCOL
#define FL_TIMER_ALIGN          FL_ALIGN_CENTER

/***** Others   *****/

#define FL_TIMER_BW             FL_BOUND_WIDTH
#define FL_TIMER_BLINKRATE      0.2

/***** Routines *****/

FL_OBJECT       *fl_create_timer(int, float, float, float, float, char []);
FL_OBJECT       *fl_add_timer(int, float, float, float, float, char []);

void    fl_set_timer(FL_OBJECT *, float);
float   fl_get_timer(FL_OBJECT *);


#if defined(_LANGUAGE_C_PLUS_PLUS) || defined(__cplusplus)
}
#endif

#endif  /* define __FORMS_H__ */
```

```
/* Header file generated with fdesign. */

/**** Callback routines ****/

extern void set_tension(FL_OBJECT *, long);
extern void set_thr2(FL_OBJECT *, long);
extern void set_thr1(FL_OBJECT *, long);
extern void spine_on_off(FL_OBJECT *, long);
extern void collision_on_off(FL_OBJECT *, long);
extern void breathe_on_off(FL_OBJECT *, long);
extern void set_elbsize(FL_OBJECT *, long);


/**** Forms and Objects ****/

extern FL_FORM *GraphTool;

extern FL_OBJECT
        *tension_slider,
        *rate_slider,
        *xy_chart,
        *intensity_slider,
        *spine_button,
        *collision_button,
        *breathe_button,
        *elbsize_slider;


/**** Creation Routine ****/

extern void create_the_forms();
```

```
#define LELBOW 0
#define RELBOW 1
#define LSHOULDER 2
#define RSHOULDER 3
#define LWRIST 4
#define RWRIST 5
#define LHIP 6
#define RHIP 7

#define HEAD 8
#define NECK 9
#define LUARM 10
#define RUARM 11
#define LLARM 12
#define RLARM 13
#define LHND 14
#define RHND 15
#define TORSO 16
#define LULG 17
#define RULG 18
#define LLLG 19
#define RLLG 20

#define MAXJNTS 8
#define MAXIMPS 21

#define FINT 0.7435
/*
#define THR 0.2
*/
#define DFWT 15.1989       /* 1/(1-fint)^2 */
#define BIGGIE  10000
#define EPS 0.01

#define quartF(x) ((x-1)*(x-1)*(x+1)*(x+1))
#define blendF(x)  (x)*(x)*(3.0-2.0*(x))


#define PB 0
#define SPHERE 1
#define CYLINDER 2
#define SPHYLINDER 3
#define CONESPHERE 4


typedef struct Ipb {
  int type;
  float weight;                 /* shape weight */
  float cc[4];                  /* poly coeffs. */
  float scval;                  /* scale up for threshold, wt */
  VERTEX  o1,o2,o3,o4,c1,c2,c3,c4;      /* local origin */
  float r1,r2,sr1,sr2;          /* radius and sc rad for sph. etc */
  VERTEX  oc,ouc,c,uc,apex;
  float ang,tang,cotang,nc;        /* additional params */
} Ipb;



typedef struct joint {
  int l1,l2;
  float comber;
} Joint;


extern Ipb ipb[MAXIMPS];
extern Joint jt[MAXJNTS];
```



```
extern VERTEX *spv[4],*spn[4];
extern int *spf[4],spvc[4],spfc[4];

extern VERTEX spcenter;
extern float r1,sr1;


extern float diffFunc();
extern float diffFunc2();
extern float cvFunc();
extern float THR;
extern float THR2;
```

```
/*************************************
*           関数名の変更              *
*                                     *
*       ファイル名：rname_fc.h        *
*************************************/
#define   initialize  c_initialize
#define   human_end   c_human_end
#define   calc_model  c_calc_model
#define   draw_model  c_draw_model

#define   pushmatrix()
#define   popmatrix()

#define   loadmatrix  c_loadmatrix
#define   getmatrix   c_getmatrix
#define   multmatrix  c_multmatrix

#define   rot         c_rot
#define   translate   c_translate
```

```
/*
 * ファイル名 : tool.h
 *
 * 機能       : 3次元変形表示確認テストプログラム関数定義
 *
 * 履歴       : 10月28日 ; 作成 ; 広瀬  正俊
 */
#include        "body_draw.h"

#ifndef EXTRN
#define EXTRN   extern
#endif

#define         NOCOMMAND       -1      /* no command select */
#define         SCALING         1
#define         TRANSLATING     2
#define         DRAW_3D_WIRE    51      /* draw 3D wire model */
#define         DRAW_3D_TEXT    52      /* draw 3D texture model */
#define         TEXT_AND_WIRE   53      /* draw 3D texture and wire model */

EXTRN   char            G_filename[MAX_NAME];   /* file name */
EXTRN   int             G_menu;                 /* menu ID */
EXTRN   long            G_winid;                /* window ID */
EXTRN   int             G_cmdst;                /* current command status */
EXTRN   float           G_rota, G_elea;                 /* 3D rotate angle */
EXTRN   float           G_black[3];
EXTRN   float           G_white[3];
EXTRN   float           G_red[3];
EXTRN   float           G_green[3];
EXTRN   float           G_blue[3];
EXTRN   float           G_yellow[3];
EXTRN   float           G_pink[3];
EXTRN   float           G_back[3];
EXTRN   float           G_3dxmin, G_3dxmax;
EXTRN   float           G_3dymin, G_3dymax;
EXTRN   float           G_3dzmin, G_3dzmax;
EXTRN   float           G_dxmin, G_dxmax;
EXTRN   float           G_dymin, G_dymax;
EXTRN   float           G_dzmin, G_dzmax;
EXTRN   float           G_xmin, G_xmax;
EXTRN   float           G_ymin, G_ymax;
EXTRN   float           G_zmin, G_zmax;
EXTRN   float           G_oxmin, G_oxmax;
EXTRN   float           G_oymin, G_oymax;
EXTRN   float           G_ozmin, G_ozmax;
EXTRN   int             G_mdlbtn;
EXTRN   float           G_movex, G_movey;
EXTRN   int             G_hand;
EXTRN   int             G_kahansin;
EXTRN   char            G_tr_data[MAX_NAME];
EXTRN   char            G_ang_data[MAX_NAME];

EXTRN   int             maptest[20];
EXTRN   int             mapn;

EXTRN   int             G_glass;
EXTRN   int             G_four;
EXTRN   int             G_munekara;

/*****     <<< main.c >>>      *****/

/*****     <<< data.c >>>      *****/
void            Information(int, void *[]);
void            CalcYubiAngle(JINBUTU *, int, float, float, Matrix *);
void            WriteWave(JINBUTU *, char *, int);
```



```
/*****     <<< draw3.c >>>     *****/
void            ReadTrackerData(int, void *[]);
void            ReadTrackerDataFile(int, void *[]);
void            ReadAngleDataFile(int, void *[]);
void            ResetTrackerData(int, void *[]);
void            DrawHuman(int, void *[]);
void            DrawTexture(int, void *[]);
void            DrawWire(int, void *[]);
void            TextureAndWire(int, void *[]);
void            Draw3DWire(JINBUTU *, int, float *);
void            Draw3DWireSeiza(JINBUTU *, int, float *);
void            Draw3DWireSeiza2(JINBUTU *, int, float *);
void            Draw3DWireSeiza3(JINBUTU *, int, float *);
void            DrawYubiKansetuWire(JINBUTU *, int);
void            DrawMap3Wire(JINBUTU *, int, VERTEX *, float *);
void            DrawSaikoro(float, float, float, float);
void            DrawSimple3DWire(JINBUTU *);

/*****     <<< init.c >>>      *****/
void            GlobalSet();
void            InitializeGL(int );
void            SetDataMinMax3(int, int, void *[], int);
void            ViewSize3(float, float, float, float, float, float);
void            InitViewSize3();
void            SetCounterValue(float *);
void            BackWhite();
void            BackBlack();
void            BackGrey();

/*****     <<< menu.c >>>      *****/
void            MainLoop(int, void *[]);
int             MenuInit();
void            QuitTool();

/*****     <<< scaling.c >>>     *****/
void            ChangeTracker(int, void *[], int, int);
void            Scaling(int, void *[], int);

/*****     <<< calib.c >>>     *****/
void            CalibMune(int, void *[]);
```

```
/*---------------------------------------------------------------------------
 * Vector macros
 *---------------------------------------------------------------------------
 */
#define VECDot(F,G)         ((F)[0]*(G)[0] + (F)[1]*(G)[1] + (F)[2]*(G)[2])
#define VECLen(F)           (sqrt((double)VECDot((F),(F))))

#define VECAsgn(a,b,c,R)    { (R)[0] = (a); (R)[1] = (b); (R)[2] = (c); }

#define VECNeg(F,R)         { (R)[0] = -(F)[0]; (R)[1] = -(F)[1]; \
                              (R)[2] = -(F)[2]; }
#define VECCopy(S,D)        { (D) = (S); }

#define VECAdd(F,G,R)       { (R)[0] = (F)[0] + (G)[0]; \
                              (R)[1] = (F)[1] + (G)[1]; \
                              (R)[2] = (F)[2] + (G)[2]; }

#define VECSub(F,G,R)       { (R)[0] = (F)[0] - (G)[0]; \
                              (R)[1] = (F)[1] - (G)[1]; \
                              (R)[2] = (F)[2] - (G)[2]; }

#define VECMpy(F,G,R)       { (R)[0] = (F)[0] * (G)[0]; \
                              (R)[1] = (F)[1] * (G)[1]; \
                              (R)[2] = (F)[2] * (G)[2]; }

#define VECCross(F,G,R)     { (R)[0] = (F)[1]*(G)[2] - (F)[2]*(G)[1]; \
                              (R)[1] = (F)[2]*(G)[0] - (F)[0]*(G)[2]; \
                              (R)[2] = (F)[0]*(G)[1] - (F)[1]*(G)[0]; }

#define VECMpyS(F,f,R)      { (R)[0] = (f)*(F)[0]; (R)[1] = (f)*(F)[1]; \
                              (R)[2] = (f)*(F)[2]; }

#define VECDivS(F,f,R)      { (R)[0] = (F)[0]/(f); (R)[1] = (F)[1]/(f); \
                              (R)[2] = (F)[2]/(f); }

#define VECComb(f,F,g,G,R) { (R)[0] = (f)*(F)[0] + (g)*(G)[0]; \
                             (R)[1] = (f)*(F)[1] + (g)*(G)[1]; \
                             (R)[2] = (f)*(F)[2] + (g)*(G)[2]; }
#define VECAddS(f,F,G,R)    { (R)[0] = (f)*(F)[0] + (G)[0]; \
                              (R)[1] = (f)*(F)[1] + (G)[1]; \
                              (R)[2] = (f)*(F)[2] + (G)[2]; }

#define VECNorm(F)         { float VECtemp = VECLen((F)); \
                             if (VECtemp != 0.0) { VECDivS((F),VECtemp,(F));} \
                             else { VECClear(F);}}

#define VECMake(F,G,R)      VECSub((G),(F),(R))

#define VECDist(F,G)       (sqrt(SQR((F)[0] - (G)[0]) + SQR((F)[1] - (G)[1]) \
                                 + SQR((F)[2] - (G)[2])))
#define VECClear(R)         { (R)[0] = 0.0; (R)[1] = 0.0; (R)[2] = 0.0; }

#define VECSet(R,G)         { (R)[0] = (G)[0]; (R)[1]=(G)[1]; (R)[2]=(G)[2]; }
```

```
/* hyouka function 1993.6.1 by H.ishii
* 右肘の位置を決定するための、エネルギー評価関数
* 入力:7つの関節角度(angk1-angt3)と
       肘の3次元位置(rhiji_xyz)および
       過去2フレームのそれらの値、（注、すべてグローバル変数）
* 出力:*p      重力によるエネルギ
       *e1     関節ストレスによるエネルギ
       *e2     摩擦によるエネルギ
       *e3     加減速によるエネルギ
       *hyouka それらの総和
* モデル  下腕を、π r＝2 7 cm、l＝2 8 cm 比重1の円筒のモデルとする。
         上腕を、π r＝3 4 cm、l＝2 8 cm 比重1の円筒のモデルとする。
        それぞれの重さは、m1 = 27*27* 28 / (4*3.14) = 1625.16 g
                          m2 = 34*34* 28 / (4*3.14) = 2577.07 g
        さらに、それぞれ太さ0、長さ28 cmの、スティックの簡単化する。
        肩、肘、手首の高さを、hk,hh,htとすると、それぞれ重力によるポテンシャルは、
            p1 = G*(ht + hh)* m1 /2
            p2 = G*(hk + hh)* m2 /2
        であるが、ここでは、hk,ht は一定なので評価すべき、ポテンシャルp は、
            p = G*hh*(m1+m2)/2
              = 980cm/sec2 * hh cm * 4202.23 g /2
              = 2059092.70 * hh g*cm2/sec2
              = 205.9 * hh ( g*m2/sec2 )
         である。
         つぎに運動エネルギは、肩、肘、手首の位置ベクトルを、kxyz,hxyz,txyzとすると、
         ここでも肩、手首位置は一定なので、評価すべき運動エネルギは、
           e3 = 力・距離 = 慣性*加速度・距離
                          (m1/2) * (d2 hxyz/dt2) • d hxyz
                         +(m2/2) * (d2 hxyz/dt2) • d hxyz
              - 2101.11 * (d2 hxyz/dt2) * d hxyz g*cm2/sec2
         ただし、ここではスティックの軸方向の回転エネルギは考えない。
         また、時間が十分短いものとし、遠心力による速度変化を無視する（無視していいの？）
         2フレーム前までの肘の位置をそれぞれ hxyz,hxyz',hxyz" とし、
         フレーム間隔 dt = 0.1 sec と仮定すると、
           e3 = 210111. * ((hxyz - hxyz')-(hxyz'-hxyz"))• (hxyz - hxyz') g *cm2/sec2
                21.0 * dd_hxyz •  d_hxyz (g*m2/sec2)
*/

#include <stdio.h>
#include <math.h>

float
ishii_hyouka( angk1, angk2, angk3, rhiji_xyz, angt1, angt2, angt3,
        exangk1, exangk2, exangk3, exrhiji_xyz, exangt1, exangt2, exangt3,
        ex2angk1, ex2angk2, ex2angk3, ex2rhiji_xyz, ex2angt1, ex2angt2, ex2angt3)
float  angk1, angk2, angk3, angt1, angt2, angt3;
float  rhiji_xyz[3];
float  exangk1, exangk2, exangk3, exangt1, exangt2, exangt3;
float  exrhiji_xyz[3];
float  ex2angk1, ex2angk2, ex2angk3, ex2angt1, ex2angt2, ex2angt3;
float  ex2rhiji_xyz[3];
{
        float  hyouka, p, e1, e2, e3;
        float  d1,d2,d3,d4,d5,d6,d7;

        float  d_rhiji_xyz[3];
        float  d2_rhiji_xyz[3];

/*
        fprintf( stderr," (%f,%f,%f) (%f,%f,%f) (%f,%f,%f)\n"
         ,angk1, angk2, angk3, angt1, angt2, angt3
         , rhiji_xyz[0], rhiji_xyz[1], rhiji_xyz[2]);
*/
         /* 肘の高さから重力によるエネルギ計算 */
         p  = 205.9 *(rhiji_xyz[1] + 28.); /* g * m2/sec2 */
```



```
        /* 無ストレス範囲を越えた量(d1-d7) からストレスによるエネルギ計算 */
        d1 = d2 = d3 = d4 = d5 = d6 = d7 = 0.;

        /* 二の腕のねじり（肩x方向） */
        if( angk1 <  -80) d1 = angk1 +80;
        if( angk1 > 60) d1 = angk1 -60;

        /* 腕のねじり（手首x方向） */
        if( angt1 < -50) d5 = angt1 +50;
        if( angt1 >  20) d5 = angt1 -20;

        /* （手首y方向） */
        if( angt2 < -20) d6 = angt2 +20;
        if( angt2 >  20) d6 = angt2 -20;

        /* （手首z方向） */
        if( angt3 < -20) d7 = angt3 +20;
        if( angt3 >  50) d7 = angt3 -50;

        /* 肩yz方向の回転のストレスは、肘の3次元位置から計算 */
        if( rhiji_xyz[0] >  -9.) d2 = rhiji_xyz[0] +9.;
        if( rhiji_xyz[1] >  -5.) d3 = rhiji_xyz[1] +5.;
        if( rhiji_xyz[2] <   5.) d4 = rhiji_xyz[2] -5.;
        e1 =  200*(d1*d1 + d5*d5 + d6*d6 + d7*d7);
        e1 +=  200*( d2*d2 + d3*d3 + d4*d4);

        /* 各関節角度の変化から摩擦によるエネルギ計算 */
        e2 =  100*(fabs(exangk1 - angk1)
                + fabs(exangk2 - angk2)
                + fabs(exangk3 - angk3)
                + fabs(exangt1 - angt1)
                + fabs(exangt2 - angt2)
                + fabs(exangt3 - angt3));

        /* 過去2フレームの肘の位置 rhiji_xyz, ex_rhiji_xyz, eex_rhiji_xyz からの
           ずれから、加減速エネルギを計算 */
        d_rhiji_xyz[0] =  rhiji_xyz[0] - exrhiji_xyz[0];
        d_rhiji_xyz[1] =  rhiji_xyz[1] - exrhiji_xyz[1];
        d_rhiji_xyz[2] =  rhiji_xyz[2] - exrhiji_xyz[2];
        d2_rhiji_xyz[0] =  rhiji_xyz[0] + ex2rhiji_xyz[0] - 2*exrhiji_xyz[0];
        d2_rhiji_xyz[1] =  rhiji_xyz[1] + ex2rhiji_xyz[1] - 2*exrhiji_xyz[1];
        d2_rhiji_xyz[2] =  rhiji_xyz[2] + ex2rhiji_xyz[2] - 2*exrhiji_xyz[2];
        e3 = fabs(21.0 * ( d2_rhiji_xyz[0] * d_rhiji_xyz[0] +
                           d2_rhiji_xyz[1] * d_rhiji_xyz[1] +
                           d2_rhiji_xyz[2] * d_rhiji_xyz[2] ));

        /* エネルギ総和 */

        hyouka = p + e1  + e2 + e3 ;

        return(hyouka);
}
```

```
#include "tool.h"


#define blendF(x)  (x)*(x)*(3.0-2.0*(x))

static  Matrix  G_idmat = {1.0, 0.0, 0.0, 0.0,
                           0.0, 1.0, 0.0, 0.0,
                           0.0, 0.0, 1.0, 0.0,
                           0.0, 0.0, 0.0, 1.0};

Matrix  sc_mat = {1.0, 0.0, 0.0, 0.0,
                  0.0, 1.0, 0.0, 0.0,
                  0.0, 0.0, 1.0, 0.0,
                  0.0, 0.0, 0.0, 1.0};

/*Matrix        mune_mat = {1.0, 0.0, 0.0, 0.0,
                  0.0, 1.0, 0.0, 0.0,
                  0.0, 0.0, 1.0, 0.0,
                  0.0, 0.0, 0.0, 1.0};
*/

float SCFAC=2.3,POWF=1.5; /* control amount, spread of outward scaling */

/*Space douspace,kosispace;*/
int ffdswitch=1;
int breatheswitch=0;
int showimp=0,collon=0;

/*VERTEX shcenter[2];
float tt[2][5];*/
float   bb[3][2];

float BRFAC=0,BRINC=0.02,BRMAX=0.3;

/* sc Angle */
/* returns a factor based on the rotation angle of a matrix m
   rr= m[0][0]+m[1][1]+m[2][2]
   value returned is 0..pi -> 0..SCFAC */

float scAngle(rr)
float rr;
{
return(SCFAC*acos((rr-1)/2.0)/M_PI);
}


/* blendF */
/* value must be in range 0..1, map has the following constraints
   f(0)=f'(0)=0,f(1)=1,f'(1)=0,f(0.5)=0.5 */
/*
float blendF(x)
float x;
{
 return(x*x*(3.0-2.0*x));
}
*/


/* bellF */
/* makes the expansion localised around 0.5 */
/* f.0 = 0 , f.1 =0 , f.0.5=1 */

float bellF(x)
float x;
{
  float tmp;
```



```
  tmp=blendF(1-fabs(2*x-1));
  return((float)pow((double)tmp,POWF));
}


/************************* space functions **********************/

/* setSpace */
/* initialises the ffd space s around buhin part k in ortho dir d
   LEEWAY allows space around bbox of k. div. of 4 planes of space
   is uniform */

#define LEEWAY 0.0


setTvals(s,hc,vhm,d,w)
Space *s;
int hc;
void   *vhm[];
int d,w;
{
  int i;
  float sz,tmp,tmp2;
  JINBUTU       *hm;

  hm=(JINBUTU   *)vhm[hc];

  hm->tval[w]=(float *)malloc(sizeof(float)*hm->vmax[w]);
  hm->tt1[w]=(float *)malloc(sizeof(float)*hm->vmax[w]);
  hm->tt2[w]=(float *)malloc(sizeof(float)*hm->vmax[w]);
  hm->tt3[w]=(float *)malloc(sizeof(float)*hm->vmax[w]);
  hm->tt4[w]=(float *)malloc(sizeof(float)*hm->vmax[w]);

  sz=s->corig[3][d]-s->corig[0][d];

  for (i=0;i<hm->vmax[w];i++)
    if (inSpace(hm->vs[w][i],s))
      {
        tmp=hm->tval[w][i]= (hm->vs[w][i][d]-s->corig[0][d])/sz;
        tmp2=1-tmp;
        hm->tt1[w][i]=tmp2*tmp2*tmp2;
        hm->tt2[w][i]=3*tmp2*tmp2*tmp;
        hm->tt3[w][i]=3*tmp2*tmp*tmp;
        hm->tt4[w][i]=tmp*tmp*tmp;
      }
    else
      {
        hm->tval[w][i]=-1;
        if (w==KOSI)
          {
            tmp=(hm->vs[w][i][d]-hm->douspace.corig[0][d])/
              (hm->douspace.corig[3][d]-hm->douspace.corig[0][d]);
            tmp2=1-tmp;
            hm->tt1[w][i]=tmp2*tmp2*tmp2;
            hm->tt2[w][i]=3*tmp2*tmp2*tmp;
            hm->tt3[w][i]=3*tmp2*tmp*tmp;
            hm->tt4[w][i]=tmp*tmp*tmp;
          }
/***** by M.Hirose 1995/2/9 *****/
/* inSpace(hm->vs[w][i],s) == 0 notokimo 1 notokito onaji shoriwo suru */
else if (w == DOU) {
    tmp=hm->tval[w][i]= (hm->vs[w][i][d]-s->corig[0][d])/sz;
    tmp2=1-tmp;
    hm->tt1[w][i]=tmp2*tmp2*tmp2;
    hm->tt2[w][i]=3*tmp2*tmp2*tmp;
```

```
    hm->tt3[w][i]=3*tmp2*tmp*tmp;
    hm->tt4[w][i]=tmp*tmp*tmp;
}
/***** by M.Hirose 1995/2/9 *****/
      }
]


setcenterT(hm,s,d,ind)
     JINBUTU *hm;
     Space *s;
int d,ind;
{
  float sz,tmp,tmp2;

  sz=s->corig[3][d]-s->corig[0][d];
  tmp=hm->tt[ind][0]= (hm->shcenter[ind][d]-s->corig[0][d])/sz;
  tmp2=1-tmp;
  hm->tt[ind][1]=tmp2*tmp2*tmp2;
  hm->tt[ind][2]=3*tmp2*tmp2*tmp;
  hm->tt[ind][3]=3*tmp2*tmp*tmp;
  hm->tt[ind][4]=tmp*tmp*tmp;
}


setSpace(s,hc,k,d,vhm)
Space *s;
int hc,k,d;
void *vhm[];
{
  int i,j,d1,d2;
  JINBUTU *hmm;

  hmm=(JINBUTU *)vhm[hc];

  c_loadmatrix(G_idmat);
  c_getmatrix(hmm->mune_mat);

/*  SetDataMinMax3(hc,vhm,k);
  bb[0][0] = G_3dxmin - (G_3dxmax-G_3dxmin)*LEEWAY;
  bb[0][1] = G_3dxmax + (G_3dxmax-G_3dxmin)*LEEWAY;
  bb[1][0] = G_3dymin - (G_3dymax-G_3dymin)*LEEWAY;
  bb[1][1] = G_3dymax + (G_3dymax-G_3dymin)*LEEWAY;
  bb[2][0] = G_3dzmin - (G_3dzmax-G_3dzmin)*LEEWAY;
  bb[2][1] = G_3dzmax + (G_3dzmax-G_3dzmin)*LEEWAY;
*/

  d1=(d+1)%3;
  d2=(d+2)%3;

  for (i=0;i<4;i++)
    {
      s->corig[i][d]=s->center[i][d]=bb[d][0]+(bb[d][1]-bb[d][0])*i/3.0;
      s->corig[i][d1]=s->center[i][d1]=(bb[d1][0]+bb[d1][1])/2.0;
      s->corig[i][d2]=s->center[i][d2]=(bb[d2][0]+bb[d2][1])/2.0;

      for (j=0;j<4;j++)
        {
          s->orig[i][j][d]=s->pt[i][j][d]=s->center[i][d];
          if (j%3)
             s->orig[i][j][d2]=s->pt[i][j][d2]=bb[d2][1];
          else
             s->orig[i][j][d2]=s->pt[i][j][d2]=bb[d2][0];
          /* bunches 0,1 and 2,3 */
          if (j>1)
             s->orig[i][j][d1]=s->pt[i][j][d1]=bb[d1][1];
```



```
          else
             s->orig[i][j][d1]=s->pt[i][j][d1]=bb[d1][0];
        ]
    }
  s->xang=s->yang=s->zang=0;

  setTvals(s,hc,vhm,d,k);

  if (k==KOSI)
    {
      setTvals(s,hc,vhm,d,SIRI_R);
      setTvals(s,hc,vhm,d,SIRI_L);
    }

  if (k==DOU)
    {
      setTvals(s,hc,vhm,d,KUBI);
      setTvals(s,hc,vhm,d,KATA_D_R);
      setTvals(s,hc,vhm,d,KATA_D_L);
      setTvals(s,hc,vhm,d,KATA_N_R);
      setTvals(s,hc,vhm,d,KATA_N_L);

      setcenterT(hmm,s,d,1);
      setcenterT(hmm,s,d,0);
    }
}


/* inSpace */
/* returns if point p lies in undeformed space s */

int inSpace(p,s)
VERTEX p;
Space *s;
{
  int i;

  for (i=0;i<3;i++)
    if ((p[i]<s->orig[0][0][i]) || (p[i]>s->orig[3][2][i]))
      return(0);
  return(1);
}


/* drawSpace */
/* draws the space in blue with the center line in black */

drawSpace(s)
Space *s;
{
  int i;
 linewidth(3);
  c3f(G_black);
  bgnline();
  v3f(s->center[0]);
  v3f(s->center[1]);
  v3f(s->center[2]);
  v3f(s->center[3]);
  endline();
  linewidth(1);

 c3f(G_blue);
 for (i=0;i<4;i++)
   {
     bgnclosedline();
     v3f(s->pt[i][0]);
```

```
      v3f(s->pt[i][1]);
      v3f(s->pt[i][2]);
      v3f(s->pt[i][3]);
      endclosedline();
      bgnline();
      v3f(s->pt[0][i]);
      v3f(s->pt[1][i]);
      v3f(s->pt[2][i]);
      v3f(s->pt[3][i]);
      endline();
    }
}


/* rotate space */

rotateSpace(JINBUTU *hm, Space *s, float *ang, int onn)
{
  int     i, j, k, l;
  Matrix  r_m, r_cm;
  VERTEX  tmp;

  s->xang = *ang++/8.0;
  s->yang = *ang++/8.0;
  s->zang = *ang/8.0;

printf("\n");
printf("s->ang : %f, %f, %f \n", s->xang, s->yang, s->zang);

  c_loadmatrix(G_idmat);
  c_rot(s->zang, 'z');
  c_rot(s->yang, 'y');
  c_rot(s->xang, 'x');
  c_getmatrix(r_cm);

  c_loadmatrix(G_idmat);
  c_translate(s->center[0][0], s->center[0][1], s->center[0][2]);
  c_multmatrix(r_cm);
  c_translate(-s->center[0][0], -s->center[0][1], -s->center[0][2]);
  c_getmatrix(r_m);

  for(k=0; k<3; k++)
    s->center[1][k] = s->corig[1][0] * r_m[0][k] +
                      s->corig[1][1] * r_m[1][k] +
                      s->corig[1][2] * r_m[2][k] +
                      r_m[3][k];

  for (i=1; i<4; i++)
    {
      c_loadmatrix(G_idmat);
      c_translate(s->center[i][0], s->center[i][1], s->center[i][2]);
      c_multmatrix(r_cm);
      c_translate(-s->center[i][0], -s->center[i][1], -s->center[i][2]);
      c_multmatrix(r_m);
      c_getmatrix(r_m);

      if (i<3)
        for(k=0; k<3; k++)
          s->center[i+1][k] = s->corig[i+1][0] * r_m[0][k] +
                              s->corig[i+1][1] * r_m[1][k] +
                              s->corig[i+1][2] * r_m[2][k] +
                              r_m[3][k];

      for(j=0; j<4; j++)
        for(k=0; k<3; k++)
          s->pt[i][j][k] = s->orig[i][j][0] * r_m[0][k] +
```



```
                           s->orig[i][j][1] * r_m[1][k] +
                           s->orig[i][j][2] * r_m[2][k] +
                           r_m[3][k];
    }
  if (onn == 1) {
    c_getmatrix(hm->mune_mat);
  } else if (onn == 2) [
    c_getmatrix(r_m);
    rev_mat(r_m, hm->kosi_mat);
  ]

}


rotateSpace2(JINBUTU *hm, Space *s, float *ang1, float *ang2, float *ang3,
             int onn)
{
  int     i, j, k, l;
  Matrix  r_cm;
  VERTEX  tmp;
  Matrix  r_m = {1.0, 0.0, 0.0, 0.0,
                 0.0, 1.0, 0.0, 0.0,
                 0.0, 0.0, 1.0, 0.0,
                 0.0, 0.0, 0.0, 1.0};

  for (i=0; i<3; i++)
    {
      s->xang = *ang1;
      s->yang = *ang2;
      s->zang = *ang3;

      c_loadmatrix(G_idmat);
      c_rot(s->zang,'z');
      c_rot(s->yang,'y');
      c_rot(s->xang,'x');
      c_getmatrix(r_cm);

      c_loadmatrix(G_idmat);
      c_translate(s->center[i][0], s->center[i][1], s->center[i][2]);
      c_multmatrix(r_cm);
      c_translate(-s->center[i][0], -s->center[i][1], -s->center[i][2]);
      c_multmatrix(r_m);
      c_getmatrix(r_m);

prt_mat(r_m, "r_m");

      for(k=0; k<3; k++)
        s->center[i+1][k] = s->corig[i+1][0] * r_m[0][k] +
                            s->corig[i+1][1] * r_m[1][k] +
                            s->corig[i+1][2] * r_m[2][k] +
                            r_m[3][k];

      for(j=0; j<4; j++)
        for(k=0; k<3; k++)
          s->pt[i+1][j][k] = s->orig[i+1][j][0] * r_m[0][k] +
                             s->orig[i+1][j][1] * r_m[1][k] +
                             s->orig[i+1][j][2] * r_m[2][k] +
                             r_m[3][k];

      if (i<2) {
          s->xang = *ang1++;
          s->yang = *ang2++;
          s->zang = *ang3++;
        }
    }
  if (onn == 1) {
```

```
    c_getmatrix(hm->mune_mat);
  } else if (onn == 2) {
    c_getmatrix(hm->kosi_mat);
  }
}


rotateSpace3(hm, s, ang1, ang2, ang3, onn)
JINBUTU *hm;
Space  *s;
float  *ang1, *ang2, *ang3;
int    onn;
{
  int     i, j, k, l;
  VERTEX  tmp;
  Matrix  r_cm, r_m;

  s->xang = ang1[0] / 2;
  s->yang = ang2[0] / 2;
  s->zang = ang3[0] / 2;

  c_loadmatrix(G_idmat);
  c_rot(s->zang,'z');
  c_rot(s->yang,'y');
  c_rot(s->xang,'x');
  c_getmatrix(r_cm);

  c_loadmatrix(G_idmat);
  c_translate(s->center[0][0], s->center[0][1], s->center[0][2]);
  c_multmatrix(r_cm);
  c_translate(-s->center[0][0], -s->center[0][1], -s->center[0][2]);
  c_getmatrix(r_m);

  for(k=0; k<3; k++)
    s->center[1][k] = s->corig[1][0] * r_m[0][k] +
                      s->corig[1][1] * r_m[1][k] +
                      s->corig[1][2] * r_m[2][k] +
                      r_m[3][k];

  c_loadmatrix(G_idmat);
  c_translate(s->center[1][0], s->center[1][1], s->center[1][2]);
  c_multmatrix(r_cm);
  c_translate(-s->center[1][0], -s->center[1][1], -s->center[1][2]);
  c_multmatrix(r_m);
  c_getmatrix(r_m);

  for(k=0; k<3; k++)
    s->pt[1][j][k] = s->orig[1][j][0] * r_m[0][k] +
                     s->orig[1][j][1] * r_m[1][k] +
                     s->orig[1][j][2] * r_m[2][k] +
                     r_m[3][k];

  for (i=1; i<3; i++)
    {
      s->xang = ang1[i] / 2;
      s->yang = ang2[i] / 2;
      s->zang = ang3[i] / 2;

      c_loadmatrix(G_idmat);
      c_translate(s->center[i][0], s->center[i][1], s->center[i][2]);
      c_rot(s->zang,'z');
      c_rot(s->yang,'y');
      c_rot(s->xang,'x');
      c_translate(-s->center[i][0], -s->center[i][1], -s->center[i][2]);
      c_multmatrix(r_m);
      c_getmatrix(r_m);
```



```
      for(k=0; k<3; k++)
        s->center[i+1][k] = s->corig[i+1][0] * r_m[0][k] +
                            s->corig[i+1][1] * r_m[1][k] +
                            s->corig[i+1][2] * r_m[2][k] +
                            r_m[3][k];

      c_loadmatrix(G_idmat);
      c_translate(s->center[i+1][0], s->center[i+1][1], s->center[i+1][2]);
      c_rot(s->zang,'z');
      c_rot(s->yang,'y');
      c_rot(s->xang,'x');
      c_translate(-s->center[i+1][0], -s->center[i+1][1], -s->center[i+1][2]);
      c_multmatrix(r_m);
      c_getmatrix(r_m);

      for(j=0; j<4; j++)
        for(k=0; k<3; k++)
          s->pt[i+1][j][k] = s->orig[i+1][j][0] * r_m[0][k] +
                             s->orig[i+1][j][1] * r_m[1][k] +
                             s->orig[i+1][j][2] * r_m[2][k] +
                             r_m[3][k];
    }

  if (onn == 1) {
    c_getmatrix(hm->mune_mat);
  } else if (onn == 2) {
    c_getmatrix(r_m);
    rev_mat(r_m, hm->kosi_mat);
  }
}


/********************** end of space functions ********************/


ffdhenkei(s,hm)
     Space     *s;
     JINBUTU *hm;
{
  int k,l;
  register float t1,t2,t3,t4,bl;
  float       *a,*vs, *va;
  Matrix r_m, mat;
  VERTEX tmp,tmp2;

 /* if (breatheswitch)
    breathehenkei(s,hm);*/

  if (breatheswitch)
    /* dou henkei */
    for (k = 0, vs = (float *)hm->vs[DOU], va = (float *)hm->va[DOU];
         k < hm->vmax[DOU]; k++, vs += 3, va += 3)
      {
        /*t=hm->tval[DOU][k];
          it=1-t;
          tsq=t*t;
          tc=tsq*t;
          tsq=3*tsq*it;
          itsq=it*it;
          itc=itsq*it;
          itsq=3*itsq*t;
          */

        t1=hm->tt1[DOU][k];
        t2=hm->tt2[DOU][k];
```

```
        t3=hm->tt3[DOU][k];
        t4=hm->tt4[DOU][k];

        /* speed up for only x rotation motion */

      va[0]=va[0];
        /*va[0] = (vs[0]-s->center[0][0])+ itc*s->center[0][0] +
          itsq*s->center[1][0] + tsq*s->center[2][0] +tc*s->center[3][0]; */

        va[0] = (vs[0]-s->center[0][0]) + t1*s->center[0][0] +
          t2*s->center[1][0] + t3*s->center[2][0] + t4*s->center[3][0];

        va[1] = t1*s->center[0][1] + t2*s->center[1][1] +
          t3*s->center[2][1] + t4*s->center[3][1];

        va[2] = (va[2]-s->center[0][2]) + t1*s->center[0][2] +
          t2*s->center[1][2] + t3*s->center[2][2] + t4*s->center[3][2];
      }
    else
       for (k = 0, vs = (float *)hm->vs[DOU], va = (float *)hm->va[DOU];
            k < hm->vmax[DOU]; k++, vs += 3, va += 3)
       {
        /*t=hm->tval[DOU][k];
          it=1-t;
          tsq=t*t;
          tc=tsq*t;
          tsq=3*tsq*it;
          itsq=it*it;
          itc=itsq*it;
          itsq=3*itsq*t;
          */

        t1=hm->tt1[DOU][k];
        t2=hm->tt2[DOU][k];
        t3=hm->tt3[DOU][k];
        t4=hm->tt4[DOU][k];

        /* speed up for only x rotation motion */

        va[0]=vs[0];
        /*va[0] = (vs[0]-s->center[0][0])+ itc*s->center[0][0] +
          itsq*s->center[1][0] + tsq*s->center[2][0] +tc*s->center[3][0]; */

        va[0] = (vs[0]-s->center[0][0]) + t1*s->center[0][0] +
          t2*s->center[1][0] + t3*s->center[2][0] + t4*s->center[3][0];

        va[1] = t1*s->center[0][1] + t2*s->center[1][1] +
          t3*s->center[2][1] + t4*s->center[3][1];

        va[2] = (vs[2]-s->center[0][2]) + t1*s->center[0][2] +
          t2*s->center[1][2] + t3*s->center[2][2] + t4*s->center[3][2];
       }

  /* other appendages on dou */

    ffdHenkeipt(hm,hm->center[KATA_D_R],hm->shcenter[1],s,1);
    ffdHenkeipt(hm,hm->center[KATA_D_L],hm->shcenter[0],s,0);

  /* for kubi we blend the rotation of 3D mat and the ffd xform */

  c_loadmatrix(G_idmat);
  c_translate(s->center[3][0],s->center[3][1],
              s->center[3][2]);
  c_rot(3*s->xang, 'x');
  c_translate(-s->corig[3][0],-s->corig[3][1],
              -s->corig[3][2]);
```



```
  c_getmatrix(r_m);

  for (k = 0, vs = (float *)hm->vs[KUBI], va = (float *)hm->va[KUBI],a=(float *)hm->a
[KUBI];
       k < hm->vmax[KUBI]; k++, vs += 3, a++,va += 3)
    {

      for(l=0;l<3;l++)
        tmp[l]=va[0]*r_m[0][l]+va[1]*r_m[1][l]+va[2]*r_m[2][l]+r_m[3][l];

      if ((/*t=*/hm->tval[KUBI][k])<0)
        {
          va[0]=tmp[0];va[1]=tmp[1];va[2]=tmp[2];
        ]
      else
      {

        t1=hm->tt1[KUBI][k];
        t2=hm->tt2[KUBI][k];
        t3=hm->tt3[KUBI][k];
        t4=hm->tt4[KUBI][k];


        tmp2[0]=va[0];

        /*tmp2[0]= (va[0]-s->center[0][0])+ itc*s->center[0][0] +
          itsq*s->center[1][0] + tsq*s->center[2][0] +tc*s->center[3][0]; */

        tmp2[0] = (va[0]-s->center[0][0]) + t1*s->center[0][0] +
          t2*s->center[1][0] + t3*s->center[2][0] + t4*s->center[3][0];

        tmp2[1] = t1*s->center[0][1] + t2*s->center[1][1] +
          t3*s->center[2][1] + t4*s->center[3][1];
        tmp2[2] = (va[2]-s->center[0][2]) + t1*s->center[0][2] +
          t2*s->center[1][2] + t3*s->center[2][2] + t4*s->center[3][2];

        bl=blendF(*a);
        va[0] = bl*tmp[0] + (1.0-bl)*tmp2[0];
        va[1] = bl*tmp[1] + (1.0-bl)*tmp2[1];
        va[2] = bl*tmp[2] + (1.0-bl)*tmp2[2];
      }
    }
  /* for kata on either side */

  ffdhenkeiKata(KATA_D_R,KATA_D_R,s,1,hm);
  ffdhenkeiKata(KATA_N_R,KATA_D_R,s,1,hm);
  ffdhenkeiKata(KATA_D_L,KATA_D_L,s,0,hm);
  ffdhenkeiKata(KATA_N_L,KATA_D_L,s,0,hm);


#if 1
  /* kosi henkei */
  for (k = 0, vs = (float *)hm->vs[KOSI], va = (float *)hm->va[KOSI];
       k < hm->vmax[KOSI]; k++, vs += 3, va += 3)
    {
      t1=hm->tt1[KOSI][k];
      t2=hm->tt2[KOSI][k];
      t3=hm->tt3[KOSI][k];
      t4=hm->tt4[KOSI][k];

      if (hm->tval[KOSI][k]<0)
        {
          /* tmp[0]=va[0]; */
          tmp[0] = (va[0]-s->center[0][0]) + t1*s->center[0][0] +
            t2*s->center[1][0] + t3*s->center[2][0] + t4*s->center[3][0];
```

```
        tmp[1] = t1*s->center[0][1] + t2*s->center[1][1] +
          t3*s->center[2][1] + t4*s->center[3][1];

        tmp[2] = (va[2]-s->center[0][2]) + t1*s->center[0][2] +
          t2*s->center[1][2] + t3*s->center[2][2] + t4*s->center[3][2];

        for(1=0;1<3;1++) {
#ifndef HIROSE
          if (trkswitch) [
            va[1]=tmp[0]*hm->mune_mat[0][1]+tmp[1]*hm->mune_mat[1][1]+
               tmp[2]*hm->mune_mat[2][1]+hm->mune_mat[3][1];
          } else {
            rev_mat(hm->kosi_mat, mat);
            va[1]=tmp[0]*mat[0][1] + tmp[1]*mat[1][1]+
               tmp[2]*mat[2][1] + mat[3][1];
          ]
#endif
        }
      }
      else
        [
        va[0]=vs[0];

        /*va[0] = (vs[0]-hm->kosispace.center[0][0])+
          itc*hm->kosispace.center[0][0] + itsq*hm->kosispace.center[1][0] +
          tsq*hm->kosispace.center[2][0] +tc*hm->kosispace.center[3][0]; */

        va[0] = (vs[0]-hm->kosispace.center[0][0]) +
          t1*hm->kosispace.center[0][0] +
          t2*hm->kosispace.center[1][0] + t3*hm->kosispace.center[2][0] +
          t4*hm->kosispace.center[3][0];

        va[1] = t1*hm->kosispace.center[0][1] +
          t2*hm->kosispace.center[1][1] +
          t3*hm->kosispace.center[2][1] + t4*hm->kosispace.center[3][1];

        va[2] = (vs[2]-hm->kosispace.center[0][2])+
          t1*hm->kosispace.center[0][2] +
          t2*hm->kosispace.center[1][2] + t3*hm->kosispace.center[2][2] +
          t4*hm->kosispace.center[3][2];
      }
    }
#endif

  ffdhenkeiSiri(&hm->kosispace,SIRI_L,hm);
  ffdhenkeiSiri(&hm->kosispace,SIRI_R,hm);

}


ffdhenkeiSiri(s,wk,hm)
     Space *s;
int wk;
JINBUTU *hm;
{
  int k,l;
  register float t1,t2,t3,t4,bl;
  float       *a,*vs, *va;
  Matrix r_m;
  VERTEX tmp,tmp2;

  for (k = 0, vs = (float *)hm->vs[wk], va = (float *)hm->va[wk],
       a=(float *)hm->a[wk];
       k < hm->vmax[wk]; k++, vs += 3, a++,va += 3)
    [
      if ((hm->tval[wk][k])>=0)
```



```
      [

        t1=hm->tt1[wk][k];
        t2=hm->tt2[wk][k];
        t3=hm->tt3[wk][k];
        t4=hm->tt4[wk][k];

        tmp2[0]=va[0];

        /*tmp2[0]= (va[0]-s->center[0][0])+ itc*s->center[0][0] +
          itsq*s->center[1][0] + tsq*s->center[2][0] +tc*s->center[3][0]; */

        tmp2[0] = (va[0]-s->center[0][0]) + t1*s->center[0][0] +
          t2*s->center[1][0] + t3*s->center[2][0] + t4*s->center[3][0];

        tmp2[1] = t1*s->center[0][1] + t2*s->center[1][1] +
          t3*s->center[2][1] + t4*s->center[3][1];
        tmp2[2] = (va[2]-s->center[0][2]) + t1*s->center[0][2] +
          t2*s->center[1][2] + t3*s->center[2][2] + t4*s->center[3][2];

        bl=blendF(*a);
        va[0] = bl*va[0] + (1.0-bl)*tmp2[0];
        va[1] = bl*va[1] + (1.0-bl)*tmp2[1];
        va[2] = bl*va[2] + (1.0-bl)*tmp2[2];
      ]
    ]
]


ffdhenkeiKata(wk,wk3,s,ind,hm)
int wk,wk3;
Space *s;
int ind;
JINBUTU *hm;
[
  int k,l;
  register float t1,t2,t3,t4,bl;
  float       *a,*vs, *va;
  Matrix r_m;
  VERTEX tmp,tmp2;

  c_loadmatrix(G_idmat);
  c_translate(hm->shcenter[ind][0], hm->shcenter[ind][1],
           hm->shcenter[ind][2]);
  c_rot(s->na[ind], 'x');
  c_translate(-hm->center[wk3][0], -hm->center[wk3][1],
           -hm->center[wk3][2]);
  c_getmatrix(r_m);

  for(k = 0, vs = (float *)hm->vs[wk], a = (float *)hm->a[wk],
      va = (float *)hm->va[wk];
     k < hm->vmax[wk]; k++, vs += 3, a++, va += 3)
    [

      for(1=0;1<3;1++)
        tmp[1]=va[0]*r_m[0][1]+va[1]*r_m[1][1]+va[2]*r_m[2][1]+r_m[3][1];

      if ((/*t=*/hm->tval[wk][k])<0)
        [
          va[0]=tmp[0];va[1]=tmp[1];va[2]=tmp[2];
        ]
      else
        [
          /*it=1-t;
          tsq=t*t;
          tc=tsq*t;
```

```
          tsq=3*tsq*it;
          itsq=it*it;
          itc=itsq*it;
          itsq=3*itsq*t;*/

          t1=hm->tt1[wk][k];
          t2=hm->tt2[wk][k];
          t3=hm->tt3[wk][k];
          t4=hm->tt4[wk][k];

          tmp2[0]=va[0];

          /*tmp2[0]= (va[0]-s->center[0][0])+ itc*s->center[0][0] +
            itsq*s->center[1][0] + tsq*s->center[2][0] +tc*s->center[3][0]; */

          tmp2[0] = (va[0]-s->center[0][0]) + t1*s->center[0][0] +
            t2*s->center[1][0] + t3*s->center[2][0] + t4*s->center[3][0];

          tmp2[1] = t1*s->center[0][1] + t2*s->center[1][1] +
            t3*s->center[2][1] + t4*s->center[3][1];
          tmp2[2] = (va[2]-s->center[0][2]) + t1*s->center[0][2] +
            t2*s->center[1][2] + t3*s->center[2][2] + t4*s->center[3][2];

          bl=blendF(*a);
          va[0] = bl*tmp[0] + (1.0-bl)*tmp2[0];
          va[1] = bl*tmp[1] + (1.0-bl)*tmp2[1];
          va[2] = bl*tmp[2] + (1.0-bl)*tmp2[2];
        ]
    ]
}


ffdHenkeipt(hm,v,xyz,s,ind)
     JINBUTU *hm;
     float *v,*xyz;
     Space *s;
     int ind;
{
  register float t1,t2,t3,t4;

  /*t=(v[1]-s->corig[0][1])/(s->corig[3][1]-s->corig[0][1]);*/

  s->na[ind]=3.0*hm->tt[ind][0]*s->xang;

  /*it=1-t;
  tsq=t*t;
  tc=tsq*t;
  tsq=3*tsq*it;
  itsq=it*it;
  itc=itsq*it;
  itsq=3*itsq*t;*/

  t1=hm->tt[ind][1];
  t2=hm->tt[ind][2];
  t3=hm->tt[ind][3];
  t4=hm->tt[ind][4];

  xyz[0]=v[0];

  /*xyz[0] = (v[0]-s->center[0][0])+ t1*s->center[0][0] +
    t2*s->center[1][0] + t3*s->center[2][0] +t4*s->center[3][0]; */

  xyz[0] = (v[0]-s->center[0][0]) + t1*s->center[0][0] +
    t2*s->center[1][0] + t3*s->center[2][0] + t4*s->center[3][0];

  xyz[1] = t1*s->center[0][1] + t2*s->center[1][1] +
```



```
    t3*s->center[2][1] + t4*s->center[3][1];
  xyz[2] = (v[2]-s->center[0][2]) + t1*s->center[0][2] +
    t2*s->center[1][2] + t3*s->center[2][2] + t4*s->center[3][2];

}


breathehenkei(s,hm)
     Space *s;
     JINBUTU *hm;
{
  int k,l;
  register float t,bl;
  float       *a,*vs, *va;

  for (k = 0, vs = (float *)hm->vs[DOU], va = (float *)hm->va[DOU];
       k < hm->vmax[DOU]; k++, vs += 3, va += 3)
    {
      va[0]=vs[0];va[1]=vs[1];va[2]=vs[2];
      if (vs[2]>s->corig[0][2])
        [
          bl=bellF(hm->tval[DOU][k]);
          va[2] = vs[2]+ (vs[2]-s->corig[0][2])*bl*BRFAC;
        ]
    ]

  for (k = 0, vs = (float *)hm->vs[KUBI], va = (float *)hm->va[KUBI];
       k < hm->vmax[KUBI]; k++, vs += 3, va += 3)
    if (((t=hm->tval[KUBI][k])>0) && (va[2]>s->corig[0][2]))
      {
        bl=bellF(t);
        va[2]= va[2] + (va[2]-s->corig[0][2])*bl*BRFAC;
      }

  for (k = 0, vs = (float *)hm->vs[KATA_D_R], va = (float *)hm->va[KATA_D_R];
       k < hm->vmax[KATA_D_R]; k++, vs += 3, va += 3)
    if (((t=hm->tval[KATA_D_R][k])>0) && (va[2]>s->corig[0][2]))
      {
        bl=bellF(t);
        va[2]= va[2] + (va[2]-s->corig[0][2])*bl*BRFAC;
      }

  for (k = 0, vs = (float *)hm->vs[KATA_N_L], va = (float *)hm->va[KATA_N_L];
       k < hm->vmax[KATA_N_L]; k++, vs += 3, va += 3)
    if (((t=hm->tval[KATA_N_L][k])>0) && (va[2]>s->corig[0][2]))
      {
        bl=bellF(t);
        va[2]= va[2] + (va[2]-s->corig[0][2])*bl*BRFAC;
      }

  for (k = 0, vs = (float *)hm->vs[KATA_D_L], va = (float *)hm->va[KATA_D_L];
     k < hm->vmax[KATA_D_L]; k++, vs += 3, va += 3)
    if (((t=hm->tval[KATA_D_L][k])>0) && (va[2]>s->corig[0][2]))
      [
        bl=bellF(t);
        va[2]= va[2] + (va[2]-s->corig[0][2])*bl*BRFAC;
      ]

  for (k = 0, vs = (float *)hm->vs[KATA_N_R], va = (float *)hm->va[KATA_N_R];
       k < hm->vmax[KATA_N_R]; k++, vs += 3, va += 3)
    if (((t=hm->tval[KATA_N_R][k])>0) && (va[2]>s->corig[0][2]))
      [
        bl=bellF(t);
        va[2]= va[2] + (va[2]-s->corig[0][2])*bl*BRFAC;
      ]
```

```
  for (k = 0, vs = (float *)hm->vs[KOSI], va = (float *)hm->va[KOSI];
       k < hm->vmax[KOSI]; k++, vs += 3, va += 3)
    {
      va[0]=vs[0];va[1]=vs[1];va[2]=vs[2];
      if (((t=hm->tval[KOSI][k])<0) && (va[2]>s->corig[0][2]))
        {
          bl=bellF(-1-t);
          va[2]= va[2] + (va[2]-s->corig[0][2])*bl*BRFAC;
        }
    }

  BRFAC+= BRINC;
  if (BRFAC<0)
    {
      BRFAC=0;
      BRINC= -BRINC;
    }
  else if
    (BRFAC>BRMAX)
      BRINC= -BRINC;
}
```

```
/*
 * ファイル名 : body_daw.c
 *
 * 機能       : 臨場感通信デモシステムに組み込む関数
 *
 * 履歴       : 1992年10月28日 ; 作成 ; 広瀬  正俊
 */
#include        "tool.h"

#define         ICHIMAI

#ifndef TANTAI
#define         read_map_tex    c_read_map_tex
#endif

static  Matrix  G_idmat = {1.0, 0.0, 0.0, 0.0,
                           0.0, 1.0, 0.0, 0.0,
                           0.0, 0.0, 1.0, 0.0,
                           0.0, 0.0, 0.0, 1.0};

/*++++++++++++++++++++++++++++++++++++++++*/

float  S1= 2.3,S2=2.3,PF1=1.5,PF2=1.5;
/*
float SCFAC= 2.3,POWF=1.5;
*/
#define blendF(x)  (x)*(x)*(3.0-2.0*(x))


/*++++++++++++++++++++++++++++++++++++++++*/



/*
 * 関数名 : InitializeGlobal()
 *
 * 引数   : なし
 *
 * 戻り値 : なし
 *
 * 機能   : グローバル変数の初期化
 *
 * 履歴   : 1992年10月28日 ; 作成 ; 広瀬  正俊
 */
void
InitializeJinbutu(JINBUTU *hm)
{
    int         i;

    for(i = 0; i < MAX_BUHIN; i++) {
        hm->vs[i] = NULL;
        hm->va[i] = NULL;
        hm->center[i][0] = 0.0;
        hm->center[i][1] = 0.0;
        hm->center[i][2] = 0.0;
        hm->vt[i] = NULL;
        hm->poly[i] = NULL;
        hm->a[i] = NULL;
        hm->vmax[i] = 0;
        hm->vtmax[i] = 0;
        hm->polmax[i] = 0;
        hm->tindex[i] = 0;
        cp_mat(hm->mat[i], G_idmat);
        hm->ipwt[i] = NULL;
    }
```



```
    hm->ske_real.atama_len = 15.0;
    hm->ske_real.kata_len = 18.0;
    hm->ske_real.ninoude_len = 28.0;
    hm->ske_real.ude_len = 28.0;
    hm->ske_real.yubi_len = 3.0;
    hm->ske_real.kubi_len = 5.0;
    hm->ske_cg.atama_len = 15.0;
    hm->ske_cg.kata_len = 18.0;
    hm->ske_cg.ninoude_len = 28.0;
    hm->ske_cg.ude_len = 28.0;
    hm->ske_cg.yubi_len = 3.0;
    hm->ske_cg.kubi_len = 5.0;

    c_loadmatrix(G_idmat);
    c_rot(90.0, 'z');
    c_getmatrix(hm->mune_calib);
    c_loadmatrix(G_idmat);
    c_rot(-90.0, 'z');
    c_rot(90.0, 'y');
    c_getmatrix(hm->atama_calib);
    cp_mat(hm->r_te_calib, G_idmat);
    cp_mat(hm->l_te_calib, G_idmat);

    hm->kosix = 0.0; hm->kosiy = -50.0; hm->kosiz = 0.0;
    hm->rhizax = 0.0; hm->rhizay = -50.0; hm->rhizaz = 50.0;
    hm->lhizax = 0.0; hm->lhizay = -50.0; hm->lhizaz = 50.0;

    strcpy(hm->hname, "ciris23");

    hm->stand = 0;
}

void
GetFileName(char *hname, char name[][MAX_NAME], float vtc[][8])
{
    FILE        *fp;
    char        buff[LINE_BUF], str1[MAX_NAME], str2[MAX_NAME];
    char        fname[MAX_NAME], dir[MAX_NAME];
    int         i;

    sprintf(fname, "%s%s", DATA_DIR, "filename.dat");
    fp = fopen(fname, "r");

    while (fgets(buff, LINE_BUF, fp) != NULL) [
        if (!strncmp(hname, buff, strlen(hname))) [
            fgets(buff, LINE_BUF, fp);
            /* read calib data */
            sscanf(buff, "%s %s %f %f %f %f %f %f %f %f",
                   str1, dir, &vtc[0][0], &vtc[0][1], &vtc[0][2], &vtc[0][3],
                   &vtc[0][4], &vtc[0][5], &vtc[0][6], &vtc[0][7]);
            for (i = BODY_COLOR; i < MAX_FILE; i++) [
                fgets(buff, LINE_BUF, fp);
                sscanf(buff, "%s %s %f %f %f", str1, str2,
                                        &vtc[i][0], &vtc[i][1], &vtc[i][2]);
                sprintf(name[i], "%s%s", dir, str2);
            ]
            break;
        ]
    ]

    fclose(fp);
}


/*
```

```
 * 関数名 : ReadObjTexture()
 *
 * 引数   : なし
 *
 * 戻り値 : なし
 *
 * 機能   : ・形状データ及びテクスチャデータの読み込み
 *          ・変形後の頂点データ格納領域の確保
 *          ・回転の中心座標を求める
 *          ・左手側データは右手側データをコピーして作る
 *
 * 履歴   : 1992年10月28日 ; 作成 ; 広瀬　正後
 */
void *ReadObjTexture(char *hname)
{
    char        name[MAX_FILE][MAX_NAME];
    float       vtc[MAX_FILE][8];
    JINBUTU     *hm;
    FILE        *fp;
    float       trd[24];
    int i, ii, n;
#ifdef ICHIMAI
    static int  mapindex = -1;
#endif

    GetFileName(hname, name, vtc);

    hm = (JINBUTU *)malloc(sizeof(JINBUTU));
    InitializeJinbutu(hm);

    hm->mata = vtc[BODY_COLOR][0];
    hm->simple = vtc[BODY_COLOR][1];
    hm->ske_real.kata_len = vtc[0][0];
    hm->ske_real.ninoude_len = vtc[0][1];
    hm->ske_real.ude_len = vtc[0][2];
    hm->ske_real.atama_len = vtc[0][3];
    hm->ske_real.kubi_len = vtc[0][4];
    hm->ske_real.dou_len = vtc[0][5];
    hm->ske_real.momo_len = vtc[0][6];
    hm->ske_real.sune_len = vtc[0][7];

#ifdef TANTAI
    /* キャリブレーションファイルの読み込み */
    if (fp = fopen(CALIB_BODY_FILE, "r")) {
        for (i = 0; i < 4; i++) {
            fscanf(fp, "%d %f %f %f %f %f %f", &n,
                   &trd[i*6], &trd[i*6+1], &trd[i*6+2], &trd[i*6+3],
                   &trd[i*6+4], &trd[i*6+5]);
        }
        fclose(fp);

        c_calib_model((void *)hm, trd);
    }
#endif

#ifdef ICHIMAI
    if (mapindex < 0) {
        hm->tindex[DOU] = read_map_tex(name[BODY_COLOR]);
    } else {
        hm->tindex[DOU] = mapindex;
    }
    mapindex = hm->tindex[DOU];
#else
    hm->tindex[DOU] = read_map_tex(name[BODY_COLOR]);
#endif
```



```
    /* 頭部 */
    hm->tindex[ATAMA] = hm->tindex[DOU];
    ReadWave(hm, name[FACE_OBJ], ATAMA, vtc[FACE_OBJ]);
[
hm->ske.atama_len = 20.0;
]
/*++++++++++++++++++++++++++++++++++++*/

 hm->va[ATAMA] = (VERTEX  *)malloc(sizeof(VERTEX)*hm->vmax[ATAMA]);

for(ii = 0; ii < 3*hm->vmax[ATAMA]; ii++)
  ((float *)hm->va[ATAMA])[ii] =  ((float *)hm->vs[ATAMA])[ii];


/*++++++++++++++++++++++++++++++++++++*/




    /* 胸部 */
    ReadWave(hm, name[DOU_OBJ], DOU, vtc[DOU_OBJ]);

/*++++++++++++++++++++++++++++++++++++*/

 hm->va[DOU] = (VERTEX  *)malloc(sizeof(VERTEX)*hm->vmax[DOU]);

/*for(ii = 0; ii < 3*hm->vmax[DOU]; ii++)
  ((float *)hm->va[DOU])[ii] =  ((float *)hm->vs[DOU])[ii];
 */

/*++++++++++++++++++++++++++++++++++++*/



    /* 右手側データ */
    /* 右上腕 */
    hm->tindex[NINOUDE_R] = hm->tindex[DOU];
    ReadWave(hm, name[NINOUDE_R_OBJ], NINOUDE_R, vtc[NINOUDE_R_OBJ]);
    hm->va[NINOUDE_R] = (VERTEX  *)malloc(sizeof(VERTEX)*hm->vmax[NINOUDE_R]);
[
hm->ske.ninoude_len = 28.0;
]

    /* 右下腕 */
    hm->tindex[UDE_R] = hm->tindex[DOU];
    ReadWave(hm, name[UDE_R_OBJ], UDE_R, vtc[UDE_R_OBJ]);
    hm->va[UDE_R] = (VERTEX  *)malloc(sizeof(VERTEX)*hm->vmax[UDE_R]);
[
hm->ske.ude_len = 28.0;
]

    /* 右手 */ /* 右手首 */
    hm->tindex[TE_R] = hm->tindex[DOU];
    ReadWave(hm, name[TE_R_OBJ], TE_R, vtc[TE_R_OBJ]);
    hm->va[TE_R] = (VERTEX *)malloc(sizeof(VERTEX)*hm->vmax[TE_R]);
hm->center[UDE_R][0] = hm->center[TE_R][0];
hm->center[UDE_R][1] = hm->center[TE_R][1];
hm->center[UDE_R][2] = hm->center[TE_R][2];

    /* 右親指 */
    ReadYubiKansetuData(hm, OYAYUBI_R_1, name[OYAYUBI_R_1_OBJ],
                        vtc[OYAYUBI_R_1_OBJ],  8);
    ReadYubiKansetuData(hm, OYAYUBI_R_2, name[OYAYUBI_R_2_OBJ],
                        vtc[OYAYUBI_R_2_OBJ],  0);
    ReadYubiKansetuData(hm, OYAYUBI_R_3, name[OYAYUBI_R_3_OBJ],
```

```
                        vtc[OYAYUBI_R_3_OBJ],  0);

    /* 右人差し指 */
    ReadYubiKansetuData(hm, HITOSASIYUBI_R_1, name[HITOSASIYUBI_R_1_OBJ],
                        vtc[HITOSASIYUBI_R_1_OBJ],  0);
    ReadYubiKansetuData(hm, HITOSASIYUBI_R_2, name[HITOSASIYUBI_R_2_OBJ],
                        vtc[HITOSASIYUBI_R_2_OBJ],  0);
    ReadYubiKansetuData(hm, HITOSASIYUBI_R_3, name[HITOSASIYUBI_R_3_OBJ],
                        vtc[HITOSASIYUBI_R_3_OBJ],  0);

    /* 右中指 */
    ReadYubiKansetuData(hm, NAKAYUBI_R_1, name[NAKAYUBI_R_1_OBJ],
                        vtc[NAKAYUBI_R_1_OBJ],  0);
    ReadYubiKansetuData(hm, NAKAYUBI_R_2, name[NAKAYUBI_R_2_OBJ],
                        vtc[NAKAYUBI_R_2_OBJ],  0);
    ReadYubiKansetuData(hm, NAKAYUBI_R_3, name[NAKAYUBI_R_3_OBJ],
                        vtc[NAKAYUBI_R_3_OBJ],  0);

    /* 右薬指 */
    ReadYubiKansetuData(hm, KUSURIYUBI_R_1, name[KUSURIYUBI_R_1_OBJ],
                        vtc[KUSURIYUBI_R_1_OBJ],  0);
    ReadYubiKansetuData(hm, KUSURIYUBI_R_2, name[KUSURIYUBI_R_2_OBJ],
                        vtc[KUSURIYUBI_R_2_OBJ],  0);
    ReadYubiKansetuData(hm, KUSURIYUBI_R_3, name[KUSURIYUBI_R_3_OBJ],
                        vtc[KUSURIYUBI_R_3_OBJ],  0);

    /* 右小指 */
    ReadYubiKansetuData(hm, KOYUBI_R_1, name[KOYUBI_R_1_OBJ],
                        vtc[KOYUBI_R_1_OBJ],  0);
    ReadYubiKansetuData(hm, KOYUBI_R_2, name[KOYUBI_R_2_OBJ],
                        vtc[KOYUBI_R_2_OBJ],  0);
    ReadYubiKansetuData(hm, KOYUBI_R_3, name[KOYUBI_R_3_OBJ],
                        vtc[KOYUBI_R_3_OBJ],  0);
{
hm->ske.yubi_len = 3.0;
}

    /* 首 */
    hm->tindex[KUBI] = hm->tindex[ATAMA];
    ReadWave(hm, name[KUBI_OBJ], KUBI, vtc[KUBI_OBJ]);
    hm->va[KUBI] = (VERTEX  *)malloc(sizeof(VERTEX)*hm->vmax[KUBI]);
    GetCenter(hm, KUBI, hm->center[KUBI], 36, 0);
{
    float       *a, *vt;
    int         i;

    a = (float *)hm->a[KUBI];

    for (i = 0; i < 36; i++) *(a++) = 0.0;
    for (i = 36; i < 72; i++) *(a++) = 0.2;
    for (i = 72; i < 108; i++) *(a++) = 0.4;
    for (i = 108; i < 144; i++) *(a++) = 0.6;
    for (i = 144; i < 180; i++) *(a++) = 0.8;
    for (i = 180; i < 216; i++) *(a++) = 1.0;
}
{
hm->ske.kubi_len = 10.0;
}

    /* 右肩関節の胸部側 */
    hm->tindex[KATA_D_R] = hm->tindex[DOU];
    ReadWave(hm, name[KATA_D_R_OBJ], KATA_D_R, vtc[KATA_D_R_OBJ]);
    hm->va[KATA_D_R] = (VERTEX  *)malloc(sizeof(VERTEX)*hm->vmax[KATA_D_R]);
    hm->tindex[KATA_N_R] = hm->tindex[DOU];
    ReadWave(hm, name[KATA_N_R_OBJ], KATA_N_R, vtc[KATA_N_R_OBJ]);
    hm->va[KATA_N_R] = (VERTEX  *)malloc(sizeof(VERTEX)*hm->vmax[KATA_N_R]);
```



```
{
hm->center[KATA_N_R][0] = hm->center[KATA_D_R][0];
hm->center[KATA_N_R][1] = hm->center[KATA_D_R][1];
hm->center[KATA_N_R][2] = hm->center[KATA_D_R][2];
hm->ske.kata_len = 18.0;
}

    /* 右肘関節の上腕側 */
    hm->tindex[HIJI_R] = hm->tindex[DOU];
    ReadWave(hm, name[HIJI_R_OBJ], HIJI_R, vtc[HIJI_R_OBJ]);
    hm->va[HIJI_R] = (VERTEX  *)malloc(sizeof(VERTEX)*hm->vmax[HIJI_R]);

    /* 左手側データ */
    /* 左上腕部 */
    hm->tindex[NINOUDE_L] = hm->tindex[DOU];
    ReadWave(hm, name[NINOUDE_L_OBJ], NINOUDE_L, vtc[NINOUDE_L_OBJ]);
    hm->va[NINOUDE_L] = (VERTEX  *)malloc(sizeof(VERTEX)*hm->vmax[NINOUDE_L]);

    /* 左下腕部 */
    hm->tindex[UDE_L] = hm->tindex[DOU];
    ReadWave(hm, name[UDE_L_OBJ], UDE_L, vtc[UDE_L_OBJ]);
    hm->va[UDE_L] = (VERTEX  *)malloc(sizeof(VERTEX)*hm->vmax[UDE_L]);

/*++++++++++++++++++++++++++++++++++++*/

for(ii = 0; ii < 3*hm->vmax[UDE_L]; ii++)
     ((float *)hm->va[UDE_L])[ii] =  ((float *)hm->vs[UDE_L])[ii];
for(ii = 0; ii < 3*hm->vmax[UDE_R]; ii++)
     ((float *)hm->va[UDE_R])[ii] =  ((float *)hm->vs[UDE_R])[ii];


/*++++++++++++++++++++++++++++++++++++*/


    /* 左手 */ /* 左手首 */
    hm->tindex[TE_L] = hm->tindex[DOU];
    ReadWave(hm, name[TE_L_OBJ], TE_L, vtc[TE_L_OBJ]);
    hm->va[TE_L] = (VERTEX *)malloc(sizeof(VERTEX)*hm->vmax[TE_L]);
hm->center[UDE_L][0] = hm->center[TE_L][0];
hm->center[UDE_L][1] = hm->center[TE_L][1];
hm->center[UDE_L][2] = hm->center[TE_L][2];

    /* 左親指 */
    ReadYubiKansetuData(hm, OYAYUBI_L_1, name[OYAYUBI_L_1_OBJ],
                        vtc[OYAYUBI_L_1_OBJ],  8);
    ReadYubiKansetuData(hm, OYAYUBI_L_2, name[OYAYUBI_L_2_OBJ],
                        vtc[OYAYUBI_L_2_OBJ],  0);
    ReadYubiKansetuData(hm, OYAYUBI_L_3, name[OYAYUBI_L_3_OBJ],
                        vtc[OYAYUBI_L_3_OBJ],  0);

    /* 左人差し指 */
    ReadYubiKansetuData(hm, HITOSASIYUBI_L_1, name[HITOSASIYUBI_L_1_OBJ],
                        vtc[HITOSASIYUBI_L_1_OBJ],  0);
    ReadYubiKansetuData(hm, HITOSASIYUBI_L_2, name[HITOSASIYUBI_L_2_OBJ],
                        vtc[HITOSASIYUBI_L_2_OBJ],  0);
    ReadYubiKansetuData(hm, HITOSASIYUBI_L_3, name[HITOSASIYUBI_L_3_OBJ],
                        vtc[HITOSASIYUBI_L_3_OBJ],  0);

    /* 左中指 */
    ReadYubiKansetuData(hm, NAKAYUBI_L_1, name[NAKAYUBI_L_1_OBJ],
                        vtc[NAKAYUBI_L_1_OBJ],  0);
    ReadYubiKansetuData(hm, NAKAYUBI_L_2, name[NAKAYUBI_L_2_OBJ],
                        vtc[NAKAYUBI_L_2_OBJ],  0);
    ReadYubiKansetuData(hm, NAKAYUBI_L_3, name[NAKAYUBI_L_3_OBJ],
                        vtc[NAKAYUBI_L_3_OBJ],  0);
```

```
    /* 左薬指 */
    ReadYubiKansetuData(hm, KUSURIYUBI_L_1, name[KUSURIYUBI_L_1_OBJ],
                        vtc[KUSURIYUBI_L_1_OBJ],  0);
    ReadYubiKansetuData(hm, KUSURIYUBI_L_2, name[KUSURIYUBI_L_2_OBJ],
                        vtc[KUSURIYUBI_L_2_OBJ],  0);
    ReadYubiKansetuData(hm, KUSURIYUBI_L_3, name[KUSURIYUBI_L_3_OBJ],
                        vtc[KUSURIYUBI_L_3_OBJ],  0);

    /* 左小指 */
    ReadYubiKansetuData(hm, KOYUBI_L_1, name[KOYUBI_L_1_OBJ],
                        vtc[KOYUBI_L_1_OBJ],  0);
    ReadYubiKansetuData(hm, KOYUBI_L_2, name[KOYUBI_L_2_OBJ],
                        vtc[KOYUBI_L_2_OBJ],  0);
    ReadYubiKansetuData(hm, KOYUBI_L_3, name[KOYUBI_L_3_OBJ],
                        vtc[KOYUBI_L_3_OBJ],  0);

    /* 左肩関節の胸部側 */
    hm->tindex[KATA_D_L] = hm->tindex[DOU];
    ReadWave(hm, name[KATA_D_L_OBJ], KATA_D_L, vtc[KATA_D_L_OBJ]);
    hm->va[KATA_D_L] = (VERTEX  *)malloc(sizeof(VERTEX)*hm->vmax[KATA_D_L]);
    hm->tindex[KATA_N_L] = hm->tindex[DOU];
    ReadWave(hm, name[KATA_N_L_OBJ], KATA_N_L, vtc[KATA_N_L_OBJ]);
    hm->va[KATA_N_L] = (VERTEX  *)malloc(sizeof(VERTEX)*hm->vmax[KATA_N_L]);
{
hm->center[KATA_N_L][0] = hm->center[KATA_D_L][0];
hm->center[KATA_N_L][1] = hm->center[KATA_D_L][1];
hm->center[KATA_N_L][2] = hm->center[KATA_D_L][2];
}
{
float x,y,z;
printf("center(KATA_D_R):%f,%f,%f\n",hm->center[KATA_D_R][0],
        hm->center[KATA_D_R][1],hm->center[KATA_D_R][2]);
printf("center(KATA_D_L):%f,%f,%f\n",hm->center[KATA_D_L][0],
        hm->center[KATA_D_L][1],hm->center[KATA_D_L][2]);
x = hm->center[KATA_D_L][0]-hm->center[KATA_D_R][0];
y = hm->center[KATA_D_L][1]-hm->center[KATA_D_R][1];
z = hm->center[KATA_D_L][2]-hm->center[KATA_D_R][2];
printf("length(KATA):%f\n",fsqrt(x*x+y*y+z*z));
}

    /* 左肘関節の上腕側 */
    hm->tindex[HIJI_L] = hm->tindex[DOU];
    ReadWave(hm, name[HIJI_L_OBJ], HIJI_L, vtc[HIJI_L_OBJ]);
    hm->va[HIJI_L] = (VERTEX  *)malloc(sizeof(VERTEX)*hm->vmax[HIJI_L]);

    hm->tindex[KOSI] = hm->tindex[DOU];
    ReadWave(hm, name[KOSI_OBJ], KOSI, vtc[KOSI_OBJ]);
    hm->va[KOSI] = (VERTEX  *)malloc(sizeof(VERTEX)*hm->vmax[KOSI]);

    hm->tindex[SIRI_R] = hm->tindex[DOU];
    ReadWave(hm, name[SIRI_R_OBJ], SIRI_R, vtc[SIRI_R_OBJ]);
    hm->va[SIRI_R] = (VERTEX  *)malloc(sizeof(VERTEX)*hm->vmax[SIRI_R]);

    hm->tindex[SIRI_L] = hm->tindex[DOU];
    ReadWave(hm, name[SIRI_L_OBJ], SIRI_L, vtc[SIRI_L_OBJ]);
    hm->va[SIRI_L] = (VERTEX  *)malloc(sizeof(VERTEX)*hm->vmax[SIRI_L]);
{
float x1,x2,y1,y2,z1,z2;
x1=(hm->center[KATA_D_R][0]+hm->center[KATA_D_L][0])/2.0;
y1=(hm->center[KATA_D_R][1]+hm->center[KATA_D_L][1])/2.0;
z1=(hm->center[KATA_D_R][2]+hm->center[KATA_D_L][2])/2.0;
x2=(hm->center[SIRI_R][0]+hm->center[SIRI_L][0])/2.0;
y2=(hm->center[SIRI_R][1]+hm->center[SIRI_L][1])/2.0;
z2=(hm->center[SIRI_R][2]+hm->center[SIRI_L][2])/2.0;
hm->ske.dou_len = fsqrt((y1-y2)*(y1-y2)+(z1-z2)*(z1-z2));
printf("length(DOU):%f\n",hm->ske.dou_len);
```



```
hm->center[DOU][0] = x1;
hm->center[DOU][1] = y1;
hm->center[DOU][2] = z1;
}
{
float x,y,z;
printf("center(SIRI_R):%f,%f,%f\n",hm->center[SIRI_R][0],
        hm->center[SIRI_R][1],hm->center[SIRI_R][2]);
printf("center(SIRI_L):%f,%f,%f\n",hm->center[SIRI_L][0],
        hm->center[SIRI_L][1],hm->center[SIRI_L][2]);
x = hm->center[SIRI_L][0]-hm->center[SIRI_R][0];
y = hm->center[SIRI_L][1]-hm->center[SIRI_R][1];
z = hm->center[SIRI_L][2]-hm->center[SIRI_R][2];
printf("length(SIRI):%f\n",fsqrt(x*x+y*y+z*z));
}

    hm->tindex[MOMO_R] = hm->tindex[DOU];
    ReadWave(hm, name[MOMO_R_OBJ], MOMO_R, vtc[MOMO_R_OBJ]);
    hm->va[MOMO_R] = (VERTEX  *)malloc(sizeof(VERTEX)*hm->vmax[MOMO_R]);

    hm->tindex[MOMO_L] = hm->tindex[DOU];
    ReadWave(hm, name[MOMO_L_OBJ], MOMO_L, vtc[MOMO_L_OBJ]);
    hm->va[MOMO_L] = (VERTEX  *)malloc(sizeof(VERTEX)*hm->vmax[MOMO_L]);

    hm->tindex[HIZA_R] = hm->tindex[DOU];
    ReadWave(hm, name[HIZA_R_OBJ], HIZA_R, vtc[HIZA_R_OBJ]);
    hm->va[HIZA_R] = (VERTEX  *)malloc(sizeof(VERTEX)*hm->vmax[HIZA_R]);
{
float x,y,z;
printf("center(HIZA_R):%f,%f,%f\n",hm->center[HIZA_R][0],
        hm->center[HIZA_R][1],hm->center[HIZA_R][2]);
x = hm->center[HIZA_R][0]-hm->center[SIRI_R][0];
y = hm->center[HIZA_R][1]-hm->center[SIRI_R][1];
z = hm->center[HIZA_R][2]-hm->center[SIRI_R][2];
hm->ske.momo_len = fsqrt(y*y+z*z);
printf("length(MOMO):%f\n",hm->ske.momo_len);
}

    hm->tindex[HIZA_L] = hm->tindex[DOU];
    ReadWave(hm, name[HIZA_L_OBJ], HIZA_L, vtc[HIZA_L_OBJ]);
    hm->va[HIZA_L] = (VERTEX  *)malloc(sizeof(VERTEX)*hm->vmax[HIZA_L]);

    hm->tindex[SUNE_R] = hm->tindex[DOU];
    ReadWave(hm, name[SUNE_R_OBJ], SUNE_R, vtc[SUNE_R_OBJ]);
    hm->va[SUNE_R] = (VERTEX  *)malloc(sizeof(VERTEX)*hm->vmax[SUNE_R]);

    hm->tindex[SUNE_L] = hm->tindex[DOU];
    ReadWave(hm, name[SUNE_L_OBJ], SUNE_L, vtc[SUNE_L_OBJ]);
    hm->va[SUNE_L] = (VERTEX  *)malloc(sizeof(VERTEX)*hm->vmax[SUNE_L]);

    hm->tindex[KUTU_R] = hm->tindex[DOU];
    ReadWave(hm, name[KUTU_R_OBJ], KUTU_R, vtc[KUTU_R_OBJ]);
    hm->va[KUTU_R] = (VERTEX  *)malloc(sizeof(VERTEX)*hm->vmax[KUTU_R]);
{
float x,y,z;
printf("center(KUTU_R):%f,%f,%f\n",hm->center[KUTU_R][0],
        hm->center[KUTU_R][1],hm->center[KUTU_R][2]);
x = hm->center[KUTU_R][0]-hm->center[HIZA_R][0];
y = hm->center[KUTU_R][1]-hm->center[HIZA_R][1];
z = hm->center[KUTU_R][2]-hm->center[HIZA_R][2];
hm->ske.sune_len = fsqrt(x*x+y*y+z*z);
printf("length(SUNE):%f\n",hm->ske.sune_len);
}

    hm->tindex[KUTU_L] = hm->tindex[DOU];
    ReadWave(hm, name[KUTU_L_OBJ], KUTU_L, vtc[KUTU_L_OBJ]);
```

```
    hm->va[KUTU_L] = (VERTEX  *)malloc(sizeof(VERTEX)*hm->vmax[KUTU_L]);

    KaradaZurasi(hm);

/*++++++++++++++++++++++++++++++++++*/


   /*** setting ninoude initial center values to kata *****/
/*    hm->center[KATA_N_R][0]=hm->center[NINOUDE_R][0];
    hm->center[KATA_N_R][1]=hm->center[NINOUDE_R][1];
    hm->center[KATA_N_R][2]=hm->center[NINOUDE_R][2];

    hm->center[KATA_N_L][0]=hm->center[NINOUDE_L][0];
    hm->center[KATA_N_L][1]=hm->center[NINOUDE_L][1];
    hm->center[KATA_N_L][2]=hm->center[NINOUDE_L][2];
*/

hm->shcenter[0][0]=hm->center[KATA_D_L][0];
hm->shcenter[0][1]=hm->center[KATA_D_L][1];
hm->shcenter[0][2]=hm->center[KATA_D_L][2];

hm->shcenter[1][0]=hm->center[KATA_D_R][0];
hm->shcenter[1][1]=hm->center[KATA_D_R][1];
hm->shcenter[1][2]=hm->center[KATA_D_R][2];


/*++++++++++++++++++++++++++++++++++*/

    return((void *)hm);
}

/*
 * 関数名 : ReadYubiKansetuData(int yubi, char *fname, float *vtc, int ofs)
 *
 * 引数   : yubi  : 指の部品番号
 *          fname : .obj ファイル名
 *          ofs   : 回転中心座標を求める時のオフセット
 *
 * 戻り値 : なし
 *
 * 機能   : ・指関節データの読み込み
 *          ・テクスチャインデックスの指定
 *          ・変形後の頂点データ格納領域の確保
 *          ・関節の回転の中心座標を求める
 *
 * 履歴   : 1992年10月28日 ; 作成 ; 広瀬  正俊
 */
void ReadYubiKansetuData(JINBUTU *hm, int yubi, char *fname, float *vtc, int ofs)
{
    hm->tindex[yubi] = hm->tindex[DOU];
    ReadWave(hm, fname, yubi, vtc);
    hm->va[yubi] =
        (VERTEX  *)malloc(sizeof(VERTEX)*hm->vmax[yubi]);
    GetCenter(hm, yubi, hm->center[yubi], 8, ofs);
}


void
KaradaZurasi(JINBUTU *hm)
{
    float       *v, *v2;
    float       vv[3], vv2[3], d[3];
    int         i, j;

    for (i = 0, v = (float *)hm->vs[DOU]; i < hm->vmax[DOU] - 36; i++, v += 3);
    v2 = (float *)hm->vs[KUBI];
```



```
    for (j = 0; j < 3; j++) {
        vv[j] = 0.0;
        vv2[j] = 0.0;
    }
    for (i = 0; i < 36; i++, v += 3, v2 += 3) {
        for (j = 0; j < 3; j++) {
            vv[j] += v[j];
            vv2[j] += v2[j];
        }
    }
    for (j = 0; j < 3; j++) {
        d[j] = (vv2[j] - vv[j])/36.0;
    }

    for (j = DOU; j < MAX_BUHIN; j++) {
        for (i = 0, v = (float *)hm->vs[j]; i < hm->vmax[j]; i++, v += 3) {
            v[0] += d[0];
            v[1] += d[1];
            v[2] += d[2];
        }
        hm->center[j][0] += d[0];
        hm->center[j][1] += d[1];
        hm->center[j][2] += d[2];
    }

    /* kubi to doutai wo kuttukeru */
    for (i = 0, v = (float *)hm->vs[DOU]; i < hm->vmax[DOU] - 36; i++, v += 3);
    for (i = 0, v2 = (float *)hm->vs[KUBI]; i < 35; i++, v2 += 3);
    for (i = 0; i < 36; i++, v += 3, v2 -= 3) {
        v[0] = v2[0];
        v[1] = v2[1];
        v[2] = v2[2];
    }
}


/*
 * 関数名 : GetCenter(int yubi, float cen, int ofs)
 *
 * 引数   : yubi : 指の部品番号
 *          cen  : 回転の中心座標
 *          ofs  : 頂点のオフセット数
 *
 * 戻り値 : なし
 *
 * 機能   : ・回転の中心座標を求める
 *          ・指関節は基本的に円筒状で円周方向に８つの頂点を持った複数の層からで *
 *
 * 履歴   : 1992年10月28日 ; 作成 ; 広瀬  正俊
 */
void
GetCenter(JINBUTU *hm, int yubi, float *cen, int rcnt, int ofs)
{
    float       *v, wx, wy, wz;
    int         i;

    wx = wy = wz = 0.0;
    v = (float *)hm->vs[yubi];
    v += ofs*3;
    for(i = 0; i < rcnt; i++) {
        wx += *(v++);
        wy += *(v++);
        wz += *(v++);
    }
    cen[0] = wx / (float)rcnt;
    cen[1] = wy / (float)rcnt;
```

```
    cen[2] = wz / (float)rcnt;
}

/*
 * 関数名 : CopyData(int buhin1, int buhin2)
 *
 * 引数   : buhin1 : 右手側データの部品番号
 *          buhin2 : 左手側データの部品番号
 *
 * 戻り値 : なし
 *
 * 機能   : ・右手側データをコピーして左手側データを作成する
 *          ・右手側データの頂点のｘ座標をマイナスして左手側頂点座標を作成する
 *
 * 履歴   : 1992年10月28日 ; 作成 ; 広瀬  正俊
 */
void
CopyData(JINBUTU *hm, int buhin1, int buhin2)
{
    float       *v1, *vt1, *a1;
    float       *v2, *vt2, *a2;
    int         *pol1, *pol2;
    int         i;

    hm->vmax[buhin2] = hm->vmax[buhin1];
    hm->vtmax[buhin2] = hm->vtmax[buhin1];
    hm->polmax[buhin2] = hm->polmax[buhin1];

    hm->center[buhin2][0] = -hm->center[buhin1][0];
    hm->center[buhin2][1] = hm->center[buhin1][1];
    hm->center[buhin2][2] = hm->center[buhin1][2];

    hm->tindex[buhin2] = hm->tindex[buhin1];

    hm->vs[buhin2] = (VERTEX  *)malloc(sizeof(VERTEX)*hm->vmax[buhin2]);
    hm->vt[buhin2] = (VTEXTURE  *)malloc(sizeof(VTEXTURE)*hm->vtmax[buhin2]);
    hm->poly[buhin2] = (POLYGON  *)malloc(sizeof(POLYGON)*hm->polmax[buhin2]);
    hm->a[buhin2] = (OMOMI  *)malloc(sizeof(OMOMI)*hm->vmax[buhin2]);
    v2 = (float *)hm->vs[buhin2];
    vt2 = (float *)hm->vt[buhin2];
    pol2 = (int *)hm->poly[buhin2];
    a2 = (float *)hm->a[buhin2];

    for(i = 0, v1 = (float *)hm->vs[buhin1]; i < hm->vmax[buhin2]; i++) {
        v2[0] = -v1[0];
        v2[1] = v1[1];
        v2[2] = v1[2];
        v1 += 3;
        v2 += 3;
    }

    for(i = 0, a1 = (float *)hm->a[buhin1]; i < hm->vmax[buhin2]; i++) {
        *(a2++) = *(a1++);
    }

    for(i = 0, vt1 = (float *)hm->vt[buhin1]; i < hm->vtmax[buhin2]; i++) {
        vt2[0] = vt1[0];
        vt2[1] = vt1[1];
        vt1 += 2;
        vt2 += 2;
    }

    for(i = 0, pol1 = (int *)hm->poly[buhin1]; i < hm->polmax[buhin2]; i++) {
        pol2[0] = pol1[0];
        pol2[1] = pol1[1];
        pol2[2] = pol1[2];
```



```
        pol2[3] = pol1[3];
        pol2[4] = pol1[4];
        pol2[5] = pol1[5];
        pol1 += 6;
        pol2 += 6;
    }
}


/*
 * 関数名 : ReadWave(char *fname, int buhin, float div, float px, float py)
 *
 * 引数   : fname       : .obj ファイル名
 *          buhin       : 部品番号
 *          div, px, py : テクスチャマッピングデータの補正値
 *
 * 戻り値 : なし
 *
 * 機能   : ・ウェーブフロントフォーマットの .obj ファイルを読み込む
 *
 * 履歴   : 1992年10月28日 ; 作成 ; 広瀬  正俊
 */
void
ReadWave(JINBUTU *hm, char *name, int buhin, float *vtc)
{
    FILE        *fp;
    char        buff[LINE_BUF];
    char        key[11];
    float       x, y, z, o;
    int         v1, v2, v3, vt1, vt2, vt3;
    int         vcnt, vtcnt, pcnt;
    int         rt;
    float       *v, *vt, *a;
    int         *pol;

    /* Open .obj file */
    if((fp = fopen(name, "r")) == NULL) {
        fprintf(stderr, "<<ERROR>> %s is not opened.\n", name);
        exit();
    }

    fprintf(stderr, "Now reading file %s\n", name);

    /* counter set */
    vcnt = vtcnt = pcnt = 0;
    while(fgets(buff, LINE_BUF, fp) != NULL) {
        if(!buff[0] || buff[0] == '#' || buff[0] == '\n')
            continue;
        sscanf(buff, "%10s", key);

        if(strcmp("v", key) == 0) {
            vcnt++;
        } else if(strcmp("vt", key) == 0) {
            vtcnt++;
        } else if((strcmp("fo", key) == 0) || (strcmp("f", key) == 0)) {
            pcnt++;
        }
    }
    hm->vmax[buhin] = vcnt;
    hm->vtmax[buhin] = vtcnt;
    hm->polmax[buhin] = pcnt;
    rewind(fp);

    /* memory allocation */
    hm->vs[buhin] = (VERTEX  *)malloc(sizeof(VERTEX)*vcnt);
    hm->vt[buhin] = (VTEXTURE  *)malloc(sizeof(VTEXTURE)*vtcnt);
```

```
    hm->poly[buhin] = (POLYGON  *)malloc(sizeof(POLYGON)*pcnt);
    hm->a[buhin] = (OMOMI  *)malloc(sizeof(OMOMI)*vcnt);
    v = (float *)hm->vs[buhin];
    vt = (float *)hm->vt[buhin];
    pol = (int *)hm->poly[buhin];
    a = (float *)hm->a[buhin];

    while(fgets(buff, LINE_BUF, fp) != NULL) {
        if(!buff[0] || buff[0] == '#' || buff[0] == '\n')
            continue;
        sscanf(buff, "%10s", key);

        if(strcmp("v", key) == 0) {
            /* add vertex to the vertex list */
            rt = sscanf(buff, "%s %f %f %f %f", key, &x, &y, &z, &o);
            if(rt == 4) {
                v[0] = x;
                v[1] = y;
                v[2] = z;
                v += 3;
            } else if(rt == 5) {
                v[0] = x;
                v[1] = y;
                v[2] = z;
                v += 3;
/*
if(buhin==KOSI)
o = 1.0-o;
*/
                *(a++) = o;
            } else {
                fprintf(stderr, "<<ERROR>> add vertex %s\n", buff);
                exit();
            }
        } else if(strcmp("vt", key) == 0) {
            /* add texture to the texture list */
            rt = sscanf(buff, "%s %f %f", key, &x, &y);
            if(rt == 3) {
                vt[0] = x * vtc[0] + vtc[1];
                vt[1] = y * vtc[0] + vtc[2];
                vt += 2;
            } else {
                fprintf(stderr, "<<ERROR>> add texture %s\n", buff);
                exit();
            }
        } else if((strcmp("fo", key) == 0) || (strcmp("f", key) == 0)) {
            /* add polygon to the polygon list */
            if(sscanf(buff, "%s %d/%d %d/%d %d/%d",
                      key, &v1, &vt1, &v2, &vt2, &v3, &vt3) != 7) {
                fprintf(stderr, "<<ERROR>> add polygon %s\n", buff);
                exit();
            } else {
                pol[0] = v1 - 1;
                pol[1] = v2 - 1;
                pol[2] = v3 - 1;
                pol[3] = vt1 - 1;
                pol[4] = vt2 - 1;
                pol[5] = vt3 - 1;
                pol += 6;
            }
        } else if(strcmp("cen", key) == 0) {
            /* add center vertex */
            if(sscanf(buff, "%s %f %f %f", key, &x, &y, &z) != 4) {
                fprintf(stderr, "<<ERROR>> add center %s\n", buff);
                exit();
            } else {
```



```
                hm->center[buhin][0] = x;
                hm->center[buhin][1] = y;
                hm->center[buhin][2] = z;
            }
        }
    }

    fclose(fp);

    fprintf(stderr, "%s read OK.\n", name);
    return;
}

/*
 * 関数名 : Henkei3D()
 *
 * 引数   : なし
 *
 * 戻り値 : なし
 *
 * 機能   : ・各関節の変形計算
 *          ・RotateVertex()で関節回転後の座標を求める
 *          ・IshiiVertex()で重み付け計算し、変形後の座標を求める
 *
 * 履歴   : 1992年10月28日 ; 作成 ; 広瀬　正俊
 */
void
Henkei3D(JINBUTU *hm)
{
    Matrix      mat;

    /* 右手側関節 */
    RotateVertex(hm, KATA_D_R);
    IshiiVertex(hm, KATA_D_R);
    RotateVertex(hm, KATA_N_R);
    IshiiVertex(hm, KATA_N_R);

/*++++++++++++++++++++++++++++++++++++*/
    SCFAC=S1;POWF=PF1;

    NewRotateVertex(hm, HIJI_R);
/*++++++++++++++++++++++++++++++++++++*/

    IshiiVertex(hm, HIJI_R);

    IshiiVertex2(hm->vs[UDE_R], hm->a[UDE_R], hm->vmax[UDE_R],
                 hm->va[UDE_R], hm->rtekubi_a, hm->center[TE_R]);

    RotateVertex(hm, TE_R);
    IshiiVertex(hm, TE_R);

    HenkeiYubiKansetu(hm, OYAYUBI_R_1);
    HenkeiYubiKansetu(hm, HITOSASIYUBI_R_1);
    HenkeiYubiKansetu(hm, NAKAYUBI_R_1);
    HenkeiYubiKansetu(hm, KUSURIYUBI_R_1);
    HenkeiYubiKansetu(hm, KOYUBI_R_1);

    /* 左手側関節 */
    RotateVertex(hm, KATA_D_L);
    IshiiVertex(hm, KATA_D_L);
    RotateVertex(hm, KATA_N_L);
    IshiiVertex(hm, KATA_N_L);

    RotateVertex(hm, HIJI_L);
    IshiiVertex(hm, HIJI_L);
```

```
    IshiiVertex2(hm->vs[UDE_L], hm->a[UDE_L], hm->vmax[UDE_L],
                 hm->va[UDE_L], hm->ltekubi_a, hm->center[TE_L]);

    RotateVertex(hm, TE_L);
    IshiiVertex(hm, TE_L);

    HenkeiYubiKansetu(hm, OYAYUBI_L_1);
    HenkeiYubiKansetu(hm, HITOSASIYUBI_L_1);
    HenkeiYubiKansetu(hm, NAKAYUBI_L_1);
    HenkeiYubiKansetu(hm, KUSURIYUBI_L_1);
    HenkeiYubiKansetu(hm, KOYUBI_L_1);

    /* 首関節 */
    RotateVertex(hm, KUBI);
    IshiiVertex(hm, KUBI);

if (trkswitch == 1 && ffdswitch == 0) {
    rev_mat(hm->mat[KOSI], mat);
    cp_mat(hm->mat[KOSI], mat);
}
    RotateVertex(hm, KOSI);
    IshiiVertex(hm, KOSI);

    RotateVertex(hm, SIRI_R);
    IshiiVertex(hm, SIRI_R);
    RotateVertex(hm, HIZA_R);
    IshiiVertex(hm, HIZA_R);

    RotateVertex(hm, SIRI_L);
    IshiiVertex(hm, SIRI_L);
    RotateVertex(hm, HIZA_L);
    IshiiVertex(hm, HIZA_L);

    /*++++++++++++++++++++++*/
    if (breatheswitch)
      breathehenkei(&hm->douspace,hm);
    if (ffdswitch)
      ffdhenkei(&hm->douspace,hm);
    if (collon)
      updateImpSpace(hm);
    /*++++++++++++++++++++++*/
}

void
Henkei3DHand(JINBUTU *hm)
{
    /* 右手側関節 */
    RotateVertex(hm, KATA_D_R);
    IshiiVertex(hm, KATA_D_R);
    RotateVertex(hm, KATA_N_R);
    IshiiVertex(hm, KATA_N_R);

    RotateVertex(hm, HIJI_R);
    IshiiVertex(hm, HIJI_R);

    IshiiVertex2(hm->vs[UDE_R], hm->a[UDE_R], hm->vmax[UDE_R],
                 hm->va[UDE_R], hm->rtekubi_a, hm->center[TE_R]);

    RotateVertex(hm, TE_R);
    IshiiVertex(hm, TE_R);

    HenkeiYubiKansetu(hm, OYAYUBI_R_1);
    HenkeiYubiKansetu(hm, HITOSASIYUBI_R_1);
    HenkeiYubiKansetu(hm, NAKAYUBI_R_1);
    HenkeiYubiKansetu(hm, KUSURIYUBI_R_1);
    HenkeiYubiKansetu(hm, KOYUBI_R_1);
```



```
    /* 左手側関節 */
    RotateVertex(hm, KATA_D_L);
    IshiiVertex(hm, KATA_D_L);
    RotateVertex(hm, KATA_N_L);
    IshiiVertex(hm, KATA_N_L);

    RotateVertex(hm, HIJI_L);
    IshiiVertex(hm, HIJI_L);

    IshiiVertex2(hm->vs[UDE_L], hm->a[UDE_L], hm->vmax[UDE_L],
                 hm->va[UDE_L], hm->ltekubi_a, hm->center[TE_L]);

    RotateVertex(hm, TE_L);
    IshiiVertex(hm, TE_L);

    HenkeiYubiKansetu(hm, OYAYUBI_L_1);
    HenkeiYubiKansetu(hm, HITOSASIYUBI_L_1);
    HenkeiYubiKansetu(hm, NAKAYUBI_L_1);
    HenkeiYubiKansetu(hm, KUSURIYUBI_L_1);
    HenkeiYubiKansetu(hm, KOYUBI_L_1);

    /* 首関節 */
    RotateVertex(hm, KUBI);
    IshiiVertex(hm, KUBI);
}

/*
 * 関数名 : HenkeiYubiKansetu(int yubi, Matrix mat)
 *
 * 引数   : yubi : 指の部品番号
 *          mat  : 回転マトリックス
 *
 * 戻り値 : なし
 *
 * 機能   : ・指関節の変形計算
 *
 * 履歴   : 1992年10月28日 ; 作成 ; 広瀬  正俊
 */
void
HenkeiYubiKansetu(JINBUTU *hm, int yubi)
{
    RotateVertex(hm, yubi);
    IshiiVertex(hm, yubi);
    RotateVertex(hm, yubi+1);
    IshiiVertex(hm, yubi+1);
    RotateVertex(hm, yubi+2);
    IshiiVertex(hm, yubi+2);
}


/*
 * 関数名 : DrawSimple3D()
 *
 *      : なし
 *
 *  :り値 : なし
 *
 *      : ・上半身モデルのテクスチャ表示
 *
 *      : 1992年10月28日 ; 作成 ; 広瀬＼敦 */
void
DrawSimple3D(void *vhm, float *linecIr)
[
    JINBUTU             *hm;
    static float        white[3] = [1.0, 1.0, 1.0};
```

```
    static float        red[3] = {1.0, 0.0, 1.0};
    static float        green[3] = {0.0, 1.0, 1.0};

    hm = (JINBUTU *)vhm;

    multmatrix(hm->mune_hosei_w);

    /* 体の位置を上方にずらす                */
    /* 上半身モデルの原点は頭部の中心にある */
    translate(0.0, -hm->center[DOU][1]+hm->muneyt, -hm->center[DOU][2]);

    /* 胸部の表示 */
#ifdef TANTAI
    multmatrix(hm->mat[DOU]);
#endif
    DrawMap3Shade(hm, DOU, hm->vs[DOU], green);

    /*                    */
    /* 右手側データの表示 */
    /*                    */
    pushmatrix();

    DrawMap3Shade(hm, KATA_D_R, hm->vs[KATA_D_R], red);
    multmatrix(hm->mat[KATA_D_R]);
    DrawMap3Shade(hm, NINOUDE_R, hm->vs[NINOUDE_R], green);

    DrawMap3Shade(hm, HIJI_R, hm->vs[HIJI_R], red);
    multmatrix(hm->mat[HIJI_R]);
    DrawMap3Shade(hm, UDE_R, hm->vs[UDE_R], green);

    DrawMap3Shade(hm, TE_R, hm->va[TE_R], red);
    multmatrix(hm->mat[TE_R]);

    pushmatrix();
    DrawMap3Shade(hm, OYAYUBI_R_1, hm->va[OYAYUBI_R_1], red);
    multmatrix(hm->mat[OYAYUBI_R_1]);
    DrawMap3Shade(hm, OYAYUBI_R_1+1, hm->va[OYAYUBI_R_1+1], red);
    multmatrix(hm->mat[OYAYUBI_R_1+1]);
    DrawMap3Shade(hm, OYAYUBI_R_1+2, hm->va[OYAYUBI_R_1+2], red);

    popmatrix();
    pushmatrix();
    DrawMap3Shade(hm, HITOSASIYUBI_R_1, hm->va[HITOSASIYUBI_R_1], red);
    multmatrix(hm->mat[HITOSASIYUBI_R_1]);
    DrawMap3Shade(hm, HITOSASIYUBI_R_1+1, hm->va[HITOSASIYUBI_R_1+1], red);
    multmatrix(hm->mat[HITOSASIYUBI_R_1+1]);
    DrawMap3Shade(hm, HITOSASIYUBI_R_1+2, hm->va[HITOSASIYUBI_R_1+2], red);

    popmatrix();
    pushmatrix();
    DrawMap3Shade(hm, NAKAYUBI_R_1, hm->va[NAKAYUBI_R_1], red);
    multmatrix(hm->mat[NAKAYUBI_R_1]);
    DrawMap3Shade(hm, NAKAYUBI_R_1+1, hm->va[NAKAYUBI_R_1+1], red);
    multmatrix(hm->mat[NAKAYUBI_R_1+1]);
    DrawMap3Shade(hm, NAKAYUBI_R_1+2, hm->va[NAKAYUBI_R_1+2], red);

    popmatrix();
    pushmatrix();
    DrawMap3Shade(hm, KUSURIYUBI_R_1, hm->va[KUSURIYUBI_R_1], red);
    multmatrix(hm->mat[KUSURIYUBI_R_1]);
    DrawMap3Shade(hm, KUSURIYUBI_R_1+1, hm->va[KUSURIYUBI_R_1+1], red);
    multmatrix(hm->mat[KUSURIYUBI_R_1+1]);
    DrawMap3Shade(hm, KUSURIYUBI_R_1+2, hm->va[KUSURIYUBI_R_1+2], red);

    popmatrix();
    DrawMap3Shade(hm, KOYUBI_R_1, hm->va[KOYUBI_R_1], red);
```



```
    multmatrix(hm->mat[KOYUBI_R_1]);
    DrawMap3Shade(hm, KOYUBI_R_1+1, hm->va[KOYUBI_R_1+1], red);
    multmatrix(hm->mat[KOYUBI_R_1+1]);
    DrawMap3Shade(hm, KOYUBI_R_1+2, hm->va[KOYUBI_R_1+2], red);

    /*                    */
    /* 左手側データの表示 */
    /*                    */
    popmatrix();
    pushmatrix();

    DrawMap3Shade(hm, KATA_D_L, hm->vs[KATA_D_L], red);
    multmatrix(hm->mat[KATA_D_L]);
    DrawMap3Shade(hm, NINOUDE_L, hm->vs[NINOUDE_L], green);

    DrawMap3Shade(hm, HIJI_L, hm->vs[HIJI_L], red);
    multmatrix(hm->mat[HIJI_L]);
    DrawMap3Shade(hm, UDE_L, hm->vs[UDE_L], green);

    DrawMap3Shade(hm, TE_L, hm->vs[TE_L], red);
    multmatrix(hm->mat[TE_L]);
    DrawMap3Shade(hm, OYAYUBI_L_1, hm->vs[OYAYUBI_L_1], green);

    /*            */
    /* 頭部の表示 */
    /*            */
    popmatrix();
    multmatrix(hm->mat[KUBI]);
    DrawMap3Shade(hm, KUBI, hm->vs[KUBI], red);

#ifdef TANTAI
    if (lineclr == NULL) {
        c3f(white);
        texbind(TX_TEXTURE_0, (long)hm->tindex[DOU]);
        tevbind(TV_ENV0, 1);
    }
    blendfunction(BF_SA, BF_MSA);
    DrawMap3(hm, ATAMA, hm->tindex[ATAMA], hm->vs[ATAMA], lineclr);
    blendfunction(BF_ONE, BF_ZERO);
    if (lineclr == NULL) {
        texbind(TX_TEXTURE_0, 0);
        tevbind(TV_ENV0, 0);
    }
#endif
}


/*
 * 関数名 : Draw3D()
 *
 * 引数   : なし
 *
 *  :り値 : なし
 *
 * 機能   : ・上半身モデルのテクスチャ表示
 *
 * 履歴   : 1992年10月28日 ; 作成 ; 広瀬＼欸 */
void
Draw3D(void *vhm, float *lineclr, int kahansin, int cup, float *p)
{
    JINBUTU             *hm;
    static float        white[3] = {1.0, 1.0, 1.0};

    hm = (JINBUTU *)vhm;

    if (lineclr == NULL) {
```

```
        c3f(white);
        texbind(TX_TEXTURE_0, (long)hm->tindex[DOU]);
        tevbind(TV_ENV0, 1);
    }

/****
    multmatrix(hm->mune_hosei_w);
****/

    /* 体の位置を上方にずらす                    */
    /* 上半身モデルの原点は頭部の中心にある */
    translate(0.0, -hm->center[DOU][1]+hm->muneyt, -hm->center[DOU][2]);

    /* 胸部の表示 */
#ifdef TANTAI
    multmatrix(hm->mat[DOU]);
#endif

    DrawMap3(hm, DOU, hm->tindex[DOU], hm->vs[DOU], lineclr);

    /* cup
    if (cup == 1) {
        GetTukamiKakudo(hm, p, 0);
    }
    */

    /*                          */
    /* 右手側データの表示 */
    /*                          */
    pushmatrix();

    DrawMap3(hm, KATA_D_R, hm->tindex[KATA_D_R], hm->va[KATA_D_R], lineclr);
    DrawMap3(hm, KATA_N_R, hm->tindex[KATA_N_R], hm->va[KATA_N_R], lineclr);

    multmatrix(hm->mat[KATA_D_R]);
    DrawMap3(hm, NINOUDE_R, hm->tindex[NINOUDE_R], hm->vs[NINOUDE_R], lineclr);

    DrawMap3(hm, HIJI_R, hm->tindex[HIJI_R], hm->va[HIJI_R], lineclr);
    multmatrix(hm->mat[HIJI_R]);

    DrawMap3(hm, UDE_R, hm->tindex[UDE_R], hm->va[UDE_R], lineclr);
    multmatrix(hm->mat[UDE_R]);

    DrawMap3(hm, TE_R, hm->tindex[TE_R], hm->va[TE_R], lineclr);
    multmatrix(hm->mat[TE_R]);

    pushmatrix();
    DrawYubiKansetu(hm, OYAYUBI_R_1, lineclr);

    popmatrix();
    pushmatrix();
    DrawYubiKansetu(hm, HITOSASIYUBI_R_1, lineclr);

    popmatrix();
    pushmatrix();
    DrawYubiKansetu(hm, NAKAYUBI_R_1, lineclr);

    popmatrix();
    pushmatrix();
    DrawYubiKansetu(hm, KUSURIYUBI_R_1, lineclr);

    popmatrix();
    DrawYubiKansetu(hm, KOYUBI_R_1, lineclr);

    /*                          */
    /* 左手側データの表示 */
```



```
    /*                          */
    popmatrix();
    pushmatrix();

    DrawMap3(hm, KATA_D_L, hm->tindex[KATA_D_L], hm->va[KATA_D_L], lineclr);
    DrawMap3(hm, KATA_N_L, hm->tindex[KATA_N_L], hm->va[KATA_N_L], lineclr);

    multmatrix(hm->mat[KATA_D_L]);
    DrawMap3(hm, NINOUDE_L, hm->tindex[NINOUDE_L], hm->vs[NINOUDE_L], lineclr);

    DrawMap3(hm, HIJI_L, hm->tindex[HIJI_L], hm->va[HIJI_L], lineclr);
    multmatrix(hm->mat[HIJI_L]);

    DrawMap3(hm, UDE_L, hm->tindex[UDE_L], hm->va[UDE_L], lineclr);
    multmatrix(hm->mat[UDE_L]);

    DrawMap3(hm, TE_L, hm->tindex[TE_L], hm->va[TE_L], lineclr);
    multmatrix(hm->mat[TE_L]);

    pushmatrix();
    DrawYubiKansetu(hm, OYAYUBI_L_1, lineclr);

    popmatrix();
    pushmatrix();
    DrawYubiKansetu(hm, HITOSASIYUBI_L_1, lineclr);

    popmatrix();
    pushmatrix();
    DrawYubiKansetu(hm, NAKAYUBI_L_1, lineclr);

    popmatrix();
    pushmatrix();
    DrawYubiKansetu(hm, KUSURIYUBI_L_1, lineclr);

    popmatrix();
    DrawYubiKansetu(hm, KOYUBI_L_1, lineclr);

if (kahansin) {
    popmatrix();
    pushmatrix();
    DrawMap3(hm, KOSI, hm->tindex[KOSI], hm->va[KOSI], lineclr);
    multmatrix(hm->mat[KOSI]);

    pushmatrix();
    DrawMap3(hm, SIRI_R, hm->tindex[SIRI_R], hm->va[SIRI_R], lineclr);
    multmatrix(hm->mat[SIRI_R]);

    DrawMap3(hm, MOMO_R, hm->tindex[MOMO_R], hm->vs[MOMO_R], lineclr);
    DrawMap3(hm, HIZA_R, hm->tindex[HIZA_R], hm->va[HIZA_R], lineclr);
    multmatrix(hm->mat[HIZA_R]);

    DrawMap3(hm, SUNE_R, hm->tindex[SUNE_R], hm->vs[SUNE_R], lineclr);
    multmatrix(hm->mat[KUTU_R]);
    DrawMap3(hm, KUTU_R, hm->tindex[KUTU_R], hm->vs[KUTU_R], lineclr);

    popmatrix();
    DrawMap3(hm, SIRI_L, hm->tindex[SIRI_L], hm->va[SIRI_L], lineclr);
    multmatrix(hm->mat[SIRI_L]);

    DrawMap3(hm, MOMO_L, hm->tindex[MOMO_L], hm->vs[MOMO_L], lineclr);
    DrawMap3(hm, HIZA_L, hm->tindex[HIZA_L], hm->va[HIZA_L], lineclr);
    multmatrix(hm->mat[HIZA_L]);

    DrawMap3(hm, SUNE_L, hm->tindex[SUNE_L], hm->vs[SUNE_L], lineclr);
    multmatrix(hm->mat[KUTU_L]);
    DrawMap3(hm, KUTU_L, hm->tindex[KUTU_L], hm->vs[KUTU_L], lineclr);
```

```
}

    /*              */
    /* 頭部の表示 */
    /*              */
    popmatrix();
    DrawMap3(hm, KUBI, hm->tindex[KUBI], hm->va[KUBI], lineclr);

    multmatrix(hm->mat[KUBI]);

#ifdef TANTAI
    DrawMap3(hm, ATAMA, hm->tindex[ATAMA], hm->vs[ATAMA], lineclr);
#endif

    if (lineclr == NULL) {
        texbind(TX_TEXTURE_0, 0);
        tevbind(TV_ENV0, 0);
    }
}


/*
 * 関数名 : Draw3DSeiza()
 *
 * 引数   : なし
 *
 * 戻り値 : なし
 *
 * 機能   : ・上半身モデルのテクスチャ表示
 *
 * 履歴   : 1992年10月28日 ; 作成 ; 広瀬＼敦 */
void
Draw3DSeiza(void *vhm, float *lineclr, int kahansin, int cup, float *p)
{
    JINBUTU             *hm;
    static float        white[3] = {1.0, 1.0, 1.0};

    hm = (JINBUTU *)vhm;

/*
    translate(0.0, hm->center[SIRI_R][1] - hm->center[KATA_D_R][1], 0.0);
*/

    if (hm->stand) {
        Draw3D(vhm, lineclr, 1, 0, white);
        return;
    }

    if (lineclr == NULL) {
        c3f(white);
        texbind(TX_TEXTURE_0, (long)hm->tindex[DOU]);
        tevbind(TV_ENV0, 1);
    }

    /* 体の位置を上方にずらす              */
    /* 上半身モデルの原点は頭部の中心にある */
    rot(180.0, 'y');
    translate(0.0, -hm->center[DOU][1], 0.0);
if(!trkswitch) {
    multmatrix(hm->mat[KOSI]);
}

/*********
    if (hm->muney > 5.0) {
*********/
    if (fsqrt(hm->kosix*hm->kosix +
```



```
        (hm->kosiy+hm->ske_real.dou_len)*(hm->kosiy+hm->ske_real.dou_len) +
        hm->kosiz*hm->kosiz) > 3.0) {
        translate(hm->kosix, hm->kosiy, hm->kosiz);
    } else {
        translate(0.0, -hm->ske_real.dou_len, 0.0);
    }

    pushmatrix();
    DrawMap3(hm, SIRI_R, hm->tindex[SIRI_R], hm->va[SIRI_R], lineclr);
    multmatrix(hm->mat[SIRI_R]);

    DrawMap3(hm, MOMO_R, hm->tindex[MOMO_R], hm->vs[MOMO_R], lineclr);
    DrawMap3(hm, HIZA_R, hm->tindex[HIZA_R], hm->va[HIZA_R], lineclr);
    multmatrix(hm->mat[HIZA_R]);

    DrawMap3(hm, SUNE_R, hm->tindex[SUNE_R], hm->vs[SUNE_R], lineclr);
    multmatrix(hm->mat[KUTU_R]);
    DrawMap3(hm, KUTU_R, hm->tindex[KUTU_R], hm->vs[KUTU_R], lineclr);

    popmatrix();
    pushmatrix();
    DrawMap3(hm, SIRI_L, hm->tindex[SIRI_L], hm->va[SIRI_L], lineclr);
    multmatrix(hm->mat[SIRI_L]);

    DrawMap3(hm, MOMO_L, hm->tindex[MOMO_L], hm->vs[MOMO_L], lineclr);
    DrawMap3(hm, HIZA_L, hm->tindex[HIZA_L], hm->va[HIZA_L], lineclr);
    multmatrix(hm->mat[HIZA_L]);

    DrawMap3(hm, SUNE_L, hm->tindex[SUNE_L], hm->vs[SUNE_L], lineclr);
    multmatrix(hm->mat[KUTU_L]);
    DrawMap3(hm, KUTU_L, hm->tindex[KUTU_L], hm->vs[KUTU_L], lineclr);

    popmatrix();
/*
if(!trkswitch) {
    multmatrix(hm->mat[KOSI]);
}
*/
prt_mat(hm->mat[KOSI], "hm->mat[KOSI]");
    DrawMap3(hm, KOSI, hm->tindex[KOSI], hm->va[KOSI], lineclr);

    /* 胸部の表示 */
/*++++++++++++++++++++++++++++++++++++++++++++++*/
    if (ffdswitch) {
        multmatrix(hm->mune_mat);
        DrawMap3(hm, DOU, hm->tindex[DOU], hm->va[DOU], lineclr);
    } else {
        multmatrix(hm->mat[KOSI]);
        DrawMap3(hm, DOU, hm->tindex[DOU], hm->vs[DOU], lineclr);
    }
/*++++++++++++++++++++++++++++++++++++++++++++++*/

    /* cup
    if (cup == 1) [
        GetTukamiKakudo(hm, p, 0);
    ]
    */

    /*              */
    /* 右手側データの表示 */
    /*              */
    pushmatrix();
    DrawMap3(hm, KATA_D_R, hm->tindex[KATA_D_R], hm->va[KATA_D_R], lineclr);
    DrawMap3(hm, KATA_N_R, hm->tindex[KATA_N_R], hm->va[KATA_N_R], lineclr);

/*++++++++++++++++++++++++++++++++++++++++++++++*/
```

```
    if (ffdswitch) {
        translate(hm->shcenter[1][0], hm->shcenter[1][1],
                  hm->shcenter[1][2]);
        rot(hm->douspace.na[1], 'x');
        translate(-hm->center[KATA_D_R][0], -hm->center[KATA_D_R][1],
                  -hm->center[KATA_D_R][2]);
    }
/*++++++++++++++++++++++++++++++++++++++++++*/

    multmatrix(hm->mat[KATA_D_R]);
    DrawMap3(hm, NINOUDE_R, hm->tindex[NINOUDE_R], hm->vs[NINOUDE_R], lineclr);

    DrawMap3(hm, HIJI_R, hm->tindex[HIJI_R], hm->va[HIJI_R], lineclr);
    multmatrix(hm->mat[HIJI_R]);

    DrawMap3(hm, UDE_R, hm->tindex[UDE_R], hm->va[UDE_R], lineclr);
    multmatrix(hm->mat[UDE_R]);

    DrawMap3(hm, TE_R, hm->tindex[TE_R], hm->va[TE_R], lineclr);
    multmatrix(hm->mat[TE_R]);

    pushmatrix();
    DrawYubiKansetu(hm, OYAYUBI_R_1, lineclr);

    popmatrix();
    pushmatrix();
    DrawYubiKansetu(hm, HITOSASIYUBI_R_1, lineclr);

    popmatrix();
    pushmatrix();
    DrawYubiKansetu(hm, NAKAYUBI_R_1, lineclr);

    popmatrix();
    pushmatrix();
    DrawYubiKansetu(hm, KUSURIYUBI_R_1, lineclr);

    popmatrix();
    DrawYubiKansetu(hm, KOYUBI_R_1, lineclr);

    /*                    */
    /* 左手側データの表示 */
    /*                    */
    popmatrix();
    pushmatrix();
    DrawMap3(hm, KATA_D_L, hm->tindex[KATA_D_L], hm->va[KATA_D_L], lineclr);
    DrawMap3(hm, KATA_N_L, hm->tindex[KATA_N_L], hm->va[KATA_N_L], lineclr);

/*++++++++++++++++++++++++++++++++++*/
    if (ffdswitch) {
        translate(hm->shcenter[0][0], hm->shcenter[0][1],
                  hm->shcenter[0][2]);
        rot(hm->douspace.na[0], 'x');
        translate(-hm->center[KATA_D_L][0], -hm->center[KATA_D_L][1],
                  -hm->center[KATA_D_L][2]);
    }
/*++++++++++++++++++++++++++++++++++*/

/*++++++++++++++++++++++++++++++++++*/
    if (lineclr == NULL) {
      texbind(TX_TEXTURE_0, 0);
      tevbind(TV_ENV0, 0);
    }

    trgrSph(1.0,spcenter,3);

     if (lineclr == NULL) {
```



```
        c3f(white);
        texbind(TX_TEXTURE_0, (long)hm->tindex[DOU]);
        tevbind(TV_ENV0, 1);
    }
/*++++++++++++++++++++++++++++++++++*/

    multmatrix(hm->mat[KATA_D_L]);
    DrawMap3(hm, NINOUDE_L, hm->tindex[NINOUDE_L], hm->vs[NINOUDE_L], lineclr);

    DrawMap3(hm, HIJI_L, hm->tindex[HIJI_L], hm->va[HIJI_L], lineclr);
    multmatrix(hm->mat[HIJI_L]);

    DrawMap3(hm, UDE_L, hm->tindex[UDE_L], hm->va[UDE_L], lineclr);
    multmatrix(hm->mat[UDE_L]);

    DrawMap3(hm, TE_L, hm->tindex[TE_L], hm->va[TE_L], lineclr);
    multmatrix(hm->mat[TE_L]);

    pushmatrix();
    DrawYubiKansetu(hm, OYAYUBI_L_1, lineclr);

    popmatrix();
    pushmatrix();
    DrawYubiKansetu(hm, HITOSASIYUBI_L_1, lineclr);

    popmatrix();
    pushmatrix();
    DrawYubiKansetu(hm, NAKAYUBI_L_1, lineclr);

    popmatrix();
    pushmatrix();
    DrawYubiKansetu(hm, KUSURIYUBI_L_1, lineclr);

    popmatrix();
    DrawYubiKansetu(hm, KOYUBI_L_1, lineclr);

    /*            */
    /* 頭部の表示 */
    /*            */
    popmatrix();
    DrawMap3(hm, KUBI, hm->tindex[KUBI], hm->va[KUBI], lineclr);

/*++++++++++++++++++++++++++++++++++++++*/
    if (ffdswitch) {
        translate(hm->douspace.center[3][0],hm->douspace.center[3][1],
                  hm->douspace.center[3][2]);
        rot(3*hm->douspace.xang, 'x');
        translate(-hm->douspace.corig[3][0],-hm->douspace.corig[3][1],
                  -hm->douspace.corig[3][2]);
    }
/*++++++++++++++++++++++++++++++++++++++*/

    multmatrix(hm->mat[KUBI]);
#ifdef TANTAI
    DrawMap3(hm, ATAMA, hm->tindex[ATAMA], hm->vs[ATAMA], lineclr);
#endif

    if (lineclr == NULL) {
        texbind(TX_TEXTURE_0, 0);
        tevbind(TV_ENV0, 0);
    }
}


/*
 * 関数名 : Draw3DWireSeiza2()
```

```
 *
 * 引数   : なし
 *
 *  :り値 : なし
 *
 * 機能   : 上半身モデルのワイヤー表示
 *
 * 履歴   : 1992年10月28日 ; 作成 ; 広瀬＼敦
 */
void
Draw3DSeiza2(void *vhm, float *lineclr, int kahansin, int cup, float *p)
[
    JINBUTU             *hm;
    static float        white[3] = {1.0, 1.0, 1.0};
    Matrix      mat;

    hm = (JINBUTU *)vhm;

    if (hm->stand) [
        Draw3D(vhm, lineclr, 1, 0, white);
        return;
    }

    if (lineclr == NULL) {
        c3f(white);
        texbind(TX_TEXTURE_0, (long)hm->tindex[DOU]);
        tevbind(TV_ENV0, 1);
    }

printf("ffdswitch:%d\n",ffdswitch);
printf("trkswitch:%d\n",trkswitch);

    /* 体の位置を上方にずらす */
/*
    rot(180.0, 'y');
    translate(0.0, -hm->center[DOU][1], 0.0);
*/

    multmatrix(hm->mat[DOU]);

    /* 胸部の表示 */
/*+++++++++++++++++++++++++++++++++++++++++++++*/
    if (ffdswitch) {
        rev_mat(hm->mune_mat, mat);
        translate(hm->center[KOSI][0], hm->center[KOSI][1],
                  hm->center[KOSI][2]);
        multmatrix(mat);
        translate(-hm->center[KOSI][0], -hm->center[KOSI][1],
                  -hm->center[KOSI][2]);
        DrawMap3(hm, DOU, hm->tindex[DOU], hm->va[DOU], lineclr);
    } else {
        DrawMap3(hm, DOU, hm->tindex[DOU], hm->vs[DOU], lineclr);
    }
/*+++++++++++++++++++++++++++++++++++++++++++++*/

    /*                      */
    /* 右手側データの表示 */
                    /*                          */
    pushmatrix();
    DrawMap3(hm, KATA_D_R, hm->tindex[KATA_D_R], hm->va[KATA_D_R], lineclr);
    DrawMap3(hm, KATA_N_R, hm->tindex[KATA_N_R], hm->va[KATA_N_R], lineclr);

/*+++++++++++++++++++++++++++++++++++++++++++++*/
    if (ffdswitch) {
        translate(hm->shcenter[1][0], hm->shcenter[1][1],
                  hm->shcenter[1][2]);
```



```
        rot(hm->douspace.na[1], 'x');
        translate(-hm->center[KATA_D_R][0], -hm->center[KATA_D_R][1],
                  -hm->center[KATA_D_R][2]);
    }
/*+++++++++++++++++++++++++++++++++++++++++++++*/

    multmatrix(hm->mat[KATA_D_R]);
    DrawMap3(hm, NINOUDE_R, hm->tindex[NINOUDE_R], hm->vs[NINOUDE_R], lineclr);

    DrawMap3(hm, HIJI_R, hm->tindex[HIJI_R], hm->va[HIJI_R], lineclr);
    multmatrix(hm->mat[HIJI_R]);

    DrawMap3(hm, UDE_R, hm->tindex[UDE_R], hm->va[UDE_R], lineclr);
    multmatrix(hm->mat[UDE_R]);

{
float p[3];

getmatrix(mat);
coord_calc(p, hm->center[TE_R], mat);
printf("[p]:%f,%f,%f\n",p[0],p[1],p[2]);
}
    DrawMap3(hm, TE_R, hm->tindex[TE_R], hm->va[TE_R], lineclr);
    multmatrix(hm->mat[TE_R]);

    pushmatrix();
    DrawYubiKansetu(hm, OYAYUBI_R_1, lineclr);

    popmatrix();
    pushmatrix();
    DrawYubiKansetu(hm, HITOSASIYUBI_R_1, lineclr);

    popmatrix();
    pushmatrix();
    DrawYubiKansetu(hm, NAKAYUBI_R_1, lineclr);

    popmatrix();
    pushmatrix();
    DrawYubiKansetu(hm, KUSURIYUBI_R_1, lineclr);

    popmatrix();
    DrawYubiKansetu(hm, KOYUBI_R_1, lineclr);

    /*                      */
    /* 左手側データの表示 */
                    /*                          */
    popmatrix();
    pushmatrix();
    DrawMap3(hm, KATA_D_L, hm->tindex[KATA_D_L], hm->va[KATA_D_L], lineclr);
    DrawMap3(hm, KATA_N_L, hm->tindex[KATA_N_L], hm->va[KATA_N_L], lineclr);

/*+++++++++++++++++++++++++++++++++*/
    if (ffdswitch) {
        translate(hm->shcenter[0][0], hm->shcenter[0][1],
                  hm->shcenter[0][2]);
        rot(hm->douspace.na[0], 'x');
        translate(-hm->center[KATA_D_L][0], -hm->center[KATA_D_L][1],
                  -hm->center[KATA_D_L][2]);
    }
/*+++++++++++++++++++++++++++++++++*/

/*+++++++++++++++++++++++++++++++++*/
    if (lineclr == NULL) [
      texbind(TX_TEXTURE_0, 0);
      tevbind(TV_ENV0, 0);
    }
```

```
    trgrSph(1.0,spcenter,3);

     if (lineclr == NULL) {
        c3f(white);
        texbind(TX_TEXTURE_0, (long)hm->tindex[DOU]);
        tevbind(TV_ENV0, 1);
    }
/*++++++++++++++++++++++++++++++++*/

    multmatrix(hm->mat[KATA_D_L]);
    DrawMap3(hm, NINOUDE_L, hm->tindex[NINOUDE_L], hm->vs[NINOUDE_L], lineclr);

    DrawMap3(hm, HIJI_L, hm->tindex[HIJI_L], hm->va[HIJI_L], lineclr);
    multmatrix(hm->mat[HIJI_L]);

    DrawMap3(hm, UDE_L, hm->tindex[UDE_L], hm->va[UDE_L], lineclr);
    multmatrix(hm->mat[UDE_L]);

    DrawMap3(hm, TE_L, hm->tindex[TE_L], hm->va[TE_L], lineclr);
    multmatrix(hm->mat[TE_L]);

    pushmatrix();
    DrawYubiKansetu(hm, OYAYUBI_L_1, lineclr);

    popmatrix();
    pushmatrix();
    DrawYubiKansetu(hm, HITOSASIYUBI_L_1, lineclr);

    popmatrix();
    pushmatrix();
    DrawYubiKansetu(hm, NAKAYUBI_L_1, lineclr);

    popmatrix();
    pushmatrix();
    DrawYubiKansetu(hm, KUSURIYUBI_L_1, lineclr);

    popmatrix();
    DrawYubiKansetu(hm, KOYUBI_L_1, lineclr);

    /*          */
    /* 頭部の表示 */
          /*                */
    popmatrix();
    pushmatrix();

    DrawMap3(hm, KUBI, hm->tindex[KUBI], hm->va[KUBI], lineclr);

/*++++++++++++++++++++++++++++++++++++*/
    if (ffdswitch) {
        translate(hm->douspace.center[3][0],hm->douspace.center[3][1],
                  hm->douspace.center[3][2]);
        rot(3*hm->douspace.xang, 'x');
        translate(-hm->douspace.corig[3][0],-hm->douspace.corig[3][1],
                  -hm->douspace.corig[3][2]);
    }
/*++++++++++++++++++++++++++++++++++++*/

    multmatrix(hm->mat[KUBI]);
    DrawMap3(hm, ATAMA, hm->tindex[ATAMA], hm->va[ATAMA], lineclr);

/***
    if (hm->muney > 5.0) [
****/

    popmatrix();
```



```
/*++++++++++++++++++++++++++++++++++++++++++++*/
    if (ffdswitch) {
        multmatrix(hm->kosi_mat);
        DrawMap3(hm, KOSI, hm->tindex[KOSI], hm->va[KOSI], lineclr);
/*
rev_mat(hm->kosi_mat, mat);
multmatrix(mat);
*/
    }
/*++++++++++++++++++++++++++++++++++++++++++++*/
    else {
        DrawMap3(hm, KOSI, hm->tindex[KOSI], hm->va[KOSI], lineclr);
        multmatrix(hm->mat[KOSI]);
    }

    if (fsqrt(hm->kosix*hm->kosix +
        (hm->kosiy+hm->ske_real.dou_len)*(hm->kosiy+hm->ske_real.dou_len) +
        hm->kosiz*hm->kosiz) > 3.0) {
printf("AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\n");
        translate(hm->kosix, hm->kosiy, hm->kosiz);

printf("\n");
printf("hm->kosi : %f, %f, %f \n", hm->kosix, hm->kosiy, hm->kosiz);

    } else {
printf("BBBBBBBBBBBBBBBBBBBBBBBBBBBBB\n");
#ifdef HIROSE
        translate(0.0, -hm->ske_real.dou_len, 0.0);
#endif
    }

    pushmatrix();
#if 1
    DrawMap3(hm, SIRI_R, hm->tindex[SIRI_R], hm->va[SIRI_R], lineclr);
#endif
    multmatrix(hm->mat[SIRI_R]);

#if 1
    DrawMap3(hm, MOMO_R, hm->tindex[MOMO_R], hm->vs[MOMO_R], lineclr);
#endif
    DrawMap3(hm, HIZA_R, hm->tindex[HIZA_R], hm->va[HIZA_R], lineclr);
    multmatrix(hm->mat[HIZA_R]);

    DrawMap3(hm, SUNE_R, hm->tindex[SUNE_R], hm->vs[SUNE_R], lineclr);
    multmatrix(hm->mat[KUTU_R]);
    DrawMap3(hm, KUTU_R, hm->tindex[KUTU_R], hm->vs[KUTU_R], lineclr);

    popmatrix();
#if 1
    DrawMap3(hm, SIRI_L, hm->tindex[SIRI_L], hm->va[SIRI_L], lineclr);
#endif
    multmatrix(hm->mat[SIRI_L]);

#if 1
    DrawMap3(hm, MOMO_L, hm->tindex[MOMO_L], hm->vs[MOMO_L], lineclr);
#endif
    DrawMap3(hm, HIZA_L, hm->tindex[HIZA_L], hm->va[HIZA_L], lineclr);
    multmatrix(hm->mat[HIZA_L]);

    DrawMap3(hm, SUNE_L, hm->tindex[SUNE_L], hm->vs[SUNE_L], lineclr);
    multmatrix(hm->mat[KUTU_L]);
    DrawMap3(hm, KUTU_L, hm->tindex[KUTU_L], hm->vs[KUTU_L], lineclr);

    if (lineclr == NULL) {
        texbind(TX_TEXTURE_0, 0);
        tevbind(TV_ENV0, 0);
```

```
    }
}


void
Draw3DHand(void *vhm, float *lineclr)
{
    JINBUTU             *hm;
    static float        white[3] = {1.0, 1.0, 1.0};

    hm = (JINBUTU *)vhm;

    if (hm->simple) {
        return;
    ]

    if (lineclr == NULL) {
        c3f(white);
        texbind(TX_TEXTURE_0, (long)hm->tindex[DOU]);
        tevbind(TV_ENV0, 1);
    ]

    multmatrix(hm->mune_hosei_w);

    /* 体の位置を上方にずらす                */
    /* 上半身モデルの原点は頭部の中心にある */
    translate(0.0, -hm->center[DOU][1]+hm->muneyt, -hm->center[DOU][2]);

    /* 胸部の表示 */
#ifdef TANTAI
    multmatrix(hm->mat[DOU]);
#endif

    /*                      */
    /* 右手側データの表示 */
    /*                      */
    pushmatrix();

    multmatrix(hm->mat[KATA_D_R]);
    DrawMap3(hm, NINOUDE_R, hm->tindex[NINOUDE_R], hm->vs[NINOUDE_R], lineclr);

    DrawMap3(hm, HIJI_R, hm->tindex[HIJI_R], hm->va[HIJI_R], lineclr);
    multmatrix(hm->mat[HIJI_R]);

    DrawMap3(hm, UDE_R, hm->tindex[UDE_R], hm->va[UDE_R], lineclr);
    multmatrix(hm->mat[UDE_R]);

    DrawMap3(hm, TE_R, hm->tindex[TE_R], hm->va[TE_R], lineclr);
    multmatrix(hm->mat[TE_R]);

    pushmatrix();
    DrawYubiKansetu(hm, OYAYUBI_R_1, lineclr);

    popmatrix();
    pushmatrix();
    DrawYubiKansetu(hm, HITOSASIYUBI_R_1, lineclr);

    popmatrix();
    pushmatrix();
    DrawYubiKansetu(hm, NAKAYUBI_R_1, lineclr);

    popmatrix();
    pushmatrix();
    DrawYubiKansetu(hm, KUSURIYUBI_R_1, lineclr);

    popmatrix();
```



```
    DrawYubiKansetu(hm, KOYUBI_R_1, lineclr);

    /*                      */
    /* 左手側データの表示 */
    /*                      */
    popmatrix();

    multmatrix(hm->mat[KATA_D_L]);
    DrawMap3(hm, NINOUDE_L, hm->tindex[NINOUDE_L], hm->vs[NINOUDE_L], lineclr);

    DrawMap3(hm, HIJI_L, hm->tindex[HIJI_L], hm->va[HIJI_L], lineclr);
    multmatrix(hm->mat[HIJI_L]);

    DrawMap3(hm, UDE_L, hm->tindex[UDE_L], hm->va[UDE_L], lineclr);
    multmatrix(hm->mat[UDE_L]);

    DrawMap3(hm, TE_L, hm->tindex[TE_L], hm->va[TE_L], lineclr);
    multmatrix(hm->mat[TE_L]);

    pushmatrix();
    DrawYubiKansetu(hm, OYAYUBI_L_1, lineclr);

    popmatrix();
    pushmatrix();
    DrawYubiKansetu(hm, HITOSASIYUBI_L_1, lineclr);

    popmatrix();
    pushmatrix();
    DrawYubiKansetu(hm, NAKAYUBI_L_1, lineclr);

    popmatrix();
    pushmatrix();
    DrawYubiKansetu(hm, KUSURIYUBI_L_1, lineclr);

    popmatrix();
    DrawYubiKansetu(hm, KOYUBI_L_1, lineclr);

    if (lineclr == NULL) {
        texbind(TX_TEXTURE_0, 0);
        tevbind(TV_ENV0, 0);
    ]
]


void
Draw3DHandSeiza(void *vhm, float *lineclr)
{
    JINBUTU             *hm;
    static float        white[3] = {1.0, 1.0, 1.0};

    hm = (JINBUTU *)vhm;

    translate(0.0, hm->center[SIRI_R][1] - hm->center[KATA_D_R][1], 0.0);

    if (hm->stand) {
        Draw3DHand(vhm, lineclr);
        return;
    ]

    if (lineclr == NULL) [
        c3f(white);
        texbind(TX_TEXTURE_0, (long)hm->tindex[DOU]);
        tevbind(TV_ENV0, 1);
    ]

    /* 体の位置を上方にずらす                */
```

```
    /* 上半身モデルの原点は頭部の中心にある */
    pushmatrix();
    rot(180.0, 'y');
    translate(0.0, -hm->center[DOU][1], 0.0);

    multmatrix(hm->mat[KOSI]);

    /* 胸部の表示 */

    /*                    */
    /* 右手側データの表示 */
    /*                    */
    pushmatrix();

    multmatrix(hm->mat[KATA_D_R]);
    DrawMap3(hm, NINOUDE_R, hm->tindex[NINOUDE_R], hm->vs[NINOUDE_R], lineclr);

    DrawMap3(hm, HIJI_R, hm->tindex[HIJI_R], hm->va[HIJI_R], lineclr);
    multmatrix(hm->mat[HIJI_R]);

    DrawMap3(hm, UDE_R, hm->tindex[UDE_R], hm->va[UDE_R], lineclr);
    multmatrix(hm->mat[UDE_R]);

    DrawMap3(hm, TE_R, hm->tindex[TE_R], hm->va[TE_R], lineclr);
    multmatrix(hm->mat[TE_R]);

    pushmatrix();
    DrawYubiKansetu(hm, OYAYUBI_R_1, lineclr);

    popmatrix();
    pushmatrix();
    DrawYubiKansetu(hm, HITOSASIYUBI_R_1, lineclr);

    popmatrix();
    pushmatrix();
    DrawYubiKansetu(hm, NAKAYUBI_R_1, lineclr);

    popmatrix();
    pushmatrix();
    DrawYubiKansetu(hm, KUSURIYUBI_R_1, lineclr);

    popmatrix();
    DrawYubiKansetu(hm, KOYUBI_R_1, lineclr);

    /*                    */
    /* 左手側データの表示 */
    /*                    */
    popmatrix();

    multmatrix(hm->mat[KATA_D_L]);
    DrawMap3(hm, NINOUDE_L, hm->tindex[NINOUDE_L], hm->vs[NINOUDE_L], lineclr);

    DrawMap3(hm, HIJI_L, hm->tindex[HIJI_L], hm->va[HIJI_L], lineclr);
    multmatrix(hm->mat[HIJI_L]);

    DrawMap3(hm, UDE_L, hm->tindex[UDE_L], hm->va[UDE_L], lineclr);
    multmatrix(hm->mat[UDE_L]);

    DrawMap3(hm, TE_L, hm->tindex[TE_L], hm->va[TE_L], lineclr);
    multmatrix(hm->mat[TE_L]);

    pushmatrix();
    DrawYubiKansetu(hm, OYAYUBI_L_1, lineclr);

    popmatrix();
    pushmatrix();
```



```
    DrawYubiKansetu(hm, HITOSASIYUBI_L_1, lineclr);

    popmatrix();
    pushmatrix();
    DrawYubiKansetu(hm, NAKAYUBI_L_1, lineclr);

    popmatrix();
    pushmatrix();
    DrawYubiKansetu(hm, KUSURIYUBI_L_1, lineclr);

    popmatrix();
    DrawYubiKansetu(hm, KOYUBI_L_1, lineclr);

    popmatrix();

    if (lineclr == NULL) {
        texbind(TX_TEXTURE_0, 0);
        tevbind(TV_ENV0, 0);
    }
}


/*
 * 関数名 : DrawYubiKansetu(int yubi, Matrix mat)
 *
 * 引数   : yubi : 指の部品番号
 *          mat  : 回転マトリックス
 *
 * 戻り値 : なし
 *
 * 機能   : ・右手指のテクスチャ表示
 *
 * 履歴   : 1992年10月28日 ; 作成 ; 広瀬  正俊
 */
void
DrawYubiKansetu(JINBUTU *hm, int yubi, float *lineclr)
{
    DrawMap3(hm, yubi, hm->tindex[yubi], hm->va[yubi], lineclr);
    multmatrix(hm->mat[yubi]);
    DrawMap3(hm, yubi+1, hm->tindex[yubi+1], hm->va[yubi+1], lineclr);
    multmatrix(hm->mat[yubi+1]);
    DrawMap3(hm, yubi+2, hm->tindex[yubi+2], hm->va[yubi+2], lineclr);
}


/*
 * 関数名 : DrawMap3(int buhin, int mapindex, VERTEX *g_v)
 *
 * 引数   : buhin     : 部品番号
 *          mapindex  : テクスチャインデックス番号
 *          g_v       : 頂点座標
 *
 * 戻り値 : なし
 *
 * 機能   : ・部品のテクスチャ表示
 *
 * 履歴   : 1992年10月28日 ; 作成 ; 広瀬  正俊
 */
void
DrawMap3(JINBUTU *hm, int buhin, int mapindex, VERTEX *g_v, float *lineclr)
{
    int                 i, *pol;

    if (lineclr == NULL) {
        for (i = 0; i < hm->polmax[buhin]; i++) {
            pol = (int *)(hm->poly[buhin]+i);
```

```
            bgnpolygon();
            t2f((float *)(hm->vt[buhin]+pol[3]));
            v3f((float *)(g_v+pol[0]));
            t2f((float *)(hm->vt[buhin]+pol[4]));
            v3f((float *)(g_v+pol[1]));
            t2f((float *)(hm->vt[buhin]+pol[5]));
            v3f((float *)(g_v+pol[2]));
            endpolygon();
        }
    } else [
        c3f(lineclr);
        for (i = 0; i < hm->polmax[buhin]; i++) {
            pol = (int *)(hm->poly[buhin]+i);
            bgnclosedline();
            v3f((float *)(g_v+pol[0]));
            v3f((float *)(g_v+pol[1]));
            v3f((float *)(g_v+pol[2]));
            endclosedline();
        }
    }
}


/*
 * 関数名 : DrawMap3Shade(int buhin, VERTEX *g_v, float *lineclr)
 *
 * 引数   : buhin   : 部品番号
 *          g_v     : 頂点座標
 *          lineclr : ワイヤーの色情報
 *
 * 戻り値 : なし
 *
 * 機能   : 右手指のワイヤー表示
 *
 * 履歴   : 1992年10月28日 ; 作成 ; 広瀬＼敦 */
void
DrawMap3Shade(JINBUTU *hm, int buhin, VERTEX *g_v, float *lineclr)
[
    int         i, *pol;
float   *v, vtx[3][3], normal[3], col[3];

    c3f(lineclr);

    for (i = 0; i < hm->polmax[buhin]; i++) {
        pol = (int *)(hm->poly[buhin]+i);
v = (float *)(g_v+pol[0]);
vtx[0][0] = v[0];
vtx[0][1] = v[1];
vtx[0][2] = v[2];
v = (float *)(g_v+pol[1]);
vtx[1][0] = v[0];
vtx[1][1] = v[1];
vtx[1][2] = v[2];
v = (float *)(g_v+pol[2]);
vtx[2][0] = v[0];
vtx[2][1] = v[1];
vtx[2][2] = v[2];
FaceNormal(vtx, normal);
normal[0] *= -1.0;
normal[1] *= -1.0;
normal[2] *= -1.0;
/*
if (normal[2] > 0.0) {
    col[0] = lineclr[0] * normal[2];
    col[1] = lineclr[1] * normal[2];
    col[2] = lineclr[2] * normal[2];
```



```
} else {
    col[0] = 0.0;
    col[1] = 0.0;
    col[2] = 0.0;
}
c3f(col);
*/
        bgnpolygon();
n3f(normal);
        v3f((float *)(g_v+pol[0]));
n3f(normal);
        v3f((float *)(g_v+pol[1]));
n3f(normal);
        v3f((float *)(g_v+pol[2]));
        endpolygon();
    }
}


/*
 * 関数名 : RotateVertex(VERTEX *g_v, VERTEX *g_xyz, int vmax, Matrix rmat,
 *                       float b1, float b2, float b3)
 *
 * 引数   : g_v        : 回転前の座標
 *          g_xyz      : 回転後の座標
 *          vmax       : 頂点数
 *          rmat       : 回転マトリックス
 *          b1, b2, b3 : 回転中心座標
 *
 * 戻り値 : なし
 *
 * 機能   : ・(b1, b2, b3) を回転中心として rmat 回転させた座標を求める
 *
 * 履歴   : 1992年10月28日 ; 作成 ; 広瀬  正俊
 */
void
RotateVertex(JINBUTU *hm, int buhin)
{
    float       *vs, *va;
    int         k;
    float       a1, a2, a3, a4;
    float       b1, b2, b3, b4;
    float       c1, c2, c3, c4;

    a1 = hm->mat[buhin][0][0];
    a2 = hm->mat[buhin][1][0];
    a3 = hm->mat[buhin][2][0];
    a4 = hm->mat[buhin][3][0];
    b1 = hm->mat[buhin][0][1];
    b2 = hm->mat[buhin][1][1];
    b3 = hm->mat[buhin][2][1];
    b4 = hm->mat[buhin][3][1];
    c1 = hm->mat[buhin][0][2];
    c2 = hm->mat[buhin][1][2];
    c3 = hm->mat[buhin][2][2];
    c4 = hm->mat[buhin][3][2];
    for (k = 0, vs = (float *)hm->vs[buhin], va = (float *)hm->va[buhin];
         k < hm->vmax[buhin]; k++, vs += 3, va += 3) {
        va[0] = a1*vs[0] + a2*vs[1] + a3*vs[2] +a4;
        va[1] = b1*vs[0] + b2*vs[1] + b3*vs[2] +b4;
        va[2] = c1*vs[0] + c2*vs[1] + c3*vs[2] +c4;
    ]
}


/*++++++++++++++++++++++++++++++++++*/
```

```
NewRotateVertex(JINBUTU *hm, int buhin)
{
    float       *vs, *va, *a;
    int         k;
    float       a1, a2, a3, a4,ta1,ta2;
    float       b1, b2, b3, b4;
    float       c1, c2, c3, c4;
    float       scalefactor, x1, x2;

    scalefactor=scAngle(hm->mat[buhin][0][0]+hm->mat[buhin][1][1]+
                        hm->mat[buhin][2][2]);

    a1 = hm->mat[buhin][0][0];
    a2 = hm->mat[buhin][1][0];
    a3 = hm->mat[buhin][2][0];
    a4 = hm->mat[buhin][3][0];
    b1 = hm->mat[buhin][0][1];
    b2 = hm->mat[buhin][1][1];
    b3 = hm->mat[buhin][2][1];
    b4 = hm->mat[buhin][3][1];
    c1 = hm->mat[buhin][0][2];
    c2 = hm->mat[buhin][1][2];
    c3 = hm->mat[buhin][2][2];
    c4 = hm->mat[buhin][3][2];

    x2=hm->center[buhin][0];

    for (k = 0, vs = (float *)hm->vs[buhin], va = (float *)hm->va[buhin],
         a = (float *)hm->a[buhin];
         k < hm->vmax[buhin]; k++, vs += 3, va += 3,a++)
      {
        ta1=blendF(1-fabs(2*(*a)-1));
        ta2=(float)pow((double)ta1,POWF);
        x1=/*sc_mat[1][1]=sc_mat[2][2]=*/
          (1.0+scalefactor*ta2);


        a1 = hm->mat[buhin][0][0]*x1;
        b1 = hm->mat[buhin][0][1]*x1;
        c1 = hm->mat[buhin][0][2]*x1;

        a4 = hm->mat[buhin][3][0]-hm->mat[buhin][0][0]*x2*(x1-1);
        b4 = hm->mat[buhin][3][1]-hm->mat[buhin][0][1]*x2*(x1-1);
        c4 = hm->mat[buhin][3][2]-hm->mat[buhin][0][2]*x2*(x1-1);

        va[0] = a1*vs[0] + a2*vs[1] + a3*vs[2] +a4;
        va[1] = b1*vs[0] + b2*vs[1] + b3*vs[2] +b4;
        va[2] = c1*vs[0] + c2*vs[1] + c3*vs[2] +c4;
      }
}


/*++++++++++++++++++++++++++++++++++*/


/*
 * 関数名 : IshiiVertex(VERTEX *g_v, OMOMI *g_a, int vmax, VERTEX *g_xyz)
 *
 * 引数   : g_v   : 変形前の座標
 *          g_a   : 各頂点の重み
 *          vmax  : 頂点数
 *          g_xyz : 変形後の座標
 *
 * 戻り値 : なし
```



```
 *
 * 機能   : ・各頂点の重み付けに従って変形後の座標を求める
 *
 * 履歴   : 1992年10月28日 ; 作成 ; 広瀬  正俊
 */
void
IshiiVertex(JINBUTU *hm, int buhin)
{
    float       *vs, *a, *va;
    int         i;

if (buhin == KOSI && trkswitch == 1 && ffdswitch == 0) {
    for (i = 0, vs = (float *)hm->vs[buhin], a = (float *)hm->a[buhin],
         va = (float *)hm->va[buhin];
        i < hm->vmax[buhin]; i++, vs += 3, a++, va += 3) {
        va[0] = (*a)*vs[0] + (1.0-(*a))*va[0];
        va[1] = (*a)*vs[1] + (1.0-(*a))*va[1];
        va[2] = (*a)*vs[2] + (1.0-(*a))*va[2];
    }
} else {
    for (i = 0, vs = (float *)hm->vs[buhin], a = (float *)hm->a[buhin],
         va = (float *)hm->va[buhin];
        i < hm->vmax[buhin]; i++, vs += 3, a++, va += 3) {
        va[0] = (*a)*va[0] + (1.0-(*a))*vs[0];
        va[1] = (*a)*va[1] + (1.0-(*a))*vs[1];
        va[2] = (*a)*va[2] + (1.0-(*a))*vs[2];
    }
}
}

/*
 * 関数名 : IshiiVertex2(VERTEX *g_v, OMOMI *g_a, int vmax, VERTEX *g_xyz,
 *                       float xang, float b1, float b2, float b3)
 *
 * 引数   : g_v         : 変形前の座標
 *          g_a         : 各頂点の重み
 *          vmax        : 頂点数
 *          g_xyz       : 変形後の座標
 *          xang        : 回転角
 *          b1, b2, b3 : 回転中心座標
 *
 * 戻り値 : なし
 *
 * 機能   : ・各頂点の重み付けに従って変形後の座標を求める
 *
 * 履歴   : 1992年10月28日 ; 作成 ; 広瀬  正俊
 */
void
IshiiVertex2(VERTEX *g_v, OMOMI *g_a, int vmax, VERTEX *g_xyz, float xang,
             float *b)
{
    float       *v, *a, *xyz;
    int         k;
    float       s, c, t1, t2;

    for(k = 0, v = (float *)g_v, a = (float *)g_a, xyz = (float *)g_xyz;
        k < vmax; k++, v += 3, a++, xyz += 3) {
        if((k%24) == 0) {
            s = (float)sin(xang*(*a)/180.0*M_PI);
            c = (float)cos(xang*(*a)/180.0*M_PI);
            t1 = -b[1]*c + b[2]*s + b[1];
            t2 = -b[1]*s - b[2]*c + b[2];
        ]

        xyz[0] = v[0];
        xyz[1] = c*v[1] - s*v[2] + t1;
```

```
        xyz[2] = s*v[1] + c*v[2] + t2;
    }
}


void
CalcKahansin(JINBUTU *hm, float *trd, int sisei)
{
    float       a, b, L1, L2, A1, A2, B1, B2;
    float       x, y1, y2, z1, z2, costh1, costh2;
    float       siri_th1, siri_th2, hiza_th1, hiza_th2;
    float       kutu_th1, kutu_th2, dou_th;

#if 0
    a = (hm->center[KATA_D_R][1]+hm->center[KATA_D_L][1])/2.0
         + G_trd[MUNE_TRAK_NO][1] - hm->ske.momo_len;
    b = (hm->center[KATA_D_R][2]+hm->center[KATA_D_L][2])/2.0
         + G_trd[MUNE_TRAK_NO][2] + hm->ske.momo_len;
    y1 = a - hm->center[HIZA_R][1];
    y2 = a - hm->center[HIZA_L][1];
    z1 = b - hm->center[HIZA_R][2];
    z2 = b - hm->center[HIZA_L][2];
    L1 = fsqrt(y1*y1 + z1*z1);
    L2 = fsqrt(y2*y2 + z2*z2);
    costh1 = (hm->ske.dou_len*hm->ske.dou_len +
              hm->ske.momo_len*hm->ske.momo_len - L1*L1) /
              (2.0*hm->ske.dou_len*hm->ske.momo_len);
    costh2 = (hm->ske.dou_len*hm->ske.dou_len +
              hm->ske.momo_len*hm->ske.momo_len - L2*L2) /
              (2.0*hm->ske.dou_len*hm->ske.momo_len);
    if (costh1 > 1.0) {
        costh1 = 1.0;
    } else if (costh1 < -1.0) {
        costh1 = -1.0;
    }
    if (costh2 > 1.0) {
        costh2 = 1.0;
    } else if (costh2 < -1.0) {
        costh2 = -1.0;
    }
    siri_th1 = acosf(costh1) / M_PI * 180.0;
    siri_th2 = acosf(costh2) / M_PI * 180.0;

    costh1 = (hm->ske.momo_len*hm->ske.momo_len + L1*L1 -
              hm->ske.dou_len*hm->ske.dou_len) / (2.0*hm->ske.momo_len*L1);
    costh2 = (hm->ske.momo_len*hm->ske.momo_len + L2*L2 -
              hm->ske.dou_len*hm->ske.dou_len) / (2.0*hm->ske.momo_len*L2);

    if (costh1 > 1.0) {
        costh1 = 1.0;
    } else if (costh1 < -1.0) {
        costh1 = -1.0;
    }
    if (costh2 > 1.0) {
        costh2 = 1.0;
    } else if (costh2 < -1.0) {
        costh2 = -1.0;
    }
    A1 = acosf(costh1) / M_PI * 180.0;
    B1 = atanf(y1/z1) / M_PI * 180.0;
    A2 = acosf(costh2) / M_PI * 180.0;
    B2 = atanf(y2/z2) / M_PI * 180.0;
    if (B1 < 0.0) {
        B1 += 180.0;
    }
    if (B2 < 0.0) {
```



```
        B2 += 180.0;
    }
    dou_th = 90.0 - siri_th1 - A1 + B1;
    hiza_th1 = 90.0 - (B1 - A1);
    hiza_th2 = 90.0 - (B2 - A2);
#endif

/*
printf("hiza_th,A,B:%f,%f,%f\n",hiza_th,A,B);
*/

printf("\n");
printf("MUNE : %f, %f, %f\n", hm->munex,
       hm->muney, hm->munez);
/*
printf("KATA_R : %f, %f, %f\n", hm->center[KATA_D_R][0],
       hm->center[KATA_D_R][1], hm->center[KATA_D_R][2]);
printf("SIRI_R : %f, %f, %f\n", hm->center[SIRI_R][0],
       hm->center[SIRI_R][1], hm->center[SIRI_R][2]);
printf("HIZA_R : %f, %f, %f\n", hm->center[HIZA_R][0],
       hm->center[HIZA_R][1], hm->center[HIZA_R][2]);
printf("KUTU_R : %f, %f, %f\n", hm->center[KUTU_R][0],
       hm->center[KUTU_R][1], hm->center[KUTU_R][2]);
*/

#ifdef TANTAI
    c_loadmatrix(G_idmat);
c_rot(-180.0, 'y');
    c_translate(hm->center[KOSI][0], hm->center[KOSI][1],
                hm->center[KOSI][2]);
    c_multmatrix(hm->mat[DOU]);
    c_translate(-hm->center[KOSI][0], -hm->center[KOSI][1],
                -hm->center[KOSI][2]);
/*
    c_translate(0.0, G_trd[MUNE_TRAK_NO][1], -G_trd[MUNE_TRAK_NO][2]);

    c_translate(hm->center[DOU][0], hm->center[DOU][1], hm->center[DOU][2]);
    c_rot(180.0, 'y');
    c_translate(-hm->center[DOU][0], -hm->center[DOU][1], -hm->center[DOU][2]);
*/
    c_getmatrix(hm->mat[KOSI]);
#endif

CalcStandingKahansin(hm, 1, &siri_th1, &siri_th2,
                     &hiza_th1, &hiza_th2, &kutu_th1, &kutu_th2);

/*
    GetKakudo_zyx(hm->mat[DOU], &z, &y, &x);
    c_loadmatrix(G_idmat);
    c_translate(hm->center[KOSI][0], hm->center[KOSI][1], hm->center[KOSI][2]);
    c_rot(siri_th-180.0, 'x');
    c_translate(-hm->center[KOSI][0], -hm->center[KOSI][1],
                -hm->center[KOSI][2]);
    c_getmatrix(hm->mat[KOSI]);
*/

    c_loadmatrix(G_idmat);
    c_translate(hm->center[SIRI_R][0], hm->center[SIRI_R][1],
                hm->center[SIRI_R][2]);

    c_rot(siri_th1 - 90.0, 'x');

    c_rot(-hm->mata, 'z');
    c_translate(-hm->center[SIRI_R][0], -hm->center[SIRI_R][1],
                -hm->center[SIRI_R][2]);
    c_getmatrix(hm->mat[SIRI_R]);
```

```
    c_loadmatrix(G_idmat);
    c_translate(hm->center[SIRI_L][0], hm->center[SIRI_L][1],
                hm->center[SIRI_L][2]);
    c_rot(siri_th2 - 90.0, 'x');
    c_rot(hm->mata, 'z');
    c_translate(-hm->center[SIRI_L][0], -hm->center[SIRI_L][1],
                -hm->center[SIRI_L][2]);
    c_getmatrix(hm->mat[SIRI_L]);

    c_loadmatrix(G_idmat);
    c_translate(hm->center[HIZA_R][0], hm->center[HIZA_R][1],
                hm->center[HIZA_R][2]);
    c_rot(hm->mata,  'z');
    c_rot(hiza_th1,  'x');
    c_rot(-hm->mata, 'z');
    c_translate(-hm->center[HIZA_R][0], -hm->center[HIZA_R][1],
                -hm->center[HIZA_R][2]);
    c_getmatrix(hm->mat[HIZA_R]);

    c_loadmatrix(G_idmat);
    c_translate(hm->center[HIZA_L][0], hm->center[HIZA_L][1],
                hm->center[HIZA_L][2]);
    c_rot(-hm->mata, 'z');
    c_rot(hiza_th2,  'x');
    c_rot(hm->mata,  'z');
    c_translate(-hm->center[HIZA_L][0], -hm->center[HIZA_L][1],
                -hm->center[HIZA_L][2]);
    c_getmatrix(hm->mat[HIZA_L]);

    c_loadmatrix(G_idmat);
    c_translate(hm->center[KUTU_R][0], hm->center[KUTU_R][1],
                hm->center[KUTU_R][2]);
    c_rot(hm->mata,  'z');
    c_rot(kutu_th1,  'x');
    c_translate(-hm->center[KUTU_R][0], -hm->center[KUTU_R][1],
                -hm->center[KUTU_R][2]);
    c_getmatrix(hm->mat[KUTU_R]);

    c_loadmatrix(G_idmat);
    c_translate(hm->center[KUTU_L][0], hm->center[KUTU_L][1],
                hm->center[KUTU_L][2]);
    c_rot(-hm->mata, 'z');
    c_rot(kutu_th2,  'x');
    c_translate(-hm->center[KUTU_L][0], -hm->center[KUTU_L][1],
                -hm->center[KUTU_L][2]);
    c_getmatrix(hm->mat[KUTU_L]);
}


void
CalcStandingKahansin(JINBUTU *hm, int sisei,
                     float *siri_th1, float *siri_th2,
                     float *hiza_th1, float *hiza_th2,
                     float *kutu_th1, float *kutu_th2)
{
    float       L1, L2, y1, y2, z1, z2, costh1, costh2;
    float       hizay1, hizay2, hizaz1, hizaz2;
    float       A, B;
    Matrix      mat;
    float       dou_len, momo_len, sune_len;

    dou_len = hm->ske_real.dou_len;
    momo_len = hm->ske_real.momo_len;
    sune_len = hm->ske_real.sune_len;
```



```
    calc_ang_geom(hm->kosiz, hm->kosiy, G_trd[TR_MIGIASHI][2],
                  G_trd[TR_MIGIASHI][1], sune_len, momo_len,
                  kutu_th1, hiza_th1);
    calc_ang_geom(hm->kosiz, hm->kosiy, G_trd[TR_HIDARIASHI][2],
                  G_trd[TR_HIDARIASHI][1], sune_len, momo_len,
                  kutu_th2, hiza_th2);

    costh1 = (90.0 - *kutu_th1) / 180.0 * M_PI;
    hizaz1 = G_trd[TR_MIGIASHI][2] - fcos(costh1) * sune_len;
    hizay1 = G_trd[TR_MIGIASHI][1] + fsin(costh1) * sune_len;

    costh2 = (90.0 - *kutu_th2) / 180.0 * M_PI;
    hizaz2 = G_trd[TR_HIDARIASHI][2] - fcos(costh2) * sune_len;
    hizay2 = G_trd[TR_HIDARIASHI][1] + fsin(costh2) * sune_len;

hm->rhizay = hizay1;
hm->rhizaz = hizaz1;
hm->lhizay = hizay2;
hm->lhizaz = hizaz2;

    y1 = fabs(hm->kosiy - hizay1);
    z1 = fabs(hm->kosiz - hizaz1);
    L1 = fsqrt(y1*y1 + z1*z1);
    y2 = fabs(hm->kosiy - hizay2);
    z2 = fabs(hm->kosiz - hizaz2);
    L2 = fsqrt(y2*y2 + z2*z2);

    costh1 = (L1*L1 + z1*z1 - y1*y1) / (2.0 * L1 * z1);
    costh2 = (L2*L2 + z2*z2 - y2*y2) / (2.0 * L2 * z2);

    if (costh1 < -1.0) [
        costh1 = -1.0;
    } else if (costh1 > 1.0){
        costh1 = 1.0;
    }
    if (costh2 < -1.0) [
        costh2 = -1.0;
    } else if (costh2 > 1.0){
        costh2 = 1.0;
    }
    costh1 = acosf(costh1) / M_PI * 180.0;
    costh2 = acosf(costh2) / M_PI * 180.0;

    *siri_th1 = costh1;
    *siri_th2 = costh2;
}


void
CalcKansetuMatrix(JINBUTU *hm, int k, Matrix mat)
[
    c_loadmatrix(G_idmat);
    c_translate(hm->center[k][0], hm->center[k][1], hm->center[k][2]);
    c_multmatrix(mat);
    c_translate(-hm->center[k][0], -hm->center[k][1], -hm->center[k][2]);
    c_getmatrix(hm->mat[k]);
]


void
c_calib_model(void *vhm, float *trd)
[
    JINBUTU     *hm;
    FILE        *fp;
    Matrix      mat;
    int         i;
```

```
/*****************************************************************
*       ファイル名 ： calc_object.c                              *
*****************************************************************/

/*
* set_trk_dg_data( trk, dg, fang )                               *
*       トラッカーおよびデータグローブのデータを                 *
*       作業用領域に格納                                         *
*                                                                *
* set_coord( trk )                                               *
*       首、胸、左右手首の座標値および首、胸の回転角度を設定     *
*                                                                *
* calc_model( trk, dg )                                          *
*       頭、胸、肩、肘、手首の回転マトリックスを計算             *
*                                                                *
* RotateVertex(g_v, g_xyz, vmax, rmat, cent)                     *
*       各部分の形状データを回転角に応じて回転させる             *
*                                                                *
* IshiiVertex(g_v, g_a, vmax, g_xyz)                             *
*       形状データの回転に重みを付けて回転させる                 *
*                                                                *
* IshiiVertex2(g_v, g_a, vmax, g_xyz, xang, cent)                *
*       形状データの回転に重みを付けて回転させる                 *
*/

#include        <gl/gl.h>
#include        <device.h>
#include        <string.h>
#include        <stdio.h>

#include        "body.h"
#include        "body_draw.h"

#ifndef TANTAI
#include        "myexte.h"
#define         CYBERGLOVE
#endif

#include        "rname_fc.h"
/*
#define         TIME_PRINT
*/

#define         r2d(a)          ((a)*180.0/3.1415926)

static  Matrix  G_idmat = {1.0, 0.0, 0.0, 0.0,
                           0.0, 1.0, 0.0, 0.0,
                           0.0, 0.0, 1.0, 0.0,
                           0.0, 0.0, 0.0, 1.0};
float   ishii_hyouka();

static float    ka2_angx = 0.0;
static float    ka1_angx = 0.0;
static float    ka2_angy = 0.0;
static float    ka1_angy = 0.0;
static float    ka2_angz = 0.0;
static float    ka1_angz = 0.0;
static float    rhiji2_pnt[3] = {0.0, 0.0, 0.0};
static float    rhiji1_pnt[3] = {0.0, 0.0, 0.0};
static float    rtekubi2_a = 0.0;
static float    rtekubi1_a = 0.0;
static float    rtekubi2_y = 0.0;
static float    rtekubi1_y = 0.0;
static float    rtekubi2_z = 0.0;
static float    rtekubi1_z = 0.0;
```



```
static float    ex_xangle = -60.0;

void    SetKubiKaiten(JINBUTU *, float *, Matrix, float *, Matrix, int);
void    SetTekubiKaiten(JINBUTU *, float *, Matrix, float *, Matrix);
extern int      rotateSpace(JINBUTU *, Space *, float *, int);
extern int      rotateSpace2(JINBUTU *, Space *, float *, float *, float *, int);
extern int      rotateSpace3(JINBUTU *, Space *, float *, float *, float *, int);
extern int      G_four;

/*
 * 関数名 : SetTrkDgData(int *dg, float fang[10][4])
 *
 * 引数   : dg   データグローブからのデータ
 *          fang 指角度の格納琉
 *
 * 戻り値 : なし
 *
 * 機能   : データグローブのデータを作業用琉茲乏頁
 *
 * 履歴   : 1994年12月29日 ; 作成 ; 広瀬＼欺
 */
void
SetTrkDgData(int *dg, float fang[10][4])
[
    float       dg1, dg2, dg3, dg4;
    int         i;

    /* 指の角度を決める */
    for (i = 0; i < 5; i++) {
        fang[i][0] = (float)*dg++;
        fang[i][1] = (float)*dg;
        fang[i][2] = (float)(*dg++/2);
    ]
    dg1 = *(dg+0) / 90.0 * 35.0;
    dg2 = *(dg+1) / 90.0 * 20.0;
    dg3 = *(dg+2) / 90.0 * 10.0;
    dg4 = *(dg+3) / 90.0 * 10.0;
    fang[0][3] = dg1 + dg2 - fabs(NAKAYUBI_HOSEI) - fabs(OYAYUBI_HOSEI);
    fang[1][3] = dg2 - fabs(NAKAYUBI_HOSEI) - fabs(HITOSASIYUBI_HOSEI);
    fang[2][3] = 0.0;
    fang[3][3] = -(dg3 + fabs(NAKAYUBI_HOSEI) - fabs(KUSURIYUBI_HOSEI));
    fang[4][3] = -(dg3 + dg4 + fabs(NAKAYUBI_HOSEI) - fabs(KOYUBI_HOSEI));
#ifdef CYBERGLOVE
    dg += 6;
#else
    dg += 4;
#endif

    for (i = 5; i < 10; i++) {
        fang[i][0] = -(float)*dg++;
        fang[i][1] = -(float)*dg;
        fang[i][2] = -(float)(*dg++/2);
    }
    dg1 = *(dg+0) / 90.0 * 35.0;
    dg2 = *(dg+1) / 90.0 * 20.0;
    dg3 = *(dg+2) / 90.0 * 10.0;
    dg4 = *(dg+3) / 90.0 * 10.0;
    fang[5][3] = -(dg1 + dg2 - fabs(NAKAYUBI_HOSEI) - fabs(OYAYUBI_HOSEI));
    fang[6][3] = -(dg2 - fabs(NAKAYUBI_HOSEI) - fabs(HITOSASIYUBI_HOSEI));
    fang[7][3] = 0.0;
    fang[8][3] = dg3 + fabs(NAKAYUBI_HOSEI) - fabs(KUSURIYUBI_HOSEI);
    fang[9][3] = dg3 + dg4 + fabs(NAKAYUBI_HOSEI) - fabs(KOYUBI_HOSEI);
    fang[5][0] *= -1.0;
]
```

```
/*****************************************************************
*       1．頭、胸、肩、肘、手首の回転マトリックスを計          *
*       2．肩、肘、手首の変形データを計                        *
*                                                               *
*       引数 ： trk （トラッカーデータ）                        *
*                  トラッカーデータ（X，Y，Z座標値、a，r，e回転角度 *
*              dg （データグローブデータ）                      *
*       リターン値 ： なし                                      *
*****************************************************************/
calc_body(vhm, trk, dg, sisei, kahansin)
    void        *vhm;
    float       *trk;
    int         *dg;
    int         sisei;
    int         kahansin;
{
    int         i;
    float       ret, min_ret;
    float       sav_ang;
    JINBUTU     *hm;
    float       trd[30];
    float       fang[10][4];
    Matrix      mune_m, atama_m, rtekubi_m, ltekubi_m;
    Matrix      mune_rm, mune_rm2;
    Matrix      rtekubi1_m, rtekubi2_m;
    Matrix      ltekubi1_m, ltekubi2_m;
    float       ltekubi_a, ltekubi_y, ltekubi_z;
    float       lhiji_pnt[3];
    float       rtekubi[3], ltekubi[3];
    static float        ka_angx, ka_angy, ka_angz;
    static float        ka_savangx, ka_savangy, ka_savangz;
    static float        rtekubi_a, rtekubi_y, rtekubi_z;
    static float        rhiji_pnt[3];
    static float        ka1_angx, ka1_angy, ka1_angz;
    static float        rtekubi1_a, rtekubi1_y, rtekubi1_z;
    static float        rhiji_pnt1[3];
    static float        ka2_angx, ka2_angy, ka2_angz;
    static float        rtekubi2_a, rtekubi2_y, rtekubi2_z;
    static float        rhiji_pnt2[3];
    static float    lyubihosei[5][3] = {
                {0, OYAYUBI_HOSEI, OYAYUBI_HOSEI},
                {HITOSASIYUBI_HOSEI, HITOSASIYUBI_HOSEI, HITOSASIYUBI_HOSEI},
                {NAKAYUBI_HOSEI, NAKAYUBI_HOSEI, NAKAYUBI_HOSEI},
                {KUSURIYUBI_HOSEI, KUSURIYUBI_HOSEI, KUSURIYUBI_HOSEI},
                {KOYUBI_HOSEI, KOYUBI_HOSEI, KOYUBI_HOSEI}
        };
    static float    ryubihosei[5][3] = {
                [0, -OYAYUBI_HOSEI, -OYAYUBI_HOSEI},
                [-HITOSASIYUBI_HOSEI, -HITOSASIYUBI_HOSEI, -HITOSASIYUBI_HOSEI},
                {-NAKAYUBI_HOSEI, -NAKAYUBI_HOSEI, -NAKAYUBI_HOSEI},
                {-KUSURIYUBI_HOSEI, -KUSURIYUBI_HOSEI, -KUSURIYUBI_HOSEI},
                {-KOYUBI_HOSEI, -KOYUBI_HOSEI, -KOYUBI_HOSEI}
        ];

printf("((trk[0]))%f,%f,%f,%f,%f,%f\n",trk[0],trk[1],trk[2],trk[3],trk[4],trk[5]);
printf("((trk[1]))%f,%f,%f,%f,%f,%f\n",trk[6],trk[7],trk[8],trk[9],trk[10],trk[11]);
printf("((trk[2]))%f,%f,%f,%f,%f,%f\n",trk[12],trk[13],trk[14],trk[15],trk[16],trk[17
]);
printf("((trk[3]))%f,%f,%f,%f,%f,%f\n",trk[18],trk[19],trk[20],trk[21],trk[22],trk[23
]);
    hm = (JINBUTU *)vhm;

/*
    if ((kahansin == 0) && hm->simple) {
        return;
```



```
    }
*/

    for (i = 0; i < 30; i++) [
        trd[i] = trk[i];
    }

/*
    if (sisei != SITTING) [
        trd[MUNE_TRAK_NO*6+1] += hm->muneyt;
        trd[MUNE_TRAK_NO*6+2] += hm->munezt;
    }
*/

    /* データグローブのデータを作業琉芸 */
    SetTrkDgData(dg, fang);

    pushmatrix();

    /* 胸と頭の回転マトリックスを求める */
    SetKubiKaiten(hm, &trd[ATAMA_TRAK_NO*6], atama_m,
                &trd[MUNE_TRAK_NO*6], mune_m, sisei);

    /* 左右の手首の磁気センサーの回転マトリックスを作成 */
    SetTekubiKaiten(hm, &trd[MIGITE_TRAK_NO*6], rtekubi_m,
                        &trd[HIDARITE_TRAK_NO*6], ltekubi_m);

    /* ワールド座標用から表示用マトリックスの作成 */
    world_disp_mat(mune_m, hm->mat1.mune);
    world_disp_mat(atama_m, hm->mat1.atama);
    world_disp_mat(rtekubi_m, hm->mat1.rtekubi);
    world_disp_mat(ltekubi_m, hm->mat1.ltekubi);

    /* 胸の回転マトリックスの逆マトリックスを作成 */
    rev_mat(hm->mat1.mune, mune_rm);
    rev_mat(mune_m, mune_rm2);

    /* 胸との相対の頭の回転マトリックスを作成 */
    loadmatrix(mune_rm);
    multmatrix(hm->mat1.atama);
    getmatrix(hm->mat1.atama);

    /* 右手首の相対位置 */
    rtekubi[0] = rtekubi_m[3][0] - mune_m[3][0];
    rtekubi[1] = rtekubi_m[3][1] - mune_m[3][1];
    rtekubi[2] = rtekubi_m[3][2] - mune_m[3][2];

    /*  手首の座標を胸の回転だけ戻す  */
/*
    coord_calc(rtekubi, rtekubi, mune_rm);
*/
    coord_calc(rtekubi, rtekubi, mune_rm2);

    min_ret = 100000000.0;
    sav_ang = -60.0;

    for (xangle = ex_xangle - 16.0; xangle < ex_xangle + 17.0; xangle += 2.0) [
        GetKataMatrix(0, rtekubi, xangle, -hm->ske_real.kata_len,
                    hm->ske_real.ninoude_len, hm->ske_real.ude_len,
                    hm->mat1.rkata, hm->mat1.rhiji, rhiji_pnt);


        /* 肩のＸ軸回転成分とＹＺ軸回転成分の取り出し */
        GetKataRotMatrix(hm->mat1.rkata, &rkata_a, rkata1_m, rkata2_m);

        /* 胸との相対の右手首の回転マトリックスを作成 */
```

```
        GetTekubiMatrix2(0, hm->mat1.mune, hm->mat1.rkata, hm->mat1.rhiji,
                    hm->mat1.rtekubi, &rtekubi_a, &rtekubi_y, &rtekubi_z,
                    rtekubi1_m, rtekubi2_m);

        GetKakudo_xyz(hm->mat1.rkata, &ka_angx, &ka_angy, &ka_angz);

        ret = ishii_hyouka(ka_angx, ka_angy, ka_angz,
                           rhiji_pnt,
                           rtekubi_a, rtekubi_y, rtekubi_z,
                           ka1_angx, ka1_angy, ka1_angz,
                           rhiji1_pnt,
                           rtekubi1_a, rtekubi1_y, rtekubi1_z,
                           ka2_angx, ka2_angy, ka2_angz,
                           rhiji2_pnt,
                           rtekubi2_a, rtekubi2_y, rtekubi2_z);

        if (ret < min_ret) [
            min_ret = ret;
            sav_ang = xangle;
            ka_savangx = ka_angx;
            ka_savangy = ka_angy;
            ka_savangz = ka_angz;
        ]
    ]

    xangle = sav_ang;

    /* 胸との相対の右肩と右肘の回転マトリックスを作成 */
    GetKataMatrix(0, rtekubi, xangle, -hm->ske_real.kata_len,
                  hm->ske_real.ninoude_len, hm->ske_real.ude_len,
                  hm->mat1.rkata, hm->mat1.rhiji, rhiji_pnt);

    /* 肩のＸ軸回転成分とＹＺ軸回転成分の取り出し */
    GetKataRotMatrix(hm->mat1.rkata, &rkata_a, rkata1_m, rkata2_m);

    /* 胸との相対の右手首の回転マトリックスを作成 */
    GetTekubiMatrix(0, hm->mat1.mune, hm->mat1.rkata, hm->mat1.rhiji,
                    hm->mat1.rtekubi, &rtekubi_a, &rtekubi_y, &rtekubi_z,
                    rtekubi1_m, rtekubi2_m);

    /* １個前のデータを２個前にする */
    ka2_angx = ka1_angx;
    ka2_angy = ka1_angy;
    ka2_angz = ka1_angz;
    rhiji2_pnt[0] = rhiji1_pnt[0];
    rhiji2_pnt[1] = rhiji1_pnt[1];
    rhiji2_pnt[2] = rhiji1_pnt[2];
    rtekubi2_a = rtekubi1_a;
    rtekubi2_y = rtekubi1_y;
    rtekubi2_z = rtekubi1_z;

    /* 今のデータを１個前にする */
    ka1_angx = ka_savangx;
    ka1_angy = ka_savangy;
    ka1_angz = ka_savangz;
    rhiji1_pnt[0] = rhiji_pnt[0];
    rhiji1_pnt[1] = rhiji_pnt[1];
    rhiji1_pnt[2] = rhiji_pnt[2];
    rtekubi1_a = rtekubi_a;
    rtekubi1_y = rtekubi_y;
    rtekubi1_z = rtekubi_z;

    if (xangle < -95.0) [
        ex_xangle = -95.0;
     ] else if (15.0 < xangle) {
        ex_xangle = 15.0;
```



```
    ] else [
        ex_xangle = xangle;
    ]

    xangle = -30.0;

    /* 胸との相対の左　蛤孤　硫鹽哨泪肇釜奪　垢鮑鄒 */
    ltekubi[0] = ltekubi_m[3][0] - mune_m[3][0];
    ltekubi[1] = ltekubi_m[3][1] - mune_m[3][1];
    ltekubi[2] = ltekubi_m[3][2] - mune_m[3][2];

    /*  手首の座標を胸の回転だけ戻す  */
/*
    coord_calc(ltekubi, ltekubi, mune_rm);
*/
    coord_calc(ltekubi, ltekubi, mune_rm2);

    GetKataMatrix(1, ltekubi, xangle, hm->ske_real.kata_len,
                  hm->ske_real.ninoude_len, hm->ske_real.ude_len,
                  hm->mat1.lkata, hm->mat1.lhiji, lhiji_pnt);

    /* 肩のＸ軸回転成分とＹＺ軸回転成分の取り出し */
    GetKataRotMatrix(hm->mat1.lkata, &lkata_a, lkata1_m, lkata2_m);

    /* 胸との相対の左手首の回転マトリックスを作成 */
    GetTekubiMatrix(1, hm->mat1.mune, hm->mat1.lkata, hm->mat1.lhiji,
                    hm->mat1.ltekubi, &ltekubi_a, &ltekubi_y, &ltekubi_z,
                    ltekubi1_m, ltekubi2_m);

    /*  指の回転マトリックスを作成  */
    mk_yubi_rotm(fang[0], 'x', ryubihosei[0], royayubi_m);
    mk_yubi_rotm(fang[1], 'z', ryubihosei[1], rhitosasiyubi_m);
    mk_yubi_rotm(fang[2], 'z', ryubihosei[2], rnakayubi_m);
    mk_yubi_rotm(fang[3], 'z', ryubihosei[3], rkusuriyubi_m);
    mk_yubi_rotm(fang[4], 'z', ryubihosei[4], rkoyubi_m);
    mk_yubi_rotm(fang[5], 'x', lyubihosei[0], loyayubi_m);
    mk_yubi_rotm(fang[6], 'z', lyubihosei[1], lhitosasiyubi_m);
    mk_yubi_rotm(fang[7], 'z', lyubihosei[2], lnakayubi_m);
    mk_yubi_rotm(fang[8], 'z', lyubihosei[3], lkusuriyubi_m);
    mk_yubi_rotm(fang[9], 'z', lyubihosei[4], lkoyubi_m);

    popmatrix();

    /* 下半身の関節マトリックスを求める */
        hm->mat1.mune[3][0] = 0.0;
        hm->mat1.mune[3][1] = 0.0;
        hm->mat1.mune[3][2] = 0.0;
    cp_mat(hm->mat[DOU], hm->mat1.mune);

    CalcKahansin(hm, trd, sisei);

    CalcKansetuMatrix(hm, KATA_D_R, hm->mat1.rkata);
    CalcKansetuMatrix(hm, KATA_N_R, hm->mat1.rkata);
    CalcKansetuMatrix(hm, HIJI_R, hm->mat1.rhiji);
    hm->rtekubi_a = rtekubi_a;
    CalcKansetuMatrix(hm, UDE_R, rtekubi1_m);
#ifdef CYBERGLOVE
    c_loadmatrix(G_idmat);
    c_rot(-(float)dg[14], 'z');
    c_rot((float)dg[15], 'y');
    c_getmatrix(rtekubi2_m);
#endif
    CalcKansetuMatrix(hm, TE_R, rtekubi2_m);
    CalcKansetuMatrix(hm, KATA_D_L, hm->mat1.lkata);
    CalcKansetuMatrix(hm, KATA_N_L, hm->mat1.lkata);
    CalcKansetuMatrix(hm, HIJI_L, hm->mat1.lhiji);
```

```
    hm->ltekubi_a = ltekubi_a;
    CalcKansetuMatrix(hm, UDE_L, ltekubi1_m);
#ifdef CYBERGLOVE
    c_loadmatrix(G_idmat);
    c_rot((float)dg[30], 'z');
    c_rot(-(float)dg[31], 'y');
    c_getmatrix(ltekubi2_m);
#endif
    CalcKansetuMatrix(hm, TE_L, ltekubi2_m);
    CalcKansetuMatrix(hm, KUBI, hm->mat1.atama);
    CalcKansetuMatrix(hm, OYAYUBI_R_1, royayubi_m[0]);
    CalcKansetuMatrix(hm, OYAYUBI_R_2, royayubi_m[1]);
    CalcKansetuMatrix(hm, OYAYUBI_R_3, royayubi_m[2]);
    CalcKansetuMatrix(hm, HITOSASIYUBI_R_1, rhitosasiyubi_m[0]);
    CalcKansetuMatrix(hm, HITOSASIYUBI_R_2, rhitosasiyubi_m[1]);
    CalcKansetuMatrix(hm, HITOSASIYUBI_R_3, rhitosasiyubi_m[2]);
    CalcKansetuMatrix(hm, NAKAYUBI_R_1, rnakayubi_m[0]);
    CalcKansetuMatrix(hm, NAKAYUBI_R_2, rnakayubi_m[1]);
    CalcKansetuMatrix(hm, NAKAYUBI_R_3, rnakayubi_m[2]);
    CalcKansetuMatrix(hm, KUSURIYUBI_R_1, rkusuriyubi_m[0]);
    CalcKansetuMatrix(hm, KUSURIYUBI_R_2, rkusuriyubi_m[1]);
    CalcKansetuMatrix(hm, KUSURIYUBI_R_3, rkusuriyubi_m[2]);
    CalcKansetuMatrix(hm, KOYUBI_R_1, rkoyubi_m[0]);
    CalcKansetuMatrix(hm, KOYUBI_R_2, rkoyubi_m[1]);
    CalcKansetuMatrix(hm, KOYUBI_R_3, rkoyubi_m[2]);
    CalcKansetuMatrix(hm, OYAYUBI_L_1, loyayubi_m[0]);
    CalcKansetuMatrix(hm, OYAYUBI_L_2, loyayubi_m[1]);
    CalcKansetuMatrix(hm, OYAYUBI_L_3, loyayubi_m[2]);
    CalcKansetuMatrix(hm, HITOSASIYUBI_L_1, lhitosasiyubi_m[0]);
    CalcKansetuMatrix(hm, HITOSASIYUBI_L_2, lhitosasiyubi_m[1]);
    CalcKansetuMatrix(hm, HITOSASIYUBI_L_3, lhitosasiyubi_m[2]);
    CalcKansetuMatrix(hm, NAKAYUBI_L_1, lnakayubi_m[0]);
    CalcKansetuMatrix(hm, NAKAYUBI_L_2, lnakayubi_m[1]);
    CalcKansetuMatrix(hm, NAKAYUBI_L_3, lnakayubi_m[2]);
    CalcKansetuMatrix(hm, KUSURIYUBI_L_1, lkusuriyubi_m[0]);
    CalcKansetuMatrix(hm, KUSURIYUBI_L_2, lkusuriyubi_m[1]);
    CalcKansetuMatrix(hm, KUSURIYUBI_L_3, lkusuriyubi_m[2]);
    CalcKansetuMatrix(hm, KOYUBI_L_1, lkoyubi_m[0]);
    CalcKansetuMatrix(hm, KOYUBI_L_2, lkoyubi_m[1]);
    CalcKansetuMatrix(hm, KOYUBI_L_3, lkoyubi_m[2]);
/************** KAHANSIN ****************/

    /* 回転マトリックスに基づた形状の変形 */
    if (kahansin == 2) {
        Henkei3DHand(hm);
    } else {
        Henkei3D(hm);
    }
}

/*
 * 関数名 : SetKubiKaiten(JINBUTU *hm, float *trd_a, Matrix atama_mat,
 *                        float *trd_m, Matrix mune_mat, int sisei)
 *
 * 引数   : hm        : 人物モデルポインター
 *          trd_m     : 頭の磁気センサーデータ
 *          atama_mat : 頭の回転マトリックス
 *          trd_a     : 胸の磁気センサーデータ
 *          mune_mat  : 胸の回転マトリックス
 *          sisei     :
 *
 * 戻り値 : なし
 *
 * 機能   : 胸と頭の回転マトリックスを求める
 *
 * 履歴   : 1995年1月19日 ; 作成 ; 広瀬＼欽
```



```
 */
void
SetKubiKaiten(JINBUTU *hm, float *trd_a, Matrix atama_mat, float *trd_m,
              Matrix mune_mat, int sisei)
[
    Matrix      mat, wm;
    float       kubic[3];               /* 首の回転中心座標 */
    float       mune[6];
    float       xyz[3], len;
    float       sx, sz;
    float       x1, x2, z1, z2;
    float       y, z, yy, zz;
    int         i;

printf("trd_a:%f,%f,%f,%f,%f,%f\n",trd_a[0],trd_a[1],trd_a[2],trd_a[3],trd_a[4],trd_a
[5]);
printf("trd_m:%f,%f,%f,%f,%f,%f\n",trd_m[0],trd_m[1],trd_m[2],trd_m[3],trd_m[4],trd_m
[5]);
    /* 頭の回転マトリックスを求める */
    c_loadmatrix(G_idmat);
    c_rot(trd_a[3], 'z');
    c_rot(trd_a[5], 'y');
    c_rot(trd_a[4], 'x');               /* 現頭センサのマトリックス */
    c_multmatrix(hm->atama_calib);
    c_getmatrix(atama_mat);

    /* 胸の回転マトリックスを求める */
        /* 首の回転中心を求める */
        c_loadmatrix(G_idmat);
        c_translate(trd_a[0], trd_a[1], trd_a[2]);
        c_multmatrix(hm->atama_itm);
        c_multmatrix(atama_mat);
        c_translate(-ATAMA_TORITUKE_X, -ATAMA_TORITUKE_Y, -ATAMA_TORITUKE_Z);
c_getmatrix(wm);
printf("pos[x,y,z] : %f, %f, %f \n", wm[3][0], wm[3][1], wm[3][2]);
        c_translate(0.0, -hm->ske_real.atama_len, 0.0);
        c_getmatrix(mat);
        kubic[0] = mat[3][0];
        kubic[1] = mat[3][1];
        kubic[2] = mat[3][2];
printf("kubi chusin:%f,%f,%f\n",kubic[0],kubic[1],kubic[2]);

        /* check standing (check kubi position) */
        if (kubic[1] > hm->ske_real.kubi_len+3.0) [
            c_translate(0.0, -(hm->ske_real.kubi_len+hm->ske_real.dou_len), 0.0);
            c_getmatrix(mat);
            hm->kosix = mat[3][0];
            hm->kosiy = mat[3][1];
            hm->kosiz = mat[3][2];
        } else if (fsqrt(kubic[0]*kubic[0] + (kubic[1]-hm->ske_real.kubi_len)*
                (kubic[1]-hm->ske_real.kubi_len) + kubic[2]*kubic[2]) < 3.0) [
            hm->kosix = 0.0;
            hm->kosiy = -hm->ske_real.dou_len;
            hm->kosiz = 0.0;
        }

        /* 胸の回転中心を求める */
printf("(SetKubiKaiten)kosix,kosiy,kosiz:%f,%f,%f\n",hm->kosix,hm->kosiy,hm->kosiz);
#if 0
        /* check kosi position */
        y = -(hm->ske_real.dou_len+hm->ske_real.sune_len);
        z = hm->ske_real.momo_len;
        yy = y - hm->kosiy;
        zz = z - hm->kosiz;
        len = fsqrt(yy*yy + zz*zz);
        if (len > hm->ske_real.momo_len+hm->ske_real.sune_len) [
```

```
            GetKosiPosition(kubic[2], kubic[1], z, y,
                            hm->ske_real.kubi_len+hm->ske_real.dou_len,
                            hm->ske_real.momo_len+hm->ske_real.sune_len,
                            &(hm->kosiz), &(hm->kosiy));
        ]
#endif

        xyz[0] = hm->kosix-kubic[0];
        xyz[1] = hm->kosiy-kubic[1];
        xyz[2] = hm->kosiz-kubic[2];
printf("xyz:%f,%f,%f\n",xyz[0],xyz[1],xyz[2]);
        len = fsqrt(xyz[0]*xyz[0] + xyz[1]*xyz[1] + xyz[2]*xyz[2]);
        mune[0] = kubic[0] + hm->ske_real.kubi_len/len*xyz[0];
        mune[1] = kubic[1] + hm->ske_real.kubi_len/len*xyz[1];
        mune[2] = kubic[2] + hm->ske_real.kubi_len/len*xyz[2];

        /* 胸の回転角度を求める */
        for (i = 0; i < 3; i++) {
            xyz[i] = kubic[i] - mune[i];
            if (fabs(xyz[i]) < EPS) {
                xyz[i] = 0.0;
            ]
        ]
printf("xyz:%f,%f,%f\n",xyz[0],xyz[1],xyz[2]);
        len = fsqrt(xyz[0]*xyz[0] + xyz[1]*xyz[1] + xyz[2]*xyz[2]);
        sx = xyz[2]/len;
        x1 = fasin(sx);
        if (x1 < 0.0) [
            x2 = -3.1415926 - x1;
        ] else [
            x2 = 3.1415926 - x1;
        ]

        for (i = 0; i < 2; i++) {
            sz = -xyz[0]/len/fcos(x1);
            z1 = fasin(sz);
            if (z1 < 0.0) [
                z2 = -3.1415926 - z1;
            ] else [
                z2 = 3.1415926 - z1;
            ]
            if (fabs(len*fcos(x1)*fcos(z1) - xyz[1]) < EPS) {
                break;
            ] else if (fabs(len*fcos(x1)*fcos(z2) - xyz[1]) < EPS) {
                z1 = z2;
                break;
            ]
            x1 = x2;
        }
        mune[3] = r2d(z1);
        mune[4] = r2d(x1);
        mune[5] = 0.0;
printf("mune[3][4][5]:%f,%f,%f\n",mune[3],mune[4],mune[5]);

#ifndef HIROSE
        if (G_four) {
            c_loadmatrix(G_idmat);
            c_translate(trd_m[0], trd_m[1], trd_m[2]);
            c_multmatrix(hm->mune_itm);
            c_getmatrix(mune_mat);
            mune[0] = mune_mat[3][0];
            mune[1] = mune_mat[3][1];
            mune[2] = mune_mat[3][2];
            c_loadmatrix(G_idmat);
            c_rot(trd_m[3], 'z');
            c_rot(trd_m[5], 'y');
```



```
            c_rot(trd_m[4], 'x');
            c_multmatrix(hm->mune_calib);
            c_getmatrix(mune_mat);
            GetKakudo_zyx(mune_mat, &mune[3], &mune[5], &mune[4]);
        ]
#endif
        /* 胸の回転マトリックスを作成 */
        c_loadmatrix(G_idmat);
        c_translate(mune[0], mune[1], mune[2]);
        c_rot(mune[3], 'z');
        c_rot(mune[5], 'y');
        c_rot(mune[4], 'x');
        c_getmatrix(mune_mat);

        c_translate(0.0, -hm->ske_real.dou_len, 0.0);
        c_getmatrix(mat);

/*++++++++++++++++++++++++++++++++++++*/
        if (ffdswitch) {
fprintf(stderr,"<<<mune>>> %f, %f, %f\n",mune[4], mune[5], mune[3]);
xyz[0] = mune[4];
xyz[1] = mune[5];
xyz[2] = mune[3];
            hm->kosispace.xang = mune[4]/6.0;
            rotateSpace(hm, &hm->kosispace, xyz, 1);
            hm->douspace.xang = mune[4]/6.0;
            rotateSpace(hm, &hm->douspace, xyz, 0);
        ]
/*++++++++++++++++++++++++++++++++++++*/

    hm->munex = mune_mat[3][0];
    hm->muney = mune_mat[3][1];
    hm->munez = mune_mat[3][2];
    hm->kosix = mat[3][0];
    hm->kosiy = mat[3][1];
    hm->kosiz = mat[3][2];
printf("mune chusin:%f,%f,%f\n",mune_mat[3][0],mune_mat[3][1],mune_mat[3][2]);
printf("kosi chusin:%f,%f,%f\n",hm->kosix,hm->kosiy,hm->kosiz);
]

/*
 * 関数名 : SetTekubiKaiten(JINBUTU *hm, float *trd_r, Matrix migite_mat,
 *                          float *trd_l, Matrix hidarite_mat)
 *
 * 引数   : hm           : 人物モデルポインター
 *          trd          : 右手の磁気センサーデータ
 *          migite_mat   : 右手首の回転マトリックス
 *          trd          : 左手の磁気センサーデータ
 *          hidarite_mat : 左手首の回転マトリックス
 *
 * 戻り値 : なし
 *
 * 機能   : 手首の回転マトリックスを求める
 *
 * 履歴   : 1994年12月9日 ; 作成 ; 広瀬＼欺
 */
void
SetTekubiKaiten(JINBUTU *hm, float *trd_r, Matrix migite_mat, float *trd_l,
                Matrix hidarite_mat)
{
    Matrix      mat;
    float       mune[3];
    float       xyz[3], len;
    float       sx, sz;
    float       x1, x2, z1, z2;
    int         i;
```

```
printf("trd_r:%f,%f,%f,%f,%f,%f\n",trd_r[0],trd_r[1],trd_r[2],trd_r[3],trd_r[4],trd_r
[5]);
prt_mat(hm->r_te_calib,"hm->r_te_calib");
prt_mat(hm->r_te_itm,"hm->r_te_itm");
    /* 右手首の回転マトリックスを求める */
    c_loadmatrix(G_idmat);
    c_rot(trd_r[3], 'z');
    c_rot(trd_r[5], 'y');
    c_rot(trd_r[4], 'x');                /* 現センサのマトリックス */
    c_multmatrix(hm->r_te_calib);
    c_getmatrix(migite_mat);

    c_loadmatrix(G_idmat);
    c_multmatrix(hm->r_te_itm);
    c_translate(trd_r[0], trd_r[1], trd_r[2]);
    c_multmatrix(migite_mat);
    c_translate(-MIGITE_TORITUKE_X, -MIGITE_TORITUKE_Y, -MIGITE_TORITUKE_Z);
    c_getmatrix(mat);
    migite_mat[3][0] = mat[3][0];
    migite_mat[3][1] = mat[3][1];
    migite_mat[3][2] = mat[3][2];

    /* 左手首の回転マトリックスを求める */
    c_loadmatrix(G_idmat);
    c_rot(trd_l[3], 'z');
    c_rot(trd_l[5], 'y');
    c_rot(trd_l[4], 'x');       /* 現センサのマトリックス */
    c_multmatrix(hm->l_te_calib);
    c_getmatrix(hidarite_mat);

    c_loadmatrix(G_idmat);
    c_multmatrix(hm->l_te_itm);
    c_translate(trd_l[0], trd_l[1], trd_l[2]);
    c_multmatrix(hidarite_mat);
    c_translate(-HIDARITE_TORITUKE_X, -HIDARITE_TORITUKE_Y, -HIDARITE_TORITUKE_Z);
    c_getmatrix(mat);
    hidarite_mat[3][0] = mat[3][0];
    hidarite_mat[3][1] = mat[3][1];
    hidarite_mat[3][2] = mat[3][2];
printf("rtekubi chusin:%f,%f,%f\n",migite_mat[3][0],migite_mat[3][1],migite_mat[3][2]
);
printf("ltekubi chusin:%f,%f,%f\n",hidarite_mat[3][0],hidarite_mat[3][1],hidarite_mat
[3][2
]);
}

void
GetKosiPosition(float x1, float y1, float x2, float y2, float r1, float r2,
                float *px, float *py)
{
    float       l, m, x, y;
    float       k, a, b, c, w;
    float       px1, py1, px2, py2;

printf("x1,y1,x2,y2,r1,r2:%f,%f,%f,%f,%f,%f\n",x1,y1,x2,y2,r1,r2);
    l = fsqrt((x1-x2)*(x1-x2) + (y1-y2)*(y1-y2));
printf("l:%f\n",l);
    m = (r1*r1 - r2*r2 + l*l) / (2.0*l);
printf("m:%f\n",m);
    x = x1 + (x2-x1)*m/l;
    y = y1 + (y2-y1)*m/l;
    k = (x2-x1) / (y1-y2);
printf("x,y,k:%f,%f,%f\n",x,y,k);
    a = 1.0 + k*k;
    b = 2.0*x + 2.0*k*k*x;
```



```
    c = x*x + k*k*x*x - (r1*r1-m*m);
printf("a,b,c:%f,%f,%f\n",a,b,c);
    w = b*b-4.0*a*c;
    if (w < 0.0) {
        return;
    }
    px1 = (b+fsqrt(w)) / (2.0*a);
    py1 = k*px1 + y - k*x;
    px2 = (b-fsqrt(w)) / (2.0*a);
    py2 = k*px2 + y - k*x;
printf("px1,py1,px2,py2:%f,%f,%f,%f\n",px1,py1,px2,py2);

    l = fsqrt((px1-(*px))*(px1-(*px)) + (py1-(*py))*(py1-(*py)));
    m = fsqrt((px2-(*px))*(px2-(*px)) + (py2-(*py))*(py2-(*py)));
    if (l < m) {
        *px = px1;
        *py = py1;
    } else {
        *px = px2;
        *py = py2;
    }
printf("*px,*py:%f,%f\n",*px,*py);
}


make_body_mat(vhm, ag, dg)
    void   *vhm;
    float  *ag;
    int    *dg;
{
    Matrix   mune_m, atama_m, rtekubi_m, ltekubi_m;
    Matrix   mune_rm, mkh_m, mkh_rm;
    Matrix   rtekubil_m, rtekubi2_m;
    Matrix   ltekubil_m, ltekubi2_m;
    int      i;
    float    ang[42], tmp_ang[3], fang[10][4];
    float    tmp_angx[3], tmp_angy[3], tmp_angz[3];
    float    rtekubi_a, rtekubi_y, rtekubi_z;
    float    ltekubi_a, ltekubi_y, ltekubi_z;
    static float  lyubihosei[5][3] = {
            {0, OYAYUBI_HOSEI, OYAYUBI_HOSEI},
            {HITOSASIYUBI_HOSEI, HITOSASIYUBI_HOSEI, HITOSASIYUBI_HOSEI},
            {NAKAYUBI_HOSEI, NAKAYUBI_HOSEI, NAKAYUBI_HOSEI},
            {KUSURIYUBI_HOSEI, KUSURIYUBI_HOSEI, KUSURIYUBI_HOSEI},
            {KOYUBI_HOSEI, KOYUBI_HOSEI, KOYUBI_HOSEI}
    };
    static float  ryubihosei[5][3] = {
            {0, -OYAYUBI_HOSEI, -OYAYUBI_HOSEI},
            {-HITOSASIYUBI_HOSEI, -HITOSASIYUBI_HOSEI, -HITOSASIYUBI_HOSEI},
            {-NAKAYUBI_HOSEI, -NAKAYUBI_HOSEI, -NAKAYUBI_HOSEI},
            {-KUSURIYUBI_HOSEI, -KUSURIYUBI_HOSEI, -KUSURIYUBI_HOSEI},
            {-KOYUBI_HOSEI, -KOYUBI_HOSEI, -KOYUBI_HOSEI}
    };
    JINBUTU  *hm;

    hm = (JINBUTU *)vhm;

    /* 関節角度データを作業領域に格納する */
    for(i = 0; i < 42; i++) {
        ang[i] = ag[i];
    }
    /* ang[3] = -ang[3]; */

    /* データグローブのデータを作業領域に格納する */
    SetTrkDgData(dg, fang);
```

```
    pushmatrix();

    /* 胸の回転マトリックスを作成する */
    c_loadmatrix(G_idmat);
    c_rot(ang[5], 'z');
    c_rot(ang[4], 'y');
    c_rot(ang[3], 'x');
/*
    c_rot(-ang[11], 'z');
    c_rot(-ang[10], 'y');
    c_rot(-ang[9],  'x');
*/
    c_getmatrix(mune_m);

    c_loadmatrix(G_idmat);
    c_multmatrix(hm->mune_itm);
    c_translate(ang[0], ang[1], ang[2]);
    c_multmatrix(mune_m);
    c_getmatrix(mune_m);

    /*++++++++++++++++++++++++++++++++++*/
    if (ffdswitch) [
        tmp_ang[0] = -ang[3] + ang[9];
        tmp_ang[1] =  ang[4] - ang[10];
        tmp_ang[2] =  ang[5] - ang[11];
/*
        tmp_ang[0] = -ang[3];
        tmp_ang[1] =  ang[4];
        tmp_ang[2] =  ang[5];
*/
        tmp_angx[0] = tmp_ang[0] * 0.167;
        tmp_angx[1] = tmp_ang[0] * 0.167;
        tmp_angx[2] = tmp_ang[0] * 0.167;
        tmp_angy[0] = tmp_ang[1] * 0.250;
        tmp_angy[1] = tmp_ang[1] * 0.250;
        tmp_angy[2] = tmp_ang[1] * 0.200;
        tmp_angz[0] = tmp_ang[2] * 0.250;
        tmp_angz[1] = tmp_ang[2] * 0.200;
        tmp_angz[2] = tmp_ang[2] * 0.200;

        rotateSpace3(hm, &hm->kosispace, &tmp_angx[0],
                     &tmp_angy[0], &tmp_angz[0], 2);
/*
        rotateSpace2(hm, &hm->kosispace, &tmp_angx[0],
                     &tmp_angy[0], &tmp_angz[0], 1);
        rotateSpace(hm, &hm->kosispace, &tmp_ang[0], 1);
*/
        tmp_ang[0] = -ang[3] + ang[9];
        tmp_ang[1] =  ang[4] - ang[10];
        tmp_ang[2] =  ang[5] - ang[11];
/*
        tmp_ang[0] = -ang[3];
        tmp_ang[1] =  ang[4];
        tmp_ang[2] =  ang[5];
*/
        tmp_angx[0] = tmp_ang[0] * 0.167;
        tmp_angx[1] = tmp_ang[0] * 0.166;
        tmp_angx[2] = tmp_ang[0] * 0.166;
        tmp_angy[0] = tmp_ang[1] * 0.150;
        tmp_angy[1] = tmp_ang[1] * 0.075;
        tmp_angy[2] = tmp_ang[1] * 0.075;
        tmp_angz[0] = tmp_ang[2] * 0.150;
        tmp_angz[1] = tmp_ang[2] * 0.100;
        tmp_angz[2] = tmp_ang[2] * 0.100;

        rotateSpace3(hm, &hm->douspace, &tmp_angx[0],
```



```
                     &tmp_angy[0], &tmp_angz[0], 0);
/*
        rotateSpace2(hm, &hm->douspace, &tmp_angx[0],
                     &tmp_angy[0], &tmp_angz[0], 0);
        rotateSpace(hm, &hm->douspace, &tmp_ang[0], 0);
*/
    ]
    /*++++++++++++++++++++++++++++++++++*/

    /* 頭の回転マトリックスを作成する */
    c_loadmatrix(G_idmat);
    c_rot(ang[8], 'z');
    c_rot(ang[7], 'y');
    c_rot(ang[6], 'x');
    c_getmatrix(atama_m);

    /* 右手首の回転マトリックスを作成する */
    c_loadmatrix(G_idmat);
    c_rot(ang[17], 'z');
    c_rot(ang[16], 'y');
    c_rot(ang[15], 'x');
    c_getmatrix(rtekubi_m);

    /* 左手首の回転マトリックスを作成する */
    c_loadmatrix(G_idmat);
    c_rot(ang[23], 'z');
    c_rot(ang[22], 'y');
    c_rot(ang[21], 'x');
    c_getmatrix(ltekubi_m);

    /* 表示用マトリックスを作成する */
    world_disp_mat(mune_m, hm->matl.mune);
    world_disp_mat(atama_m, hm->matl.atama);
    world_disp_mat(rtekubi_m, hm->matl.rtekubi);
    world_disp_mat(ltekubi_m, hm->matl.ltekubi);

    /* 胸の逆回転マトリックスを作成する */
    rev_mat(hm->matl.mune, mune_rm);

    /* 胸と相対の頭の回転マトリックスを作成する */
    c_loadmatrix(hm->matl.atama);
    c_rot(180.0, 'y');
    c_getmatrix(hm->matl.atama);

    /* 右肩の回転マトリックスを作成する */
    c_loadmatrix(G_idmat);
    c_rot(ang[14], 'z');
    c_rot(ang[13], 'y');
    c_rot(ang[12], 'x');
    c_getmatrix(hm->matl.rkata);

    /* 右肘の回転マトリックスを作成する */
    c_loadmatrix(G_idmat);
    c_rot(ang[30], 'y');
    c_getmatrix(hm->matl.rhiji);

    /* 左肩の回転マトリックスを作成する */
    c_loadmatrix(G_idmat);
    c_rot(ang[20], 'z');
    c_rot(ang[19], 'y');
    c_rot(ang[18], 'x');
    c_getmatrix(hm->matl.lkata);

    /* 左肘の回転マトリックスを作成する */
    c_loadmatrix(G_idmat);
    c_rot(ang[31], 'y');
```

```
    c_getmatrix(hm->mat1.lhiji);

    /* 胸と相対の右手首の回転マトリックスを作成する */
/*
    c_loadmatrix(hm->mat1.mune);
    c_multmatrix(hm->mat1.rkata);
    c_multmatrix(hm->mat1.rhiji);
    c_getmatrix(mkh_m);
    rev_mat(mkh_m, mkh_rm);

    c_loadmatrix(mkh_rm);
    c_multmatrix(hm->mat1.rtekubi);
    c_getmatrix(hm->mat1.rtekubi);
*/

    /* 右手首のＸ軸回転を取り出す */
    GetTekubiX(0, rtekubi_m, &rtekubi_a, rtekubi1_m);
/*
    GetTekubiX(0, hm->mat1.rtekubi, &rtekubi_a, rtekubi1_m);
*/

    /* 右手首のＹＺ軸回転を取り出す */
    GetTekubiYZ(0, rtekubi_m, &rtekubi_y, &rtekubi_z, rtekubi2_m);
/*
    GetTekubiYZ(0, hm->mat1.rtekubi, &rtekubi_y, &rtekubi_z, rtekubi2_m);
*/

    /* 胸と相対の左手首の回転マトリックスを作成する */
/*
    c_loadmatrix(hm->mat1.mune);
    c_multmatrix(hm->mat1.lkata);
    c_multmatrix(hm->mat1.lhiji);
    c_getmatrix(mkh_m);
    rev_mat(mkh_m, mkh_rm);

    c_loadmatrix(mkh_rm);
    c_multmatrix(hm->mat1.ltekubi);
    c_getmatrix(hm->mat1.ltekubi);
*/

    /* 左手首のＸ軸回転を取り出す */
    GetTekubiX(1, ltekubi_m, &ltekubi_a, ltekubi1_m);
/*
    GetTekubiX(1, hm->mat1.ltekubi, &ltekubi_a, ltekubi1_m);
*/

    /* 左手首のＹＺ軸回転を取り出す */
    GetTekubiYZ(1, ltekubi_m, &ltekubi_y, &ltekubi_z, ltekubi2_m);
/*
    GetTekubiYZ(1, hm->mat1.ltekubi, &ltekubi_y, &ltekubi_z, ltekubi2_m);
*/

    /* 指の回転マトリックスを作成する */
    mk_yubi_rotm(fang[0], 'x', ryubihosei[0], royayubi_m);
    mk_yubi_rotm(fang[1], 'z', ryubihosei[1], rhitosasiyubi_m);
    mk_yubi_rotm(fang[2], 'z', ryubihosei[2], rnakayubi_m);
    mk_yubi_rotm(fang[3], 'z', ryubihosei[3], rkusuriyubi_m);
    mk_yubi_rotm(fang[4], 'z', ryubihosei[4], rkoyubi_m);
    mk_yubi_rotm(fang[5], 'x', lyubihosei[0], loyayubi_m);
    mk_yubi_rotm(fang[6], 'z', lyubihosei[1], lhitosasiyubi_m);
    mk_yubi_rotm(fang[7], 'z', lyubihosei[2], lnakayubi_m);
    mk_yubi_rotm(fang[8], 'z', lyubihosei[3], lkusuriyubi_m);
    mk_yubi_rotm(fang[9], 'z', lyubihosei[4], lkoyubi_m);

    popmatrix();
```



```
    /* 腰の回転マトリックスを作成する */
    c_loadmatrix(G_idmat);
    c_translate(hm->center[KOSI][0], hm->center[KOSI][1],
                hm->center[KOSI][2]);
    c_rot(ang[11], 'z');
    c_rot(ang[10], 'y');
    c_rot(ang[9],  'x');
/*  c_rot(ang[5] + ang[11], 'z');
    c_rot(ang[4] + ang[10], 'y');
    c_rot(ang[3] + ang[9],  'x');  */
    c_translate(-hm->center[KOSI][0], -hm->center[KOSI][1],
                -hm->center[KOSI][2]);
    c_getmatrix(hm->mat[KOSI]);

    /* 右股関節の回転マトリックスを作成する */
    c_loadmatrix(G_idmat);
    c_translate(hm->center[SIRI_R][0], hm->center[SIRI_R][1],
                hm->center[SIRI_R][2]);
    c_rot(ang[26], 'z');
    c_rot(ang[25], 'y');
    c_rot(ang[24], 'x');
    c_translate(-hm->center[SIRI_R][0], -hm->center[SIRI_R][1],
                -hm->center[SIRI_R][2]);
    c_getmatrix(hm->mat[SIRI_R]);

    /* 左股関節の回転マトリックスを作成する */
    c_loadmatrix(G_idmat);
    c_translate(hm->center[SIRI_L][0], hm->center[SIRI_L][1],
                hm->center[SIRI_L][2]);
    c_rot(ang[29], 'z');
    c_rot(ang[28], 'y');
    c_rot(ang[27], 'x');
    c_translate(-hm->center[SIRI_L][0], -hm->center[SIRI_L][1],
                -hm->center[SIRI_L][2]);
    c_getmatrix(hm->mat[SIRI_L]);

    /* 右膝の回転マトリックスを作成する */
    c_loadmatrix(G_idmat);
    c_translate(hm->center[HIZA_R][0], hm->center[HIZA_R][1],
                hm->center[HIZA_R][2]);
    c_rot(ang[32], 'x');
    c_translate(-hm->center[HIZA_R][0], -hm->center[HIZA_R][1],
                -hm->center[HIZA_R][2]);
    c_getmatrix(hm->mat[HIZA_R]);

    /* 左膝の回転マトリックスを作成する */
    c_loadmatrix(G_idmat);
    c_translate(hm->center[HIZA_L][0], hm->center[HIZA_L][1],
                hm->center[HIZA_L][2]);
    c_rot(ang[33], 'x');
    c_translate(-hm->center[HIZA_L][0], -hm->center[HIZA_L][1],
                -hm->center[HIZA_L][2]);
    c_getmatrix(hm->mat[HIZA_L]);

    /* 右足首の回転マトリックスを作成する */
    c_loadmatrix(G_idmat);
    c_translate(hm->center[KUTU_R][0], hm->center[KUTU_R][1],
                hm->center[KUTU_R][2]);
    c_rot(ang[38], 'z');
    c_rot(ang[37], 'y');
    c_rot(ang[36], 'x');
    c_translate(-hm->center[KUTU_R][0], -hm->center[KUTU_R][1],
                -hm->center[KUTU_R][2]);
    c_getmatrix(hm->mat[KUTU_R]);

    /* 左足首の回転マトリックスを作成する */
```

```
    c_loadmatrix(G_idmat);
    c_translate(hm->center[KUTU_L][0], hm->center[KUTU_L][1],
                hm->center[KUTU_L][2]);
    c_rot(ang[41], 'z');
    c_rot(ang[40], 'y');
    c_rot(ang[39], 'x');
    c_translate(-hm->center[KUTU_L][0], -hm->center[KUTU_L][1],
                -hm->center[KUTU_L][2]);
    c_getmatrix(hm->mat[KUTU_L]);

    /* 右上肢パーツの表示用回転マトリックスを作成する */
    CalcKansetuMatrix(hm, KATA_D_R, hm->mat1.rkata);
    CalcKansetuMatrix(hm, KATA_N_R, hm->mat1.rkata);
    CalcKansetuMatrix(hm, HIJI_R, hm->mat1.rhiji);
    hm->rtekubi_a = rtekubi_a;
    CalcKansetuMatrix(hm, UDE_R, rtekubi1_m);
    CalcKansetuMatrix(hm, TE_R, rtekubi2_m);

    /* 左上肢パーツの表示用回転マトリックスを作成する */
    CalcKansetuMatrix(hm, KATA_D_L, hm->mat1.lkata);
    CalcKansetuMatrix(hm, KATA_N_L, hm->mat1.lkata);
    CalcKansetuMatrix(hm, HIJI_L, hm->mat1.lhiji);
    hm->ltekubi_a = ltekubi_a;
    CalcKansetuMatrix(hm, UDE_L, ltekubi1_m);
    CalcKansetuMatrix(hm, TE_L, ltekubi2_m);

    /* 首の表示用回転マトリックスを作成する */
    CalcKansetuMatrix(hm, KUBI, hm->mat1.atama);

    /* 指の表示用回転マトリックスを作成する */
    CalcKansetuMatrix(hm, OYAYUBI_R_1, royayubi_m[0]);
    CalcKansetuMatrix(hm, OYAYUBI_R_2, royayubi_m[1]);
    CalcKansetuMatrix(hm, OYAYUBI_R_3, royayubi_m[2]);
    CalcKansetuMatrix(hm, HITOSASIYUBI_R_1, rhitosasiyubi_m[0]);
    CalcKansetuMatrix(hm, HITOSASIYUBI_R_2, rhitosasiyubi_m[1]);
    CalcKansetuMatrix(hm, HITOSASIYUBI_R_3, rhitosasiyubi_m[2]);
    CalcKansetuMatrix(hm, NAKAYUBI_R_1, rnakayubi_m[0]);
    CalcKansetuMatrix(hm, NAKAYUBI_R_2, rnakayubi_m[1]);
    CalcKansetuMatrix(hm, NAKAYUBI_R_3, rnakayubi_m[2]);
    CalcKansetuMatrix(hm, KUSURIYUBI_R_1, rkusuriyubi_m[0]);
    CalcKansetuMatrix(hm, KUSURIYUBI_R_2, rkusuriyubi_m[1]);
    CalcKansetuMatrix(hm, KUSURIYUBI_R_3, rkusuriyubi_m[2]);
    CalcKansetuMatrix(hm, KOYUBI_R_1, rkoyubi_m[0]);
    CalcKansetuMatrix(hm, KOYUBI_R_2, rkoyubi_m[1]);
    CalcKansetuMatrix(hm, KOYUBI_R_3, rkoyubi_m[2]);
    CalcKansetuMatrix(hm, OYAYUBI_L_1, loyayubi_m[0]);
    CalcKansetuMatrix(hm, OYAYUBI_L_2, loyayubi_m[1]);
    CalcKansetuMatrix(hm, OYAYUBI_L_3, loyayubi_m[2]);
    CalcKansetuMatrix(hm, HITOSASIYUBI_L_1, lhitosasiyubi_m[0]);
    CalcKansetuMatrix(hm, HITOSASIYUBI_L_2, lhitosasiyubi_m[1]);
    CalcKansetuMatrix(hm, HITOSASIYUBI_L_3, lhitosasiyubi_m[2]);
    CalcKansetuMatrix(hm, NAKAYUBI_L_1, lnakayubi_m[0]);
    CalcKansetuMatrix(hm, NAKAYUBI_L_2, lnakayubi_m[1]);
    CalcKansetuMatrix(hm, NAKAYUBI_L_3, lnakayubi_m[2]);
    CalcKansetuMatrix(hm, KUSURIYUBI_L_1, lkusuriyubi_m[0]);
    CalcKansetuMatrix(hm, KUSURIYUBI_L_2, lkusuriyubi_m[1]);
    CalcKansetuMatrix(hm, KUSURIYUBI_L_3, lkusuriyubi_m[2]);
    CalcKansetuMatrix(hm, KOYUBI_L_1, lkoyubi_m[0]);
    CalcKansetuMatrix(hm, KOYUBI_L_2, lkoyubi_m[1]);
    CalcKansetuMatrix(hm, KOYUBI_L_3, lkoyubi_m[2]);

    /* 回転マトリックスに基づいて形状を変形する */
    Henkei3D(hm);
}

make_body_mat2(vhm, ag, dg)
```



```
    void   *vhm;
    float  *ag;
    int    *dg;
[
    Matrix   mune_m, atama_m, rtekubi_m, ltekubi_m;
    Matrix   mune_rm, mkh_m, mkh_rm, mat;
    Matrix   rtekubi1_m, rtekubi2_m;
    Matrix   ltekubi1_m, ltekubi2_m;
    int      i;
    float    ang[42], tmp_ang[3], fang[10][4];
    float    tmp_angx[3], tmp_angy[3], tmp_angz[3];
    float    rtekubi_a, rtekubi_y, rtekubi_z;
    float    ltekubi_a, ltekubi_y, ltekubi_z;
    static float  lyubihosei[5][3] = [
            [0, OYAYUBI_HOSEI, OYAYUBI_HOSEI],
            [HITOSASIYUBI_HOSEI, HITOSASIYUBI_HOSEI, HITOSASIYUBI_HOSEI],
            [NAKAYUBI_HOSEI, NAKAYUBI_HOSEI, NAKAYUBI_HOSEI],
            [KUSURIYUBI_HOSEI, KUSURIYUBI_HOSEI, KUSURIYUBI_HOSEI],
            [KOYUBI_HOSEI, KOYUBI_HOSEI, KOYUBI_HOSEI]
    ];
    static float  ryubihosei[5][3] = [
            [0, -OYAYUBI_HOSEI, -OYAYUBI_HOSEI],
            [-HITOSASIYUBI_HOSEI, -HITOSASIYUBI_HOSEI, -HITOSASIYUBI_HOSEI],
            [-NAKAYUBI_HOSEI, -NAKAYUBI_HOSEI, -NAKAYUBI_HOSEI],
            [-KUSURIYUBI_HOSEI, -KUSURIYUBI_HOSEI, -KUSURIYUBI_HOSEI],
            [-KOYUBI_HOSEI, -KOYUBI_HOSEI, -KOYUBI_HOSEI]
    ];
    JINBUTU  *hm;

printf("\nmake_body_mat2\n");
    hm = (JINBUTU *)vhm;

    /* 関節角度データを作業領域に格納する */
    for(i = 0; i < 42; i++) [
        ang[i] = ag[i];
    ]
    /* ang[3] = -ang[3]; */

    /* データグローブのデータを作業領域に格納する */
    SetTrkDgData(dg, fang);

    pushmatrix();

    /*++++++++++++++++++++++++++++++++++*/
    if (ffdswitch) [
/*
        tmp_ang[0] = -ang[3] + ang[9];
        tmp_ang[1] =  ang[4] - ang[10];
        tmp_ang[2] =  ang[5] - ang[11];
*/
        tmp_ang[0] = ang[9];
        tmp_ang[1] = ang[10];
        tmp_ang[2] = ang[11];
        tmp_angx[0] = tmp_ang[0] / 2;
        tmp_angx[1] = tmp_ang[0] / 3;
        tmp_angx[2] = tmp_ang[0] / 6;
        tmp_angy[0] = tmp_ang[1] / 2;
        tmp_angy[1] = tmp_ang[1] / 3;
        tmp_angy[2] = tmp_ang[1] / 6;
        tmp_angz[0] = tmp_ang[2] / 6;
        tmp_angz[1] = tmp_ang[2] / 3;
        tmp_angz[2] = tmp_ang[2] / 2;

/*
        rotateSpace3(hm, &hm->kosispace, &tmp_angx[0],
                     &tmp_angy[0], &tmp_angz[0], 2);
```

```
        rotateSpace2(hm, &hm->kosispace, &tmp_angx[0],
                     &tmp_angy[0], &tmp_angz[0], 1);
*/
        rotateSpace(hm, &hm->kosispace, &tmp_ang[0], 2); /* kosi_mat */

/*
        tmp_ang[0] = -ang[3] + ang[9];
        tmp_ang[1] =  ang[4] - ang[10];
        tmp_ang[2] =  ang[5] - ang[11];
*/
        tmp_ang[0] = ang[9];
        tmp_ang[1] = ang[10];
        tmp_ang[2] = ang[11];
        tmp_angx[0] = tmp_angx[1] = tmp_angx[2] = 0.0;
        tmp_angy[0] = tmp_angy[1] = tmp_angy[2] = 0.0;
        tmp_angz[0] = tmp_angz[1] = tmp_angz[2] = 0.0;

/*
        rotateSpace3(hm, &hm->douspace, &tmp_angx[0],
                     &tmp_angy[0], &tmp_angz[0], 0);
        rotateSpace2(hm, &hm->douspace, &tmp_angx[0],
                     &tmp_angy[0], &tmp_angz[0], 0);
*/
        rotateSpace(hm, &hm->douspace, &tmp_ang[0], 1); /* mune_mat */
    }
    /*++++++++++++++++++++++++++++++++++*/

    /* 腰の回転マトリックスを作成する */
    c_loadmatrix(G_idmat);
    c_rot(ang[11], 'z');
    c_rot(ang[10], 'y');
    c_rot(ang[9],  'x');
    c_getmatrix(mat);
    rev_mat(mat, hm->mat[KOSI]);
    c_loadmatrix(G_idmat);
    c_translate(hm->center[KOSI][0], hm->center[KOSI][1],
                hm->center[KOSI][2]);
    c_multmatrix(hm->mat[KOSI]);
    c_translate(-hm->center[KOSI][0], -hm->center[KOSI][1],
                -hm->center[KOSI][2]);
    c_getmatrix(hm->mat[KOSI]);

    /* 胸の回転マトリックスを作成する */
    c_loadmatrix(G_idmat);
    c_translate(ang[0], ang[1], ang[2]);
    c_translate(hm->center[DOU][0], hm->center[DOU][1],
                hm->center[DOU][2]);
    c_rot(ang[5], 'z');
    c_rot(ang[4], 'y');
    c_rot(ang[3], 'x');
    c_translate(-hm->center[DOU][0], -hm->center[DOU][1],
                -hm->center[DOU][2]);
    c_translate(hm->center[KOSI][0], hm->center[KOSI][1],
                hm->center[KOSI][2]);
#if 0
    if (!ffdswitch) {
        c_multmatrix(mat);
    ]
#else
    c_multmatrix(mat);
#endif
    c_translate(-hm->center[KOSI][0], -hm->center[KOSI][1],
                -hm->center[KOSI][2]);
    c_getmatrix(hm->mat[DOU]);
    cp_mat(mune_m, hm->mat[DOU]);
```



```
    /* 頭の回転マトリックスを作成する */
    c_loadmatrix(G_idmat);
    c_rot(ang[8], 'z');
    c_rot(ang[7], 'y');
    c_rot(ang[6], 'x');
    c_getmatrix(atama_m);

    /* 右手首の回転マトリックスを作成する */
    c_loadmatrix(G_idmat);
    c_rot(ang[17], 'z');
    c_rot(ang[16], 'y');
    c_rot(ang[15], 'x');
    c_getmatrix(rtekubi_m);

    /* 左手首の回転マトリックスを作成する */
    c_loadmatrix(G_idmat);
    c_rot(ang[23], 'z');
    c_rot(ang[22], 'y');
    c_rot(ang[21], 'x');
    c_getmatrix(ltekubi_m);

    /* 表示用マトリックスを作成する */
    world_disp_mat(mune_m, hm->mat1.mune);
    world_disp_mat(atama_m, hm->mat1.atama);
    world_disp_mat(rtekubi_m, hm->mat1.rtekubi);
    world_disp_mat(ltekubi_m, hm->mat1.ltekubi);

    /* 胸の逆回転マトリックスを作成する */
    rev_mat(hm->mat1.mune, mune_rm);

    /* 胸と相対の頭の回転マトリックスを作成する */
    c_loadmatrix(hm->mat1.atama);
    c_rot(180.0, 'y');
    c_getmatrix(hm->mat1.atama);

    /* 右肩の回転マトリックスを作成する */
    c_loadmatrix(G_idmat);
    c_rot(ang[14], 'z');
    c_rot(ang[13], 'y');
    c_rot(ang[12], 'x');
    c_getmatrix(hm->mat1.rkata);

    /* 右肘の回転マトリックスを作成する */
    c_loadmatrix(G_idmat);
    c_rot(ang[30], 'y');
    c_getmatrix(hm->mat1.rhiji);

    /* 左肩の回転マトリックスを作成する */
    c_loadmatrix(G_idmat);
    c_rot(ang[20], 'z');
    c_rot(ang[19], 'y');
    c_rot(ang[18], 'x');
    c_getmatrix(hm->mat1.lkata);

    /* 左肘の回転マトリックスを作成する */
    c_loadmatrix(G_idmat);
    c_rot(ang[31], 'y');
    c_getmatrix(hm->mat1.lhiji);

    /* 胸と相対の右手首の回転マトリックスを作成する */
/*
    c_loadmatrix(hm->mat1.mune);
    c_multmatrix(hm->mat1.rkata);
    c_multmatrix(hm->mat1.rhiji);
    c_getmatrix(mkh_m);
    rev_mat(mkh_m, mkh_rm);
```

```
    c_loadmatrix(mkh_rm);
    c_multmatrix(hm->mat1.rtekubi);
    c_getmatrix(hm->mat1.rtekubi);
*/

    /* 右手首のＸ軸回転を取り出す */
    GetTekubiX(0, rtekubi_m, &rtekubi_a, rtekubi1_m);
/*
    GetTekubiX(0, hm->mat1.rtekubi, &rtekubi_a, rtekubi1_m);
*/

    /* 右手首のＹＺ軸回転を取り出す */
    GetTekubiYZ(0, rtekubi_m, &rtekubi_y, &rtekubi_z, rtekubi2_m);
/*
    GetTekubiYZ(0, hm->mat1.rtekubi, &rtekubi_y, &rtekubi_z, rtekubi2_m);
*/

    /* 胸と相対の左手首の回転マトリックスを作成する */
/*
    c_loadmatrix(hm->mat1.mune);
    c_multmatrix(hm->mat1.lkata);
    c_multmatrix(hm->mat1.lhiji);
    c_getmatrix(mkh_m);
    rev_mat(mkh_m, mkh_rm);

    c_loadmatrix(mkh_rm);
    c_multmatrix(hm->mat1.ltekubi);
    c_getmatrix(hm->mat1.ltekubi);
*/

    /* 左手首のＸ軸回転を取り出す */
    GetTekubiX(1, ltekubi_m, &ltekubi_a, ltekubi1_m);
/*
    GetTekubiX(1, hm->mat1.ltekubi, &ltekubi_a, ltekubi1_m);
*/

    /* 左手首のＹＺ軸回転を取り出す */
    GetTekubiYZ(1, ltekubi_m, &ltekubi_y, &ltekubi_z, ltekubi2_m);
/*
    GetTekubiYZ(1, hm->mat1.ltekubi, &ltekubi_y, &ltekubi_z, ltekubi2_m);
*/

    /* 指の回転マトリックスを作成する */
    mk_yubi_rotm(fang[0], 'x', ryubihosei[0], royayubi_m);
    mk_yubi_rotm(fang[1], 'z', ryubihosei[1], rhitosasiyubi_m);
    mk_yubi_rotm(fang[2], 'z', ryubihosei[2], rnakayubi_m);
    mk_yubi_rotm(fang[3], 'z', ryubihosei[3], rkusuriyubi_m);
    mk_yubi_rotm(fang[4], 'z', ryubihosei[4], rkoyubi_m);
    mk_yubi_rotm(fang[5], 'x', lyubihosei[0], loyayubi_m);
    mk_yubi_rotm(fang[6], 'z', lyubihosei[1], lhitosasiyubi_m);
    mk_yubi_rotm(fang[7], 'z', lyubihosei[2], lnakayubi_m);
    mk_yubi_rotm(fang[8], 'z', lyubihosei[3], lkusuriyubi_m);
    mk_yubi_rotm(fang[9], 'z', lyubihosei[4], lkoyubi_m);

    popmatrix();

    /* 右股関節の回転マトリックスを作成する */
    c_loadmatrix(G_idmat);
    c_translate(hm->center[SIRI_R][0], hm->center[SIRI_R][1],
                hm->center[SIRI_R][2]);
    c_rot(ang[26], 'z');
    c_rot(ang[25], 'y');
    c_rot(ang[24], 'x');
    c_translate(-hm->center[SIRI_R][0], -hm->center[SIRI_R][1],
                -hm->center[SIRI_R][2]);
```



```
    c_getmatrix(hm->mat[SIRI_R]);

    /* 左股関節の回転マトリックスを作成する */
    c_loadmatrix(G_idmat);
    c_translate(hm->center[SIRI_L][0], hm->center[SIRI_L][1],
                hm->center[SIRI_L][2]);
    c_rot(ang[29], 'z');
    c_rot(ang[28], 'y');
    c_rot(ang[27], 'x');
    c_translate(-hm->center[SIRI_L][0], -hm->center[SIRI_L][1],
                -hm->center[SIRI_L][2]);
    c_getmatrix(hm->mat[SIRI_L]);

    /* 右膝の回転マトリックスを作成する */
    c_loadmatrix(G_idmat);
    c_translate(hm->center[HIZA_R][0], hm->center[HIZA_R][1],
                hm->center[HIZA_R][2]);
    c_rot(ang[32], 'x');
    c_translate(-hm->center[HIZA_R][0], -hm->center[HIZA_R][1],
                -hm->center[HIZA_R][2]);
    c_getmatrix(hm->mat[HIZA_R]);

    /* 左膝の回転マトリックスを作成する */
    c_loadmatrix(G_idmat);
    c_translate(hm->center[HIZA_L][0], hm->center[HIZA_L][1],
                hm->center[HIZA_L][2]);
    c_rot(ang[33], 'x');
    c_translate(-hm->center[HIZA_L][0], -hm->center[HIZA_L][1],
                -hm->center[HIZA_L][2]);
    c_getmatrix(hm->mat[HIZA_L]);

    /* 右足首の回転マトリックスを作成する */
    c_loadmatrix(G_idmat);
    c_translate(hm->center[KUTU_R][0], hm->center[KUTU_R][1],
                hm->center[KUTU_R][2]);
    c_rot(ang[38], 'z');
    c_rot(ang[37], 'y');
    c_rot(ang[36], 'x');
    c_translate(-hm->center[KUTU_R][0], -hm->center[KUTU_R][1],
                -hm->center[KUTU_R][2]);
    c_getmatrix(hm->mat[KUTU_R]);

    /* 左足首の回転マトリックスを作成する */
    c_loadmatrix(G_idmat);
    c_translate(hm->center[KUTU_L][0], hm->center[KUTU_L][1],
                hm->center[KUTU_L][2]);
    c_rot(ang[41], 'z');
    c_rot(ang[40], 'y');
    c_rot(ang[39], 'x');
    c_translate(-hm->center[KUTU_L][0], -hm->center[KUTU_L][1],
                -hm->center[KUTU_L][2]);
    c_getmatrix(hm->mat[KUTU_L]);

    /* 右上肢パーツの表示用回転マトリックスを作成する */
    CalcKansetuMatrix(hm, KATA_D_R, hm->mat1.rkata);
    CalcKansetuMatrix(hm, KATA_N_R, hm->mat1.rkata);
    CalcKansetuMatrix(hm, HIJI_R, hm->mat1.rhiji);
    hm->rtekubi_a = rtekubi_a;
    CalcKansetuMatrix(hm, UDE_R, rtekubi1_m);
    CalcKansetuMatrix(hm, TE_R, rtekubi2_m);

    /* 左上肢パーツの表示用回転マトリックスを作成する */
    CalcKansetuMatrix(hm, KATA_D_L, hm->mat1.lkata);
    CalcKansetuMatrix(hm, KATA_N_L, hm->mat1.lkata);
    CalcKansetuMatrix(hm, HIJI_L, hm->mat1.lhiji);
    hm->ltekubi_a = ltekubi_a;
```

```
    CalcKansetuMatrix(hm, UDE_L, ltekubi1_m);
    CalcKansetuMatrix(hm, TE_L, ltekubi2_m);

    /* 首の表示用回転マトリックスを作成する */
    CalcKansetuMatrix(hm, KUBI, hm->matl.atama);

    /* 指の表示用回転マトリックスを作成する */
    CalcKansetuMatrix(hm, OYAYUBI_R_1, royayubi_m[0]);
    CalcKansetuMatrix(hm, OYAYUBI_R_2, royayubi_m[1]);
    CalcKansetuMatrix(hm, OYAYUBI_R_3, royayubi_m[2]);
    CalcKansetuMatrix(hm, HITOSASIYUBI_R_1, rhitosasiyubi_m[0]);
    CalcKansetuMatrix(hm, HITOSASIYUBI_R_2, rhitosasiyubi_m[1]);
    CalcKansetuMatrix(hm, HITOSASIYUBI_R_3, rhitosasiyubi_m[2]);
    CalcKansetuMatrix(hm, NAKAYUBI_R_1, rnakayubi_m[0]);
    CalcKansetuMatrix(hm, NAKAYUBI_R_2, rnakayubi_m[1]);
    CalcKansetuMatrix(hm, NAKAYUBI_R_3, rnakayubi_m[2]);
    CalcKansetuMatrix(hm, KUSURIYUBI_R_1, rkusuriyubi_m[0]);
    CalcKansetuMatrix(hm, KUSURIYUBI_R_2, rkusuriyubi_m[1]);
    CalcKansetuMatrix(hm, KUSURIYUBI_R_3, rkusuriyubi_m[2]);
    CalcKansetuMatrix(hm, KOYUBI_R_1, rkoyubi_m[0]);
    CalcKansetuMatrix(hm, KOYUBI_R_2, rkoyubi_m[1]);
    CalcKansetuMatrix(hm, KOYUBI_R_3, rkoyubi_m[2]);
    CalcKansetuMatrix(hm, OYAYUBI_L_1, loyayubi_m[0]);
    CalcKansetuMatrix(hm, OYAYUBI_L_2, loyayubi_m[1]);
    CalcKansetuMatrix(hm, OYAYUBI_L_3, loyayubi_m[2]);
    CalcKansetuMatrix(hm, HITOSASIYUBI_L_1, lhitosasiyubi_m[0]);
    CalcKansetuMatrix(hm, HITOSASIYUBI_L_2, lhitosasiyubi_m[1]);
    CalcKansetuMatrix(hm, HITOSASIYUBI_L_3, lhitosasiyubi_m[2]);
    CalcKansetuMatrix(hm, NAKAYUBI_L_1, lnakayubi_m[0]);
    CalcKansetuMatrix(hm, NAKAYUBI_L_2, lnakayubi_m[1]);
    CalcKansetuMatrix(hm, NAKAYUBI_L_3, lnakayubi_m[2]);
    CalcKansetuMatrix(hm, KUSURIYUBI_L_1, lkusuriyubi_m[0]);
    CalcKansetuMatrix(hm, KUSURIYUBI_L_2, lkusuriyubi_m[1]);
    CalcKansetuMatrix(hm, KUSURIYUBI_L_3, lkusuriyubi_m[2]);
    CalcKansetuMatrix(hm, KOYUBI_L_1, lkoyubi_m[0]);
    CalcKansetuMatrix(hm, KOYUBI_L_2, lkoyubi_m[1]);
    CalcKansetuMatrix(hm, KOYUBI_L_3, lkoyubi_m[2]);

    /* 回転マトリックスに基づいて形状を変形する */
    Henkei3D(hm);
}
/* END OF FILE */
```

```
#include        "tool.h"

#define         SAMPLECOUNT     20

void
CalibMune(int hcnt, void *vhm[])
{
    int         wid;
    float       sx, sy, sz;
    JINBUTU     *hm;
    char        *hname = "ciris0";
    int         i;

    hm = (JINBUTU *)vhm[0];

    prefposition(490, 790, 200, 300);
    wid = winopen("Mune_Calibrate");
    doublebuffer();
    gconfig();

    color(BLACK);
    clear();
    color(WHITE);
    cmov2i(10, 10);
    charstr("Sit Down, Please" );
    swapbuffers();

    sleep(5);
    sx = sy = sz = 0.0;
    for (i = 0; i < SAMPLECOUNT; i++) {
        ReadFastrak(hname, G_trd);
        sx += G_trd[MUNE_TRAK_NO][0];
        sy += G_trd[MUNE_TRAK_NO][1];
        sz += G_trd[MUNE_TRAK_NO][2];
    }

    G_trd[MUNE_TRAK_NO][0] = sx/(float)SAMPLECOUNT;
    G_trd[MUNE_TRAK_NO][1] = sy/(float)SAMPLECOUNT;
    G_trd[MUNE_TRAK_NO][2] = sz/(float)SAMPLECOUNT;
    for (i = 0; i < hcnt; i++) {
        c_calib_model(vhm[i], (float *)G_trd);
    }

    color(BLACK);
    clear();
    color(WHITE);
    cmov2i(10, 10);
    charstr("Now Relax, Please");
    swapbuffers();

    sleep(2);
    winclose(wid);
}
```

```
#include "tool.h"
#include "vector.h"

VERTEX contact[3];


VERTEX spcenter;
float r1,sr1;
float ht[2];
#define TSTEP 0.01

float ttime=2,tdir=1;
extern float G_c;

setContact(JINBUTU *hm)
{
  int k;
  float *vs;

  contact[0][0]=(hm->center[HIJI_L][0]+hm->center[TE_L][0])/2;
  contact[0][1]=(hm->center[HIJI_L][1]+hm->center[TE_L][1])/2 + 9.5-G_c*3;
  contact[0][2]=(hm->center[HIJI_L][2]+hm->center[TE_L][2])/2;

  contact[2][0]=(hm->center[HIJI_R][0]+hm->center[TE_R][0])/2;
  contact[2][1]=(hm->center[HIJI_R][1]+hm->center[TE_R][1])/2 + 8.5-G_c*3;
  contact[2][2]=(hm->center[HIJI_R][2]+hm->center[TE_R][2])/2;

  contact[1][0]=(contact[2][0]+contact[0][0])/2;
  contact[1][1]=contact[0][1]+29 -G_c*3;
  contact[1][2]=(contact[2][2]+contact[0][2])/2;

  ht[0]=ht[1]=contact[1][1]+100-G_c*15.0;


  sr1=10;
#if 0
  r1=sr1/sqrt(1-sqrt(THR2));
#endif

  for(k=0, vs=(float *)spv[2]; k<spvc[2]; k++, vs+=3)
    {
      vs[0]*=(2*sr1);
      vs[1]*=(2*sr1);
      vs[2]*=(2*sr1);
     }
}


placeSphere()
{
  int wh;
  float nt;

  wh= (int) ttime;
  nt= ttime- (float)wh;
  ht[0]=ht[1]=contact[1][1]+100-G_c*5.0;
  nt= 1- quartF(nt);

  spcenter[0]=contact[wh][0]+(contact[wh+1][0]-contact[wh][0])*nt;
  spcenter[1]=contact[wh][1]+ (contact[wh+1][1]-contact[wh][1])*nt
            + ht[wh]*nt*(1-nt);
  spcenter[2]=contact[0][2]/**(contact[wh+1][2]-contact[wh][2])*nt*/;

  ttime+= (tdir*TSTEP);

  if (ttime>2)
```



```
    [
      ttime=2;
      tdir= -tdir;
    ]
  if (ttime<0)
    {
      ttime=0;
      tdir= -tdir;
    ]
}



deformSphere(JINBUTU *hm, int a)
[
  int k,j;
  VERTEX dumc,dumc2,ctr,dir,p;
  float addup,addup2,*vs,*va,*vn,*vn2,d2,tmp,far,near,denf,denn,dist,
  den,mval,pdn;

  r1=sr1/sqrt(1-sqrt(THR2));
  if (sphyVal(a,spcenter,dumc,&addup,0))
    pdn=diffFunc2(1,ipb[a].weight*THR*quartF(addup/ipb[a].r1),4.0,1.0);
  else
#if 0
    if (addup>=(sr1+ipb[a].sr1))
#else
    if (addup>=(r1+ipb[a].sr1))
#endif
      {
        return;
      ]
    else
      pdn=1;

  for(k=0, vs=(float *)spv[2],va=(float *)spv[3],
           vn=(float *)spn[2],vn2=(float *)spn[3];
      k<spvc[2];
      k++, vs+=3, va+=3,vn+=3,vn2+=3)
    {
      near=0;
      denn=pdn;

      VECAdd(spcenter,vs,dumc2);

      if (!sphyVal(a,dumc2,dumc,&addup,0))
        {
          VECSet(va,vs);
          VECSet(vn2,vn);
          continue;
        ]

      d2=ipb[a].weight*THR*quartF(addup/ipb[a].r1);

        den=diffFunc2(THR2,d2,4.0,1.0);

      if (fabs(den-THR2)<=EPS)
        [
          VECSet(va,vs);
          VECSet(vn2,vn);
          continue;
        ]

      far=VECLen(vs);
      denf=den;
```

```
      j=0;
      while ((far-near>EPS) && (j<4))
        {
          j++;

          if ((denf-THR2)*(denn-THR2)>0)
            dist=(near+far)/2;
          else
            dist= (near*(denf-THR2)-far*(denn-THR2))/(denf-denn);

          p[0]=spcenter[0] - vn[0]*dist;
          p[1]=spcenter[1] - vn[1]*dist;
          p[2]=spcenter[2] - vn[2]*dist;

          d2=0;
          if (sphyVal(a,p,dumc,&addup,0))
            d2=ipb[a].weight*THR*quartF(addup/ipb[a].r1);

          den=diffFunc2((tmp=quartF(dist/r1)),d2,4.0,1.0);

          if (fabs(mval=(den-THR2))<=EPS)
            break;

          if (mval<0)
            [
              far=dist;
              denf=den;
            ]
          else
            [
              near=dist;
              denn=den;
            ]
        ]
      VECSub(p,spcenter,va);
      VECSub(p,dumc,dumc2);
      VECNorm(dumc2);
      vn2[0]=vn[0]*tmp+d2*dumc2[0];
      vn2[1]=vn[0]*tmp+d2*dumc2[1];
      vn2[2]=vn[0]*tmp+d2*dumc2[2];
      VECNorm(vn2);
    }
]


deformPart(JINBUTU *hm, int a, int b)
[
  int k,j;
  VERTEX dumc,dumc2,ctr,dir,p;
  float addup,addup2,*vs,*va,*wt,*mp,
d2,d3,tmp,far,near,denf,denn,dist,den,mval,pdn;

  r1=sr1/sqrt(1-sqrt(THR2));
  if (!sphyVal(a,spcenter,dumc,&addup,0))
    if (addup>=(sr1+ipb[a].r1))
      return;

  for(k=0, vs=(float *)hm->vs[b],va=(float *)hm->va[b],wt=(float *)hm->ipwt[b]
      ,mp=(float *)hm->map[b]; k<hm->vmax[b];
      k++, vs+=3, va+=3,wt++,mp+=3)
    [
      if (!sphereVal(spcenter,sr1,va,&addup))
        continue;

      VECSub(va,mp,dir);
      dist=far=VECLen(dir);
```



```
      VECNorm(dir);

      denf=diffFunc(THR,quartF(addup/r1),4.0,*wt);;

      if (fabs(denf-THR)<=EPS)
        continue;


      near=0;
      if (sphereVal(spcenter,sr1,mp,&addup))
        denn=diffFunc((*wt)*ipb[a].weight*THR,quartF(addup/r1),4.0,*wt);
      else
        denn=(*wt)*ipb[a].weight*THR;

      VECSet(p,va);

      j=0;
      while ((far-near>EPS) && (fabs(denf-denn)>EPS) && (j<5))
        [
          j++;

          if ((denf-THR)*(denn-THR)>0)
            dist=(near+far)/2;
          else
            dist= (near*(denf-THR)-far*(denn-THR))/(denf-denn);

          p[0]=mp[0] + dir[0]*dist;
          p[1]=mp[1] + dir[1]*dist;
          p[2]=mp[2] + dir[2]*dist;

          d2=0;
          if (sphereVal(spcenter,r1,p,&addup))
            d2=quartF(addup/r1);

          sphyVal(a,p,dumc,&addup,0);
          d3=(*wt)*ipb[a].weight*THR*quartF(addup/ipb[a].r1);

          den=diffFunc(d3,d2,4.0,*wt);

          if (fabs(mval=(den-THR))<=EPS)
            break;

          if (mval<0)
            [
              far=dist;
              denf=den;
            ]
          else
            [
              near=dist;
              denn=den;
            ]
        }
      /*if ((dist>ipb[a].r1) || (dist <0))
        printf("hola %d %f %f, mp %f %f %f, p %f %f %f\n",j,dist,den,
             mp[0],mp[1],mp[2],p[0],p[1],p[2]);*/
      VECSet(va,p);

    }
]



deformSpherePartPlastic(JINBUTU *hm, int a, int b)
[
  int k,j;
```

```
  VERTEX dumc,dumc2,ctr,dir,p;
  float addup,addup2,*vs,*va,*wt,*mp,
d2,d3,tmp,far,near,denf,denn,dist,den,mval,pdn;


  if (!sphVal(a,spcenter,&addup,1))
    if (addup>=(sr1+ipb[a].r1))
      return;

  for(k=0, vs=(float *)hm->vs[b],va=(float *)hm->va[b],wt=(float *)hm->ipwt[b]
      ,mp=(float *)hm->map[b]; k<hm->vmax[b];
      k++, vs+=3, va+=3,wt++,mp+=3)
    {
      if (!sphereVal(spcenter,sr1,va,&addup))
        continue;

      denf=diffFunc(THR,quartF(addup/r1),4.0,*wt);;

      if (fabs(denf-THR)<=EPS)
        continue;

      VECSub(va,mp,dir);
      dist=far=VECLen(dir);
      VECNorm(dir);


      near=0;
      if (sphereVal(spcenter,sr1,mp,&addup))
        denn=diffFunc((*wt)*ipb[a].weight*THR,quartF(addup/r1),4.0,*wt);
      else
        denn=(*wt)*ipb[a].weight*THR;

      VECSet(p,va);

      j=0;
      while ((far-near>EPS) && (fabs(denf-denn)>EPS) && (j<5))
        {
          j++;

          if ((denf-THR)*(denn-THR)>0)
            dist=(near+far)/2;
          else
            dist= (near*(denf-THR)-far*(denn-THR))/(denf-denn);

          p[0]=mp[0] + dir[0]*dist;
          p[1]=mp[1] + dir[1]*dist;
          p[2]=mp[2] + dir[2]*dist;

          d2=0;
          if (sphereVal(spcenter,r1,p,&addup))
            d2=quartF(addup/r1);

          sphVal(a,p,&addup,1);
          d3=(*wt)*ipb[a].weight*THR*quartF(addup/ipb[a].r1);

          den=diffFunc(d3,d2,4.0,*wt);

          if (fabs(mval=(den-THR))<=EPS)
            break;

          if (mval<0)
            [
              far=dist;
              denf=den;
            ]
          else
```



```
            {
              near=dist;
              denn=den;
            ]
        ]
     /* if ((dist>ipb[a].r1) || (dist <0))
        printf("hola %d %f %f, mp %f %f %f, p %f %f %f\n",j,dist,den,
               mp[0],mp[1],mp[2],p[0],p[1],p[2]);*/
      VECSet(va,p);

    }
}


deformSpherePart(JINBUTU *hm, int a, int b)
{
  int k,j;
  VERTEX dumc,dumc2,ctr,dir,p;
  float addup,addup2,*vs,*va,*wt,*mp,
d2,d3,tmp,far,near,denf,denn,dist,den,mval,pdn;

        r1=sr1/sqrt(1-sqrt(THR2));
  if (!sphVal(a,spcenter,&addup,1))
    if (addup>=(sr1+ipb[a].r1))
      return;

  for(k=0, vs=(float *)hm->vs[b],va=(float *)hm->va[b],wt=(float *)hm->ipwt[b]
      ,mp=(float *)hm->map[b]; k<hm->vmax[b];
      k++, vs+=3, va+=3,wt++,mp+=3)
    {
      if (!sphereVal(spcenter,sr1,vs,&addup))
        {
          VECSet(va,vs);
          continue;
        }

      denf=diffFunc(THR,quartF(addup/r1),4.0,*wt);;

      if (fabs(denf-THR)<=EPS)
        {
          VECSet(va,vs);
          continue;
        ]

      VECSub(vs,mp,dir);
      dist=far=VECLen(dir);
      VECNorm(dir);


      near=0;
      if (sphereVal(spcenter,sr1,mp,&addup))
        denn=diffFunc((*wt)*ipb[a].weight*THR,quartF(addup/r1),4.0,*wt);
      else
        denn=(*wt)*ipb[a].weight*THR;

      VECSet(p,vs);

      j=0;
      while ((far-near>EPS) && (fabs(denf-denn)>EPS) && (j<5))
        {
          j++;

          if ((denf-THR)*(denn-THR)>0)
            dist=(near+far)/2;
          else
            dist= (near*(denf-THR)-far*(denn-THR))/(denf-denn);
```

```
          p[0]=mp[0] + dir[0]*dist;
          p[1]=mp[1] + dir[1]*dist;
          p[2]=mp[2] + dir[2]*dist;

          d2=0;
          if (sphereVal(spcenter,r1,p,&addup))
            d2=quartF(addup/r1);

          sphVal(a,p,&addup,1);
          d3=(*wt)*ipb[a].weight*THR*quartF(addup/ipb[a].r1);

          den=diffFunc(d3,d2,4.0,*wt);

          if (fabs(mval=(den-THR))<=EPS)
            break;

          if (mval<0)
            {
              far=dist;
              denf=den;
            }
          else
            {
              near=dist;
              denn=den;
            }
        }
     /* if ((dist>ipb[a].r1) || (dist <0))
        printf("hola %d %f %f, mp %f %f %f, p %f %f %f\n",j,dist,den,
               mp[0],mp[1],mp[2],p[0],p[1],p[2]);*/
      VECSet(va,p);

    }
}


deformSphereSphere(JINBUTU *hm, int a)
{
  int k,j;
  VERTEX dumc,dumc2,ctr,dir,p;
  float addup,addup2,*vs,*va,*vn,*vn2,d2,tmp,far,near,denf,denn,dist,
  den,mval,pdn;

        r1=sr1/sqrt(1-sqrt(THR2));
  if (sphVal(a,spcenter,&addup,1))
    pdn=diffFunc2(1.0,ipb[a].weight*THR*quartF(addup/ipb[a].r1),4.0,1.0);
  else
#if 0
    if (addup>=(sr1+ipb[a].sr1))
#else
    if (addup>=(r1+ipb[a].sr1))
#endif
      {
        return;
      }
    else
      pdn=1;
        /*printf("sp %f %f, %f %f %f, hp %f %f %f\n",sr1,ipb[a].sr1,
              spcenter[0],spcenter[1],spcenter[2],
              ipb[a].o1[0],ipb[a].o1[1],ipb[a].o1[2]);*/

  for(k=0, vs=(float *)spv[2],va=(float *)spv[3],
          vn=(float *)spn[2],vn2=(float *)spn[3];
      k<spvc[2];
      k++, vs+=3, va+=3,vn+=3,vn2+=3)
```



```
{
  near=0;
  denn=pdn;

  VECAdd(spcenter,vs,dumc2);

  if (!sphVal(a,dumc2,&addup,1))
    {
      VECSet(va,vs);
      VECSet(vn2,vn);
      continue;
    }

  d2=ipb[a].weight*THR*quartF(addup/ipb[a].r1);

    den=diffFunc2(THR2,d2,4.0,1.0);

  if (fabs(den-THR2)<=EPS)
    {
      VECSet(va,vs);
      VECSet(vn2,vn);
      continue;
    }

  far=VECLen(vs);
  denf=den;

  j=0;
  while ((far-near>EPS) && (j<4))
    {
      j++;

      if ((denf-THR2)*(denn-THR2)>0)
        dist=(near+far)/2;
      else
        dist= (near*(denf-THR2)-far*(denn-THR2))/(denf-denn);

      p[0]=spcenter[0] - vn[0]*dist;
      p[1]=spcenter[1] - vn[1]*dist;
      p[2]=spcenter[2] - vn[2]*dist;

      d2=0;
      if (sphVal(a,p,&addup,1))
        d2=ipb[a].weight*THR*quartF(addup/ipb[a].r1);

      den=diffFunc2((tmp=quartF(dist/r1)),d2,4.0,1.0);

      if (fabs(mval=(den-THR2))<=EPS)
        break;

      if (mval<0)
        {
          far=dist;
          denf=den;
        }
      else
        {
          near=dist;
          denn=den;
        }
    }
  VECSub(p,spcenter,va);
  VECSub(p,dumc,dumc2);
  VECNorm(dumc2);
  vn2[0]=vn[0]*tmp+d2*dumc2[0];
  vn2[1]=vn[0]*tmp+d2*dumc2[1];
```

```
      vn2[2]=vn[0]*tmp+d2*dumc2[2];
      VECNorm(vn2);
    }
}


deformPartPart(JINBUTU *hm, int a, int b, int c)
[
  int k,j;
  VERTEX dumc,dumc2,ctr,dir,p;
  float addup,addup2,*vs,*va,*wt,*mp,
d2,d3,tmp,far,near,denf,denn,dist,den,mval,pdn;

        rl=srl/sqrt(1-sqrt(THR2));
  for(k=0, vs=(float *)hm->vs[b],va=(float *)hm->va[b],wt=(float *)hm->ipwt[b]
      ,mp=(float *)hm->map[b]; k<hm->vmax[b];
      k++, vs+=3, va+=3,wt++,mp+=3)
    [
      if (!sphyVal(c,va,dumc,&addup,0))
        continue;

      VECSub(va,mp,dir);
      dist=far=VECLen(dir);
      VECNorm(dir);

      denf=diffFunc(THR,quartF(addup/rl),4.0,*wt);;

      if (fabs(denf-THR)<=EPS)
        continue;


      near=0;
      if (sphyVal(c,mp,dumc,&addup,0))
        denn=diffFunc((*wt)*ipb[a].weight*THR,quartF(addup/rl),4.0,*wt);
      else
        denn=(*wt)*ipb[a].weight*THR;

      VECSet(p,va);

      j=0;
      while ((far-near>EPS) && (fabs(denf-denn)>EPS) && (j<5))
        [
          j++;

          if ((denf-THR)*(denn-THR)>0)
            dist=(near+far)/2;
          else
            dist= (near*(denf-THR)-far*(denn-THR))/(denf-denn);

          p[0]=mp[0] + dir[0]*dist;
          p[1]=mp[1] + dir[1]*dist;
          p[2]=mp[2] + dir[2]*dist;

          d2=0;
          if (sphyVal(c,p,dumc,&addup,0))
            d2=quartF(addup/rl);

          sphyVal(a,p,dumc,&addup,0);
          d3=(*wt)*ipb[a].weight*THR*quartF(addup/ipb[a].rl);

          den=diffFunc(d3,d2,4.0,*wt);

          if (fabs(mval=(den-THR))<=EPS)
            break;

          if (mval<0)
```



```
            [
              far=dist;
              denf=den;
            ]
          else
            [
              near=dist;
              denn=den;
            ]
        ]
      /*if ((dist>ipb[a].rl) || (dist <0))
        printf("hola %d %f %f, mp %f %f %f, p %f %f %f\n",j,dist,den,
             mp[0],mp[1],mp[2],p[0],p[1],p[2]);*/
      VECSet(va,p);

    ]
}
```

```
#include  <gl.h>
#include  <math.h>

#define  IN
#define  OUT

static Matrix  mat;

void    cp_mat( Matrix m1, Matrix m2 );

c_loadmatrix( m )
IN  Matrix  m;
{
    cp_mat( mat, m );
}

c_getmatrix( m )
OUT  Matrix  m;
{
    cp_mat( m, mat );
}

c_multmatrix( m )
IN  Matrix  m;
{
    int     i;
    float   w;
    Matrix  wm;

    for( i=0; i<4; i++ ) {
        w  = m[i][0] * mat[0][0];
        w += m[i][1] * mat[1][0];
        w += m[i][2] * mat[2][0];
        w += m[i][3] * mat[3][0];
        wm[i][0] = w;

        w  = m[i][0] * mat[0][1];
        w += m[i][1] * mat[1][1];
        w += m[i][2] * mat[2][1];
        w += m[i][3] * mat[3][1];
        wm[i][1] = w;

        w  = m[i][0] * mat[0][2];
        w += m[i][1] * mat[1][2];
        w += m[i][2] * mat[2][2];
        w += m[i][3] * mat[3][2];
        wm[i][2] = w;

        w  = m[i][0] * mat[0][3];
        w += m[i][1] * mat[1][3];
        w += m[i][2] * mat[2][3];
        w += m[i][3] * mat[3][3];
        wm[i][3] = w;
    }

    cp_mat( mat, wm );
}

c_rot( a, axis )
IN  float  a;
IN  char   axis;
{
    float  r;
    float  s, c;

    r = a / 180.0 * M_PI;
```



```
    s = fsin( r );
    c = fcos( r );

    switch( axis ) {
    case 'x' :
        rot_x( s, c );
        break;
    case 'y' :
        rot_y( s, c );
        break;
    default :
        rot_z( s, c );
        break;
    ]
}

rot_x( s, c )
IN  float  s, c;
{
    float  a1, a2, a3, a4;
    float  b1, b2, b3, b4;

    a1 = c * mat[1][0] + s * mat[2][0];
    a2 = c * mat[1][1] + s * mat[2][1];
    a3 = c * mat[1][2] + s * mat[2][2];
    a4 = c * mat[1][3] + s * mat[2][3];

    b1 = c * mat[2][0] - s * mat[1][0];
    b2 = c * mat[2][1] - s * mat[1][1];
    b3 = c * mat[2][2] - s * mat[1][2];
    b4 = c * mat[2][3] - s * mat[1][3];

    mat[1][0] = a1;
    mat[1][1] = a2;
    mat[1][2] = a3;
    mat[1][3] = a4;
    mat[2][0] = b1;
    mat[2][1] = b2;
    mat[2][2] = b3;
    mat[2][3] = b4;
}

rot_y( s, c )
IN  float  s, c;
{
    float  a1, a2, a3, a4;
    float  b1, b2, b3, b4;

    a1 = c * mat[0][0] - s * mat[2][0];
    a2 = c * mat[0][1] - s * mat[2][1];
    a3 = c * mat[0][2] - s * mat[2][2];
    a4 = c * mat[0][3] - s * mat[2][3];

    b1 = c * mat[2][0] + s * mat[0][0];
    b2 = c * mat[2][1] + s * mat[0][1];
    b3 = c * mat[2][2] + s * mat[0][2];
    b4 = c * mat[2][3] + s * mat[0][3];

    mat[0][0] = a1;
    mat[0][1] = a2;
    mat[0][2] = a3;
    mat[0][3] = a4;
    mat[2][0] = b1;
    mat[2][1] = b2;
    mat[2][2] = b3;
    mat[2][3] = b4;
```

```
}

rot_z( s, c )
IN  float  s, c;
{
    float  a1, a2, a3, a4;
    float  b1, b2, b3, b4;

    a1 = c * mat[0][0] + s * mat[1][0];
    a2 = c * mat[0][1] + s * mat[1][1];
    a3 = c * mat[0][2] + s * mat[1][2];
    a4 = c * mat[0][3] + s * mat[1][3];

    b1 = c * mat[1][0] - s * mat[0][0];
    b2 = c * mat[1][1] - s * mat[0][1];
    b3 = c * mat[1][2] - s * mat[0][2];
    b4 = c * mat[1][3] - s * mat[0][3];

    mat[0][0] = a1;
    mat[0][1] = a2;
    mat[0][2] = a3;
    mat[0][3] = a4;
    mat[1][0] = b1;
    mat[1][1] = b2;
    mat[1][2] = b3;
    mat[1][3] = b4;
}

c_translate( x, y, z )
IN  Coord  x, y, z;
{
    float  w;

    w  = x * mat[0][0];
    w += y * mat[1][0];
    w += z * mat[2][0];
    mat[3][0] = w + mat[3][0];

    w  = x * mat[0][1];
    w += y * mat[1][1];
    w += z * mat[2][1];
    mat[3][1] = w + mat[3][1];

    w  = x * mat[0][2];
    w += y * mat[1][2];
    w += z * mat[2][2];
    mat[3][2] = w + mat[3][2];
}
```

```
/*
 * ファイル名 : data.c
 *
 * 機能       : 上半身モデルの頂点数、パッチ数のリスト及び、各関節の回転マトリッ *
 *
 * 履歴       : １０月２８日 ; 作成 ; 広瀬  正俊
 */
#include        "tool.h"

static  Matrix  G_idmat = {1.0, 0.0, 0.0, 0.0,
                           0.0, 1.0, 0.0, 0.0,
                           0.0, 0.0, 1.0, 0.0,
                           0.0, 0.0, 0.0, 1.0};

/*
 * 関数名 : Information()
 *
 * 引数   : なし
 *
 * 戻り値 : なし
 *
 * 機能   : 上半身モデルの頂点数、パッチ数を表示する
 *
 * 履歴   : １０月２８日 ; 作成 ; 広瀬  正俊
 */
void
Information(int hcnt, void *vhm[])
{
    JINBUTU     *hm;
    int         vcnt, pcnt;
    int         i, j;

    for (j = 0; j < hcnt; j++) {
        hm = (JINBUTU *)vhm[j];

        for(i = 0, vcnt = 0, pcnt = 0; i < MAX_BUHIN; i++) {
            vcnt += hm->vmax[i];
            pcnt += hm->polmax[i];
        }
        fprintf(stderr, "\nnumber of vertex  : %d\n", vcnt);
        fprintf(stderr, "\nnumber of polygon : %d\n", pcnt);
    }
    printf("\nSCALE FACTOR (%f),POWER FACTOR (%f) :", SCFAC,POWF);
        scanf("%f %f",&S1,&PF1);
]


void
CalcYubiAngle(JINBUTU *hm, int yubi, float hosei, float angle, Matrix *mat)
[
    loadmatrix(G_idmat);
    translate(hm->center[yubi][0], hm->center[yubi][1], hm->center[yubi][2]);
    rot(hosei, 'y');
    rot(angle, 'z');
    rot(-hosei, 'y');
    translate(-hm->center[yubi][0], -hm->center[yubi][1], -hm->center[yubi][2]);
    getmatrix(mat[0]);
    loadmatrix(G_idmat);
    translate(hm->center[yubi+1][0], hm->center[yubi+1][1],
              hm->center[yubi+1][2]);
    rot(hosei, 'y');
    rot(angle, 'z');
    rot(-hosei, 'y');
    translate(-hm->center[yubi+1][0], -hm->center[yubi+1][1],
              -hm->center[yubi+1][2]);
    getmatrix(mat[1]);
```



```
    loadmatrix(G_idmat);
    translate(hm->center[yubi+2][0], hm->center[yubi+2][1],
              hm->center[yubi+2][2]);
クスの作成osei, 'y');
    rot(angle, 'z');
    rot(-hosei, 'y');
    translate(-hm->center[yubi+2][0], -hm->center[yubi+2][1],
              -hm->center[yubi+2][2]);
    getmatrix(mat[2]);
]

/*
 * 関数名 : WriteWave(char *fname, int yubi)
 *
 * 引数   : fname : データファイル名
 *          yubi  : 部品番号
 *
 * 戻り値 : なし
 *
 * 機能   : ・ウェーブフロントフォーマットでデータをファイルに出力する
 *
 * 履歴   : １０月２８日 ; 作成 ; 広瀬  正俊
 */
void
WriteWave(JINBUTU *hm, char *fname, int yubi)
[
    FILE        *fp;
    char        name[MAX_NAME];
    float       *v, *vt;
    int         *pol;
    int         i;

    sprintf(name, "%s%s", DATA_DIR, fname);

    /* open .obj file */
    if((fp = fopen(name, "w")) == NULL) {
        fprintf(stderr, "<<ERROR>> %s is not opened.\n", name);
        return;
    }

    fprintf(stderr, "Now writing file %s\n", name);

    /* write data */
    fprintf(fp, "#   Wavefront data format\n");

    for(i = 0, v = (float *)hm->vs[yubi]; i < hm->vmax[yubi]; i++, v += 3) [
        fprintf(fp, "v %f %f %f\n", vt[0], v[1], v[2]);
    }

    for(i = 0, vt = (float *)hm->vt[yubi]; i < hm->vtmax[yubi]; i++, vt += 2) [
        fprintf(fp, "vt %f %f\n", vt[0], vt[1]);
    }

    for(i = 0, pol = (int *)hm->poly[yubi]; i < hm->polmax[yubi]; i++, pol += 6) [
        fprintf(fp, "f %d/%d %d/%d %d/%d\n", pol[0]+1, pol[3]+1,
                    pol[1]+1, pol[4]+1, pol[2]+1, pol[5]+1);
    }

    fclose(fp);
    fprintf(stderr, "%s write OK.\n", name);
    return;
}
```

```
/*
 * ファイル名 : draw3.c
 *
 * 機能       : 3次元表示用関数
 *
 * 履歴       : 1992年10月28日 ; 作成 ; 広瀬  正俊
 */
#include        "tool.h"

#include        <sys/time.h>

/*
#define         TIME_PRINT
*/

/* cup */
static Matrix   idmat = {1.0,0.0,0.0,0.0,
                         0.0,1.0,0.0,0.0,
                         0.0,0.0,1.0,0.0,
                         0.0,0.0,0.0,1.0};
int     kakudokeisan(float, float, float *, float *, float, Matrix, Matrix);
/*
int     DrawKampai(Matrix);
*/
#define         XANGLE          -30.0
void    GetKataMatrix(int, float *, float, float, float, float, Matrix,
                      Matrix, float *);


void
ReadTrackerData(int hcnt, void *vhm[])
{
    int         i, n;
    Device      dev;
    short       val;
    static char *hname = "ciris0";
#if 0
int cyuba;
JINBUTU *hm;
hm = (JINBUTU *)vhm[0];
cyuba = 0;
#endif

    while (1) {
        if (qtest()) {
            dev = qread(&val);
            if (dev == LEFTMOUSE) {
                break;
            }
        }

        ReadFastrak(hname, G_trd);
/********
        ReadCyber(hname, G_dgd);
*********/
/*
        ReadTR(hname, G_trd);
*/
#if 0
if (cyuba == 0) {
    hm->munex = G_trd[TR_MUNE][0];
    hm->muney = G_trd[TR_MUNE][1];
    hm->munez = G_trd[TR_MUNE][2];
    cyuba = 1;
}
#endif
```



```
/*
        ReadDG(hname, G_dgd);
*/
        DrawHuman(hcnt, vhm);
    }
}


void
ReadTrackerDataFile(int hcnt, void *vhm[])
{
    FILE        *fpt, *fpd;
    char        buff[LINE_BUF];
    int         i, n;
    Device      dev;
    short       val;
    int         fff;
    JINBUTU     *hm;

    fpt = fopen("tr_data", "r");
/*
    fpd = fopen("dg_data", "r");
*/
    fff = 0;
    hm = (JINBUTU *)vhm;

    while (1) {
        if (qtest()) {
            dev = qread(&val);
            if (dev == LEFTMOUSE) {
                break;
            }
        }

        for (i = 0; i < MAX_TR; i++) {
            if (fgets(buff, LINE_BUF, fpt) == NULL) {
                rewind(fpt);
                break;
            }
            sscanf(buff, "%d %f %f %f %f %f %f", &n,
                        &G_trd[i][0], &G_trd[i][1], &G_trd[i][2],
                        &G_trd[i][3], &G_trd[i][4], &G_trd[i][5]);
#if 0
G_trd[i][0] -= 4.0;
G_trd[i][1] -= 50.0;
G_trd[i][2] -= 40.0;
#endif
/*
            if (fgets(buff, LINE_BUF, fpd) == NULL) {
                rewind(fpd);
                break;
            }
            sscanf(buff, "%d %d %d %d %d %d %d %d %d %d",
                    &G_dgd[0], &G_dgd[1], &G_dgd[2], &G_dgd[3], &G_dgd[4],
                    &G_dgd[5], &G_dgd[6], &G_dgd[7], &G_dgd[8], &G_dgd[9]);
*/
        }

        if (fff == 0) {
            hm->munex = G_trd[1][0];
            hm->muney = G_trd[1][1];
            hm->munez = G_trd[1][2];
            fff = 1;
        }

        DrawHuman(hcnt, vhm);
```

```
    }

    fclose(fpt);
/*
    fclose(fpd);
*/
]


void
ReadAngleDataFile(int hcnt, void *vhm[])
{
    FILE     *fpa, *fpd;
    char     buff[LINE_BUF];
    int      i, n;
    Device   dev;
    short    val;
    int      fff;
    JINBUTU  *hm;

    fpa = fopen("ang_data", "r");
/*
    fpd = fopen("dg_data", "r");
*/
    fff = 0;
    hm = (JINBUTU *)vhm;

    while (1) {
        if (qtest()) [
            dev = qread(&val);
            if (dev == LEFTMOUSE) [
                break;
            ]
        ]

        for (i = 0; i < 7; i++) [
            if (fgets(buff, LINE_BUF, fpa) == NULL) [
                rewind(fpa);
                break;
            ]
            sscanf(buff, "%d %f %f %f %f %f %f", &n,
                        &G_ang[i][0], &G_ang[i][1], &G_ang[i][2],
                        &G_ang[i][3], &G_ang[i][4], &G_ang[i][5]);

/*
            if (fgets(buff, LINE_BUF, fpd) == NULL) [
                rewind(fpd);
                break;
            ]
            sscanf(buff, "%d %d %d %d %d %d %d %d %d %d",
                   &G_dgd[0], &G_dgd[1], &G_dgd[2], &G_dgd[3], &G_dgd[4],
                   &G_dgd[5], &G_dgd[6], &G_dgd[7], &G_dgd[8], &G_dgd[9]);
*/
        ]

        if (fff == 0) {
            hm->munex = G_ang[0][0];
            hm->muney = G_ang[0][1];
            hm->munez = G_ang[0][2];
            fff = 1;
        ]

        DrawHuman(hcnt, vhm);
    ]
    fclose(fpa);
/*
```



```
    fclose(fpd);
*/
]


/*
 * 関数名 : DrawHuman(int hcnt, void *vhm[])
 *
 * 引数   : hcnt  人物モデルの数
 *          vhm   人物モデルポインター
 *
 * 戻り値 : なし
 *
 * 機能   : 人物モデルの表示
 *
 * 履歴   : 1994年12月26日 ; 作成 ; 広瀬＼欸 */
void
DrawHuman(int hcnt, void *vhm[])
{
    int         i;
    Matrix      mat;
    float       p[3];

    if (trkswitch) [
        for (i = 0; i < hcnt; i++) [
            if (G_hand) [
                calc_body(vhm[i], G_trd, G_dgd, G_sisei, 2);
            ] else [
                calc_body(vhm[i], G_trd, G_dgd, G_sisei, G_kahansin);
            ]
        ]
    ] else [
        for (i = 0; i < hcnt; i++) [
            if (G_munekara) [
printf("Adsafkedfksdgks\n");
                make_body_mat2(vhm[i], G_ang, G_dgd);
            ] else [
                make_body_mat(vhm[i], G_ang, G_dgd);
            ]
        ]
    ]

    c3f(G_back);
    clear();
    zclear();

    pushmatrix();
printf("G_rota,G_elea:%f,%f\n",G_rota,G_elea);
    rot(G_rota, 'y');
    rot(G_elea, 'x');

#if 0
    if (G_sisei == SEIZA) [
        pushmatrix();
        translate(-30.0, -88.0, -15.0);
        pushmatrix();
        rot(180.0, 'y');
        translate(0.0, -5.0, 50.0);
        RGBcolor(255, 255, 255);
        DrawZabuton(&G_obj[0], NULL);
        popmatrix();
        RGBcolor(255, 0, 0);
        DrawZabuton(&G_obj[1], NULL);
        popmatrix();
    ] else [
        /* floor */
```

```
        pushmatrix();
        translate(0.0, 36.5, 0.0);
        DrawFloor(NULL);
        popmatrix();
    }
#endif

    /* コップをつかんでる場合 */
    if (G_glass) {
        /* 仮想空間でのコップの位置を求める */
        getmatrix(mat);
        coord_calc(p, G_trd[4], mat);
    }
printf("ATAMA_TRAK_NO_trd:%f,%f,%f\n",G_trd[ATAMA_TRAK_NO][0],G_trd[ATAMA_TRAK_NO][1]
 ,G_trd[ATAMA_TRAK_NO][2]);
printf("MIGITE_TRAK_NO:%f,%f,%f\n",G_trd[MIGITE_TRAK_NO][0],G_trd[MIGITE_TRAK_NO][1],
 G_trd[MIGITE_TRAK_NO][2]);
printf("G_trd:%f,%f,%f\n",G_trd[4][0],G_trd[4][1],G_trd[4][2]);
printf("p:%f,%f,%f\n",p[0],p[1],p[2]);

if (G_cmdst == DRAW_3D_TEXT) {
    /* テクスチャモデルの表示 */
    c3f(G_white);
    for (i = 0; i < hcnt; i++) {
        pushmatrix();
        translate((float)(2*i-1)*30.0, 0.0, 0.0);
        if (((JINBUTU *)vhm[i])->simple) {
            DrawSimple3D(vhm[i], NULL);
        } else {
            switch (G_sisei) {
            case STANDING:
                translate(0.0, 36.5, 0.0);
                Draw3D(vhm[i], NULL, G_kahansin, G_glass, p);
                break;
            case SITTING:
            case SEIZA:
                if (!trkswitch) {
                    if (G_munekara) {
                        Draw3DSeiza2(vhm[i], NULL, G_kahansin, G_glass, p);
                    } else {
                        Draw3DSeiza(vhm[i], NULL, G_kahansin, G_glass, p);
                    }
                } else {
#ifdef HIROSE
                    Draw3DSeiza3(vhm[i], NULL, G_kahansin, G_glass, p);
#else
                    Draw3DSeiza(vhm[i], NULL, G_kahansin, G_glass, p);
#endif
                }
                break;
            default:
                break;
            }
        }
        popmatrix();
    }
} else if (G_cmdst == DRAW_3D_WIRE) {
    /* ワイヤーモデルの表示 */
    for (i = 0; i < hcnt; i++) {
        pushmatrix();
        translate((float)(2*i-1)*30.0, 0.0, 0.0);

        if (((JINBUTU *)vhm[i])->simple) {
            DrawSimple3DWire((JINBUTU *)vhm[i]);
        } else {
            switch (G_sisei) {
```



```
            case STANDING:
                translate(0.0, 36.5, 0.0);
                Draw3DWire((JINBUTU *)vhm[i], G_glass, p);
                break;
            case SITTING:
            case SEIZA:
                if (!trkswitch) {
/*
                    Draw3DWire((JINBUTU *)vhm[i], G_glass, p);
*/
                    if (G_munekara) {
                        Draw3DWireSeiza2((JINBUTU *)vhm[i], G_glass, p);
                    } else {
                        Draw3DWireSeiza((JINBUTU *)vhm[i], G_glass, p);
                    }
                } else {
#ifdef HIROSE
                    Draw3DWireSeiza3((JINBUTU *)vhm[i], G_glass, p);
#else
                    Draw3DWireSeiza((JINBUTU *)vhm[i], G_glass, p);
#endif
                }
                break;
            default:
                break;
            }
        }
        popmatrix();
    }
}

    /* コップをつかんでる場合、コップを表示 */
/*
    if (G_glass) {
        translate(G_trd[4][0], G_trd[4][1],G_trd[4][2]);
        DrawKampai(idmat);
        translate(-G_trd[4][0], -G_trd[4][1], -G_trd[4][2]);
        lmbind(MATERIAL, 1);
    }
*/
    swapbuffers();

    popmatrix();

    gflush();
}

void
ResetTrackerData(int hcnt, void *vhm[])
{
    GlobalSet();
    DrawHuman(hcnt, vhm);
}


/*
 * 関数名 : Draw3DWire()
 *
 * 引数   : なし
 *
 * 戻り値 : なし
 *
 * 機能   : 上半身モデルのワイヤー表示
 *
 * 履歴   : 1992年10月28日 ; 作成 ; 広瀬　正俊
 */
```

```
void
Draw3DWire(JINBUTU *hm, int cup, float *p)
{

/* ***** */
    c3f(G_red);
    DrawSaikoro( 0.0, 0.0, 0.0, 2.0);
/* ***** */

    /* 体の位置を上方にずらす */
    translate(0.0, -hm->center[DOU][1]+hm->muneyt, -hm->center[DOU][2]);

    /* 胸部の表示 */
    multmatrix(hm->mat[DOU]);

/*+++++++++++++++++++++++++++++++++++++++++++++*/
    if (ffdswitch) {
        DrawMap3Wire(hm, DOU, hm->va[DOU], G_green);
    } else{
        DrawMap3Wire(hm, DOU, hm->vs[DOU], G_green);
    }

    drawSpace(&hm->douspace);
    DrawMap3Wire(hm, DOU, hm->vs[DOU], G_green);

/*+++++++++++++++++++++++++++++++++++++++++++++*/

    /* cup
    if (cup == 1) {
        GetTukamiKakudo(hm, p, 1);
    }
    */

    /*                    */
    /* 右手側データの表示 */
    /*                    */
    pushmatrix();

/* 11.4 (ochi) */
/* *****
    c3f(G_blue);
    DrawSaikoro(hm->center[KATA_D_R][0], hm->center[KATA_D_R][1],
                hm->center[KATA_D_R][2], 5.0);
***** */

    DrawMap3Wire(hm, KATA_D_R, hm->va[KATA_D_R], G_red);
    DrawMap3Wire(hm, KATA_N_R, hm->va[KATA_N_R], G_red);

 /*+++++++++++++++++++++++++++++++++++++++++++++*/
if(ffdswitch) {
    c3f(G_yellow);
    DrawSaikoro(hm->shcenter[1][0], hm->shcenter[1][1],
                hm->shcenter[1][2], 1.0);

    translate(hm->shcenter[1][0], hm->shcenter[1][1],
              hm->shcenter[1][2]);

    rot(hm->douspace.na[1], 'x');

    translate(-hm->center[KATA_D_R][0], -hm->center[KATA_D_R][1],
              -hm->center[KATA_D_R][2]);
}
 /*+++++++++++++++++++++++++++++++++++++++++++++*/

    multmatrix(hm->mat[KATA_D_R]);
```



```
printf("\n");
printf("showimp : %d \n", showimp);

 /*+++++++++++++++++++++++++++++++++++++++++++++*/
if (showimp) {
        trSphyl(ipb[RUARM].ouc,ipb[RUARM].nc,ipb[RUARM].r1,
                ipb[RUARM].o1,G_black);
        /*trSphyl(ipb[RUARM].ouc,ipb[RUARM].nc,ipb[RUARM].sr1,
                ipb[RUARM].o1,G_pink);*/

        trSph(ipb[RUARM].r1,ipb[RUARM].o1,G_black,0);
        /*trSph(ipb[RUARM].sr1,ipb[RUARM].o1,G_pink,0);*/
        trSph(ipb[RUARM].r1,ipb[RUARM].o2,G_black,0);
        /*trSph(ipb[RUARM].sr1,ipb[RUARM].o2,G_pink,0);*/
]
 /*+++++++++++++++++++++++++++++++++++++++++++++*/

    DrawMap3Wire(hm, NINOUDE_R, hm->vs[NINOUDE_R], G_green);

/* 11.4 (ochi) <KATA> */

/* *****
    c3f(G_blue);
    DrawSaikoro(hm->center[HIJI_R][0], hm->center[HIJI_R][1],
                hm->center[HIJI_R][2], 5.0);
***** */

    DrawMap3Wire(hm, HIJI_R, hm->va[HIJI_R], G_red);
    multmatrix(hm->mat[HIJI_R]);

 /*+++++++++++++++++++++++++++++++++++++++++++++*/
if (showimp) {
        trSphyl(ipb[RLARM].ouc,ipb[RLARM].nc,ipb[RLARM].r1,
                ipb[RLARM].o1,G_black);
        /*trSphyl(ipb[RLARM].ouc,ipb[RLARM].nc,ipb[RLARM].sr1,
                ipb[RLARM].o1,G_pink);*/
}
 /*+++++++++++++++++++++++++++++++++++++++++++++*/

/* *****
    c3f(G_blue);
    DrawSaikoro(hm->center[UDE_R][0], hm->center[UDE_R][1],
                hm->center[UDE_R][2], 1.5);
***** */
    DrawMap3Wire(hm, UDE_R, hm->va[UDE_R], G_green);
    multmatrix(hm->mat[UDE_R]);

/* *****
    c3f(G_blue);
    DrawSaikoro(hm->center[TE_R][0], hm->center[TE_R][1],
                hm->center[TE_R][2], 5.0);
***** */
    DrawMap3Wire(hm, TE_R, hm->va[TE_R], G_red);
    multmatrix(hm->mat[TE_R]);

    pushmatrix();
    DrawYubiKansetuWire(hm, OYAYUBI_R_1);

    popmatrix();
    pushmatrix();
    DrawYubiKansetuWire(hm, HITOSASIYUBI_R_1);

    popmatrix();
    pushmatrix();
    DrawYubiKansetuWire(hm, NAKAYUBI_R_1);
```

```
    popmatrix();
    pushmatrix();
    DrawYubiKansetuWire(hm, KUSURIYUBI_R_1);

    popmatrix();
    DrawYubiKansetuWire(hm, KOYUBI_R_1);

    /*                      */
    /* 左手側データの表示 */
    /*                      */
    popmatrix();
    pushmatrix();
/* *****
    c3f(G_blue);
    DrawSaikoro(hm->center[KATA_D_L][0], hm->center[KATA_D_L][1],
                hm->center[KATA_D_L][2], 1.0);
***** */
    DrawMap3Wire(hm, KATA_D_L, hm->va[KATA_D_L], G_red);
    DrawMap3Wire(hm, KATA_N_L, hm->va[KATA_N_L], G_red);


/*++++++++++++++++++++++++++++++++*/
    c3f(G_yellow);
    DrawSaikoro(hm->shcenter[0][0], hm->shcenter[0][1],
                hm->shcenter[0][2], 1.0);

    translate(hm->shcenter[0][0], hm->shcenter[0][1],
              hm->shcenter[0][2]);

    rot(hm->douspace.na[0], 'x');

    translate(-hm->center[KATA_D_L][0], -hm->center[KATA_D_L][1],
              -hm->center[KATA_D_L][2]);
/*++++++++++++++++++++++++++++++++*/
trSph(1.0,spcenter,G_blue,3);

    multmatrix(hm->mat[KATA_D_L]);

 /*++++++++++++++++++++++++++++++++++++++++++++*/
if (showimp) [
        /*trSphyl(ipb[LUARM].ouc,ipb[LUARM].nc,ipb[LUARM].r1,
                ipb[LUARM].o1,G_yellow);*/
        trSphyl(ipb[LUARM].ouc,ipb[LUARM].nc,ipb[LUARM].sr1,
                ipb[LUARM].o1,G_black);

        /*trSph(ipb[LUARM].r1,ipb[LUARM].o1,G_black,0);*/
        trSph(ipb[LUARM].sr1,ipb[LUARM].o1,G_black,0);
        /*trSph(ipb[LUARM].r1,ipb[LUARM].o2,G_black,0);*/
        trSph(ipb[LUARM].sr1,ipb[LUARM].o2,G_black,0);
]
/*++++++++++++++++++++++++++++++++++++++++++++*/

    DrawMap3Wire(hm, NINOUDE_L, hm->vs[NINOUDE_L], G_green);

/* *****
    c3f(G_blue);
    DrawSaikoro(hm->center[HIJI_L][0], hm->center[HIJI_L][1],
                hm->center[HIJI_L][2], 1.0);
***** */

    DrawMap3Wire(hm, HIJI_L, hm->va[HIJI_L], G_red);
    multmatrix(hm->mat[HIJI_L]);

/* *****
    c3f(G_blue);
    DrawSaikoro(hm->center[UDE_L][0], hm->center[UDE_L][1],
```



```
                hm->center[UDE_L][2], 1.0);
***** */

 /*++++++++++++++++++++++++++++++++++++++++++++*/
  /* trSph(1.0,spcenter,G_blue,3);*/

if (showimp) {
       /*trSphyl(ipb[LLARM].ouc,ipb[LLARM].nc,ipb[LLARM].r1,
               ipb[LLARM].o1,G_yellow);*/
       trSphyl(ipb[LLARM].ouc,ipb[LLARM].nc,ipb[LLARM].sr1,
               ipb[LLARM].o1,G_black);
}
 /*++++++++++++++++++++++++++++++++++++++++++++*/

    DrawMap3Wire(hm, UDE_L, hm->va[UDE_L], G_green);
    multmatrix(hm->mat[UDE_L]);

/* *****
    c3f(G_blue);
    DrawSaikoro(hm->center[TE_L][0], hm->center[TE_L][1],
                hm->center[TE_L][2], 1.0);
***** */

    DrawMap3Wire(hm, TE_L, hm->va[TE_L], G_red);
    multmatrix(hm->mat[TE_L]);

    DrawMap3Wire(hm, OYAYUBI_L_1, hm->vs[OYAYUBI_L_1], G_green);
    DrawMap3Wire(hm, OYAYUBI_L_2, hm->vs[OYAYUBI_L_2], G_green);
    DrawMap3Wire(hm, OYAYUBI_L_3, hm->vs[OYAYUBI_L_3], G_green);
    DrawMap3Wire(hm, HITOSASIYUBI_L_1, hm->vs[HITOSASIYUBI_L_1], G_green);
    DrawMap3Wire(hm, HITOSASIYUBI_L_2, hm->vs[HITOSASIYUBI_L_2], G_green);
    DrawMap3Wire(hm, HITOSASIYUBI_L_3, hm->vs[HITOSASIYUBI_L_3], G_green);
    DrawMap3Wire(hm, NAKAYUBI_L_1, hm->vs[NAKAYUBI_L_1], G_green);
    DrawMap3Wire(hm, NAKAYUBI_L_2, hm->vs[NAKAYUBI_L_2], G_green);
    DrawMap3Wire(hm, NAKAYUBI_L_3, hm->vs[NAKAYUBI_L_3], G_green);
    DrawMap3Wire(hm, KUSURIYUBI_L_1, hm->vs[KUSURIYUBI_L_1], G_green);
    DrawMap3Wire(hm, KUSURIYUBI_L_2, hm->vs[KUSURIYUBI_L_2], G_green);
    DrawMap3Wire(hm, KUSURIYUBI_L_3, hm->vs[KUSURIYUBI_L_3], G_green);
    DrawMap3Wire(hm, KOYUBI_L_1, hm->vs[KOYUBI_L_1], G_green);
    DrawMap3Wire(hm, KOYUBI_L_2, hm->vs[KOYUBI_L_2], G_green);
    DrawMap3Wire(hm, KOYUBI_L_3, hm->vs[KOYUBI_L_3], G_green);

    /*          */
    /* 頭部の表示 */
    /*          */
    popmatrix();
    pushmatrix();


    DrawMap3Wire(hm, KUBI, hm->va[KUBI], G_red);

/*++++++++++++++++++++++++++++++++++++*/
/*
    translate(hm->douspace.center[3][0],hm->douspace.center[3][1],
              hm->douspace.center[3][2]);
    rot(3*hm->douspace.xang, 'x');
    translate(-hm->douspace.corig[3][0],-hm->douspace.corig[3][1],
              -hm->douspace.corig[3][2]);
*/
/*++++++++++++++++++++++++++++++++++++*/


    multmatrix(hm->mat[KUBI]);

/*++++++++++++++++++++++++++++++++++++*/
if (showimp) [
```

```
        trSph(ipb[HEAD].r1,ipb[HEAD].o1,G_black,0);
        /*trSph(ipb[HEAD].sr1,ipb[HEAD].o1,G_pink,0);*/
]
/*++++++++++++++++++++++++++++++++++++*/

    DrawMap3Wire(hm, ATAMA, hm->va[ATAMA], G_green);

    popmatrix();

 /*++++++++++++++++++++++++++++++++++++++++++++*/
    popmatrix();

    drawSpace(&hm->kosispace);

 /*++++++++++++++++++++++++++++++++++++++++++++*/

/* Nariyama add */
rot(180.0, 'y');
translate(0.0, -hm->center[DOU][1], 0.0);
/* Nariyama add */

    DrawMap3Wire(hm, KOSI, hm->va[KOSI], G_red);
    multmatrix(hm->mat[KOSI]);
printf("Path 1 \n");
prt_mat(hm->mat[KOSI], "hm->mat[KOSI]");

    pushmatrix();
    DrawMap3Wire(hm, SIRI_R, hm->va[SIRI_R], G_red);
    multmatrix(hm->mat[SIRI_R]);

    DrawMap3Wire(hm, MOMO_R, hm->vs[MOMO_R], G_green);
    DrawMap3Wire(hm, HIZA_R, hm->va[HIZA_R], G_red);

/*++++++++++++++++++++++++++++++++++++++++++++*/
if (showimp) [
       trSphyl(ipb[RULG].ouc,ipb[RULG].nc,ipb[RULG].r1,
               ipb[RULG].o2,G_black);
       /*trSphyl(ipb[RULG].ouc,ipb[RULG].nc,ipb[RULG].sr1,
               ipb[RULG].o2,G_pink);*/
       trSph(ipb[RULG].r1,ipb[RULG].o1,G_black,0);
       trSph(ipb[RULG].r1,ipb[RULG].o2,G_black,0);

}
/*++++++++++++++++++++++++++++++++++++++++++++*/

    multmatrix(hm->mat[HIZA_R]);

/*++++++++++++++++++++++++++++++++++++++++++++*/
if (showimp) [
       trSphyl(ipb[RLLG].ouc,ipb[RLLG].nc,ipb[RLLG].r1,
               ipb[RLLG].o2,G_black);
       /*trSphyl(ipb[RLLG].ouc,ipb[RLLG].nc,ipb[RLLG].sr1,
               ipb[RLLG].o2,G_pink);*/
}
/*++++++++++++++++++++++++++++++++++++++++++++*/

    DrawMap3Wire(hm, SUNE_R, hm->vs[SUNE_R], G_green);
    multmatrix(hm->mat[KUTU_R]);
    DrawMap3Wire(hm, KUTU_R, hm->vs[KUTU_R], G_green);

    popmatrix();
    DrawMap3Wire(hm, SIRI_L, hm->va[SIRI_L], G_red);
    multmatrix(hm->mat[SIRI_L]);

    DrawMap3Wire(hm, MOMO_L, hm->vs[MOMO_L], G_green);
    DrawMap3Wire(hm, HIZA_L, hm->va[HIZA_L], G_red);
```



```
/*++++++++++++++++++++++++++++++++++++++++++++*/
if (showimp) {
       /*trSphyl(ipb[LULG].ouc,ipb[LULG].nc,ipb[LULG].r1,
               ipb[LULG].o2,G_yellow);*/
       trSphyl(ipb[LULG].ouc,ipb[LULG].nc,ipb[LULG].sr1,
               ipb[LULG].o2,G_black);
       trSph(ipb[LULG].sr1,ipb[LULG].o1,G_black,0);
       trSph(ipb[LULG].sr1,ipb[LULG].o2,G_black,0);
}
/*++++++++++++++++++++++++++++++++++++++++++++*/

    multmatrix(hm->mat[HIZA_L]);

/*++++++++++++++++++++++++++++++++++++++++++++*/
if (showimp) {
       /*trSphyl(ipb[LLLG].ouc,ipb[LLLG].nc,ipb[LLLG].r1,
               ipb[LLLG].o2,G_yellow);*/
       trSphyl(ipb[LLLG].ouc,ipb[LLLG].nc,ipb[LLLG].sr1,
               ipb[LLLG].o2,G_black);
}
/*++++++++++++++++++++++++++++++++++++++++++++*/

    DrawMap3Wire(hm, SUNE_L, hm->vs[SUNE_L], G_green);
    multmatrix(hm->mat[KUTU_L]);
    DrawMap3Wire(hm, KUTU_L, hm->vs[KUTU_L], G_green);

    popmatrix();
]


/*
 * 関数名 : Draw3DWireSeiza()
 *
 * 引数   : なし
 *
 * 戻り値 : なし
 *
 * 機能   : 上半身モデルのワイヤー表示
 *
 * 履歴   : 1992年10月28日 ; 作成 ; 広瀬＼敦 */
void
Draw3DWireSeiza(JINBUTU *hm, int cup, float *p)
{
    float               dumy;

    if (hm->stand) {
        Draw3DWire(hm, 0, &dumy);
        return;
    }

    /* 体の位置を上方にずらす */
/*
    rot(180.0, 'y');
*/
    translate(0.0, -hm->center[DOU][1], 0.0);

prt_mat(hm->mat[KOSI], "hm->mat[KOSI]");
prt_mat(hm->mune_mat, "hm->mune_mat");
prt_mat(hm->mat1.mune, "hm->mat1.mune");

    if(!trkswitch) [
        multmatrix(hm->mat[KOSI]);
        /* multmatrix(hm->mune_mat); */
    }
```

```
/***
    if (hm->muney > 5.0) {
****/

    if (fsqrt(hm->kosix*hm->kosix +
        (hm->kosiy+hm->ske_real.dou_len)*(hm->kosiy+hm->ske_real.dou_len) +
        hm->kosiz*hm->kosiz) > 3.0) {
        translate(hm->kosix, hm->kosiy, hm->kosiz);

printf("\n");
printf("hm->kosi : %f, %f, %f \n", hm->kosix, hm->kosiy, hm->kosiz);

    ] else {
        translate(0.0, -hm->ske_real.dou_len, 0.0);
    ]

    pushmatrix();
    DrawMap3Wire(hm, SIRI_R, hm->va[SIRI_R], G_red);
    multmatrix(hm->mat[SIRI_R]);

    DrawMap3Wire(hm, MOMO_R, hm->vs[MOMO_R], G_green);
    DrawMap3Wire(hm, HIZA_R, hm->va[HIZA_R], G_red);

/*++++++++++++++++++++++++++++++++++++++++++++*/
if (showimp) {
       trSphyl(ipb[RULG].ouc,ipb[RULG].nc,ipb[RULG].r1,
               ipb[RULG].o2,G_black);
       /*trSphyl(ipb[RULG].ouc,ipb[RULG].nc,ipb[RULG].sr1,
               ipb[RULG].o2,G_pink);*/
       trSph(ipb[RULG].r1,ipb[RULG].o1,G_black,0);
       trSph(ipb[RULG].r1,ipb[RULG].o2,G_black,0);
}
/*++++++++++++++++++++++++++++++++++++++++++++*/

    multmatrix(hm->mat[HIZA_R]);

/*++++++++++++++++++++++++++++++++++++++++++++*/
if (showimp) {
       trSphyl(ipb[RLLG].ouc,ipb[RLLG].nc,ipb[RLLG].r1,
               ipb[RLLG].o2,G_black);
       /*trSphyl(ipb[RLLG].ouc,ipb[RLLG].nc,ipb[RLLG].sr1,
               ipb[RLLG].o2,G_pink);*/
}
/*++++++++++++++++++++++++++++++++++++++++++++*/

    DrawMap3Wire(hm, SUNE_R, hm->vs[SUNE_R], G_green);
    multmatrix(hm->mat[KUTU_R]);
    DrawMap3Wire(hm, KUTU_R, hm->vs[KUTU_R], G_green);

    popmatrix();
    pushmatrix();
    DrawMap3Wire(hm, SIRI_L, hm->va[SIRI_L], G_red);
    multmatrix(hm->mat[SIRI_L]);

    DrawMap3Wire(hm, MOMO_L, hm->vs[MOMO_L], G_green);
    DrawMap3Wire(hm, HIZA_L, hm->va[HIZA_L], G_red);

/*++++++++++++++++++++++++++++++++++++++++++++*/
if (showimp) {
       /*trSphyl(ipb[LULG].ouc,ipb[LULG].nc,ipb[LULG].r1,
               ipb[LULG].o2,G_yellow);*/
       trSphyl(ipb[LULG].ouc,ipb[LULG].nc,ipb[LULG].sr1,
               ipb[LULG].o2,G_black);
       trSph(ipb[LULG].sr1,ipb[LULG].o1,G_black,0);
       trSph(ipb[LULG].sr1,ipb[LULG].o2,G_black,0);
}
```



```
/*++++++++++++++++++++++++++++++++++++++++++++*/

    multmatrix(hm->mat[HIZA_L]);

/*++++++++++++++++++++++++++++++++++++++++++++*/
if (showimp) {
       /*trSphyl(ipb[LLLG].ouc,ipb[LLLG].nc,ipb[LLLG].r1,
               ipb[LLLG].o2,G_yellow);*/
       trSphyl(ipb[LLLG].ouc,ipb[LLLG].nc,ipb[LLLG].sr1,
               ipb[LLLG].o2,G_black);
}
/*++++++++++++++++++++++++++++++++++++++++++++*/

    DrawMap3Wire(hm, SUNE_L, hm->vs[SUNE_L], G_green);
    multmatrix(hm->mat[KUTU_L]);
    DrawMap3Wire(hm, KUTU_L, hm->vs[KUTU_L], G_green);

    popmatrix();
/*++++++++++++++++++++++++++++++++++++++++++++*/
    if (ffdswitch) {
        drawSpace(&hm->kosispace);
    ]
/*++++++++++++++++++++++++++++++++++++++++++++*/
    DrawMap3Wire(hm, KOSI, hm->va[KOSI], G_red);

    /* 胸部の表示 */
/*++++++++++++++++++++++++++++++++++++++++++++*/
    if (ffdswitch) [
        multmatrix(hm->mune_mat);
        drawSpace(&hm->douspace);
        DrawMap3Wire(hm, DOU, hm->va[DOU], G_black);
    ] else {
        multmatrix(hm->mat[KOSI]);
        DrawMap3Wire(hm, DOU, hm->vs[DOU], G_green);
    ]
/*++++++++++++++++++++++++++++++++++++++++++++*/

    /* cup
    if (cup == 1) {
        GetTukamiKakudo(hm, p, l);
    ]
    */

    /*                  */
    /* 右手側データの表示 */
    /*                  */
    pushmatrix();
    c3f(G_blue);
    DrawSaikoro(hm->center[KATA_D_R][0], hm->center[KATA_D_R][1],
                hm->center[KATA_D_R][2], 1.0);
    DrawMap3Wire(hm, KATA_D_R, hm->va[KATA_D_R], G_red);
    DrawMap3Wire(hm, KATA_N_R, hm->va[KATA_N_R], G_red);

/*++++++++++++++++++++++++++++++++++++++++++++*/
    if (ffdswitch) {
        c3f(G_yellow);
        DrawSaikoro(hm->shcenter[1][0], hm->shcenter[1][1],
                    hm->shcenter[1][2], 1.0);
        translate(hm->shcenter[1][0], hm->shcenter[1][1],
                  hm->shcenter[1][2]);
        rot(hm->douspace.na[1], 'x');
        translate(-hm->center[KATA_D_R][0], -hm->center[KATA_D_R][1],
                  -hm->center[KATA_D_R][2]);
    ]
/*++++++++++++++++++++++++++++++++++++++++++++*/
```

```
    multmatrix(hm->mat[KATA_D_R]);

/*+++++++++++++++++++++++++++++++++++++++++++*/
if (showimp) {
        trSphyl(ipb[RUARM].ouc,ipb[RUARM].nc,ipb[RUARM].r1,
                ipb[RUARM].o1,G_black);
        /*trSphyl(ipb[RUARM].ouc,ipb[RUARM].nc,ipb[RUARM].sr1,
                ipb[RUARM].o1,G_pink);*/

        trSph(ipb[RUARM].r1,ipb[RUARM].o1,G_black,0);
        /*trSph(ipb[RUARM].sr1,ipb[RUARM].o1,G_pink,0);*/
        trSph(ipb[RUARM].r1,ipb[RUARM].o2,G_black,0);
        /*trSph(ipb[RUARM].sr1,ipb[RUARM].o2,G_pink,0);*/
}
/*+++++++++++++++++++++++++++++++++++++++++++*/

    DrawMap3Wire(hm, NINOUDE_R, hm->vs[NINOUDE_R], G_green);

    c3f(G_blue);
    DrawSaikoro(hm->center[HIJI_R][0], hm->center[HIJI_R][1],
                hm->center[HIJI_R][2], 1.0);
    DrawMap3Wire(hm, HIJI_R, hm->va[HIJI_R], G_red);
    multmatrix(hm->mat[HIJI_R]);

/*+++++++++++++++++++++++++++++++++++++++++++*/
if (showimp) {
        trSphyl(ipb[RLARM].ouc,ipb[RLARM].nc,ipb[RLARM].r1,
                ipb[RLARM].o1,G_black);
        /*trSphyl(ipb[RLARM].ouc,ipb[RLARM].nc,ipb[RLARM].sr1,
                ipb[RLARM].o1,G_pink);*/
}
/*+++++++++++++++++++++++++++++++++++++++++++*/

    c3f(G_blue);
    DrawSaikoro(hm->center[UDE_R][0], hm->center[UDE_R][1],
                hm->center[UDE_R][2], 1.0);
    DrawMap3Wire(hm, UDE_R, hm->va[UDE_R], G_green);
    multmatrix(hm->mat[UDE_R]);

{
Matrix mat;
float p[3];

getmatrix(mat);
coord_calc(p, hm->center[TE_R], mat);
printf("[p]:%f,%f,%f\n",p[0],p[1],p[2]);
}
    c3f(G_blue);
    DrawSaikoro(hm->center[TE_R][0], hm->center[TE_R][1],
                hm->center[TE_R][2], 1.0);
    DrawMap3Wire(hm, TE_R, hm->va[TE_R], G_red);
    multmatrix(hm->mat[TE_R]);

    pushmatrix();
    DrawYubiKansetuWire(hm, OYAYUBI_R_1);

    popmatrix();
    pushmatrix();
    DrawYubiKansetuWire(hm, HITOSASIYUBI_R_1);

    popmatrix();
    pushmatrix();
    DrawYubiKansetuWire(hm, NAKAYUBI_R_1);

    popmatrix();
    pushmatrix();
```



```
    DrawYubiKansetuWire(hm, KUSURIYUBI_R_1);

    popmatrix();
    DrawYubiKansetuWire(hm, KOYUBI_R_1);

    /*                      */
    /* 左手側データの表示 */
    /*                      */
    popmatrix();
    pushmatrix();
    c3f(G_blue);
    DrawSaikoro(hm->center[KATA_D_L][0], hm->center[KATA_D_L][1],
                hm->center[KATA_D_L][2], 1.0);
    DrawMap3Wire(hm, KATA_D_L, hm->va[KATA_D_L], G_red);
    DrawMap3Wire(hm, KATA_N_L, hm->va[KATA_N_L], G_red);

/*++++++++++++++++++++++++++++++++*/
    if (ffdswitch) {
        c3f(G_yellow);
        DrawSaikoro(hm->shcenter[0][0], hm->shcenter[0][1],
                    hm->shcenter[0][2], 1.0);
        translate(hm->shcenter[0][0], hm->shcenter[0][1],
                  hm->shcenter[0][2]);
        rot(hm->douspace.na[0], 'x');
        translate(-hm->center[KATA_D_L][0], -hm->center[KATA_D_L][1],
                  -hm->center[KATA_D_L][2]);
    ]
/*++++++++++++++++++++++++++++++++*/
if (collon) [
trSph(1.0,spcenter,G_blue,3);
}

    multmatrix(hm->mat[KATA_D_L]);

/*++++++++++++++++++++++++++++++++++++++++++++*/
if (showimp) {
        /*trSphyl(ipb[LUARM].ouc,ipb[LUARM].nc,ipb[LUARM].r1,
                ipb[LUARM].o1,G_yellow);*/
        trSphyl(ipb[LUARM].ouc,ipb[LUARM].nc,ipb[LUARM].sr1,
                ipb[LUARM].o1,G_black);

        /*trSph(ipb[LUARM].r1,ipb[LUARM].o1,G_black,0);*/
        trSph(ipb[LUARM].sr1,ipb[LUARM].o1,G_black,0);
        /*trSph(ipb[LUARM].r1,ipb[LUARM].o2,G_black,0);*/
        trSph(ipb[LUARM].sr1,ipb[LUARM].o2,G_black,0);
}
/*++++++++++++++++++++++++++++++++++++++++++++*/

    DrawMap3Wire(hm, NINOUDE_L, hm->vs[NINOUDE_L], G_green);

    c3f(G_blue);
    DrawSaikoro(hm->center[HIJI_L][0], hm->center[HIJI_L][1],
                hm->center[HIJI_L][2], 1.0);
    DrawMap3Wire(hm, HIJI_L, hm->va[HIJI_L], G_red);
    multmatrix(hm->mat[HIJI_L]);

    c3f(G_blue);
    DrawSaikoro(hm->center[UDE_L][0], hm->center[UDE_L][1],
                hm->center[UDE_L][2], 1.0);

/*++++++++++++++++++++++++++++++++++++++++++++*/
  /* trSph(1.0,spcenter,G_blue,3);*/

if (showimp) {
      /*trSphyl(ipb[LLARM].ouc,ipb[LLARM].nc,ipb[LLARM].r1,
              ipb[LLARM].o1,G_yellow);*/
```

```
        trSphyl(ipb[LLARM].ouc,ipb[LLARM].nc,ipb[LLARM].sr1,
                ipb[LLARM].ol,G_black);
]
/*++++++++++++++++++++++++++++++++++++++++++++*/

    DrawMap3Wire(hm, UDE_L, hm->va[UDE_L], G_green);
    multmatrix(hm->mat[UDE_L]);

    c3f(G_blue);
    DrawSaikoro(hm->center[TE_L][0], hm->center[TE_L][1],
                hm->center[TE_L][2], 1.0);
    DrawMap3Wire(hm, TE_L, hm->va[TE_L], G_red);
    multmatrix(hm->mat[TE_L]);
    DrawMap3Wire(hm, OYAYUBI_L_1, hm->vs[OYAYUBI_L_1], G_green);
    DrawMap3Wire(hm, OYAYUBI_L_2, hm->vs[OYAYUBI_L_2], G_green);
    DrawMap3Wire(hm, OYAYUBI_L_3, hm->vs[OYAYUBI_L_3], G_green);
    DrawMap3Wire(hm, HITOSASIYUBI_L_1, hm->vs[HITOSASIYUBI_L_1], G_green);
    DrawMap3Wire(hm, HITOSASIYUBI_L_2, hm->vs[HITOSASIYUBI_L_2], G_green);
    DrawMap3Wire(hm, HITOSASIYUBI_L_3, hm->vs[HITOSASIYUBI_L_3], G_green);
    DrawMap3Wire(hm, NAKAYUBI_L_1, hm->vs[NAKAYUBI_L_1], G_green);
    DrawMap3Wire(hm, NAKAYUBI_L_2, hm->vs[NAKAYUBI_L_2], G_green);
    DrawMap3Wire(hm, NAKAYUBI_L_3, hm->vs[NAKAYUBI_L_3], G_green);
    DrawMap3Wire(hm, KUSURIYUBI_L_1, hm->vs[KUSURIYUBI_L_1], G_green);
    DrawMap3Wire(hm, KUSURIYUBI_L_2, hm->vs[KUSURIYUBI_L_2], G_green);
    DrawMap3Wire(hm, KUSURIYUBI_L_3, hm->vs[KUSURIYUBI_L_3], G_green);
    DrawMap3Wire(hm, KOYUBI_L_1, hm->vs[KOYUBI_L_1], G_green);
    DrawMap3Wire(hm, KOYUBI_L_2, hm->vs[KOYUBI_L_2], G_green);
    DrawMap3Wire(hm, KOYUBI_L_3, hm->vs[KOYUBI_L_3], G_green);

    /*          */
    /* 頭部の表示 */
    /*          */
    popmatrix();

    DrawMap3Wire(hm, KUBI, hm->va[KUBI], G_red);

/*++++++++++++++++++++++++++++++++++++++*/
    if (ffdswitch) [
        translate(hm->douspace.center[3][0],hm->douspace.center[3][1],
                  hm->douspace.center[3][2]);
        rot(3*hm->douspace.xang, 'x');
        translate(-hm->douspace.corig[3][0],-hm->douspace.corig[3][1],
                  -hm->douspace.corig[3][2]);
    ]
/*++++++++++++++++++++++++++++++++++++++*/

    multmatrix(hm->mat[KUBI]);

/*++++++++++++++++++++++++++++++++++++++*/
if (showimp) [
        trSph(ipb[HEAD].r1,ipb[HEAD].ol,G_black,0);
        /*trSph(ipb[HEAD].sr1,ipb[HEAD].ol,G_pink,0);*/
]
/*++++++++++++++++++++++++++++++++++++++*/

    DrawMap3Wire(hm, ATAMA, hm->va[ATAMA], G_green);
}


/*
 * 関数名 : Draw3DWireSeiza2()
 *
 * 引数   : なし
 *
 * 戻り値 : なし
 *
```



```
 * 機能   : 上半身モデルのワイヤー表示
 *
 * 履歴   : 1992年10月28日 ; 作成 ; 広瀬＼敦 */
void
Draw3DWireSeiza2(JINBUTU *hm, int cup, float *p)
{
    float               dumy;
    Matrix      mat;

printf("ffdswitch:%d\n",ffdswitch);
printf("trkswitch:%d\n",trkswitch);

    /* 体の位置を上方にずらす */
/*
    rot(180.0, 'y');
    translate(0.0, -hm->center[DOU][1], 0.0);
*/

    multmatrix(hm->mat[DOU]);

    /* 胸部の表示 */
/*++++++++++++++++++++++++++++++++++++++++++++*/
    if (ffdswitch) [
        rev_mat(hm->mune_mat, mat);
        translate(hm->center[KOSI][0], hm->center[KOSI][1],
                  hm->center[KOSI][2]);
        multmatrix(mat);
        translate(-hm->center[KOSI][0], -hm->center[KOSI][1],
                  -hm->center[KOSI][2]);
        drawSpace(&hm->douspace);
        DrawMap3Wire(hm, DOU, hm->va[DOU], G_black);
    ] else [
        DrawMap3Wire(hm, DOU, hm->vs[DOU], G_green);
    ]
/*++++++++++++++++++++++++++++++++++++++++++++*/

    /*                  */
    /* 右手側データの表示 */
    /*                  */
    pushmatrix();
    c3f(G_blue);
    DrawSaikoro(hm->center[KATA_D_R][0], hm->center[KATA_D_R][1],
                hm->center[KATA_D_R][2], 1.0);
    DrawMap3Wire(hm, KATA_D_R, hm->va[KATA_D_R], G_red);
    DrawMap3Wire(hm, KATA_N_R, hm->va[KATA_N_R], G_red);

/*++++++++++++++++++++++++++++++++++++++++++++*/
    if (ffdswitch) [
        c3f(G_yellow);
        DrawSaikoro(hm->shcenter[1][0], hm->shcenter[1][1],
                    hm->shcenter[1][2], 1.0);
        translate(hm->shcenter[1][0], hm->shcenter[1][1],
                  hm->shcenter[1][2]);
        rot(hm->douspace.na[1], 'x');
        translate(-hm->center[KATA_D_R][0], -hm->center[KATA_D_R][1],
                  -hm->center[KATA_D_R][2]);
    ]
/*++++++++++++++++++++++++++++++++++++++++++++*/

    multmatrix(hm->mat[KATA_D_R]);

/*++++++++++++++++++++++++++++++++++++++++++++*/
if (showimp) [
        trSphyl(ipb[RUARM].ouc,ipb[RUARM].nc,ipb[RUARM].r1,
                ipb[RUARM].ol,G_black);
        /*trSphyl(ipb[RUARM].ouc,ipb[RUARM].nc,ipb[RUARM].sr1,
```

```
                ipb[RUARM].o1,G_pink);*/

        trSph(ipb[RUARM].r1,ipb[RUARM].o1,G_black,0);
        /*trSph(ipb[RUARM].sr1,ipb[RUARM].o1,G_pink,0);*/
        trSph(ipb[RUARM].r1,ipb[RUARM].o2,G_black,0);
        /*trSph(ipb[RUARM].sr1,ipb[RUARM].o2,G_pink,0);*/
]
/*++++++++++++++++++++++++++++++++++++++++++++*/

    DrawMap3Wire(hm, NINOUDE_R, hm->vs[NINOUDE_R], G_green);

    c3f(G_blue);
    DrawSaikoro(hm->center[HIJI_R][0], hm->center[HIJI_R][1],
                hm->center[HIJI_R][2], 1.0);
    DrawMap3Wire(hm, HIJI_R, hm->va[HIJI_R], G_red);
    multmatrix(hm->mat[HIJI_R]);

/*++++++++++++++++++++++++++++++++++++++++++++*/
if (showimp) {
        trSphyl(ipb[RLARM].ouc,ipb[RLARM].nc,ipb[RLARM].r1,
                ipb[RLARM].o1,G_black);
        /*trSphyl(ipb[RLARM].ouc,ipb[RLARM].nc,ipb[RLARM].sr1,
                ipb[RLARM].o1,G_pink);*/
}
/*++++++++++++++++++++++++++++++++++++++++++++*/

    c3f(G_blue);
    DrawSaikoro(hm->center[UDE_R][0], hm->center[UDE_R][1],
                hm->center[UDE_R][2], 1.0);
    DrawMap3Wire(hm, UDE_R, hm->va[UDE_R], G_green);
    multmatrix(hm->mat[UDE_R]);

{
Matrix mat;
float p[3];

getmatrix(mat);
coord_calc(p, hm->center[TE_R], mat);
printf("{p}:%f,%f,%f\n",p[0],p[1],p[2]);
}
    c3f(G_blue);
    DrawSaikoro(hm->center[TE_R][0], hm->center[TE_R][1],
                hm->center[TE_R][2], 1.0);
    DrawMap3Wire(hm, TE_R, hm->va[TE_R], G_red);
    multmatrix(hm->mat[TE_R]);

    pushmatrix();
    DrawYubiKansetuWire(hm, OYAYUBI_R_1);

    popmatrix();
    pushmatrix();
    DrawYubiKansetuWire(hm, HITOSASIYUBI_R_1);

    popmatrix();
    pushmatrix();
    DrawYubiKansetuWire(hm, NAKAYUBI_R_1);

    popmatrix();
    pushmatrix();
    DrawYubiKansetuWire(hm, KUSURIYUBI_R_1);

    popmatrix();
    DrawYubiKansetuWire(hm, KOYUBI_R_1);

    /*                  */
    /* 左手側データの表示 */
```



```
    /*                  */
    popmatrix();
    pushmatrix();
    c3f(G_blue);
    DrawSaikoro(hm->center[KATA_D_L][0], hm->center[KATA_D_L][1],
                hm->center[KATA_D_L][2], 1.0);
    DrawMap3Wire(hm, KATA_D_L, hm->va[KATA_D_L], G_red);
    DrawMap3Wire(hm, KATA_N_L, hm->va[KATA_N_L], G_red);

/*++++++++++++++++++++++++++++++++*/
    if (ffdswitch) {
        c3f(G_yellow);
        DrawSaikoro(hm->shcenter[0][0], hm->shcenter[0][1],
                    hm->shcenter[0][2], 1.0);
        translate(hm->shcenter[0][0], hm->shcenter[0][1],
                  hm->shcenter[0][2]);
        rot(hm->douspace.na[0], 'x');
        translate(-hm->center[KATA_D_L][0], -hm->center[KATA_D_L][1],
                  -hm->center[KATA_D_L][2]);
    ]
/*++++++++++++++++++++++++++++++++*/
if (collon) {
trSph(1.0,spcenter,G_blue,3);
}

    multmatrix(hm->mat[KATA_D_L]);

/*++++++++++++++++++++++++++++++++++++++++++++*/
if (showimp) {
        /*trSphyl(ipb[LUARM].ouc,ipb[LUARM].nc,ipb[LUARM].r1,
                ipb[LUARM].o1,G_yellow);*/
        trSphyl(ipb[LUARM].ouc,ipb[LUARM].nc,ipb[LUARM].sr1,
                ipb[LUARM].o1,G_black);

        /*trSph(ipb[LUARM].r1,ipb[LUARM].o1,G_black,0);*/
        trSph(ipb[LUARM].sr1,ipb[LUARM].o1,G_black,0);
        /*trSph(ipb[LUARM].r1,ipb[LUARM].o2,G_black,0);*/
        trSph(ipb[LUARM].sr1,ipb[LUARM].o2,G_black,0);
}
/*++++++++++++++++++++++++++++++++++++++++++++*/

    DrawMap3Wire(hm, NINOUDE_L, hm->vs[NINOUDE_L], G_green);

    c3f(G_blue);
    DrawSaikoro(hm->center[HIJI_L][0], hm->center[HIJI_L][1],
                hm->center[HIJI_L][2], 1.0);
    DrawMap3Wire(hm, HIJI_L, hm->va[HIJI_L], G_red);
    multmatrix(hm->mat[HIJI_L]);

    c3f(G_blue);
    DrawSaikoro(hm->center[UDE_L][0], hm->center[UDE_L][1],
                hm->center[UDE_L][2], 1.0);

/*++++++++++++++++++++++++++++++++++++++++++++*/
  /* trSph(1.0,spcenter,G_blue,3);*/

if (showimp) {
       /*trSphyl(ipb[LLARM].ouc,ipb[LLARM].nc,ipb[LLARM].r1,
                ipb[LLARM].o1,G_yellow);*/
        trSphyl(ipb[LLARM].ouc,ipb[LLARM].nc,ipb[LLARM].sr1,
                ipb[LLARM].o1,G_black);
}
/*++++++++++++++++++++++++++++++++++++++++++++*/

    DrawMap3Wire(hm, UDE_L, hm->va[UDE_L], G_green);
    multmatrix(hm->mat[UDE_L]);
```

```
    c3f(G_blue);
    DrawSaikoro(hm->center[TE_L][0], hm->center[TE_L][1],
                hm->center[TE_L][2], 1.0);
    DrawMap3Wire(hm, TE_L, hm->va[TE_L], G_red);
    multmatrix(hm->mat[TE_L]);
    DrawMap3Wire(hm, OYAYUBI_L_1, hm->vs[OYAYUBI_L_1], G_green);
    DrawMap3Wire(hm, OYAYUBI_L_2, hm->vs[OYAYUBI_L_2], G_green);
    DrawMap3Wire(hm, OYAYUBI_L_3, hm->vs[OYAYUBI_L_3], G_green);
    DrawMap3Wire(hm, HITOSASIYUBI_L_1, hm->vs[HITOSASIYUBI_L_1], G_green);
    DrawMap3Wire(hm, HITOSASIYUBI_L_2, hm->vs[HITOSASIYUBI_L_2], G_green);
    DrawMap3Wire(hm, HITOSASIYUBI_L_3, hm->vs[HITOSASIYUBI_L_3], G_green);
    DrawMap3Wire(hm, NAKAYUBI_L_1, hm->vs[NAKAYUBI_L_1], G_green);
    DrawMap3Wire(hm, NAKAYUBI_L_2, hm->vs[NAKAYUBI_L_2], G_green);
    DrawMap3Wire(hm, NAKAYUBI_L_3, hm->vs[NAKAYUBI_L_3], G_green);
    DrawMap3Wire(hm, KUSURIYUBI_L_1, hm->vs[KUSURIYUBI_L_1], G_green);
    DrawMap3Wire(hm, KUSURIYUBI_L_2, hm->vs[KUSURIYUBI_L_2], G_green);
    DrawMap3Wire(hm, KUSURIYUBI_L_3, hm->vs[KUSURIYUBI_L_3], G_green);
    DrawMap3Wire(hm, KOYUBI_L_1, hm->vs[KOYUBI_L_1], G_green);
    DrawMap3Wire(hm, KOYUBI_L_2, hm->vs[KOYUBI_L_2], G_green);
    DrawMap3Wire(hm, KOYUBI_L_3, hm->vs[KOYUBI_L_3], G_green);

    /*            */
    /* 頭部の表示 */
    /*            */
    popmatrix();
    pushmatrix();

    DrawMap3Wire(hm, KUBI, hm->va[KUBI], G_red);

/*++++++++++++++++++++++++++++++++++++*/
    if (ffdswitch) {
        translate(hm->douspace.center[3][0],hm->douspace.center[3][1],
                  hm->douspace.center[3][2]);
        rot(3*hm->douspace.xang, 'x');
        translate(-hm->douspace.corig[3][0],-hm->douspace.corig[3][1],
                  -hm->douspace.corig[3][2]);
    }
/*++++++++++++++++++++++++++++++++++++*/

    multmatrix(hm->mat[KUBI]);

/*++++++++++++++++++++++++++++++++++++*/
if (showimp) {
        trSph(ipb[HEAD].rl,ipb[HEAD].ol,G_black,0);
        /*trSph(ipb[HEAD].srl,ipb[HEAD].ol,G_pink,0);*/
}
/*++++++++++++++++++++++++++++++++++++*/

    DrawMap3Wire(hm, ATAMA, hm->va[ATAMA], G_green);

/***
    if (hm->muney > 5.0) {
****/

    popmatrix();
/*++++++++++++++++++++++++++++++++++++++++++++++*/
    if (ffdswitch) {
        multmatrix(hm->kosi_mat);
        drawSpace(&hm->kosispace);
        DrawMap3Wire(hm, KOSI, hm->va[KOSI], G_red);
/*
rev_mat(hm->kosi_mat, mat);
multmatrix(mat);
*/
    }
```



```
/*++++++++++++++++++++++++++++++++++++++++++++++*/
    else {
        DrawMap3Wire(hm, KOSI, hm->va[KOSI], G_red);
        multmatrix(hm->mat[KOSI]);
    }

    if (fsqrt(hm->kosix*hm->kosix +
        (hm->kosiy+hm->ske_real.dou_len)*(hm->kosiy+hm->ske_real.dou_len) +
        hm->kosiz*hm->kosiz) > 3.0) {
printf("AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\n");
        translate(hm->kosix, hm->kosiy, hm->kosiz);

printf("\n");
printf("hm->kosi : %f, %f, %f \n", hm->kosix, hm->kosiy, hm->kosiz);

    } else {
printf("BBBBBBBBBBBBBBBBBBBBBBBBBBBBBB\n");
#ifdef HIROSE
        translate(0.0, -hm->ske_real.dou_len, 0.0);
#endif
    }

    pushmatrix();
#if 1
    DrawMap3Wire(hm, SIRI_R, hm->va[SIRI_R], G_red);
#endif
    multmatrix(hm->mat[SIRI_R]);

#if 1
    DrawMap3Wire(hm, MOMO_R, hm->vs[MOMO_R], G_green);
#endif
    DrawMap3Wire(hm, HIZA_R, hm->va[HIZA_R], G_red);

/*++++++++++++++++++++++++++++++++++++++++++++++*/
if (showimp) {
       trSphyl(ipb[RULG].ouc,ipb[RULG].nc,ipb[RULG].rl,
               ipb[RULG].o2,G_black);
       /*trSphyl(ipb[RULG].ouc,ipb[RULG].nc,ipb[RULG].srl,
               ipb[RULG].o2,G_pink);*/
       trSph(ipb[RULG].rl,ipb[RULG].o1,G_black,0);
       trSph(ipb[RULG].rl,ipb[RULG].o2,G_black,0);
}
/*++++++++++++++++++++++++++++++++++++++++++++++*/

    multmatrix(hm->mat[HIZA_R]);

/*++++++++++++++++++++++++++++++++++++++++++++++*/
if (showimp) {
       trSphyl(ipb[RLLG].ouc,ipb[RLLG].nc,ipb[RLLG].rl,
               ipb[RLLG].o2,G_black);
       /*trSphyl(ipb[RLLG].ouc,ipb[RLLG].nc,ipb[RLLG].srl,
               ipb[RLLG].o2,G_pink);*/
}
/*++++++++++++++++++++++++++++++++++++++++++++++*/

    DrawMap3Wire(hm, SUNE_R, hm->vs[SUNE_R], G_green);
    multmatrix(hm->mat[KUTU_R]);
    DrawMap3Wire(hm, KUTU_R, hm->vs[KUTU_R], G_green);

    popmatrix();
#if 1
    DrawMap3Wire(hm, SIRI_L, hm->va[SIRI_L], G_red);
#endif
    multmatrix(hm->mat[SIRI_L]);

#if 1
```

```
    DrawMap3Wire(hm, MOMO_L, hm->vs[MOMO_L], G_green);
#endif
    DrawMap3Wire(hm, HIZA_L, hm->va[HIZA_L], G_red);

/*++++++++++++++++++++++++++++++++++++++++++++*/
if (showimp) {
       /*trSphyl(ipb[LULG].ouc,ipb[LULG].nc,ipb[LULG].r1,
                ipb[LULG].o2,G_yellow);*/
       trSphyl(ipb[LULG].ouc,ipb[LULG].nc,ipb[LULG].sr1,
                ipb[LULG].o2,G_black);
       trSph(ipb[LULG].sr1,ipb[LULG].o1,G_black,0);
       trSph(ipb[LULG].sr1,ipb[LULG].o2,G_black,0);
]
/*++++++++++++++++++++++++++++++++++++++++++++*/

    multmatrix(hm->mat[HIZA_L]);

/*++++++++++++++++++++++++++++++++++++++++++++*/
if (showimp) {
       /*trSphyl(ipb[LLLG].ouc,ipb[LLLG].nc,ipb[LLLG].r1,
                ipb[LLLG].o2,G_yellow);*/
       trSphyl(ipb[LLLG].ouc,ipb[LLLG].nc,ipb[LLLG].sr1,
                ipb[LLLG].o2,G_black);
]
/*++++++++++++++++++++++++++++++++++++++++++++*/

    DrawMap3Wire(hm, SUNE_L, hm->vs[SUNE_L], G_green);
    multmatrix(hm->mat[KUTU_L]);
    DrawMap3Wire(hm, KUTU_L, hm->vs[KUTU_L], G_green);
}


/*
 * 関数名 : Draw3DWireSeiza3()
 *
 * 引数   : なし
 *
 * 戻り値 : なし
 *
 * 機能   : 上半身モデルのワイヤー表示
 *
 * 履歴   : 1995年12月22日 ; 作成 ; 広瀬＼欽 */
void
Draw3DWireSeiza3(JINBUTU *hm, int cup, float *p)
{
    float               dumy;
    Matrix      mat;

    if (hm->stand) {
        Draw3DWire(hm, 0, &dumy);
        return;
    }

    /* 体の位置を上方にずらす */
    rot(180.0, 'y');
    translate(0.0, -hm->center[DOU][1], 0.0);

prt_mat(hm->mat[KOSI], "hm->mat[KOSI]");
prt_mat(hm->mune_mat, "hm->mune_mat");
prt_mat(hm->mat1.mune, "hm->mat1.mune");

    if(!trkswitch) {
        multmatrix(hm->mat[KOSI]);
        /* multmatrix(hm->mune_mat); */
    }
```



```
/***
    if (hm->muney > 5.0) {
****/

    if (fsqrt(hm->kosix*hm->kosix +
        (hm->kosiy+hm->ske_real.dou_len)*(hm->kosiy+hm->ske_real.dou_len) +
        hm->kosiz*hm->kosiz) > 3.0) {
        translate(hm->kosix, hm->kosiy, hm->kosiz);

printf("\n");
printf("hm->kosi : %f, %f, %f \n", hm->kosix, hm->kosiy, hm->kosiz);

    } else {
        translate(0.0, -hm->ske_real.dou_len, 0.0);
    }

    /* 胸部の表示 */
/*++++++++++++++++++++++++++++++++++++++++++++*/
    if (ffdswitch) {
        multmatrix(hm->mune_mat); /* kosi wo chuusintosita kosi(mune) no kaiten */
        drawSpace(&hm->douspace);
        DrawMap3Wire(hm, DOU, hm->va[DOU], G_black);
        pushmatrix();
        rev_mat(hm->mune_mat, mat);
        multmatrix(mat);
    } else {
        multmatrix(hm->mat[KOSI]);
        DrawMap3Wire(hm, DOU, hm->vs[DOU], G_green);
        pushmatrix();
        rev_mat(hm->mat[KOSI], mat);
        multmatrix(mat);
    }
/*++++++++++++++++++++++++++++++++++++++++++++*/

    pushmatrix();
    DrawMap3Wire(hm, SIRI_R, hm->va[SIRI_R], G_red);
    multmatrix(hm->mat[SIRI_R]);

    DrawMap3Wire(hm, MOMO_R, hm->vs[MOMO_R], G_green);
    DrawMap3Wire(hm, HIZA_R, hm->va[HIZA_R], G_red);

/*++++++++++++++++++++++++++++++++++++++++++++*/
if (showimp) {
       trSphyl(ipb[RULG].ouc,ipb[RULG].nc,ipb[RULG].r1,
                ipb[RULG].o2,G_black);
       /*trSphyl(ipb[RULG].ouc,ipb[RULG].nc,ipb[RULG].sr1,
                ipb[RULG].o2,G_pink);*/
       trSph(ipb[RULG].r1,ipb[RULG].o1,G_black,0);
       trSph(ipb[RULG].r1,ipb[RULG].o2,G_black,0);
}
/*++++++++++++++++++++++++++++++++++++++++++++*/

    multmatrix(hm->mat[HIZA_R]);

/*++++++++++++++++++++++++++++++++++++++++++++*/
if (showimp) {
       trSphyl(ipb[RLLG].ouc,ipb[RLLG].nc,ipb[RLLG].r1,
                ipb[RLLG].o2,G_black);
       /*trSphyl(ipb[RLLG].ouc,ipb[RLLG].nc,ipb[RLLG].sr1,
                ipb[RLLG].o2,G_pink);*/
}
/*++++++++++++++++++++++++++++++++++++++++++++*/

    DrawMap3Wire(hm, SUNE_R, hm->vs[SUNE_R], G_green);
    multmatrix(hm->mat[KUTU_R]);
    DrawMap3Wire(hm, KUTU_R, hm->vs[KUTU_R], G_green);
```

```
    popmatrix();
    pushmatrix();
    DrawMap3Wire(hm, SIRI_L, hm->va[SIRI_L], G_red);
    multmatrix(hm->mat[SIRI_L]);

    DrawMap3Wire(hm, MOMO_L, hm->vs[MOMO_L], G_green);
    DrawMap3Wire(hm, HIZA_L, hm->va[HIZA_L], G_red);

/*++++++++++++++++++++++++++++++++++++++++++++*/
if (showimp) {
       /*trSphyl(ipb[LULG].ouc,ipb[LULG].nc,ipb[LULG].r1,
               ipb[LULG].o2,G_yellow);*/
       trSphyl(ipb[LULG].ouc,ipb[LULG].nc,ipb[LULG].sr1,
               ipb[LULG].o2,G_black);
       trSph(ipb[LULG].sr1,ipb[LULG].o1,G_black,0);
       trSph(ipb[LULG].sr1,ipb[LULG].o2,G_black,0);
}
/*++++++++++++++++++++++++++++++++++++++++++++*/

    multmatrix(hm->mat[HIZA_L]);

/*++++++++++++++++++++++++++++++++++++++++++++*/
if (showimp) [
       /*trSphyl(ipb[LLLG].ouc,ipb[LLLG].nc,ipb[LLLG].r1,
               ipb[LLLG].o2,G_yellow);*/
       trSphyl(ipb[LLLG].ouc,ipb[LLLG].nc,ipb[LLLG].sr1,
               ipb[LLLG].o2,G_black);
]
/*++++++++++++++++++++++++++++++++++++++++++++*/

    DrawMap3Wire(hm, SUNE_L, hm->vs[SUNE_L], G_green);
    multmatrix(hm->mat[KUTU_L]);
    DrawMap3Wire(hm, KUTU_L, hm->vs[KUTU_L], G_green);

    popmatrix();
/*++++++++++++++++++++++++++++++++++++++++++++*/
    if (ffdswitch) {
        drawSpace(&hm->kosispace);
    ]
/*++++++++++++++++++++++++++++++++++++++++++++*/
    DrawMap3Wire(hm, KOSI, hm->va[KOSI], G_red);

    popmatrix(); /* mune wo hyouji sita tokino matrix */

    /* cup
    if (cup == 1) [
        GetTukamiKakudo(hm, p, 1);
    ]
    */

    /*                      */
    /* 右手側データの表示 */
    /*                      */
    pushmatrix();
    c3f(G_blue);
    DrawSaikoro(hm->center[KATA_D_R][0], hm->center[KATA_D_R][1],
                hm->center[KATA_D_R][2], 1.0);
    DrawMap3Wire(hm, KATA_D_R, hm->va[KATA_D_R], G_red);
    DrawMap3Wire(hm, KATA_N_R, hm->va[KATA_N_R], G_red);

/*++++++++++++++++++++++++++++++++++++++++++++*/
    if (ffdswitch) {
        c3f(G_yellow);
        DrawSaikoro(hm->shcenter[1][0], hm->shcenter[1][1],
                    hm->shcenter[1][2], 1.0);
```



```
        translate(hm->shcenter[1][0], hm->shcenter[1][1],
                  hm->shcenter[1][2]);
        rot(hm->douspace.na[1], 'x');
        translate(-hm->center[KATA_D_R][0], -hm->center[KATA_D_R][1],
                  -hm->center[KATA_D_R][2]);
    ]
/*++++++++++++++++++++++++++++++++++++++++++++*/

    multmatrix(hm->mat[KATA_D_R]);

/*++++++++++++++++++++++++++++++++++++++++++++*/
if (showimp) {
        trSphyl(ipb[RUARM].ouc,ipb[RUARM].nc,ipb[RUARM].r1,
                ipb[RUARM].o1,G_black);
        /*trSphyl(ipb[RUARM].ouc,ipb[RUARM].nc,ipb[RUARM].sr1,
                ipb[RUARM].o1,G_pink);*/

        trSph(ipb[RUARM].r1,ipb[RUARM].o1,G_black,0);
        /*trSph(ipb[RUARM].sr1,ipb[RUARM].o1,G_pink,0);*/
        trSph(ipb[RUARM].r1,ipb[RUARM].o2,G_black,0);
        /*trSph(ipb[RUARM].sr1,ipb[RUARM].o2,G_pink,0);*/
}
/*++++++++++++++++++++++++++++++++++++++++++++*/

    DrawMap3Wire(hm, NINOUDE_R, hm->vs[NINOUDE_R], G_green);

    c3f(G_blue);
    DrawSaikoro(hm->center[HIJI_R][0], hm->center[HIJI_R][1],
                hm->center[HIJI_R][2], 1.0);
    DrawMap3Wire(hm, HIJI_R, hm->va[HIJI_R], G_red);
    multmatrix(hm->mat[HIJI_R]);

/*++++++++++++++++++++++++++++++++++++++++++++*/
if (showimp) [
        trSphyl(ipb[RLARM].ouc,ipb[RLARM].nc,ipb[RLARM].r1,
                ipb[RLARM].o1,G_black);
        /*trSphyl(ipb[RLARM].ouc,ipb[RLARM].nc,ipb[RLARM].sr1,
                ipb[RLARM].o1,G_pink);*/
]
/*++++++++++++++++++++++++++++++++++++++++++++*/

    c3f(G_blue);
    DrawSaikoro(hm->center[UDE_R][0], hm->center[UDE_R][1],
                hm->center[UDE_R][2], 1.0);
    DrawMap3Wire(hm, UDE_R, hm->va[UDE_R], G_green);
    multmatrix(hm->mat[UDE_R]);

{
Matrix mat;
float p[3];

getmatrix(mat);
coord_calc(p, hm->center[TE_R], mat);
printf("{p}:%f,%f,%f\n",p[0],p[1],p[2]);
}
    c3f(G_blue);
    DrawSaikoro(hm->center[TE_R][0], hm->center[TE_R][1],
                hm->center[TE_R][2], 1.0);
    DrawMap3Wire(hm, TE_R, hm->va[TE_R], G_red);
    multmatrix(hm->mat[TE_R]);

    pushmatrix();
    DrawYubiKansetuWire(hm, OYAYUBI_R_1);

    popmatrix();
    pushmatrix();
```

```
    DrawYubiKansetuWire(hm, HITOSASIYUBI_R_1);

    popmatrix();
    pushmatrix();
    DrawYubiKansetuWire(hm, NAKAYUBI_R_1);

    popmatrix();
    pushmatrix();
    DrawYubiKansetuWire(hm, KUSURIYUBI_R_1);

    popmatrix();
    DrawYubiKansetuWire(hm, KOYUBI_R_1);

    /*                      */
    /* 左手側データの表示 */
    /*                      */
    popmatrix();
    pushmatrix();
    c3f(G_blue);
    DrawSaikoro(hm->center[KATA_D_L][0], hm->center[KATA_D_L][1],
                hm->center[KATA_D_L][2], 1.0);
    DrawMap3Wire(hm, KATA_D_L, hm->va[KATA_D_L], G_red);
    DrawMap3Wire(hm, KATA_N_L, hm->va[KATA_N_L], G_red);

/*++++++++++++++++++++++++++++++++*/
    if (ffdswitch) {
        c3f(G_yellow);
        DrawSaikoro(hm->shcenter[0][0], hm->shcenter[0][1],
                    hm->shcenter[0][2], 1.0);
        translate(hm->shcenter[0][0], hm->shcenter[0][1],
                  hm->shcenter[0][2]);
        rot(hm->douspace.na[0], 'x');
        translate(-hm->center[KATA_D_L][0], -hm->center[KATA_D_L][1],
                  -hm->center[KATA_D_L][2]);
    }
/*++++++++++++++++++++++++++++++++*/
if (collon) {
trSph(1.0,spcenter,G_blue,3);
}

    multmatrix(hm->mat[KATA_D_L]);

/*++++++++++++++++++++++++++++++++++++++++++++++*/
if (showimp) {
        /*trSphyl(ipb[LUARM].ouc,ipb[LUARM].nc,ipb[LUARM].r1,
                ipb[LUARM].o1,G_yellow);*/
        trSphyl(ipb[LUARM].ouc,ipb[LUARM].nc,ipb[LUARM].sr1,
                ipb[LUARM].o1,G_black);

        /*trSph(ipb[LUARM].r1,ipb[LUARM].o1,G_black,0);*/
        trSph(ipb[LUARM].sr1,ipb[LUARM].o1,G_black,0);
        /*trSph(ipb[LUARM].r1,ipb[LUARM].o2,G_black,0);*/
        trSph(ipb[LUARM].sr1,ipb[LUARM].o2,G_black,0);
}
/*++++++++++++++++++++++++++++++++++++++++++++++*/

    DrawMap3Wire(hm, NINOUDE_L, hm->vs[NINOUDE_L], G_green);

    c3f(G_blue);
    DrawSaikoro(hm->center[HIJI_L][0], hm->center[HIJI_L][1],
                hm->center[HIJI_L][2], 1.0);
    DrawMap3Wire(hm, HIJI_L, hm->va[HIJI_L], G_red);
    multmatrix(hm->mat[HIJI_L]);

    c3f(G_blue);
    DrawSaikoro(hm->center[UDE_L][0], hm->center[UDE_L][1],
```



```
                hm->center[UDE_L][2], 1.0);

/*++++++++++++++++++++++++++++++++++++++++++++++*/
  /* trSph(1.0,spcenter,G_blue,3);*/

if (showimp) {
        /*trSphyl(ipb[LLARM].ouc,ipb[LLARM].nc,ipb[LLARM].r1,
                ipb[LLARM].o1,G_yellow);*/
        trSphyl(ipb[LLARM].ouc,ipb[LLARM].nc,ipb[LLARM].sr1,
                ipb[LLARM].o1,G_black);
}
/*++++++++++++++++++++++++++++++++++++++++++++++*/

    DrawMap3Wire(hm, UDE_L, hm->va[UDE_L], G_green);
    multmatrix(hm->mat[UDE_L]);

    c3f(G_blue);
    DrawSaikoro(hm->center[TE_L][0], hm->center[TE_L][1],
                hm->center[TE_L][2], 1.0);
    DrawMap3Wire(hm, TE_L, hm->va[TE_L], G_red);
    multmatrix(hm->mat[TE_L]);
    DrawMap3Wire(hm, OYAYUBI_L_1, hm->vs[OYAYUBI_L_1], G_green);
    DrawMap3Wire(hm, OYAYUBI_L_2, hm->vs[OYAYUBI_L_2], G_green);
    DrawMap3Wire(hm, OYAYUBI_L_3, hm->vs[OYAYUBI_L_3], G_green);
    DrawMap3Wire(hm, HITOSASIYUBI_L_1, hm->vs[HITOSASIYUBI_L_1], G_green);
    DrawMap3Wire(hm, HITOSASIYUBI_L_2, hm->vs[HITOSASIYUBI_L_2], G_green);
    DrawMap3Wire(hm, HITOSASIYUBI_L_3, hm->vs[HITOSASIYUBI_L_3], G_green);
    DrawMap3Wire(hm, NAKAYUBI_L_1, hm->vs[NAKAYUBI_L_1], G_green);
    DrawMap3Wire(hm, NAKAYUBI_L_2, hm->vs[NAKAYUBI_L_2], G_green);
    DrawMap3Wire(hm, NAKAYUBI_L_3, hm->vs[NAKAYUBI_L_3], G_green);
    DrawMap3Wire(hm, KUSURIYUBI_L_1, hm->vs[KUSURIYUBI_L_1], G_green);
    DrawMap3Wire(hm, KUSURIYUBI_L_2, hm->vs[KUSURIYUBI_L_2], G_green);
    DrawMap3Wire(hm, KUSURIYUBI_L_3, hm->vs[KUSURIYUBI_L_3], G_green);
    DrawMap3Wire(hm, KOYUBI_L_1, hm->vs[KOYUBI_L_1], G_green);
    DrawMap3Wire(hm, KOYUBI_L_2, hm->vs[KOYUBI_L_2], G_green);
    DrawMap3Wire(hm, KOYUBI_L_3, hm->vs[KOYUBI_L_3], G_green);

    /*            */
    /* 頭部の表示 */
    /*            */
    popmatrix();

    DrawMap3Wire(hm, KUBI, hm->va[KUBI], G_red);

/*++++++++++++++++++++++++++++++++++++*/
    if (ffdswitch) {
        translate(hm->douspace.center[3][0],hm->douspace.center[3][1],
                  hm->douspace.center[3][2]);
        rot(3*hm->douspace.xang, 'x');
        translate(-hm->douspace.corig[3][0],-hm->douspace.corig[3][1],
                  -hm->douspace.corig[3][2]);
    }
/*++++++++++++++++++++++++++++++++++++*/

    multmatrix(hm->mat[KUBI]);

/*++++++++++++++++++++++++++++++++++++*/
if (showimp) {
        trSph(ipb[HEAD].r1,ipb[HEAD].o1,G_black,0);
        /*trSph(ipb[HEAD].sr1,ipb[HEAD].o1,G_pink,0);*/
}
/*++++++++++++++++++++++++++++++++++++*/

    DrawMap3Wire(hm, ATAMA, hm->va[ATAMA], G_green);
}
```

```
/*
 * 関数名 : DrawYubiKansetuWire(int yubi, Matrix mat)
 *
 * 引数   : yubi : 指の部品番号
 *          mat  : 回転マトリックス
 *
 * 戻り値 : なし
 *
 * 機能   : 右手指のワイヤー表示
 *
 * 履歴   : 1992年10月28日 ; 作成 ; 広瀬  正俊
 */
void
DrawYubiKansetuWire(JINBUTU *hm, int yubi)
{
    DrawMap3Wire(hm, yubi, hm->va[yubi], G_green);
    multmatrix(hm->mat[yubi]);
    DrawMap3Wire(hm, yubi+1, hm->va[yubi+1], G_green);
    multmatrix(hm->mat[yubi+1]);
    DrawMap3Wire(hm, yubi+2, hm->va[yubi+2], G_green);
}

/*
 * 関数名 : DrawMap3Wire(int buhin, VERTEX *g_v, float *lineclr)
 *
 * 引数   : buhin   : 部品番号
 *          g_v     : 頂点座標
 *          lineclr : ワイヤーの色情報
 *
 * 戻り値 : なし
 *
 * 機能   : 右手指のワイヤー表示
 *
 * 履歴   : 1992年10月28日 ; 作成 ; 広瀬  正俊
 */
void
DrawMap3Wire(JINBUTU *hm, int buhin, VERTEX *g_v, float *lineclr)
{
    int         i, *pol;

    c3f(lineclr);

    for(i = 0; i < hm->polmax[buhin]; i++) {
        pol = (int *)(hm->poly[buhin]+i);
        bgnclosedline();
        v3f((float *)(g_v+pol[0]));
        v3f((float *)(g_v+pol[1]));
        v3f((float *)(g_v+pol[2]));
        endclosedline();
    }
}

void
DrawSaikoro(float cx, float cy, float cz, float w)
{
    float       v[8][3];
    static int  cn[6][4] = [ 0, 3, 2, 1,  4, 5, 6, 7,  0, 1, 5, 4,
                             1, 2, 6, 5,  2, 3, 7, 6,  3, 0, 4, 7
                           ];
    int         i;

    v[0][0] = v[3][0] = v[4][0] = v[7][0] = cx - w;
    v[1][0] = v[2][0] = v[5][0] = v[6][0] = cx + w;
    v[0][1] = v[1][1] = v[4][1] = v[5][1] = cy - w;
    v[2][1] = v[3][1] = v[6][1] = v[7][1] = cy + w;
```



```
    v[0][2] = v[1][2] = v[2][2] = v[3][2] = cz - w;
    v[4][2] = v[5][2] = v[6][2] = v[7][2] = cz + w;

    for (i = 0; i < 6; i++) {
        bgnpolygon();
        v3f(v[cn[i][0]]);
        v3f(v[cn[i][1]]);
        v3f(v[cn[i][2]]);
        v3f(v[cn[i][3]]);
        endpolygon();
    }
}


/*
 * 関数名 : void DrawSimple3DWire()
 *
 * 引数   : なし
 *
 * 戻り値 : なし
 *
 * 機能   : 上半身モデルのワイヤー表示
 *
 * 履歴   : 1993年7月30日 ; 作成 ; 広瀬＼敦 */
void
DrawSimple3DWire(JINBUTU *hm)
{
    pushmatrix();

    /* 体の位置を上方にずらす */
    translate(0.0, -hm->center[DOU][1], 0.0);

    /* 胸部の表示 */
    multmatrix(hm->mat[DOU]);
    DrawMap3Wire(hm, DOU, hm->vs[DOU], G_green);

    /*                    */
    /* 右手側データの表示 */
    /*                    */
    pushmatrix();

    DrawMap3Wire(hm, KATA_D_R, hm->vs[KATA_D_R], G_red);
    multmatrix(hm->mat[KATA_D_R]);
    DrawMap3Wire(hm, NINOUDE_R, hm->vs[NINOUDE_R], G_green);

    DrawMap3Wire(hm, HIJI_R, hm->vs[HIJI_R], G_red);
    multmatrix(hm->mat[HIJI_R]);
    DrawMap3Wire(hm, UDE_R, hm->vs[UDE_R], G_green);

    DrawMap3Wire(hm, TE_R, hm->va[TE_R], G_red);

/* 11.2 (ochi)
    multmatrix(hm->mat[TE_R]);
*/
/*****
    DrawMap3Wire(hm, OYAYUBI_R_1, hm->vs[OYAYUBI_R_1], G_green);
*****/

    pushmatrix();
    DrawMap3Wire(hm, OYAYUBI_R_1, hm->va[OYAYUBI_R_1], G_red);
    multmatrix(hm->mat[OYAYUBI_R_1]);
    DrawMap3Wire(hm, OYAYUBI_R_1+1, hm->va[OYAYUBI_R_1+1], G_red);
    multmatrix(hm->mat[OYAYUBI_R_1+1]);
    DrawMap3Wire(hm, OYAYUBI_R_1+2, hm->va[OYAYUBI_R_1+2], G_red);

    popmatrix();
```

```
    pushmatrix();
    DrawMap3Wire(hm, HITOSASIYUBI_R_1, hm->va[HITOSASIYUBI_R_1], G_red);
    multmatrix(hm->mat[HITOSASIYUBI_R_1]);
    DrawMap3Wire(hm, HITOSASIYUBI_R_1+1, hm->va[HITOSASIYUBI_R_1+1], G_red);
    multmatrix(hm->mat[HITOSASIYUBI_R_1+1]);
    DrawMap3Wire(hm, HITOSASIYUBI_R_1+2, hm->va[HITOSASIYUBI_R_1+2], G_red);

    popmatrix();
    pushmatrix();
    DrawMap3Wire(hm, NAKAYUBI_R_1, hm->va[NAKAYUBI_R_1], G_red);
    multmatrix(hm->mat[NAKAYUBI_R_1]);
    DrawMap3Wire(hm, NAKAYUBI_R_1+1, hm->va[NAKAYUBI_R_1+1], G_red);
    multmatrix(hm->mat[NAKAYUBI_R_1+1]);
    DrawMap3Wire(hm, NAKAYUBI_R_1+2, hm->va[NAKAYUBI_R_1+2], G_red);

    popmatrix();
    pushmatrix();
    DrawMap3Wire(hm, KUSURIYUBI_R_1, hm->va[KUSURIYUBI_R_1], G_red);
    multmatrix(hm->mat[KUSURIYUBI_R_1]);
    DrawMap3Wire(hm, KUSURIYUBI_R_1+1, hm->va[KUSURIYUBI_R_1+1], G_red);
    multmatrix(hm->mat[KUSURIYUBI_R_1+1]);
    DrawMap3Wire(hm, KUSURIYUBI_R_1+2, hm->va[KUSURIYUBI_R_1+2], G_red);

    popmatrix();
    DrawMap3Wire(hm, KOYUBI_R_1, hm->va[KOYUBI_R_1], G_red);
    multmatrix(hm->mat[KOYUBI_R_1]);
    DrawMap3Wire(hm, KOYUBI_R_1+1, hm->va[KOYUBI_R_1+1], G_red);
    multmatrix(hm->mat[KOYUBI_R_1+1]);
    DrawMap3Wire(hm, KOYUBI_R_1+2, hm->va[KOYUBI_R_1+2], G_red);

    /*                      */
    /* 左手側データの表示 */
    /*                      */
    popmatrix();
    pushmatrix();

    DrawMap3Wire(hm, KATA_D_L, hm->vs[KATA_D_L], G_red);
    multmatrix(hm->mat[KATA_D_L]);
    DrawMap3Wire(hm, NINOUDE_L, hm->vs[NINOUDE_L], G_green);

    DrawMap3Wire(hm, HIJI_L, hm->vs[HIJI_L], G_red);
    multmatrix(hm->mat[HIJI_L]);
    DrawMap3Wire(hm, UDE_L, hm->vs[UDE_L], G_green);

    DrawMap3Wire(hm, TE_L, hm->vs[TE_L], G_red);
    multmatrix(hm->mat[TE_L]);
    DrawMap3Wire(hm, OYAYUBI_L_1, hm->vs[OYAYUBI_L_1], G_green);

    /*              */
    /* 頭部の表示 */
    /*              */
    popmatrix();

    DrawMap3Wire(hm, KUBI, hm->vs[KUBI], G_red);
    multmatrix(hm->mat[KUBI]);
    DrawMap3Wire(hm, ATAMA, hm->vs[ATAMA], G_green);

    popmatrix();
}


/* 1993.2.5 */
#if 0
static VERTEX   *G_n[MAX_BUHIN];        /* normal */
static VERTEX   *G_n[MAX_BUHIN];        /* normal */
```



```
/*
 * 関数名 : DrawMap3Wire(int buhin, VERTEX *g_v, float *linec1r)
 *
 * 引数   : buhin   : 部品番号
 *          g_v     : 頂点座標
 *          linec1r : ワイヤーの色情報
 *
 * 戻り値 : なし
 *
 * 機能   : 右手指のワイヤー表示
 *
 * 履歴   : 1992年10月28日 ; 作成 ; 広瀬  正俊
 */
void
DrawMap3Shade(buhin, g_v, linec1r)
    int         buhin;
    VERTEX      *g_v;
    float       *linec1r;
{
    int         i, *pol;

    c3f(linec1r);

    for(i = 0; i < G_polmax[buhin]; i++) {
        pol = (int *)(G_pol[buhin]+i);
        bgnpolygon();
        v3f((float *)(g_v+pol[0]));
        v3f((float *)(g_v+pol[1]));
        v3f((float *)(g_v+pol[2]));
        endpolygon();
    }
}

/*
 * 関数名 : CalcNormal()
 *
 * 引数   : なし
 *
 * 戻り値 : なし
 *
 * 機能   : ・頂点の法線ベクトルを求める
 *
 * 履歴   : 1992年10月28日 ; 作成 ; 広瀬  正俊
 */
void
CalcNormal()
{
    PATCHPTR            ptr;
    NODES3PTR           tmp;
    float               vtx[3][3];
    int                 lg, lt;
    int                 i;

    /* counter initialize */
    for(tmp = G_Snod3; tmp != NULL; tmp = tmp->next) {
        tmp->con = 0;
    }

    /* face normal vector */
    for(ptr = G_Sptc; ptr != NULL; ptr = ptr->next) {
        vtx[0][0] = (ptr->nod3[0])->xyz[0];
        vtx[0][1] = (ptr->nod3[0])->xyz[1];
        vtx[0][2] = (ptr->nod3[0])->xyz[2];
        (ptr->nod3[0])->co[(ptr->nod3[0])->con] = (unsigned long)ptr;
        (ptr->nod3[0])->con++;
        vtx[1][0] = (ptr->nod3[2])->xyz[0];
```

```
        vtx[1][1] = (ptr->nod3[2])->xyz[1];
        vtx[1][2] = (ptr->nod3[2])->xyz[2];
        (ptr->nod3[2])->co[(ptr->nod3[2])->con] = (unsigned long)ptr;
        (ptr->nod3[2])->con++;
        vtx[2][0] = (ptr->nod3[1])->xyz[0];
        vtx[2][1] = (ptr->nod3[1])->xyz[1];
        vtx[2][2] = (ptr->nod3[1])->xyz[2];
        (ptr->nod3[1])->co[(ptr->nod3[1])->con] = (unsigned long)ptr;
        (ptr->nod3[1])->con++;
        FaceNormal(vtx, ptr->normal);
    }

    /* point normal vector */
    for(tmp = G_Snod3; tmp != NULL; tmp = tmp->next) {
        tmp->normal[0] = 0.0;
        tmp->normal[1] = 0.0;
        tmp->normal[2] = 0.0;
        for(i = 0; i < tmp->con; i++) {
            tmp->normal[0] += ((PATCHPTR)(tmp->co[i]))->normal[0];
            tmp->normal[1] += ((PATCHPTR)(tmp->co[i]))->normal[1];
            tmp->normal[2] += ((PATCHPTR)(tmp->co[i]))->normal[2];
        }
        tmp->normal[0] /= (float)tmp->con;
        tmp->normal[1] /= (float)tmp->con;
        tmp->normal[2] /= (float)tmp->con;
    }
}
#endif
```

```
/*****************************************************************
*       ファイル名 : get_all_mat.c                               *
*****************************************************************/
/*
* get_tr_mat( trdata, chmat )                                    *
*       トラッカーデータから回転マトリックスを作成               *
*                                                                *
* toritsuke_mat( num, orgmat, chgmat )                           *
*       トラッカーの取り付け位置を基準とした                     *
*       回転マトリックスを作成                                   *
*                                                                *
* world_disp_mat( wrdmat, dispmat )                              *
*       回転マトリックスをワールド座標から表示系に変換           *
*                                                                *
* disp_world_mat( dispmat, wrdmat )                              *
*       回転マトリックスを表示系からワールド座標に変換           *
*/

#include        <gl/gl.h>

#include        "body.h"
#include        "body_draw.h"

#include        "rname_fc.h"

/* トラッカーの取り付け位置による回転補正角度 */
/* 資料の「磁気センサーの取り付け位置について」を参照 */

#define         MUNE_ROT_X      0.0     /* 胸のX軸回転補正 */
#define         MUNE_ROT_Y      0.0     /* 胸のY軸回転補正 */
#define         MUNE_ROT_Z      90.0    /* 胸のZ軸回転補正 */

#define         ATAMA_ROT_X     0.0     /* 頭のX軸回転補正 */
#define         ATAMA_ROT_Y     90.0    /* 頭のY軸回転補正 */
#define         ATAMA_ROT_Z     0.0     /* 頭のZ軸回転補正 */

#define         RTEKUBI_ROT_X   90.0    /* 右手首のX軸回転補正 */
#define         RTEKUBI_ROT_Y   0.0     /* 右手首のY軸回転補正 */
#define         RTEKUBI_ROT_Z   0.0     /* 右手首のZ軸回転補正 */

#define         LTEKUBI_ROT_X   90.0    /* 左手首のX軸回転補正 */
#define         LTEKUBI_ROT_Y   180.0   /* 左手首のY軸回転補正 */
#define         LTEKUBI_ROT_Z   0.0     /* 左手首のZ軸回転補正 */

static float    tr_zure[4][3] = {
                        MUNE_ROT_X, MUNE_ROT_Y, MUNE_ROT_Z,
                        ATAMA_ROT_X, ATAMA_ROT_Y, ATAMA_ROT_Z,
                        RTEKUBI_ROT_X, RTEKUBI_ROT_Y, RTEKUBI_ROT_Z,
                        LTEKUBI_ROT_X, LTEKUBI_ROT_Y, LTEKUBI_ROT_Z };

/*****************************************************************
*       トラッカーデータから回転マトリックスを作成               *
*                                                                *
*       引数 : trdata (トラッカーデータ)                         *
*              chmat (回転マトリックス)                          *
*       リターン値 : なし                                        *
*****************************************************************/
void
get_tr_mat( trdata, chmat )
float   trdata[6];
Matrix  chmat;
{
        float   rx, ry, rz;

        loadmatrix( idmat );
        translate( trdata[0], trdata[1], trdata[2] );
        rot( trdata[3], 'z');
        rot( trdata[5], 'y');
        rot( trdata[4], 'x');

                        /* 生の回転マトリックス */
        getmatrix( chmat );
/*
        GetKakudo_zyx( chmat,&rz,&ry,&rx);
        fprintf(stderr,"chmat = %f, %f, %f, %f, %f, %f\n",
                rx, ry, rz, chmat[3][0], chmat[3][1], chmat[3][2] );
*/
}

/*****************************************************************
*       トラッカーの取り付け位置を基準とした                     *
*       回転マトリックスを作成                                   *
*                                                                *
*       引数 : num (トラッカーの区別)                            *
*              orgmat (元の回転マトリックス)                     *
*              chgmat (補正した回転マトリックス)                 *
*       リターン値 : なし                                        *
*****************************************************************/
void
toritsuke_mat( num, orgmat, chgmat )
int     num;
Matrix  orgmat;
Matrix  chgmat;
{
        Matrix  m_wk;
        float   rx, ry, rz;

        loadmatrix( idmat );

        /* 取り付け位置の回転ずれ */
        rot( tr_zure[num][2], 'z' );
        rot( tr_zure[num][1], 'y' );
        rot( tr_zure[num][0], 'x' );
        getmatrix( m_wk );

        /* ずれた回転の逆マトリックス */
        rev_mat( m_wk, m_wk );

        loadmatrix( orgmat );
        multmatrix( m_wk );

        getmatrix( chgmat );
/*
        GetKakudo_zyx( chgmat,&rz,&ry,&rx);
        fprintf(stderr,"chgmat(%d) = %f, %f, %f, %f, %f, %f\n",
                num, rx, ry, rz, chgmat[3][0], chgmat[3][1], chgmat[3][2] );
*/
}

/*****************************************************************
*       キャリブレーション用マトリックスを掛けて                 *
*       トラッカーのズレ (位置, 回転) を補正                     *
*                                                                *
*       引数 : orgmat (元の回転マトリックス)                     *
*              calmat (キャリブレーション用マトリックス)         *
*              chgmat (補正した回転マトリックス)                 *
*       リターン値 : なし                                        *
*****************************************************************/
void
calib_mat( orgmat, calmat, chgmat )
```

```
Matrix  orgmat;
Matrix  calmat;
Matrix  chgmat;
{
        loadmatrix( orgmat );
        multmatrix( calmat );
        getmatrix( chgmat );
}

/*****************************************************************
*       回転マトリックスをワールド座標から表示系に変換          *
*                                                               *
*       引数  :  wrdmat (ワールド座標回転マトリックス)              *
*                dispmat (表示系回転マトリックス)               *
*       リターン値 :  なし                                      *
*****************************************************************/
void
world_disp_mat( wrdmat, dispmat )
Matrix  wrdmat;
Matrix  dispmat;
{

        loadmatrix( wrdmat );
        rot( 180.0, 'y' );
        getmatrix( dispmat );
        dispmat[3][0] = 0.0;
        dispmat[3][1] = 0.0;
        dispmat[3][2] = 0.0;

}

/*****************************************************************
*       回転マトリックスを表示系からワールド座標に変換          *
*                                                               *
*       引数  :  dispmat (表示系回転マトリックス)               *
*                wrdmat (ワールド座標回転マトリックス)          *
*       リターン値 :  なし                                      *
*****************************************************************/
void
disp_world_mat( dispmat, wrdmat )
Matrix  dispmat;
Matrix  wrdmat;
{
        float   rx, ry, rz;

        GetKakudo_zyx( dispmat,&rz,&ry,&rx);

        pushmatrix();
        loadmatrix( idmat );
        rot( -rz, 'z' );
        rot( ry, 'y' );
        rot( -rx, 'x' );
        getmatrix( wrdmat );
        popmatrix();

}

/* END OF FILE */
```

```
/* Form definition file generated with fdesign. */

#include "forms.h"
#include "hirose.h"

FL_FORM *GraphTool;

FL_OBJECT
        *tension_slider,
        *rate_slider,
        *xy_chart,
        *intensity_slider,
        *spine_button,
        *collision_button,
        *breathe_button,
        *elbsize_slider;

void create_form_GraphTool()
{
  FL_OBJECT *obj;
  GraphTool = fl_bgn_form(FL_NO_BOX,300.0,330.0);
  obj = fl_add_box(FL_SHADOW_BOX,0.0,0.0,300.0,330.0,"");
    fl_set_object_lstyle(obj,FL_BOLD_STYLE);
  tension_slider = obj = fl_add_valslider(FL_HOR_SLIDER,50.0,10.0,180.0,20.0,"Muscle
tension");
    fl_set_object_color(obj,47,1);
    fl_set_object_lcol(obj,59);
    fl_set_object_lsize(obj,FL_SMALL_FONT);
    fl_set_object_align(obj,FL_ALIGN_TOP);
    fl_set_object_lstyle(obj,FL_BOLD_STYLE);
    fl_set_call_back(obj,set_tension,0);
  rate_slider = obj = fl_add_valslider(FL_HOR_SLIDER,140.0,250.0,150.0,20.0,"THR2:");
    fl_set_object_color(obj,47,1);
    fl_set_object_lsize(obj,FL_SMALL_FONT);
    fl_set_object_align(obj,FL_ALIGN_LEFT);
    fl_set_object_lstyle(obj,FL_BOLD_STYLE);
    fl_set_call_back(obj,set_thr2,0);
  xy_chart = obj = fl_add_chart(FL_LINE_CHART,70.0,50.0,140.0,140.0,"Density Function
");
    fl_set_object_boxtype(obj,FL_SHADOW_BOX);
    fl_set_object_lcol(obj,58);
    fl_set_object_lsize(obj,FL_SMALL_FONT);
    fl_set_object_align(obj,FL_ALIGN_TOP);
  intensity_slider = obj = fl_add_valslider(FL_HOR_SLIDER,140.0,270.0,150.0,20.0,"THR
1:");
    fl_set_object_color(obj,47,1);
    fl_set_object_lsize(obj,FL_SMALL_FONT);
    fl_set_object_align(obj,FL_ALIGN_LEFT);
    fl_set_object_lstyle(obj,FL_BOLD_STYLE);
    fl_set_call_back(obj,set_thr1,0);
  spine_button = obj = fl_add_roundbutton(FL_PUSH_BUTTON,10.0,300.0,100.0,20.0,"Anima
te Spine");
    fl_set_object_color(obj,7,1);
    fl_set_object_lcol(obj,4);
    fl_set_object_lsize(obj,FL_SMALL_FONT);
    fl_set_object_lstyle(obj,FL_BOLD_STYLE);
    fl_set_call_back(obj,spine_on_off,0 );
  collision_button = obj = fl_add_roundbutton(FL_PUSH_BUTTON,150.0,300.0,120.0,20.0,"
Collision Model");
    fl_set_object_color(obj,7,1);
    fl_set_object_lcol(obj,4);
    fl_set_object_lsize(obj,FL_SMALL_FONT);
    fl_set_object_lstyle(obj,FL_BOLD_STYLE);
    fl_set_call_back(obj,collision_on_off,0);
  breathe_button = obj = fl_add_roundbutton(FL_PUSH_BUTTON,0.0,260.0,80.0,20.0,"Breat
he");
```



```
    fl_set_object_color(obj,7,1);
    fl_set_object_lsize(obj,FL_SMALL_FONT);
    fl_set_object_lstyle(obj,FL_BOLD_STYLE);
    fl_set_call_back(obj,breathe_on_off,0);
  elbsize_slider = obj = fl_add_valslider(FL_HOR_SLIDER,140.0,220.0,150.0,20.0,"Elbow
 Primitive Size:");
    fl_set_object_color(obj,47,1);
    fl_set_object_lcol(obj,4);
    fl_set_object_lsize(obj,FL_SMALL_FONT);
    fl_set_object_align(obj,FL_ALIGN_LEFT);
    fl_set_object_lstyle(obj,FL_BOLD_STYLE);
    fl_set_call_back(obj,set_elbsize,0);
  fl_end_form();
}

/*---------------------------------------*/

void create_the_forms()
{
  create_form_GraphTool();
}
```

```
/*
 * ファイル名 : init.c
 *
 * 機能       : データ初期化関数
 *
 * 履歴       : 1992年10月28日 ; 作成 ; 広瀬　正俊
 */
#include        "tool.h"

#include        "forms.h"
#include        "hirose.h"

/*
 * 関数名 : GlobalSet()
 *
 * 引数   : なし
 *
 * 戻り値 : なし
 *
 * 機能   : グローバル変数の初期値のセット
 *
 * 履歴   : 1992年10月28日 ; 作成 ; 広瀬　正俊
 */
void
GlobalSet()
[
    int         i, j;

    for (i = 0; i < MAX_TR; i++) {
        for (j = 0; j < 6; j++) {
            G_trd[i][j] = 0.0;
        ]
    ]
    G_trd[MUNE_TRAK_NO][3] = 0.0;  /* 90.0 */
    G_trd[ATAMA_TRAK_NO][1] = 37.0;  /* 30.0 */
    G_trd[ATAMA_TRAK_NO][3] = 90.0;  /* 0.0 */
    G_trd[ATAMA_TRAK_NO][5] = -90.0;  /* 90.0 */
    G_trd[MIGITE_TRAK_NO][0] = -71.0;  /* 74.0 */
    G_trd[MIGITE_TRAK_NO][4] = 90.0;
    G_trd[MIGITE_TRAK_NO][5] = 180.0;  /* 90.0 */
    G_trd[HIDARITE_TRAK_NO][0] = 71.0;  /* -74.0 */
    G_trd[HIDARITE_TRAK_NO][4] = 90.0;
    G_trd[HIDARITE_TRAK_NO][5] = 0.0;  /* 180.0 */
    G_trd[TR_MIGIASHI][1] = -95.0;
    G_trd[TR_MIGIASHI][2] =  32.9;
    G_trd[TR_HIDARIASHI][1] = -95.0;
    G_trd[TR_HIDARIASHI][2] =  32.9;

#ifdef CYBERGLOVE
    for (i = 0; i < 32; i++) [
        G_dgd[i] = 0;
    ]
#else
    for (i = 0; i < 28; i++) {
        G_dgd[i] = 0;
    ]
#endif
    G_dgd[10] = G_dgd[24] = 35;
    G_dgd[11] = G_dgd[25] = 20;
    G_dgd[12] = G_dgd[26] = 10;
    G_dgd[13] = G_dgd[27] = 10;

    G_black[0]  = 0.0; G_black[1]  = 0.0; G_black[2]  = 0.0;
    G_white[0]  = 1.0; G_white[1]  = 1.0; G_white[2]  = 1.0;
    G_red[0]    = 1.0; G_red[1]    = 0.0; G_red[2]    = 0.0;
    G_green[0]  = 0.0; G_green[1]  = 1.0; G_green[2]  = 0.0;
```



```
    G_blue[0]   = 0.0; G_blue[1]   = 0.0; G_blue[2]   = 1.0;
    G_yellow[0] = 1.0; G_yellow[1] = 1.0; G_yellow[2] = 0.0;
    G_pink[0]   = 1.0; G_pink[1]   = 0.0; G_pink[2]   = 0.5;
    G_back[0]   = 1.0; G_back[1]   = 1.0; G_back[2]   = 1.0;
    G_movex = G_movey = 0.0;
    G_rota = G_elea = 0.0;
    G_hand = 0;
    G_kahansin = 1;
    trkswitch = 1;
    strcpy(G_tr_data, "tr_data");
    strcpy(G_ang_data, "ang_data");

    G_glass = 0;
    G_sisei = SITTING;
    G_cmdst = DRAW_3D_WIRE;
    G_four = 1;
    G_munekara = 0;
]

/*
 * 関数名 : InitializeGL()
 *
 * 引数   : なし
 *
 * 戻り値 : なし
 *
 * 機能   : ・ｇｌの初期化
 *          ・ウインドウのオープン
 *          ・ＲＧＢモードの設定
 *          ・ｚバッファーを有効にする
 *          ・ダブルバッファーを有効にする
 *          ・光源モデルの設定
 *          ・キューデバイスの設定
 *
 * 履歴   : 1992年10月28日 ; 作成 ; 広瀬　正俊
 */
void
InitializeGL(aa)
int aa;
{
    static float        white_light[] = [LCOLOR,   0.8,  0.8,  0.8,
                                         POSITION, 0.0,  0.0, 20.0, 1.0,
                                         LMNULL];
    static float        mtbuf[] = [EMISSION, 0.0, 0.0, 0.0,
                                   AMBIENT,  0.1, 0.1, 0.1,
                                   DIFFUSE,  0.6, 0.2, 0.2,
                                   SPECULAR, 0.5, 0.5, 0.5,
                                   SHININESS,60.0,
                                   ALPHA,    1.0,
                                   LMNULL];

    fl_init();
    create_the_forms();
    initialize_form();
    fl_show_form( GraphTool, FL_PLACE_SIZE, TRUE, "Human Figure Parameters" );
    draw_graph();

    fl_qdevice(LEFTMOUSE);
    fl_qdevice(MIDDLEMOUSE);
    fl_qdevice(RIGHTMOUSE);
    fl_qdevice(ESCKEY);

    /* prefposition(10, 550, 400, 950); */
    G_winid = winopen("TOOL 3D");
    RGBmode();
    zbuffer(TRUE);
```

```
    doublebuffer();

/*+++++++++++++++++++++++++++++++*/

    if (aa)
      {
        zbsize(0);
        mssize(8,32,0);
      }
 /*+++++++++++++++++++++++++++++++*/

   gconfig();
    c3f(G_black);
    clear();
    swapbuffers();

    mmode(MVIEWING);
    lmdef(DEFMATERIAL, 1, 21, mtbuf);
    lmdef(DEFLIGHT, 1, 10, white_light);
    lmdef(DEFLMODEL, 1, 0, NULL);
    lmbind(MATERIAL, 1);
    lmbind(LIGHT0, 1);
    lmbind(LMODEL, 1);
}

/*
 * 関数名 : SetDataMinMax3(int buhin)
 *
 * 引数   : buhin : 部品番号
 *
 * 戻り値 : なし
 *
 * 機能   : ・3次元データの最小、最大値を求める
 *
 * 履歴   : 1992年10月28日 ; 作成 ; 広瀬　正俊
 */
void
SetDataMinMax3(int stpt, int hcnt, void *vhm[], int buhin)
{
    int         h, i, j, k, k1, k2;
    float       *v, xmin, xmax, ymin, ymax, zmin, zmax;
    JINBUTU     *hm;

    for (h = stpt; h < stpt+hcnt; h++) {
        hm = (JINBUTU *)vhm[h];
        xmin = 1.0e32;  xmax = -1.0e32;
        ymin = 1.0e32;  ymax = -1.0e32;
        zmin = 1.0e32;  zmax = -1.0e32;

        if(buhin == MAX_BUHIN) {
            k1 = 0;
            k2 = MAX_BUHIN - 1;
        } else {
            k1 = buhin;
            k2 = buhin;
        }
        for(k = k1; k <= k2; k++) {
            for(j = 0; j < hm->vmax[k]; j++) {
                v = (float *)(hm->vs[k]+j);
                for(i = 0; i < 3; i++) {
                    xmin = min(xmin, v[0]);
                    xmax = max(xmax, v[0]);
                    ymin = min(ymin, v[1]);
                    ymax = max(ymax, v[1]);
                    zmin = min(zmin, v[2]);
                    zmax = max(zmax, v[2]);
```



```
                }
            }
        }
        printf("human(%d) xsize:%f\n", h, xmax-xmin);
        printf("          ysize:%f\n", ymax-ymin);
        printf("          zsize:%f\n", zmax-zmin);
    }
    G_3dxmin = xmin;    G_3dxmax = xmax;
    G_3dymin = ymin;    G_3dymax = ymax;
    G_3dzmin = zmin;    G_3dzmax = zmax;
}

/*
 * 関数名 : ViewSize3(float xmin, float xmax, float ymin, float ymax,
 *                    float zmin, float zmax)
 *
 * 引数   : xmin, xmax, ymin, ymax, zmin, zmax : 表示領域の範囲
 *
 * 戻り値 : なし
 *
 * 機能   : ・ortho() により表示領域を決定する
 *
 * 履歴   : 1992年10月28日 ; 作成 ; 広瀬　正俊
 */
void
ViewSize3(float xmin, float xmax, float ymin, float ymax,
          float zmin, float zmax)
{
    long        sizex, sizey;
    float       xy, gxy, s;
    float       width;

    getsize(&sizex, &sizey);
    xy = (float)sizex / (float)sizey;
    gxy = (xmax-xmin) / (ymax-ymin);
    if(gxy > xy) {
        s = (xmax-xmin) / xy - (ymax-ymin);
        s *= 0.5;
        ymin -= s;
        ymax += s;
    } else {
        s = (ymax-ymin) * xy - (xmax-xmin);
        s *= 0.5;
        xmin -= s;
        xmax += s;
    }

    width = max(xmax-xmin, ymax-ymin);
    width = max(width, zmax-zmin);
    mmode(MPROJECTION);
    ortho(xmin, xmax, ymin, ymax, -((zmax+zmin)/2.0+width),
                                  -((zmax+zmin)/2.0-width));
#ifndef HIROSE
    lookat(0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0);
#else
    lookat(0.0, 0.0, -1.0, 0.0, 0.0, 0.0, 0);
#endif
    mmode(MVIEWING);
}

/*
 * 関数名 : InitViewSize3(float xmin, float xmax, float ymin, float ymax,
 *                        float zmin, float zmax)
 *
 * 引数   : xmin, xmax, ymin, ymax, zmin, zmax : 表示データの最小、最大値
 *
```

```
 * 戻り値 : なし
 *
 * 機能   : ・３次元データの最小、最大値よりビューサイズを求める
 *
 * 履歴   : 1992年10月28日 ; 作成 ; 広瀬　正俊
 */
void
InitViewSize3()
{
    float       xmin, xmax, ymin, ymax, zmin, zmax;

    xmin = G_3dxmin - (G_3dxmax-G_3dxmin)*0.25;
    xmax = G_3dxmax + (G_3dxmax-G_3dxmin)*0.25;
    ymin = G_3dymin - (G_3dymax-G_3dymin)*0.25;
    ymax = G_3dymax + (G_3dymax-G_3dymin)*0.25;
    zmin = G_3dzmin - (G_3dzmax-G_3dzmin)*2.65;
    zmax = G_3dzmax + (G_3dzmax-G_3dzmin)*2.65;
    if(xmin == xmax)
        xmax += 1.0;
    if(ymin == ymax)
        ymax += 1.0;
    if(zmin == zmax)
        zmax += 1.0;

    ViewSize3(xmin, xmax, ymin, ymax, zmin, zmax);
}


/*
 * 関数名 : BackWhite()
 *
 * 引数   : なし
 *
 * 戻り値 : なし
 *
 * 機能   : ・背景色を白にする
 *
 * 履歴   : １０月２８日 ; 作成 ; 広瀬　正俊
 */
void
BackWhite()
{
    G_back[0] = 1.0; G_back[1] = 1.0; G_back[2] = 1.0;
}

/*
 * 関数名 : BackBlack()
 *
 * 引数   : なし
 *
 * 戻り値 : なし
 *
 * 機能   : ・背景色を黒にする
 *
 * 履歴   : １０月２８日 ; 作成 ; 広瀬　正俊
 */
void
BackBlack()
{
    G_back[0] = 0.0; G_back[1] = 0.0; G_back[2] = 0.0;
}

void
BackGrey()
{
    G_back[0] = 0.5; G_back[1] = 0.5; G_back[2] = 0.5;
```



```
}
```

```
/*
 * ファイル名 : main.c
 *
 * 機能       : 3次元変形表示確認テストプログラムメイン関数
 *
 * 履歴       : 1992年10月28日 ; 作成 ; 広瀬　正俊
 */
#define EXTRN

#include        "body.h"
#include        "tool.h"

main(argc, argv)
    int         argc;
    char        **argv;
[
    void        *vhm[2];
    int         hcnt,d;
    int         ii;

/*loadobjects();*/


    /* Textports can not be used unless the GL program is run in the
       foreground. */
    foreground();                       /* 標準入出力を使用するために必要 */

    GlobalSet();                        /* グローバル変数の初期値のセット */

    /*++++++++++++++++++++*/
    d=0;
    if ((argc>1) && !strcmp(argv[1],"aa"))
          {
            InitializeGL(1);
            d=1;
          }

    else
      InitializeGL(0);                  /* ｇｌの初期化 */
 /*++++++++++++++++++++*/
    MenuInit();                         /* ポップアップメニューの作成 */

    /* データの読み込み */


 /*++++++++++++++++++++*/
    hcnt = 0;
    if (argc > d+1) {
        vhm[hcnt++] = ReadObjTexture(*(argv+d+1));
        if (argc == d+3) {
            vhm[hcnt++] = ReadObjTexture(*(argv+d+2));
        }
    } else {
        vhm[hcnt++] = ReadObjTexture("Ochi");
    }
 /*++++++++++++++++++++*/
{
  char  s[100];
  int   i;

  mapn = 0;
  for(i=0;i<mapn;i++){
    sprintf(s,"ochi_body%d.color", i+1);
    maptest[i] = read_map_tex(s);
  ]
]
```



```
/*********************
WriteWave("../gattai/ver6/ochi_body.obj", MUNE);
WriteWave("../gattai/ver6/ochi_body_ninoude_b_r.obj", KATA_R_M);
WriteWave("../gattai/ver6/ochi_body_ninoude_n_r.obj", KATA_R_N);
WriteWave("../gattai/ver6/ochi_ninoude_r.obj", NINOUDE_R);
WriteWave("../gattai/ver6/ochi_ninoude_ude_n_r.obj", HIJI_R_N);
WriteWave("../gattai/ver6/ochi_ninoude_ude_u_r.obj", HIJI_R_U);
WriteWave("../gattai/ver6/ochi_ude_r.obj", UDE_R);
WriteWave("../gattai/ver6/ochi_ude_te_u_r.obj", TEKUBI_R_U);
WriteWave("../gattai/ver6/ochi_body_ninoude_b_l.obj", KATA_L_M);
WriteWave("../gattai/ver6/ochi_body_ninoude_n_l.obj", KATA_L_N);
WriteWave("../gattai/ver6/ochi_ninoude_l.obj", NINOUDEE_L);
WriteWave("../gattai/ver6/ochi_ninoude_ude_n_l.obj", HIJI_L_N);
WriteWave("../gattai/ver6/ochi_ninoude_ude_u_l.obj", HIJI_L_U);
WriteWave("../gattai/ver6/ochi_ude_l.obj", UDEE_L);
WriteWave("../gattai/ver6/ochi_ude_te_u_l.obj", TEKUBI_L_U);
*********************/

/*++++++++++++++++++++++++++++++++++++*/
for (ii=0;ii<hcnt;ii++)
{
SetDataMinMax3(ii,1,vhm,DOU);
bb[0][0] = G_3dxmin;
bb[0][1] = G_3dxmax;
bb[1][0] = G_3dymin;
bb[1][1] = G_3dymax;
bb[2][0] = G_3dzmin;
bb[2][1] = G_3dzmax;

SetDataMinMax3(ii,1,vhm,KOSI);
if (bb[2][0] > G_3dzmin)
     bb[2][0]=G_3dzmin;

if (bb[2][1] < G_3dzmax)
     bb[2][1] = G_3dzmax;

setSpace(&(((JINBUTU *)vhm[ii])->douspace),ii,DOU,1,vhm);

bb[1][1] = bb[1][0];
bb[1][0] = G_3dymin;

setSpace(&(((JINBUTU *)vhm[ii])->kosispace),ii,KOSI,1,vhm);

calibImpSpace((JINBUTU *)vhm[0]);
}
/*++++++++++++++++++++++++++++++++++++*/


    SetDataMinMax3(0, hcnt, vhm, MAX_BUHIN);/* 3次元データの最小、最大を求める */


    /* ワイヤーモデルの表示 */
    InitViewSize3();
    DrawHuman(hcnt, vhm);

    /* イベント処理メインループ */
    MainLoop(hcnt, vhm);
}
```

```
/*
 * ファイル名 : menu.c
 *
 * 機能       : 3次元変形表示確認テストプログラムメニュー作成関数
 *
 * 履歴       : 1992年10月28日 ; 作成 ; 広瀬　正俊
 */
#include        "tool.h"

#include        "forms.h"
#include        "hirose.h"

#define         MAX_POINT       80
#define         MIN_XVAL        0.0
#define         MAX_XVAL        1.0
#define         MIN_YVAL        0.0
#define         MAX_YVAL        1.0

static int     G_breathe, G_spine, G_collision;
float   G_intensity, G_rate;
float G_c = 0.0;

/*
 * 関数名 : MainLoop()
 *
 * 引数   : なし
 *
 * 戻り値 : なし
 *
 * 機能   : イベント処理メインループ
 *
 * 履歴   : 1992年10月28日 ; 作成 ; 広瀬　正俊
 */
void
MainLoop(int hcnt, void *vhm[])
{
    Device      dev;
    short       val;
    long        rtn;
    int         trn, jiku;
    int         mdlbtn = SCALING;
    JINBUTU     *hm;
    FL_OBJECT   *retobj;

    hm = (JINBUTU *)vhm;

  while (1) {
    retobj = fl_check_forms();
      if (retobj == FL_EVENT) {
        if (!fl_qtest())
            continue;
        dev = fl_qread(&val);

        switch (dev) {
        case REDRAW:            /* 再描画 */
            reshapeviewport();
            InitViewSize3();
            DrawHuman(hcnt, vhm);
            break;
        case LEFTMOUSE:         /* マウスの左ボタンクリック */
            if (val == 1) {                     /* push button */
                if (jiku == 6) {
                    Rotate3(hcnt, vhm);
                } else if (trn < 0) {
                    Scaling(hcnt, vhm, TRANSLATING);
                } else {
```



```
                    ChangeTracker(hcnt, vhm, trn, jiku);
                }
            }
            qreset();
            break;
        case MIDDLEMOUSE:       /* マウスの中ボタンクリック */
            if (val == 1) {                     /* push button */
                if (jiku == 6) {
                    Rotate3(hcnt, vhm);
                } else if (trn < 0) {
                    Scaling(hcnt, vhm, SCALING);
                } else {
                    ChangeTracker(hcnt, vhm, trn, jiku+2);
                }
            }
            qreset();
            break;
        case RIGHTMOUSE:        /* マウスの右ボタンクリック */
            if ((G_cmdst != NOCOMMAND) && (G_cmdst != DRAW_3D_WIRE)
                                       && (G_cmdst != DRAW_3D_TEXT)
                                       && (G_cmdst != TEXT_AND_WIRE))
                G_cmdst = NOCOMMAND;
            rtn = dopup(G_menu);        /* ポップアップメニューの表示及び選択 */
            switch (rtn) {

            case 1:
                trn = MUNE_TRAK_NO;
                jiku = 0;
                break;
            case 2:
                trn = MUNE_TRAK_NO;
                jiku = 1;
                break;

            case 3:
                trn = ATAMA_TRAK_NO;
                jiku = 0;
                break;
            case 4:
                trn = ATAMA_TRAK_NO;
                jiku = 1;
                break;
            case 5:
                trn = MIGITE_TRAK_NO;
                jiku = 0;
                break;
            case 6:
                trn = MIGITE_TRAK_NO;
                jiku = 1;
                break;
            case 7:
                trn = HIDARITE_TRAK_NO;
                jiku = 0;
                break;
            case 8:
                trn = HIDARITE_TRAK_NO;
                jiku = 1;
                break;
            case 25:
                trn = TR_MIGIASHI;
                jiku = 1;
                break;
            case 26:
                trn = TR_HIDARIASHI;
                jiku = 1;
                break;
```

```
            case 9:
                trn = MAX_TR;
                jiku = 4;
                break;
            case 16:
                trn = MAX_TR;
                jiku = 5;
                break;
            case 20:
                jiku = 6;
                break;
            case 10:
                jiku = 7;
                if (mdlbtn == SCALING) {
                    mdlbtn = TRANSLATING;
                } else {
                    mdlbtn = SCALING;
                }
                trn = -1;
                break;
            case 11:
                G_cmdst = DRAW_3D_WIRE;
                DrawHuman(hcnt, vhm);
                break;
            case 12:
                G_cmdst = DRAW_3D_TEXT;
                DrawHuman(hcnt, vhm);
                break;
/*
            case 13:
                G_cmdst = TEXT_AND_WIRE;
                DrawHuman(hcnt, vhm);
                break;
*/
            case 14:
                showimp=1-showimp;
                Information(hcnt, vhm);
                break;
/*
            case 30:
                G_four = 1-G_four;
                break;
*/
            case 17:
                G_four = 1;
                trkswitch = 1;
                ReadTrackerData(hcnt, vhm);
                break;
            case 15:
                trkswitch = 1;
                ReadTrackerDataFile(hcnt, vhm);
                break;
/*
            case 27:
                G_munekara = 0;
                trkswitch = 0;
                ReadAngleDataFile(hcnt, vhm);
                break;
            case 31:
                G_munekara = 1;
                trkswitch = 0;
                ReadAngleDataFile(hcnt, vhm);
                break;
*/
            case 18:
                ResetTrackerData(hcnt, vhm);
```



```
                break;
            case 19:
                CalibMune(hcnt, vhm);
                winset(G_winid);
                break;
                /*++++++++++++++++++++++++++++++++*/
            case 21:
                ffdswitch=1-ffdswitch;
                printf("ffdtoggle=%d\n",ffdswitch);
                break;
            case 22:
                breatheswitch=1-breatheswitch;
                break;
            case 23:
                showimp=1-showimp;
                break;
            case 24:
                collon=1-collon;
                break;

                /*++++++++++++++++++++++++++++++++*/

            default:
                break;
            }
            qreset();
            break;
        case ESCKEY:
            greset();
            gexit();
            exit(1);
            break;
        default:
            break;
        }
    }
  }
}


/*
 * 関数名 : MenuInit()
 *
 * 引数   : なし
 *
 * 戻り値 : なし
 *
 * 機能   : ポップアップメニューの作成
 *
 * 履歴   : 1992年10月28日 ; 作成 ; 広瀬　正俊
 */
int
MenuInit()
{
    int         submenu1;
    char        menu[1024];

    submenu1 = newpup();
    strcpy(menu, "BACK COLOR MENU %t");
    strcat(menu, "|white %f");
    strcat(menu, "|black %f");
    strcat(menu, "|grey %f");
    addtopup(submenu1, menu, BackWhite, BackBlack, BackGrey);

    G_menu = newpup();
    strcpy(menu, "TOOL 3D MENU %t");
```

```
    strcat(menu, "|wire %x11");
    strcat(menu, "|texture %x12");
/*
    strcat(menu, "|texture and wire %x13");
*/
    strcat(menu, "|all %x20");

    strcat(menu, "|mune(x,y) %x1");
    strcat(menu, "|mune(z,y) %x2");

    strcat(menu, "|atama(x,y) %x3");
    strcat(menu, "|atama(z,y) %x4");
    strcat(menu, "|rtekubi(x,y) %x5");
    strcat(menu, "|rtekubi(z,y) %x6");
    strcat(menu, "|ltekubi(x,y) %x7");
    strcat(menu, "|ltekubi(z,y) %x8");
    strcat(menu, "|rkutu(z,y) %x25");
    strcat(menu, "|lkutu(z,y) %x26");
    strcat(menu, "|ryubi %x9");
    strcat(menu, "|lyubi %x16");
/*
    strcat(menu, "|four tracker %x30");
*/
    strcat(menu, "|read tracker data %x17");
    strcat(menu, "|read tracker data file %x15");
/*
    strcat(menu, "|read angle data file (KOSI) %x27");
    strcat(menu, "|read angle data file (MUNE) %x31");
*/
    strcat(menu, "|reset tracker data %x18");
    strcat(menu, "|mune calibration %x19");
    /*++++++++++++++++++++++++*/
    strcat(menu, "|animate spine %x21");
    strcat(menu, "|breathe %x22");
    strcat(menu, "|show implicit space %x23");
    strcat(menu, "|collision %x24");

    /*++++++++++++++++++++++++*/
    strcat(menu, "|information %x14");
    strcat(menu, "|zoom / move %x10");
    strcat(menu, "|back color %m");
    strcat(menu, "|Quit %f");

    /* コールバック関数の設定 */
    addtopup(G_menu, menu, submenu1, QuitTool);
}


/*
 * 関数名 : QuitTool()
 *
 * 引数   : なし
 *
 * 戻り値 : なし
 *
 * 機能   : ・メニューで Quit が選択された時に呼ばれる
 *          ・ｇｌを終了した後、プログラムを終了する
 *
 * 履歴   : 1992年10月28日 ; 作成 ; 広瀬　正俊
 */
void
QuitTool()
{
    greset();
    gexit();
    exit(0);
```



```
}


initialize_form()
{
  fl_set_chart_maxnumb( xy_chart, MAX_POINT );
  fl_set_chart_autosize( xy_chart, FALSE );
  fl_set_chart_bounds( xy_chart, MIN_YVAL, MAX_YVAL );

  fl_set_slider_bounds(intensity_slider, 0.01, 0.90);
  fl_set_slider_bounds(rate_slider, 0.01, 0.90);
  fl_set_slider_bounds(tension_slider, -2.5, 0.0);

  fl_set_slider_value(intensity_slider, THR);
  fl_set_slider_value(rate_slider, THR2);
  fl_set_slider_value(tension_slider, G_c);

  fl_set_button(breathe_button, breatheswitch);
  fl_set_button(spine_button, ffdswitch);
  fl_set_button(collision_button, collon);
}


draw_graph()
{
  int           i;
  float         x, y;

  fl_clear_chart( xy_chart );
  fl_freeze_object( xy_chart );

  for (i = 0; i <= MAX_POINT; i++) {
    x = (float)(i)/(float)MAX_POINT*(MAX_XVAL-MIN_XVAL);
    y = (G_c+2.0)*x*x*x - (2.0*G_c+3.0)*x*x + G_c*x + 1.0;

    /* forms のバグで、グラフが境界を越えてしまうので、それに対する処理 */
    if( y >=  MAX_YVAL*1.2 ){
      y = MAX_YVAL*1.2;
      fl_add_chart_value( xy_chart, y, "", 0 );
    }
    else if( y <= -MAX_YVAL*1.2 ){
      y = -MAX_YVAL*1.2;
      fl_add_chart_value( xy_chart, y, "", 0 );
    }
    else{       /* 描画(プロット) */
      fl_add_chart_value( xy_chart, y, "", 1 );
      /* fl_insert_chart_value( xy_chart, i, y, "", 1 ); */
    }
  }
  fl_unfreeze_object( xy_chart );
}


void
set_thr1(FL_OBJECT *obj, long l)
{
#if 0
  G_intensity = fl_get_slider_value(obj);
#else
        THR = fl_get_slider_value(obj);
#endif
/*printf("G_intensity:%f\n",G_intensity);*/
}


void
```

```
set_thr2(FL_OBJECT *obj, long l)
{
#if 0
  G_rate = fl_get_slider_value(obj);
#else
        THR2 = fl_get_slider_value(obj);
#endif
/*printf("G_rate:%f\n",G_rate);*/
}


void
set_tension(FL_OBJECT *obj, long l)
{
  G_c = fl_get_slider_value(obj);
/*printf("G_c:%f\n",G_c);*/
  draw_graph();
}


void
set_elbsize(FL_OBJECT *obj, long l)
{
/*  G_c = fl_get_slider_value(obj);*/
/*printf("G_c:%f\n",G_c);*/
}


void
breathe_on_off(FL_OBJECT *obj, long l)
{
  breatheswitch = fl_get_button(obj);
/*printf("G_breathe:%d\n",G_breathe);*/
}


void
spine_on_off(FL_OBJECT *obj, long l)
{
  ffdswitch = fl_get_button(obj);
/*printf("G_spine:%d\n",G_spine);*/
}


void
collision_on_off(FL_OBJECT *obj, long l)
{
  collon = fl_get_button(obj);
/*printf("G_collision:%d\n",G_collision);*/
}
```

```
/*****************************************************************
*       ファイル名 : mkrotm.c                                    *
*****************************************************************/

/*
* mk_atama_rotm()                                                *
*       頭の回転マトリックスを作成                               *
*                                                                *
* mk_kata_rotm( num, kata_x, tekubi, ka_m, hi_m )                *
*       肩と肘の回転マトリックスを作成                           *
*                                                                *
* mk_tekubi_rotm( num )                                          *
*       手首の回転マトリックスを作成                             *
*                                                                *
* mk_wrl_kata_hiji()                                             *
*       肩と肘のワールド座標系での絶対回転マトリックスを作成     *
*                                                                *
* mk_yubi_rotm( fang, axis, hosei, m )                           *
*       指の回転マトリックスを作成                               *
*                                                                *
* GetKataRotMatrix( ka_m, a, m1, m2)                             *
*       肩のＸ軸回転、肩のＹ，Ｚ軸回転を別々に取り出す           *
*                                                                *
* coord_calc( d1, d2, mat )                                      *
*       マトリックスにより座標値を変換                           *
*                                                                *
* get_rot_coord( pnt1, pnt2, angle, axis )                       *
*       回転させた時の座標値を計算                               *
*                                                                *
* get_teku_x( num, m )                                           *
*       手首のＸ軸回転を取り出す                                 *
*                                                                *
* get_teku_yz( num, m )                                          *
*       手首のＹ，Ｚ軸回転を取り出す                             *
*                                                                *
* rev_mat( m1, m2 )                                              *
*       逆マトリックスを作成                                     *
*                                                                *
* cp_mat( m1, m2 )                                               *
*       マトリックスのコピー                                     *
*/

#include        <gl/gl.h>

#include        "body.h"
#include        "body_draw.h"

#include        "rname_fc.h"

/*****************************************************************
*       頭の回転マトリックスを作成                               *
*                                                                *
*       引数 : なし                                              *
*       リターン値 : なし                                        *
*****************************************************************/
void
mk_atama_rotm()
{

        /* 胸の逆マトリックスをロードする */
        loadmatrix( mune_rm );
        /* 頭のマトリックスを掛ける */
        multmatrix( atama_m );

        /* 頭の回転マトリックスを作成 */
```



```
        getmatrix( atama_m );

}


/*
 * 関数名 : GetKataMatrix(int num, float *tekubi, float xangle, float kata_x,
 *            float ninoude_l, float ude_l, Matrix kata_m, Matrix hiji_m,
 *            float *hiji_pnt)
 *
 * 引数   : num       左右の区別  0 : 右手、 1 : 左手
 *          tekubi    手首の位置
 *          xangle    肩のx軸回転角度
 *          kata_x    肩幅
 *          ninoude_l 二の腕長さ
 *          ude_l     腕長さ
 *          kata_m    肩の回転マトリックス
 *          hiji_m    肘の回転マトリックス
 *          hiji_pnt  肘の位置
 *
 * 戻り値 : なし
 *
 * 機能   : 肩と肘の回転マトリックスを作成
 *          (表示座標系で計算
 *
 * 履歴   : 1995年1月13日 ; 作成 ; 広瀬＼敦
 */
void
GetKataMatrix(int num, float *tekubi, float xangle, float kata_x,
              float ninoude_l, float ude_l, Matrix kata_m, Matrix hiji_m,
              float *hiji_pnt)
{
    float       pnt1[4], pnt2[4];
    float       theta;
    float       angle1, angle2;
    float       l, cos_theta;
    float       tmp_ang;
    Matrix      tmp_m;
    float       kata_angy, kata_angz, hiji_angy;
    int         i, j;

    /*  手首のワールド座標を肩が原点 (0,0,0) にくるように   */
    /*  移動したものを pnt1[] に設定                         */
    pnt1[0] = tekubi[X] - kata_x;
    pnt1[1] = tekubi[Y];
    pnt1[2] = tekubi[Z];

    /*  Ｙ軸回りの回転角度を計察複擦鮮佑┐覆茲∠砲垢襯燭領   */
    theta = atan2f(pnt1[2], pnt1[0]);

    /*  右腕は－Ｘ軸(-(ninoude_l+ude_l),0,0)から、
        左腕は＋Ｘ軸(+(ninoude_l+ude_l),0,0)からの回転にするため   */
    if ((!num) && (pnt1[2] < 0.0)) {
        theta += M_PI;
    } else if ((!num) && (pnt1[2] >= 0.0)) {
        theta -= M_PI;
    ]

    angle1 = theta / M_PI * 180.0;

    /*  pnt1 をＹ軸に angle1 回転させた時の座標値を pnt2 に設定   */
    get_rot_coord(pnt1, pnt2, angle1, 'y');

    theta = atan2f(pnt2[1], pnt2[0]);
    if ((!num) && (pnt2[1] < 0.0)) {
        theta += M_PI;
```

```
    } else if ((!num) && (pnt2[1] >= 0.0)) {
        theta -= M_PI;
    }
    angle2 = theta / M_PI * 180.0;

    /*  原点(0,0) から pnt2 までの距離を計  */
    l = sqrt(pnt2[0]*pnt2[0] + pnt2[1]*pnt2[1]);

    /*  余弦定理によりｌと原点から引かれる辺とで作られる角度を計  */
    cos_theta = (l*l + ninoude_l*ninoude_l - ude_l*ude_l) / (2.0*l*ninoude_l);

    if (cos_theta > 1.0) {
        cos_theta = 1.0;
    } else if (cos_theta < -1.0) {
        cos_theta = -1.0;
    }

    tmp_ang = acosf(cos_theta) / M_PI * 180.0;

    /*  肘の位置より肩の回転角度を計  */
    pushmatrix();
    loadmatrix(idmat);
    rot(-angle1, 'y');
    rot(angle2, 'z');
    rot(xangle, 'x');
    if (!num) {
        rot(-tmp_ang, 'y');
    } else {
        rot(tmp_ang, 'y');
    }
    getmatrix(kata_m);
    popmatrix();
/*
printf("xangle,angle1,angle2,tmp_ang:%f,%f,%f,%f\n",xangle,angle1,angle2,tmp_ang);
*/

    /* 現在の肘の位置を検出 */
    if (!num) {
        pnt1[0] = -ninoude_l;
    } else {
        pnt1[0] = ninoude_l;
    }
    pnt1[1] = 0.0;
    pnt1[2] = 0.0;
    pnt1[3] = 1.0;
    for (i = 0; i < 4; i++) {
        pnt2[i] = 0.0;
        for (j = 0; j < 4; j++) {
            pnt2[i] += pnt1[j] * kata_m[j][i];
        }
    }
    hiji_pnt[0] = pnt2[0];
    hiji_pnt[1] = pnt2[1];
    hiji_pnt[2] = pnt2[2];

    /* 余弦定理によりｌと原点から引かれる辺とで作られる角度を計 */
    cos_theta = (ninoude_l*ninoude_l + ude_l*ude_l - l*l) /
                (2.0*ninoude_l*ude_l);

    if (cos_theta > 1.0) {
        cos_theta = 1.0;
    } else if (cos_theta < -1.0) {
        cos_theta = -1.0;
    }

    hiji_angy = acosf(cos_theta);
```



```
    hiji_angy = hiji_angy / M_PI * 180.0;
    hiji_angy = 180 - hiji_angy;

    if (num == 1) {
        hiji_angy = -1.0 * hiji_angy;
    }

    /* 肘のＹ軸の回転マトリックスを求める */
    pushmatrix();
    loadmatrix(idmat);
    rot(hiji_angy, 'y');
    getmatrix(hiji_m);
    popmatrix();
}


/*****************************************************************
*       手首の回転マトリックスを作成                              *
*                                                                 *
*       引数 :  num （左右の区別）  0 : 右、 1 : 左               *
*       リターン値 :  なし                                        *
*****************************************************************/
void
GetTekubiMatrix(int num, Matrix mune, Matrix kata, Matrix hiji, Matrix tekubi,
                float *tekubi_a, float *tekubi_y, float *tekubi_z,
                Matrix tekubi1_m, Matrix tekubi2_m)
{
    Matrix      mkh_m, mkh_rm;

    loadmatrix(mune);
    multmatrix(kata);
    multmatrix(hiji);
    getmatrix(mkh_m);
    rev_mat(mkh_m, mkh_rm);

    loadmatrix(mkh_rm);
    multmatrix(tekubi);

    /* 手首の相対マトリックスを作成 */
    getmatrix(tekubi);

    /* 手首のＸ軸回転のマトリックスを作成 */
    GetTekubiX(num, tekubi, tekubi_a, tekubi1_m);

    /* 手首のＹＺ軸回転のマトリックスを作成 */
    GetTekubiYZ(num, tekubi, tekubi_y, tekubi_z, tekubi2_m);
}


/*****************************************************************
*       手首の回転マトリックスを作成                              *
*                                                                 *
*       引数 :  num （左右の区別）  0 : 右、 1 : 左               *
*       リターン値 :  なし                                        *
*****************************************************************/
void
GetTekubiMatrix2(int num, Matrix mune, Matrix kata, Matrix hiji, Matrix tekubi,
                float *tekubi_a, float *tekubi_y, float *tekubi_z,
                Matrix tekubi1_m, Matrix tekubi2_m)
{
    Matrix      mkh_m, mkh_rm, wrk_m;

    loadmatrix(mune);
    multmatrix(kata);
    multmatrix(hiji);
    getmatrix(mkh_m);
```

```
    rev_mat(mkh_m, mkh_rm);

    loadmatrix(mkh_rm);
    multmatrix(tekubi);

    /* 手首の相対マトリックスを作成 */
    getmatrix(wrk_m);

    /* 手首のＸ軸回転のマトリックスを作成 */
    GetTekubiX(num, wrk_m, tekubi_a, tekubi1_m);

    /* 手首のＹＺ軸回転のマトリックスを作成 */
    GetTekubiYZ(num, wrk_m, tekubi_y, tekubi_z, tekubi2_m);
]


/*****************************************************************
*       指の回転マトリックスを作成                                *
*                                                                 *
*       引数 :  fang[3]（指の回転角度）                           *
*               axis（回転軸）                                    *
*               hosei[3]（補正角度）                              *
*               m[3]（指の回転マトリックス）                      *
*       リターン値 : なし                                         *
*****************************************************************/
void
mk_yubi_rotm(fang, axis, hosei, m)
IN  float   *fang;
IN  char    axis;
IN  float   *hosei;
OUT Matrix  *m;
{
    int i;

    /*  0 : 第１関節   1 : 第２関節   2 : 第３関節  */

    pushmatrix();

    loadmatrix(idmat);
    rot(fang[3], 'y');
    rot(-hosei[0], 'y');
    rot(fang[0], axis);
    rot(hosei[0], 'y');
    getmatrix(m[0]);

    for (i = 1; i < 3; i++) [
        loadmatrix(idmat);
        rot(-hosei[i], 'y');
        rot(fang[i], 'z');      /*  第２関節以降は全てＹ軸回転  */
        rot(hosei[i], 'y');
        getmatrix(m[i]);
    ]

    popmatrix();
]


/* HIROSE 1993.5.31 */
/*
 * 関数名 : GetKataRotMatrix(float *a, Matrix m1, Matrix m2)
 *
 * 引数   : a   肩のＸ軸回転角度
 *          m1  肩のＸ軸回転成分のマトリックス
 *          m2  肩のＹ，Ｚ軸回転成分のマトリックス
 *
 * 戻り値 : なし
```



```
 *
 * 機能   : 肩のＸ軸回転、肩のＹ，Ｚ軸回転を別々に取り出す。
 *
 * 履歴   : 1993年5月31日 ; 作成 ; 広瀬　正俊
 */
void
GetKataRotMatrix( ka_m, a, m1, m2)
IN   Matrix  ka_m;
OUT  float   *a;
OUT  Matrix  m1, m2;
[
        float   e, r;

        /* 肩のＸＹＺの回転角度を取り出す */
        GetKakudo_xyz(ka_m, a, &e, &r);

        pushmatrix();

        loadmatrix(idmat);
        rot(*a, 'x');

        /* 肩のＸ軸方向の回転マトリックスの作成 */
        getmatrix(m1);

        loadmatrix(idmat);
        rot(*a, 'x');
        rot( e, 'y' );
        rot( r, 'z' );
        rot(-(*a), 'x');

        /* 肩のＹ、Ｘ軸方向の回転マトリックスを作成 */
        getmatrix(m2);

        popmatrix();

}
/* HIROSE */


/*****************************************************************
*       マトリックスにより座標値を変換                            *
*                                                                 *
*       引数 :  *d1（変換後の座標値格納）                         *
*               *d2（変換前の座標値格納）                         *
*               mat（変換用のマトリックス）                       *
*       リターン値 : なし                                         *
*****************************************************************/
void
coord_calc( d1, d2, mat )
OUT float   *d1;
IN  float   *d2;
IN  Matrix  mat;
[
        int     i, j;
        float   a[4];
        float   c[4];

        a[0] = *d2++;
        a[1] = *d2++;
        a[2] = *d2;
        a[3] = 1.0;

        for( i=0; i<4; i++ ) [
                c[i] = 0.0;
                for( j=0; j<4; j++ )
                        c[i] += a[j] * mat[j][i];
```

```
        }

        *dl++ = c[0];
        *dl++ = c[1];
        *dl   = c[2];

]


/*****************************************************************
*       回転させた時の座標値を計算                                *
*                                                                 *
*       引数 :  pnt1（回転前の座標値）                            *
*               pnt2（回転後の座標値設定領域）                    *
*               theta（回転角度）                                 *
*               axis（回転軸の指定 'x', 'y', 'z'）                *
*       リターン値 : なし                                         *
*****************************************************************/
void
get_rot_coord( pnt1, pnt2, angle, axis )
IN  float  pnt1[4];
OUT float  pnt2[4];
IN  float  angle;
IN  char   axis;
{
        int     i, j;
        Matrix  rot_m;

        pushmatrix();

        loadmatrix( idmat );

        rot( angle, axis );
        getmatrix( rot_m );

        pnt1[3] = 1.0;
        for( i=0; i<4; i++ ) [
                pnt2[i] = 0.0;
                for( j=0; j<4; j++ ) pnt2[i] += pnt1[j] * rot_m[j][i];
        }

        popmatrix();
]


/*****************************************************************
*       手首のＸ軸回転を取り出す                                  *
*                                                                 *
*       引数 :  m（手首の回転マトリックス）                       *
*       リターン値 : なし                                         *
*****************************************************************/
void
GetTekubiX(int num, Matrix m, float *tekubi_a, Matrix tekubi1_m)
{
    *tekubi_a = fatan2(m[1][2], m[2][2]) / M_PI * 180.0;

    pushmatrix();
    loadmatrix(idmat);
    rot(*tekubi_a, 'x');
    getmatrix(tekubi1_m);
    popmatrix();
]


/*****************************************************************
*       手首のＹ，Ｚ軸回転を取り出す                              *
```



```
*                                                                 *
*       引数 :  m（手首の回転マトリックス）                       *
*       リターン値 : なし                                         *
*****************************************************************/
void
GetTekubiYZ(int num, Matrix m, float *tekubi_y, float *tekubi_z, Matrix tekubi2_m)
{
    float       a, e, r;

    a = fatan2(m[0][1], m[0][0]);
    e = fatan(-m[0][2]/(m[0][0]/fcos(a)));
    r = fatan2(m[1][2], m[2][2]);
    a = a / M_PI * 180.0;
    e = e / M_PI * 180.0;
    r = r / M_PI * 180.0;

    pushmatrix();
    loadmatrix(idmat);
    rot(-r, 'x');
    rot(a, 'z');
    rot(e, 'y');
    rot(r, 'x');
    getmatrix(tekubi2_m);
    *tekubi_y = e;
    *tekubi_z = a;
    popmatrix();
]


/*****************************************************************
*       マトリックスより角度を取り出す                            *
*                                                                 *
*       引数 :  m （４×４のマトリックス）                        *
*               *rx （Ｘ軸回転の角度）                            *
*               *ry （Ｙ軸回転の角度）                            *
*               *rz （Ｚ軸回転の角度）                            *
*       リターン値 : なし                                         *
*****************************************************************/

GetKakudo_xyz(m,rx,ry,rz)
float           m[4][4];
float           *rz, *ry, *rx;
{

        *rx = fatan2(m[2][1], m[2][2] );
        *ry = fatan2(-m[2][0], m[2][2]/fcos(*rx));
        *rz = fatan2(m[1][0], m[0][0]);

        *rz = - *rz/M_PI * 180.0;
        *ry = - *ry/M_PI * 180.0;
        *rx = - *rx/M_PI * 180.0;
        if( *rx < -90){
                *rx += 180.;
                *ry =  90. + (90. - *ry);
                *rz += 180.;
        ]
]


/*****************************************************************
*       マトリックスより角度を取り出す                            *
*                                                                 *
*       引数 :  m （４×４のマトリックス）                        *
*               *rz （Ｚ軸回転の角度）                            *
*               *ry （Ｙ軸回転の角度）                            *
*               *rx （Ｘ軸回転の角度）                            *
```

```
*       リターン値 : なし                                              *
*****************************************************************/

GetKakudo_zyx(m,rz,ry,rx)
float           m[4][4];
float           *rz, *ry, *rx;
{

        *rz = fatan2(m[0][1], m[0][0] );
        *ry = fatan2(-m[0][2], m[0][0]/fcos(*rz));
        *rx = fatan2(m[1][2], m[2][2]);

        *rz = *rz/M_PI * 180.0;
        *ry = *ry/M_PI * 180.0;
        *rx = *rx/M_PI * 180.0;

}


/*****************************************************************
*       逆マトリックスを作成                                           *
*                                                                *
*       引数 : m1 (逆を作るマトリックス)                          *
*              m2 (逆マトリックス)                                *
*       リターン値 : なし                                              *
*****************************************************************/
void
rev_mat( m1, m2 )
IN  Matrix  m1;
OUT Matrix  m2;
{
#if 0
        int     i, j, k;
        double  t, u;

        cp_mat( m2, m1 );

        for( i=0; i<4; i++ ) {
                t = m2[i][i];
                for( j=0; j<4; j++ )  m2[i][j] /= t;

                m2[i][i] = 1 / t;
                for( j=0; j<4; j++ ) {
                        if( i != j ) {
                                u = m2[j][i];
                                for( k=0; k<4; k++ ) {
                                        if( i != k )    m2[j][k] -= m2[i][k] * u;
                                        else            m2[j][k] = -u / t;
                                ]
                        }
                }
        }
#else
    float       det, eps;
    int         rt;

    eps = 1.0e-6;
    cp_mat(m2, m1);
    rt = MatrixInversion((float *)m2, 4, 4, eps, &det);
    if (rt != 0) {
        fprintf(stderr, "MatrixInversion error:%d\n",rt);
    }
#endif
}
```



```
/*
 * 関数名 : MatrixInversion(float a, int l, int m, float eps, float *det)
 *
 * 引数   : a   : a[m][l]なるfloat型の配列名で、aijを格納する。逆行列が得られる。
 *          l   : メインプログラムで宣言した配列aの第2添字の値をint型の定数か変数名で与える。  l >= m
 *          m   : 配列aの内、逆行列を求めたい部分の行数をint型の定数か変数名で与える。
 *                500 >= m >= 2
 *          eps : 収束判定値（標準値1x10-6）をfloat型の定数か変数名で与える。
 *                eps > 0
 *          det : float型の変数名で、行列式の値が得られる。
 *
 * 戻り値 : 0   : エラーなし
 *          1   : 掃き出しの途中、ピボット要素がepsより小さい
 *          999 : m < 2, m > 500, eps <= 0 のいずれかである。
 *
 * 機能   : 正方行列の逆行列と行列式の値をGauss-Jordan法により求める。掃き出す際、軸（Pivot）として列要素の絶対値最大を選択するため行入れ換えを行なう。
 *
 * 履歴   : 1994年10月6日 ; 作成 ; 広瀬＼敦
 */
int
MatrixInversion(a, l, m, eps, det)
    float       *a, eps, *det;
    int         l, m;
{
    int         work[500], i, j, k, r, iw, s, t, u, v;
    double      w, wmax, pivot, api, wl;
    extern double       fabs();

    if (m < 2 || m > 500 || eps <= 0.0) {
        return(999);
    }
    wl = 1.0;
    for (i = 0; i < m; i++) {
        work[i] = i;
    }
    for (k = 0; k < m; k++) {
        wmax = 0.0;
        for (i = k; i < m; i++) {
            w = fabs(a[i*l+k]);
            if (w > wmax) {
                wmax = w;
                r = i;
            }
        }
        pivot = a[r*l+k];
        api = fabs(pivot);
        if (api <= eps) {
            *det = wl;
            return(1);
        }
        wl *= pivot;
        u = k * l;
        v = r * l;
        if (r != k) {
            wl = -wl;
            iw = work[k];
            work[k] = work[r];
            work[r] = iw;
            for (j = 0; j < m; j++) {
                s = u + j;
                t = v + j;
```

```
                w = a[s];
                a[s] = a[t];
                a[t] = w;
            }
        }
        for (i = 0; i < m; i++) {
            a[u+i] /= pivot;
        }
        for (i = 0; i < m; i++) {
            if (i != k) {
                v = i * l;
                s = v + k;
                w = a[s];
                if ( w != 0.0) {
                    for (j = 0; j < m; j++) [
                        if ( j != k) [
                            a[v+j] -= w * a[u+j];
                        }
                    }
                    a[s] = -w / pivot;
                }
            ]
        }
        a[u+k] = 1.0 / pivot;
    }

    for (i = 0; i < m; i++) {
        while(1) {
            k = work[i];
            if (k == i) [
                break;
            }
            iw = work[k];
            work[k] = work[i];
            work[i] = iw;
            for (j = 0; j < m; j++) {
                u = j * l;
                s = u + i;
                t = u + k;
                w = a[s];
                a[s] = a[t];
                a[t] = w;
            }
        }
    }
    *det = w1;
    return(0);
}


/*****************************************************************
*       マトリックスのコピー                                      *
*                                                                 *
*       引数  :  m1 (コピーされたマトリックス)                    *
*                m2 (コピーの基マトリックス)                      *
*       リターン値  :  なし                                       *
*****************************************************************/
void
cp_mat( m1, m2 )
OUT Matrix  m1;
IN  Matrix  m2;
{
    m1[0][0] = m2[0][0];
    m1[0][1] = m2[0][1];
    m1[0][2] = m2[0][2];
    m1[0][3] = m2[0][3];
```



```
    m1[1][0] = m2[1][0];
    m1[1][1] = m2[1][1];
    m1[1][2] = m2[1][2];
    m1[1][3] = m2[1][3];
    m1[2][0] = m2[2][0];
    m1[2][1] = m2[2][1];
    m1[2][2] = m2[2][2];
    m1[2][3] = m2[2][3];
    m1[3][0] = m2[3][0];
    m1[3][1] = m2[3][1];
    m1[3][2] = m2[3][2];
    m1[3][3] = m2[3][3];
]

prt_mat( mat, mname )
IN  Matrix  mat;
IN  char    *mname;
[
    int  i, j;

    fprintf( stderr, "---< %s >---\n", mname );
    for( i=0; i<4; i++ ) [
        for( j=0; j<4; j++ ) [
            fprintf( stderr, "%5.2f ",mat[i][j] );
        ]
        fprintf( stderr, "\n" );
    }
}

get_time( msg )
char  *msg;
{
    int    misec;
    static int      pmisec;
    struct timeval  tv;
    struct timezone tz;

    gettimeofday( &tv, &tz );

    misec = 1000000 * tv.tv_sec + tv.tv_usec;

    if( msg && pmisec )
        printf( "%s : TIME : %7d msec\n", msg,(misec-pmisec)/1000 );

    pmisec = misec;
}

/* END OF FILE */
```

```
/*=========================================================================
 * file:     impbody.c
 * author:   Karansher Singh
 * created:  July 28, 1994
 *=========================================================================
 */



#include "tool.h"
#include "vector.h"

Ipb ipb[MAXIMPS];
Joint jt[MAXJNTS];
float THR = 0.2;
float THR2 = 0.2;

extern float r1,sr1;



/* setting the implicit space functions and calibrating*/

/* a is a  imp prim.
   sets the prim centers to p1 p2, calcs lengths and
   sets a huge radius for calc max and avg distances.
*/

setPart(a,p1,p2,typ)
     int a;
     VERTEX p1,p2;
     int typ;
[
  ipb[a].type=typ;
  if (typ==SPHERE)
    [
      VECSet(ipb[a].c1,p1);
      VECSet(ipb[a].o1,p1);
    ]
  else
    [
      VECSet(ipb[a].o1,p1);
      VECSet(ipb[a].o2,p2);
      VECSet(ipb[a].c1,p1);
      VECSet(ipb[a].c2,p2);
      VECSub(ipb[a].o2,ipb[a].o1,ipb[a].oc);
      ipb[a].nc=VECLen(ipb[a].oc);
      ipb[a].ang=ipb[a].nc/2;
      VECSet(ipb[a].ouc,ipb[a].oc);
      VECNorm(ipb[a].ouc);
      VECSet(ipb[a].uc,ipb[a].ouc);
    ]
  ipb[a].r1=BIGGIE;
]

/* a is a sph imp prim.
   sets the prim center to p  calcs lengths and
   set radii to r1, sr1  and calculates weight of prim.
   also sets jt space values...
*/


setJoint(a,p,r,sr,l1,l2)
int a;
VERTEX p;
float r,sr;
```



```
int l1,l2;
{
  jt[a].l1=l1;
  jt[a].l2=l2;
  jt[a].comber=4;
  VECSet(ipb[a].o1,p);
  VECSet(ipb[a].c1,p);

  ipb[a].r1=r;
  ipb[a].sr1=sr;
#if 0
  ipb[a].weight=THR/quartF(sr/r);
#else
  ipb[a].weight=1.0/quartF(sr/r);
#endif
]




/*
    a : is an implicit prim
    b : is buhin part
    creates map and ipwt mem. for b if first time around
    returns the adds to max and avg distances of pts. of b
    from the skeleton of a. assumes raduis of a is BIGGIE
*/

fixpart(hm,a,b,mx,av,ct)
JINBUTU *hm;
int a,b;
float *mx,*av;
int *ct;
[
  int k,first=0,ini;
  float addup,*vs;
  VERTEX dumc;

  if (hm->ipwt[b]==NULL)
    {
      hm->ipwt[b]=(float *)malloc(sizeof(float)*hm->vmax[b]);
      hm->map[b]=(VERTEX *)malloc(sizeof(VERTEX)*hm->vmax[b]);
      first=1;
    ]

  for(k=0, vs=(float *)hm->vs[b]; k<hm->vmax[b]; k++, vs+=3)
    {
      if (ipb[a].type==SPHERE)
        ini=sphVal(a,vs,&addup,1);
      else
        ini=sphyVal(a,vs,dumc,&addup,1);

      if (ini)
        {
          *av+=addup;
          hm->ipwt[b][k]=addup;
          (*ct)++;
          if (addup>*mx)
            *mx=addup;
        ]
      else if
      (first)
        hm->ipwt[b][k]=-1;
    ]
]
```

```
calibImpSpace(JINBUTU *hm)
[
  float maxr;
  float sum;
  int cnt;

  readwaver("../wf/sph.obj",0);
  readwaver("../wf/sph2.obj",2);
  readwaver("../wf/sph2.obj",3);
  readwaver("../wf/cyl.obj",1);

 setContact(hm);

  /* set arms */

  setPart(LUARM,hm->center[KATA_D_L],hm->center[HIJI_L],SPHYLINDER);
  setPart(RUARM,hm->center[KATA_D_R],hm->center[HIJI_R],SPHYLINDER);
  setPart(LLARM,hm->center[HIJI_L],hm->center[UDE_L],SPHYLINDER);
  setPart(RLARM,hm->center[HIJI_R],hm->center[UDE_R],SPHYLINDER);

  setPart(HEAD,hm->center[ATAMA],hm->center[UDE_R],SPHERE);

  setPart(LULG,hm->center[SIRI_L],hm->center[HIZA_L],SPHYLINDER);
  setPart(RULG,hm->center[SIRI_R],hm->center[HIZA_R],SPHYLINDER);
  setPart(LLLG,hm->center[HIZA_L],hm->center[KUTU_L],SPHYLINDER);
  setPart(RLLG,hm->center[HIZA_R],hm->center[KUTU_R],SPHYLINDER);

  /* arm settings */

  sum=0;cnt=0;maxr=0;
  fixpart(hm,LUARM,KATA_D_L,&maxr,&sum,&cnt);
  sum=0;cnt=0;maxr=0;
  fixpart(hm,LUARM,KATA_N_L,&maxr,&sum,&cnt);
  fixpart(hm,LUARM,NINOUDE_L,&maxr,&sum,&cnt);
  fixpart(hm,LUARM,HIJI_L,&maxr,&sum,&cnt);
  sum/=(float)cnt;

  printf("luarm setting = %f %f %f\n",maxr,sum,maxr/sum);
  ipb[LUARM].r1=maxr;
  ipb[LUARM].sr1=sum;
#if 0
  ipb[LUARM].weight=THR/quartF(sum/maxr);
#else
  ipb[LUARM].weight=1.0/quartF(sum/maxr);
#endif


  sum=0;cnt=0;maxr=0;
  fixpart(hm,RUARM,KATA_D_R,&maxr,&sum,&cnt);
   sum=0;cnt=0;maxr=0;
  fixpart(hm,RUARM,KATA_N_R,&maxr,&sum,&cnt);
  fixpart(hm,RUARM,NINOUDE_R,&maxr,&sum,&cnt);
  fixpart(hm,RUARM,HIJI_R,&maxr,&sum,&cnt);
  sum/=(float)cnt;

  printf("ruarm setting = %f %f %f\n",maxr,sum,maxr/sum);
  ipb[RUARM].r1=maxr;
  ipb[RUARM].sr1=sum;
#if 0
  ipb[RUARM].weight=THR/quartF(sum/maxr);
#else
  ipb[RUARM].weight=1.0/quartF(sum/maxr);
#endif
```



```
  sum=0;cnt=0;maxr=0;
  fixpart(hm,RLARM,TE_R,&maxr,&sum,&cnt);
  sum=0;cnt=0;maxr=0;
  fixpart(hm,RLARM,UDE_R,&maxr,&sum,&cnt);
  fixpart(hm,RLARM,HIJI_R,&maxr,&sum,&cnt);
  sum/=(float)cnt;

  printf("rlarm setting = %f %f %f\n",maxr,sum,maxr/sum);
  ipb[RLARM].r1=maxr;
  ipb[RLARM].sr1=sum;
#if 0
  ipb[RLARM].weight=THR/quartF(sum/maxr);
#else
  ipb[RLARM].weight=1.0/quartF(sum/maxr);
#endif


  sum=0;cnt=0;maxr=0;
  fixpart(hm,LLARM,TE_L,&maxr,&sum,&cnt);
  sum=0;cnt=0;maxr=0;
  fixpart(hm,LLARM,UDE_L,&maxr,&sum,&cnt);
  fixpart(hm,LLARM,HIJI_L,&maxr,&sum,&cnt);
  sum/=(float)cnt;

  ipb[LLARM].r1=maxr;
  ipb[LLARM].sr1=sum;
#if 0
  ipb[LLARM].weight=THR/quartF(sum/maxr);
#else
  ipb[LLARM].weight=1.0/quartF(sum/maxr);
#endif

  printf("llarm setting = %f %f %f %f\n",maxr,sum,maxr/sum,ipb[LLARM].weight);

  setJoint(LELBOW,ipb[LLARM].o1,ipb[LUARM].r1,ipb[LUARM].sr1,LUARM,LLARM);
  setJoint(RELBOW,ipb[RLARM].o1,ipb[RUARM].r1,ipb[RUARM].sr1,RUARM,RLARM);

  hm->bl[KATA_D_L]=LSHOULDER;
  hm->bl[KATA_D_R]=RSHOULDER;
  hm->bl[KATA_N_L]=LSHOULDER;
  hm->bl[KATA_N_R]=RSHOULDER;

  hm->bl[NINOUDE_L]=LUARM;
  hm->bl[NINOUDE_R]=RUARM;

  hm->bl[HIJI_L]=LELBOW;
  hm->bl[HIJI_R]=RELBOW;

  hm->bl[UDE_L]=LELBOW;
  hm->bl[UDE_R]=RELBOW;

  /* leg settings */

  sum=0;cnt=0;maxr=0;
  fixpart(hm,RULG,SIRI_R,&maxr,&sum,&cnt);
  fixpart(hm,RULG,MOMO_R,&maxr,&sum,&cnt);
  fixpart(hm,RULG,HIZA_R,&maxr,&sum,&cnt);
  sum/=(float)cnt;

  printf("ruleg setting = %f %f %f\n",maxr,sum,maxr/sum);
  ipb[RULG].r1=maxr;
  ipb[RULG].sr1=sum;
#if 0
  ipb[RULG].weight=THR/quartF(sum/maxr);
#else
  ipb[RULG].weight=1.0/quartF(sum/maxr);
```

```
#endif

  sum=0;cnt=0;maxr=0;
 /* fixpart(hm,RLLG,HIZA_R,&maxr,&sum,&cnt);*/
  fixpart(hm,RLLG,SUNE_R,&maxr,&sum,&cnt);
  sum/=(float)cnt;

  printf("rlleg setting = %f %f %f\n",maxr,sum,maxr/sum);
  ipb[RLLG].rl=maxr-2.5;
  ipb[RLLG].srl=sum;
#if 0
  ipb[RLLG].weight=THR/quartF(sum/maxr);
#else
  ipb[RLLG].weight=1.0/quartF(sum/maxr);
#endif

sum=0;cnt=0;maxr=0;
  fixpart(hm,LULG,SIRI_L,&maxr,&sum,&cnt);
  fixpart(hm,LULG,MOMO_L,&maxr,&sum,&cnt);
  fixpart(hm,LULG,HIZA_L,&maxr,&sum,&cnt);
  sum/=(float)cnt;

  printf("luleg setting = %f %f %f\n",maxr,sum,maxr/sum);
  ipb[LULG].rl=maxr;
  ipb[LULG].srl=sum;
#if 0
  ipb[LULG].weight=THR/quartF(sum/maxr);
#else
  ipb[LULG].weight=1.0/quartF(sum/maxr);
#endif

  sum=0;cnt=0;maxr=0;
 /* fixpart(hm,LLLG,HIZA_L,&maxr,&sum,&cnt);*/
  fixpart(hm,LLLG,SUNE_L,&maxr,&sum,&cnt);
  sum/=(float)cnt;

  printf("llleg setting = %f %f %f\n",maxr,sum,maxr/sum);
  ipb[LLLG].rl=maxr-2.5;
  ipb[LLLG].srl=sum;
#if 0
  ipb[LLLG].weight=THR/quartF(sum/maxr);
#else
  ipb[LLLG].weight=1.0/quartF(sum/maxr);
#endif



  sum=0;cnt=0;maxr=0;
  fixpart(hm,HEAD,ATAMA,&maxr,&sum,&cnt);
  sum/=(float)cnt;

  ipb[HEAD].rl=maxr;
  ipb[HEAD].srl=sum;
#if 0
  ipb[HEAD].weight=THR/quartF(sum/maxr);
#else
  ipb[HEAD].weight=1.0/quartF(sum/maxr);
#endif

  printf("torso settings... %f %f %f,%f %f %f\n",
         hm->kosispace.orig[0][0][0],hm->kosispace.orig[0][0][1],
         hm->kosispace.orig[0][0][2],
         hm->douspace.orig[3][2][0],hm->douspace.orig[3][2][1],
         hm->douspace.orig[3][2][2]);

  printf("torso set center %f %f %f, %f %f %f\n",
```



```
         hm->shcenter[0][0],hm->shcenter[0][1],
         hm->shcenter[0][2],
         hm->shcenter[1][0],hm->shcenter[1][1],
         hm->shcenter[1][2]);

  printf("torso set center %f %f %f, %f %f %f\n",
         hm->center[SIRI_L][0],hm->center[SIRI_L][1],
         hm->center[SIRI_L][2],
         hm->center[SIRI_R][0],hm->center[SIRI_R][1],
         hm->center[SIRI_R][2]);

  setptWt(hm,KATA_D_L,LUARM,TORSO,LSHOULDER);
  setptWt(hm,KATA_N_L,LUARM,TORSO,LSHOULDER);
  setptWt(hm,NINOUDE_L,LUARM,LLARM,LELBOW);
  setptWt(hm,HIJI_L,LUARM,LLARM,LELBOW);
  setptWt(hm,UDE_L,LUARM,LLARM,LELBOW);


  setptWt(hm,KATA_D_R,RUARM,TORSO,RSHOULDER);
  setptWt(hm,KATA_N_R,RUARM,TORSO,RSHOULDER);
  setptWt(hm,NINOUDE_R,RUARM,RLARM,RELBOW);
  setptWt(hm,HIJI_R,RUARM,RLARM,RELBOW);
  setptWt(hm,UDE_R,RUARM,RLARM,RELBOW);

  setptWt(hm,ATAMA,HEAD,RLARM,RELBOW);



]

/***** in setting wt and pt. use c as far as possible and b otherwise ****/

setptWt(hm,a,b,c,d)
JINBUTU *hm;
int a,b,c,d;
{

 int k,f1=0,f2=0,tp;
  float addup,addup2,*vs,d1,d2,d3,tmp;
  VERTEX dumc,v1,v2;


  for(k=0, vs=(float *)hm->vs[a]; k<hm->vmax[a]; k++, vs+=3)
    [
      d1=d2=d3=0;
      tp=0;

      if (ipb[b].type==SPHERE)
        {
          if ((tp=sphVal(b,vs,&addup,1)))
            [
#if 0
              d1=ipb[b].weight*quartF(addup/ipb[b].r1);
#else
              d1=ipb[b].weight*quartF(addup/ipb[b].r1)*THR;
#endif
              f1=1;
              hm->ipwt[a][k]=THR/(d1);
              VECSet(hm->map[a][k],ipb[b].o1);
            }
          else printf("sineadie\n");
        }
      else
        {
          if ((tp=sphyVal(b,vs,dumc,&addup,1)))
            [
```

```
#if 0
              d1=ipb[b].weight*quartF(addup/ipb[b].r1);
#else
              d1=ipb[b].weight*quartF(addup/ipb[b].r1)*THR;
#endif
              f1=1;
              hm->ipwt[a][k]=THR/(d1);
              VECSet(hm->map[a][k],dumc);
            ]

          if ((c!=TORSO) && ((tp=sphyVal(c,vs,dumc,&addup,1))))
            [
#if 0
              d2=ipb[c].weight*quartF(addup/ipb[c].r1);
#else
              d2=ipb[c].weight*quartF(addup/ipb[c].r1)*THR;
#endif
              f2=1;
              VECSet(hm->map[a][k],dumc);
              hm->ipwt[a][k]=THR/(d2);
            ]
        }
    ]
]

oldsetptWt(hm,a,b,c,d)
JINBUTU *hm;
int a,b,c,d;
{

 int k,f1=0,f2=0,tp;
  float addup,addup2,*vs,d1,d2,d3,tmp;
  VERTEX dumc,v1,v2;


  for(k=0, vs=(float *)hm->vs[a]; k<hm->vmax[a]; k++, vs+=3)
    [
      d1=d2=d3=0;
      tp=0;
      if (hm->ipwt[a][k]<0)
      printf("yohoho and a b o r \n");

   /*   if ((tp=sphyVal(b,vs,dumc,&addup,1)))
        [
#if 0
          d1=ipb[b].weight*quartF(addup/ipb[b].r1);
#else
          d1=ipb[b].weight*quartF(addup/ipb[b].r1)*THR;
#endif
          f1=1;
          if (tp==3)
            VECSet(hm->map[a][k],dumc);
        }*/
      if ((c!=TORSO) && ((tp=sphyVal(c,vs,dumc,&addup,1))))
        [
#if 0
          d2=ipb[c].weight*quartF(addup/ipb[c].r1);
#else
          d2=ipb[c].weight*quartF(addup/ipb[c].r1)*THR;
#endif
          f2=1;
            VECSet(hm->map[a][k],dumc);
        ]

 /*     if (sphVal(d,vs,&addup,1))
        [
```



```
#if 0
          d3=ipb[d].weight*quartF(addup/ipb[d].r1);
#else
          d3=ipb[d].weight*quartF(addup/ipb[d].r1)*THR;
#endif
          if (tp!=3)
            VECSet(hm->map[a][k],ipb[d].c1);
        ]
   */
      if (f1 && f2)
        [
          tmp=diffFunc(d1,d2,jt[d].comber,1);
          hm->ipwt[a][k]=THR/(tmp+d3);
        ]
      else
        hm->ipwt[a][k]=THR/(d1+d2+d3);

    ]

}

/********************** animation routines ***************/


updateImpSpace(JINBUTU *hm)
{
  int i,j;
  VERTEX tmp;

  /* only transform the hiji */

  for (i=0;i<3;i++)
    {
      ipb[RLARM].c2[i]=ipb[RLARM].o2[0]*hm->mat[HIJI_R][0][i]+
        ipb[RLARM].o2[1]*hm->mat[HIJI_R][1][i]+
          ipb[RLARM].o2[2]*hm->mat[HIJI_R][2][i]+hm->mat[HIJI_R][3][i];

      ipb[LLARM].c2[i]=ipb[LLARM].o2[0]*hm->mat[HIJI_L][0][i]+
        ipb[LLARM].o2[1]*hm->mat[HIJI_L][1][i]+
          ipb[LLARM].o2[2]*hm->mat[HIJI_L][2][i]+hm->mat[HIJI_L][3][i];
    }

  VECSub(ipb[LLARM].c2,ipb[LLARM].c1,ipb[LLARM].uc);
  VECNorm(ipb[LLARM].uc);

  VECSub(ipb[RLARM].c2,ipb[RLARM].c1,ipb[RLARM].uc);
  VECNorm(ipb[RLARM].uc);

#if 1
  setptWt(hm,KATA_D_L,LUARM,TORSO,LSHOULDER);
  setptWt(hm,KATA_N_L,LUARM,TORSO,LSHOULDER);
  setptWt(hm,NINOUDE_L,LUARM,LLARM,LELBOW);
  setptWt(hm,HIJI_L,LUARM,LLARM,LELBOW);
  setptWt(hm,UDE_L,LUARM,LLARM,LELBOW);


  setptWt(hm,KATA_D_R,RUARM,TORSO,RSHOULDER);
  setptWt(hm,KATA_N_R,RUARM,TORSO,RSHOULDER);
  setptWt(hm,NINOUDE_R,RUARM,RLARM,RELBOW);
  setptWt(hm,HIJI_R,RUARM,RLARM,RELBOW);
  setptWt(hm,UDE_R,RUARM,RLARM,RELBOW);

  setptWt(hm,ATAMA,HEAD,RLARM,RELBOW);
#endif

  placeSphere();
```

```
  deformSphere(hm,LLARM);
  deformSphere(hm,RLARM);
  deformSphereSphere(hm,HEAD);

  deformPart(hm,LLARM,UDE_L);
  deformPart(hm,RLARM,UDE_R);
  deformSpherePart(hm,HEAD,ATAMA);

  if (collide(LLARM,RLARM))
    {
      deformPartPart(hm,LLARM,UDE_L,RLARM);
      deformPartPart(hm,RLARM,UDE_R,LLARM);
    }

  /* henkeipart(hm,HIJI_R);
  henkeipart(hm,HIJI_L); */


]


float df(hm,a,b,c,d,p)
JINBUTU *hm;
int a,b,c,d;
VERTEX p;
{
  int f1=0,f2=0;
  float addup,addup2,d1,d2,d3,tmp;
  VERTEX dumc,dumc2;

  d1=d2=d3=0;

  if (cylVal(b,p,dumc,&addup,&addup2,0))
    {
      tmp=fabs(addup2/ipb[b].ang -1 );
#if 0
      d1=ipb[b].weight*quartF(addup/ipb[b].r1)*sqrt(quartF(tmp));
#else
      d1=ipb[b].weight*quartF(addup/ipb[b].r1)*sqrt(quartF(tmp))*THR;
#endif
      f1=1;
    }
  if ((c!=TORSO) && (cylVal(c,p,dumc2,&addup,&addup2,0)))
    {
      tmp=fabs(addup2/ipb[c].ang -1 );
#if 0
      d2=ipb[c].weight*quartF(addup/ipb[c].r1)*sqrt(quartF(tmp));
#else
      d2=ipb[c].weight*quartF(addup/ipb[c].r1)*sqrt(quartF(tmp))*THR;
#endif
      f2=1;
    }

  if (sphVal(d,p,&addup,0))
    [
#if 0
      d3=ipb[d].weight*quartF(addup/ipb[d].r1);
#else
      d3=ipb[d].weight*quartF(addup/ipb[d].r1)*THR;
#endif
    ]

  if (f1 && f2)
    [
      addup=(float)pow((double)d1,jt[d].comber);
```



```
      addup2=(float)pow((double)d2,jt[d].comber);
      tmp=fabs(addup-addup2);
      addup=(float)pow((double)tmp,1.0/(float)jt[d].comber);
      return(addup+d3);
    ]
  else
    return(d1+d2+d3);
]



henkeipart(hm,a)
JINBUTU *hm;
int a;
[
  int k,f1=0,f2=0,b,c,d,j;
  float addup,addup2,*vs,d1,d2,d3,tmp,far,near,denf,denn,dist,den,mval;
  VERTEX dumc,dumc2,ctr,dir,p;


  if ((d=hm->bl[a])>MAXJNTS-1)
    printf("sine a die\n");

  b=jt[d].11;
  c=jt[d].12;


  for(k=0, vs=(float *)hm->va[a]; k<hm->vmax[a]; k++, vs+=3)
    {
      d1=d2=d3=0;

      if (cylVal(b,vs,dumc,&addup,&addup2,0))
        {
          tmp=fabs(addup2/ipb[b].ang -1 );
#if 0
          d1=ipb[b].weight*quartF(addup/ipb[b].r1)*sqrt(quartF(tmp));
#else
          d1=ipb[b].weight*quartF(addup/ipb[b].r1)*sqrt(quartF(tmp))*THR;
#endif
          f1=1;
          dist=addup;
        ]
      if ((c!=TORSO) && (cylVal(c,vs,dumc2,&addup,&addup2,0)))
        [
          tmp=fabs(addup2/ipb[c].ang -1 );
#if 0
          d2=ipb[c].weight*quartF(addup/ipb[c].r1)*sqrt(quartF(tmp));
#else
          d2=ipb[c].weight*quartF(addup/ipb[c].r1)*sqrt(quartF(tmp))*THR;
#endif
          f2=1;
          dist=addup;
        ]

      if (sphVal(d,vs,&addup,0))
        [
#if 0
          d3=ipb[d].weight*quartF(addup/ipb[d].r1);
#else
          d3=ipb[d].weight*quartF(addup/ipb[d].r1)*THR;
#endif
          if ((f1 && f2) || (!f1 && !f2))
            dist=addup;
        ]
```

```
      if (f1 && f2)
        {
          VECSet(ctr,ipb[d].c1);
          far=ipb[d].r1;
        }
      else if (f1)
        {
          VECSet(ctr,dumc);
          far=ipb[d].r1;
        }
      else if (f2)
        {
          VECSet(ctr,dumc2);
          far=ipb[d].r1;
        }
      else
        {
          VECSet(ctr,ipb[d].c1);
          far=ipb[d].r1;
        }



      if (f1 && f2)
        {
          addup=(float)pow((double)d1,jt[d].comber);
          addup2=(float)pow((double)d2,jt[d].comber);
          tmp=fabs(addup-addup2);
          addup=(float)pow((double)tmp,1.0/(float)jt[d].comber);
          den=(addup+d3)*hm->ipwt[a][k];
        }
      else
        den=hm->ipwt[a][k]*(d1+d2+d3);

      near=0;
      denn=hm->ipwt[a][k];
      denf=0;

      if (fabs(mval=(den-THR))<=EPS)
        {
          continue;
        }
      if (mval<0)
        {
          far=dist;
          denf=den;
        }
        else
         {
           near=k;
           denn=den;
         ]

      VECSub(vs,ctr,dir);
      VECNorm(dir);


      j=0;
      while ((far-near>EPS) && (j<5))
        {
          j++;
          dist= (near*(denf-THR)-far*(denn-THR))/(denf-denn);

          p[0]=ctr[0] + dir[0]*dist;
          p[1]=ctr[1] + dir[1]*dist;
          p[2]=ctr[2] + dir[2]*dist;
```



```
          den=df(hm,a,b,c,d,p)*hm->ipwt[a][k];

          if (fabs(mval=(den-THR))<=EPS)
            continue;

          if (mval<0)
            {
              far=dist;
              denf=den;
            }
          else
            {
              near=dist;
              denn=den;
            ]
        }
      /*printf("den=%f\n",den);*/
      VECSet(vs,p);
    }
]
```

```
/*=============================================================================
 * file:     impbody.c
 * author:   Karansher Singh
 * created:  July 28, 1994
 *=============================================================================
 */



#include "tool.h"
#include "vector.h"


VERTEX *spv[4],*spn[4];
int *spf[4],spvc[4],spfc[4];


/****************** display of imp space related routines *****************/

readwaver(filename,w)
char *filename;
int w;
{
  FILE  *fp;
  char  buff[LINE_BUF];
  char  key[11];
  float x, y, z, o;
  int           tv[4], tvt[4],vals,jj;
  int           vcnt, vtcnt, pcnt;
  int           rt;
  float *v, *vt,*vn, *a;
  int           *pol;

  /* Open .obj file */
  if((fp = fopen(filename, "r")) == NULL) {
        fprintf(stderr, "<<ERROR>> %s is not opened.\n",filename);
        exit();
    }

    /* counter set */
    vcnt = vtcnt = pcnt = 0;
    while(fgets(buff, LINE_BUF, fp) != NULL) {
        if(!buff[0] || buff[0] == '#' || buff[0] == '\n')
            continue;
        sscanf(buff, "%10s", key);

        if(strcmp("v", key) == 0) {
            vcnt++;
        } else if(strcmp("vt", key) == 0) {
            vtcnt++;
        } else if((strcmp("fo", key) == 0) || (strcmp("f", key) == 0)) {
            pcnt++;
        }
    }
    spvc[w] = vcnt;
    spfc[w] = pcnt;
    rewind(fp);

  /* memory allocation */
  spv[w] = (VERTEX  *)malloc(sizeof(VERTEX)*vcnt);
  spn[w] = (VERTEX  *)malloc(sizeof(VERTEX)*vcnt);
  spf[w] = (int  *)malloc(sizeof(int)*pcnt*6);


  v=(float *)spv[w];
  vn=(float *)spn[w];
```



```
  pol=(int  *)spf[w];

  while(fgets(buff, LINE_BUF, fp) != NULL) [
    if(!buff[0] || buff[0] == '#' || buff[0] == '\n')
      continue;
    sscanf(buff, "%10s", key);

    if(strcmp("v", key) == 0) [
      /* add vertex to the vertex list */
      rt = sscanf(buff, "%s %f %f %f", key, &x, &y, &z);
      if(rt == 4) {
        v[0] = x; vn[0]=-x;
        v[1] = y; vn[1]=-y;
        v[2] = z; vn[2]=-z;
        v += 3;
        VECNorm(vn);
        vn+=3;
      ]
      else [
        fprintf(stderr, "<<ERROR>> add vertex %s\n", buff);
        exit();
      ]
    ]
    else if((strcmp("fo", key) == 0) || (strcmp("f", key) == 0)) [
      /* add polygon to the polygon list */
      if((vals=sscanf(buff, "%s %d/%d %d/%d %d/%d %d/%d",
                key, &tv[0], &tvt[0], &tv[1], &tvt[1],
                &tv[2], &tvt[2], &tv[3], &tvt[3])) < 7) [
        fprintf(stderr, "<<ERROR>> add polygon %s\n", buff);
        exit();
      } else {
        pol[0]=(vals-1)/2;
        for (jj=1;jj<=pol[0];jj++)
          pol[jj]=tv[jj-1]-1;

        pol+=(pol[0]+1);
      ]
    ]
  ]
  fclose(fp);
  fprintf(stderr, "sphyl read OK.\n");
  return;
}


drawwaver(float *lineclr,int w)
{
  int         i, *pol,j;

  c3f(lineclr);
  pol = (int *)(spf[w]);
  for(i = 0; i < spfc[w]; i++)
    {
      bgnclosedline();
      for (j=1;j<=pol[0];j++)
        v3f((float *)(spv[w]+pol[j]));

      endclosedline();
      pol+=(pol[0]+1);
    ]
]

drawgrwaver(int w)
{
  int         i, *pol,j;
```

```
  pol = (int *)(spf[w]);
  for(i = 0; i < spfc[w]; i++)
    {
      bgnpolygon();
      for (j=1;j<=pol[0];j++)
        [
          n3f((float *)(spn[w]+pol[j]));
          v3f((float *)(spv[w]+pol[j]));
        ]
      endpolygon();
      pol+=(pol[0]+1);
    ]
]



trSphyl(c,sc,rr,w,lineclr)
     VERTEX c;
     float sc,rr;
     VERTEX w;
     float *lineclr;
{

  pushmatrix();

  translate(w[0],w[1],w[2]);
  rot(asin(-c[2])*180/3.1415, 'x');
  rot(asin(-c[0])*180/3.1415, 'z');
  scale(rr,sc,rr);
  drawwaver(lineclr,1);
  popmatrix();
]

trSph(sc,w,lineclr,ss)
     float sc;
     VERTEX w;
     float *lineclr;
     int ss;
{

  pushmatrix();
  translate(w[0],w[1],w[2]);
  scale(sc,sc,sc);
  drawwaver(lineclr,ss);
  popmatrix();
]

trgrSph(sc,w,ss)
     float sc;
     VERTEX w;
     int ss;
{

  pushmatrix();
  translate(w[0],w[1],w[2]);
  scale(sc,sc,sc);
  drawgrwaver(ss);
  popmatrix();
]


/******************** end of display related rtns ********************/

int solve_quad(aa,bb,cc,r1,r2)
float aa,bb,cc,*r1,*r2;
{float temp;
```



```
 if (fabs(aa)<EPS)
   return(0);
 temp = bb*bb - 4*aa*cc;
 if (temp >= 0)
   [
     *r1 = (-bb - sqrt(temp))/(2*aa);
     *r2 = (-bb + sqrt(temp))/(2*aa);
     return(1);
   ]
 else
   [
     printf("ERROR : complex roots\n");
     return(0);
   ]
]

float diffFunc(a,b,n,w)
float a,b,n,w;
{
  float tp1,tp2,tp3;

       if (b>THR)
         return(a+THR-w*b);
       else
         return (a);

/*  tp1=(float)pow((double)a,n);
  tp2=(float)pow((double)b,n);
  tp3=fabs(tp1-tp2);
  tp1=(float)pow((double)tp3,1/n);

  return(tp1);*/
]



float diffFunc2(a,b,n,w)
float a,b,n,w;
{
       if (b>THR2)
         return(a+THR2-w*b);
       else
         return (a);
/*
  float tp1,tp2,tp3;

  tp1=(float)pow((double)a,n);
  tp2=(float)pow((double)b,n);
  tp3=fabs(tp1-tp2);
  tp1=(float)pow((double)tp3,1/n);

  return(tp1);
*/
]

float cvFunc(a,b,c)
     int a;
     float b,c;
[
  float tmp;

  tmp=fabs(c/ipb[a].ang -1 );
  return(ipb[a].weight*quartF(b/ipb[a].r1)*sqrt(quartF(tmp)));
]
```

```
/* returns normal vector through  cpt */

int oldsphyVal(ind,p,cpt,s,wh)
int ind;
VERTEX p;
VERTEX cpt;
float *s;
int wh;
{
  VERTEX     c1,c2,c,r,ans;
  float      len,dot;

  if (wh)
    {
      VECSet(c1 , ipb[ind].o1);
      VECSet(c2 , ipb[ind].o2);
      VECSet(c ,  ipb[ind].ouc);
      len= ipb[ind].nc;
    }
  else
    {
      VECSet(c1 , ipb[ind].c1);
      VECSet(c2 , ipb[ind].c2);
      VECSet(c ,  ipb[ind].uc);
      len= ipb[ind].nc;
    }

  VECSub(p,c1,r);
  dot = VECDot(c,r);
  if (dot <= 0)
    {
      *s=VECLen(r);
      cpt=r;
      if (*s<=ipb[ind].r1)
        return(1);
      else
        return(0);
    }
  if (dot >= len)
    {
      VECSub(p,c2,ans);
      *s=VECLen(ans);
      cpt=ans;
      if (*s<=ipb[ind].r1)
        return(2);
      else
        return(0);
    }

  ans[0] = r[0] - dot*c[0];
  ans[1] = r[1] - dot*c[1];
  ans[2] = r[2] - dot*c[2];
  *s=VECLen(ans);
  cpt=ans;
  if (*s<=ipb[ind].r1)
    return(3);
  else
    return(0);
}


int sphyVal(ind,p,cpt,s,wh)
```



```
int ind;
VERTEX p;
VERTEX cpt;
float *s;
int wh;
{
  VERTEX     c1,c2,c,r,ans;
  float      len,dot;

  if (wh)
    {
      VECSet(c1 , ipb[ind].o1);
      VECSet(c2 , ipb[ind].o2);
      VECSet(c ,  ipb[ind].ouc);
      len= ipb[ind].nc;
    }
  else
    {
      VECSet(c1 , ipb[ind].c1);
      VECSet(c2 , ipb[ind].c2);
      VECSet(c ,  ipb[ind].uc);
      len= ipb[ind].nc;
    }

  VECSub(p,c1,r);
  dot = VECDot(c,r);
  if (dot <= 0)
    {
      *s=VECLen(r);
      VECSet(cpt,c1);
      if (*s<=ipb[ind].r1)
        return(1);
      else
        return(0);
    }
  if (dot >= len)
    {
      VECSub(p,c2,ans);
      *s=VECLen(ans);
      VECSet(cpt,c2);
      if (*s<=ipb[ind].r1)
        return(2);
      else
        return(0);
    }

  ans[0] = r[0] - dot*c[0];
  ans[1] = r[1] - dot*c[1];
  ans[2] = r[2] - dot*c[2];
  *s=VECLen(ans);
  VECSub(p,ans,cpt);
  if (*s<=ipb[ind].r1)
    return(3);
  else
    return(0);
}


int cylVal(ind,p,cpt,s,s2,wh)
int ind;
VERTEX p;
VERTEX cpt;
float *s,*s2;
int wh;
{
  VERTEX     c1,c2,c,r,ans;
```

```
  float      len,dot;

  if (wh)
    {
      VECSet(c1 , ipb[ind].o1);
      VECSet(c2 , ipb[ind].o2);
      VECSet(c ,  ipb[ind].ouc);
      len= ipb[ind].nc;
    }
  else
    {
      VECSet(c1 , ipb[ind].c1);
      VECSet(c2 , ipb[ind].c2);
      VECSet(c ,  ipb[ind].uc);
      len= ipb[ind].nc;
    }

  VECSub(p,c1,r);
  dot = VECDot(c,r);
  if ((dot > 0) && (dot < len))
    {
      ans[0] = r[0] - dot*c[0];
      ans[1] = r[1] - dot*c[1];
      ans[2] = r[2] - dot*c[2];
      *s=VECLen(ans);
      *s2=dot;
      VECSub(p,ans,cpt);
      if (*s<=ipb[ind].r1)
        return(1);
      else
        return(0);
    }
  else
    return(0);
}


int sphVal(ind,p,s,wh)
int ind;
VERTEX p;
float *s;
int wh;
{
  VERTEX     c1,c2,c,r,ans;
  float      len,dot;

  if (wh)
    {
      VECSet(c1 , ipb[ind].o1);
    }
  else
    {
      VECSet(c1 , ipb[ind].c1);
    }
  VECSub(p,c1,r);
  dot = VECDot(r,r);
  *s=VECLen(r);
  if (*s <= ipb[ind].r1)
    return(1);
  else
    return(0);
}

int sphereVal(c,rr,pp,s)
VERTEX c;
float rr;
```



```
VERTEX pp;
float *s;
{
  VERTEX     r,ans;
  float      len,dot;

  VECSub(pp,c,r);
  *s=VECLen(r);;
  if (*s <= rr)
    return(1);
  else
    return(0);
}

/**************** prim func calc rtns ***********************/


int collide(a,b)
int a,b;
{
  VERTEX     c1,c2,c,r,ans;
  float      len,dot;

  sphyVal(a,ipb[b].c2,r,&dot,0);
  if (dot<=ipb[b].sr1+ipb[a].sr1)
    return(1);
  sphyVal(b,ipb[a].c2,r,&dot,0);
  if (dot<=ipb[a].sr1+ipb[b].sr1)
    return(1);

  VECCross(ipb[a].uc,ipb[b].uc,c);
  VECNorm(c);
  VECSub(ipb[a].c1,ipb[b].c1,r);

  dot=VECDot(r,c);
  if (dot>ipb[b].sr1+ipb[a].sr1)
    return(0);

/*  if ((ipb[a].c2[0]*ipb[a].c1[0]<0) && (ipb[b].c2[0]*ipb[b].c1[0]<0))
  */  return(1);
  /*else
    return(0);*/
}
```

```
/*                                  */
/*  readmap.c                       */
/*                                  */
/* map image file read libraliy */
/*                                  */

#include <gl/gl.h>
#include <gl/image.h>
#include <string.h>
#include <stdio.h>

#define MAXMAPNAME 50  /* マップファイル名の最大長 */
#define MAXMAPDB   50  /* マップの数の最大数       */

/*                                  */
/* マップファイルデータベース       */
/*                                  */

struct MAPDB {
  long mapindex;
  char mapname[MAXMAPNAME];
};

static struct MAPDB mapdb[MAXMAPDB];

static int mapdbindex = 0;

static float texprops[] = {TX_MINFILTER, TX_POINT,
                           TX_MAGFILTER, TX_BILINEAR,
                           TX_WRAP, TX_REPEAT, TX_NULL};

static float tevprops[] = { TV_MODULATE, TV_NULL };

/* wavefront 用定義 */

/* for wavefront rla file read */

typedef struct {
    short left, right, bottom, top;
} WINDOW_S;

typedef struct {
    WINDOW_S    window;
    WINDOW_S    active_window;
    short frame;
    short storage_type;
    short num_chan;
    short num_matte;
    short num_aux;
    short aux_mask;
} RLA_HEADER;

static RLA_HEADER header;

/* for wavefront tex file read */

typedef struct {
    int tex_id;
    int tex_type;
    int tex_size;
    int tex_chan;
    int tex_dim;
} TEX_HEADER;

static TEX_HEADER texheader;
```



```
/* イメージ読み込み用バッファ */

static int data_xsize, data_ysize, data_zsize;
static RGBvalue crbuf[2048], cgbuf[2048], cbbuf[2048];
static char  cbuf[4096];
static short rbuf[4096], gbuf[4096], bbuf[4096];
static unsigned long dbuf[1280 * 1024];

static int      load_iris_image();
static int      load_wave_image();
static int      load_tex_image();
static void     seekerror();
static void     decode();

read_map_tex(s)
  char *s;
{
  char *p, *t;
  int val;

printf("%s\n",s);

  if(mapdbindex == MAXMAPDB){
    fprintf(stderr, "MAPLIB: error map database full \n");
    return 0;
  }

  if(search_map_index(s) != 0){
    fprintf(stderr, "MAPLIB: map %s is already read.\n", s);
    return 0;
  }

  p = strrchr(s, '.');

  if(p == NULL){
    val = load_iris_image(s);
  }
  else if(strcmp(p, ".rla") == 0){
printf(".rla file read ! \n");
    val = load_wave_image(s);
  }
  else if(strcmp(p, ".tex") == 0){
printf(".tex file read ! \n");
    val = load_tex_image(s);
  }
  else{
printf(".iris file read ! \n");
    val = load_iris_image(s);
  }

  if(val){
    fprintf(stderr, "MAPLIB: image file read error %s \n", s);
    return 0;
  }

  if(data_zsize == 3){
    texdef2d((long)(mapdbindex + 1), (long)4, (long)data_xsize,
             (long)data_ysize, dbuf, 0, texprops);
  }
  else{
    texdef2d((long)(mapdbindex + 1), (long)1, (long)data_xsize,
             (long)data_ysize, dbuf, 0, texprops);
  }

  if(mapdbindex == 0)
    tevdef(1, 0, tevprops);
```

```
  strcpy(mapdb[mapdbindex].mapname, s);
  mapdb[mapdbindex].mapindex = mapdbindex+1;

  mapdbindex++;

  printf("MAPLIB: Map file %s is read as index %d \n", s, mapdbindex);

  return mapdbindex;
}

static int
load_tex_image(s)
  char *s;
{
  FILE *fdir, *fopen();
  int ix, iy;
  char rd, gd, bd;
  unsigned long *lpl;


  fprintf(stderr, "MAPLIB: start read tex file %s \n", s);

  if((fdir = fopen(s, "r")) == NULL){
     fprintf(stderr,"MAPLIB: error cannot open tex file %s \n", s);
     return 1;
   }

  if(fread(&texheader, 1, sizeof(texheader), fdir) != sizeof(texheader)) {
     fprintf(stderr,"MAPLIB: error on read of tex file header\n");
     fclose(fdir);
     return 1;
   }

  if(texheader.tex_dim != 2){
     fprintf(stderr,"MAPLIB: error .tex file is not 2-D image\n");
     return 1;
   }

  if(texheader.tex_chan != 3){
    fprintf(stderr, "MAPLIB: error .tex file must be in color \n");
    return 1;
  }

  data_xsize = 1 << texheader.tex_size;
  data_ysize = data_xsize;
  data_zsize = texheader.tex_chan;

  for(ix=0; ix < 7+texheader.tex_chan; ix++){
    fgets(cbuf, 128, fdir);
/*      printf("%s", cbuf);  */
  }

  fseek(fdir, (long)(data_xsize * data_xsize - 1), 1);

  lpl = dbuf;

  for(iy = data_ysize - 1; iy >=0 ; iy--){

    for(ix = 0; ix < data_xsize; ix++){
      fread(&bd, sizeof(char), 1, fdir);
      fread(&gd, sizeof(char), 1, fdir);
      fread(&rd, sizeof(char), 1, fdir);
      *lpl++ = (long)(rd)|((long)(gd) << 8)|((long)(bd) <<16);
    }
  }
```



```
  fclose(fdir);
}

static int
load_wave_image(s)
  char *s;
{
  FILE *fdir;
  int ix, iy;
  RGBvalue *ibr, *ibg, *ibb;
  unsigned long offset, *lpl;
  short len;

    if((fdir = fopen(s, "r")) == NULL){
      printf("MAPLIB: error can't open rla file %s\n", s);
      return 1;
    }

    if(fread(&header,  1, sizeof(header),  fdir) != sizeof(header)) {
        fprintf(stderr,"MAPLIB: error in reading rla file headert\n");
        fclose(fdir);
        return 1;
    }
    if(header.num_chan != 3) {
        fprintf(stderr,"MAPLIB: error rla file channels must be 3\n");
        fclose(fdir);
        return 1;
    }
    if(header.storage_type!=0) {
        fprintf(stderr,"MAPLIB: error rla storage type must be 0\n");
        fclose(fdir);
        return 1;
    }

    data_xsize = header.active_window.right-header.active_window.left+1;
    data_ysize = header.active_window.top-header.active_window.bottom+1;
    data_zsize = 3;

    lpl = dbuf;

    for(iy=0; iy<data_ysize; iy++) {

        ibr = crbuf;
        ibg = cgbuf;
        ibb = cbbuf;

        if(fseek(fdir,740+4*iy,0))
            seekerror();
        fread(&offset,sizeof(long),1,fdir);
        if(fseek(fdir,offset,0))
            seekerror();

        fread(&len,sizeof(short),1,fdir);
        fread(cbuf,len,1,fdir);
        decode(cbuf,ibr,len);

        fread(&len,sizeof(short),1,fdir);
        fread(cbuf,len,1,fdir);
        decode(cbuf,ibg,len);

        fread(&len,sizeof(short),1,fdir);
        fread(cbuf,len,1,fdir);
        decode(cbuf,ibb,len);

        for(ix=0; ix<data_xsize; ix++){
          *lpl++ = (long)(*ibr++)
```

```
                |((long)(*ibg++) << 8)
                |((long)(*ibb++) <<16);
        }
    }
    fclose(fdir);
}

static int
load_iris_image(s)
  char *s;
{
  IMAGE *imagefd;
  int pos, ix, iy;
  short int *rp1, *gp1, *bp1;
  unsigned long *lp1;
  char *cp1;

  if((imagefd = iopen(s, "r")) != NULL){ /* not IRIS image file */
    data_xsize = imagefd->xsize;
    data_ysize = imagefd->ysize;
    data_zsize = imagefd->zsize;
    if(data_zsize >= 3){
      lp1 = dbuf;
      for(iy=0; iy<data_ysize; iy++) {
        getrow(imagefd, rbuf, iy, 0);
        getrow(imagefd, gbuf, iy, 1);
        getrow(imagefd, bbuf, iy, 2);
        rp1 = rbuf;
        gp1 = gbuf;
        bp1 = bbuf;
        for(ix=0; ix<data_xsize; ix++){
          *lp1++ = (long)(*rp1++)
                |((long)(*gp1++) << 8)
                |((long)(*bp1++) <<16);
        }
      }
    }
    else {
      cp1 = (char *)dbuf;
      for(iy=0; iy<data_ysize; iy++) {
        getrow(imagefd, rbuf, iy, 0);
        rp1 = rbuf;
        for(ix=0; ix<data_xsize; ix++){
          *cp1++ = *rp1++;
        }
      }
    }
    iclose(imagefd);
    return 0;
  }
  else{
    fprintf(stderr, "MAPLIB: error image file is not IRIS format\n");
    return 1;
  }
}

static void
decode(c_in,c_out,ct)
register signed char *c_in;
register unsigned char *c_out;
register int ct;
{
    register int count;
    register short val;

    while(ct>0) {
```



```
        if(*c_in < 0) {
            count = - *c_in++;
            ct -= count+1;
            while(count--)
                *c_out++ = (*c_in++)&0xff;
        } else {
            count = *c_in++ + 1;
            ct -= 2;
            val = (*c_in++)&0xff;
            while(count--)
                *c_out++ = val;
        }
    }
}

static void
seekerror()
{
    printf("imagedisp: seek error in reading rla filen");
    exit(1);
}

int search_map_index(s)
  char *s;
{
  int i;

  for(i = mapdbindex - 1; i >=0; i--){
    if(strcmp(s, mapdb[i].mapname) == 0) return mapdb[i].mapindex;
  }
  return 0;
}


void
mapSize(int *x, int *y)
{
  *x = data_xsize;      *y = data_ysize;
}
```

```
#include "cyber/cyber_glove.h"

#define TIME_GET

char    *xdr_cyber(), *xdr_cyber_angle();

int
ReadCyber(char *hname, int *dgd)
{
    static struct CYBERSTR      cy_str;
    int         stat;

    initrpcudp(hname, CG_PROG_NUM, CG_PROG_VER);

    if ((stat = callrpc(hname, CG_PROG_NUM, CG_PROG_VER, CG_PROG_PR2,
                        xdr_void, 0, xdr_cyber_angle, &cy_str)) != 0) {
        fprintf(stderr, "Not Responce %s Machine\n", hname);
    }
/*
    fprintf(stdout,
            "%3d %3d %3d %3d %3d %3d %3d %3d %3d %3d %3d %3d %3d %3d\n",
            cy_str.joint_angle[0][0], cy_str.joint_angle[0][1],
            cy_str.joint_angle[1][0], cy_str.joint_angle[1][1],
            cy_str.joint_angle[2][0], cy_str.joint_angle[2][1],
            cy_str.joint_angle[3][0], cy_str.joint_angle[3][1],
            cy_str.joint_angle[4][0], cy_str.joint_angle[4][1],
            cy_str.joint_space[0], cy_str.joint_space[1],
            cy_str.joint_space[2], cy_str.joint_space[3]);
    fflush(stdout);
*/

    dgd[0] = cy_str.joint_angle[0][0];
    dgd[1] = cy_str.joint_angle[0][1];
    dgd[2] = cy_str.joint_angle[1][0];
    dgd[3] = cy_str.joint_angle[1][1];
    dgd[4] = cy_str.joint_angle[2][0];
    dgd[5] = cy_str.joint_angle[2][1];
    dgd[6] = cy_str.joint_angle[3][0];
    dgd[7] = cy_str.joint_angle[3][1];
    dgd[8] = cy_str.joint_angle[4][0];
    dgd[9] = cy_str.joint_angle[4][1];
    dgd[10] = cy_str.joint_space[0];
    dgd[11] = cy_str.joint_space[1];
    dgd[12] = cy_str.joint_space[2];
    dgd[13] = cy_str.joint_space[3];

    exitrpcudp();
}
```

```
#include "fastrak/fastrak.h"

char    *xdr_tracker();

int
ReadFastrak(char *hname, float trd[][6])
{
    static  PRCSTRUCT  rpcval[ARRAYSIZE];
    int     stat, cnt, m;
    double  x, y, z, a, e, r;

    initrpcudp(hname, TR_PROG_NUM, TR_PROG_VER);

    if ((stat = callrpcudp(TR_PROG_PR, xdr_void, 0,
                           xdr_tracker, &rpcval[0])) != 0) {
        fprintf(stderr, "Not Responce %s Machine\n", hname);
    }
    for (m = 0; m < ARRAYSIZE && rpcval[m].sens_num != 0; m++) {
/*
        fprintf(stdout, "%d - %7.2f, %7.2f, %7.2f, %7.2f, %7.2f, %7.2f\n",
              rpcval[m].sens_num, rpcval[m].x, rpcval[m].y, rpcval[m].z,
              rpcval[m].a, rpcval[m].e, rpcval[m].r);
*/
#ifndef HIROSE
        trd[m][0] = -rpcval[m].x;
        trd[m][1] = rpcval[m].y;
        trd[m][2] = -rpcval[m].z;
        trd[m][3] = -rpcval[m].a;
        trd[m][4] = -rpcval[m].e;
        trd[m][5] = rpcval[m].r;
#else
        trd[m][0] = rpcval[m].x;
        trd[m][1] = rpcval[m].y;
        trd[m][2] = rpcval[m].z;
        trd[m][3] = rpcval[m].a;
        trd[m][4] = rpcval[m].e;
        trd[m][5] = rpcval[m].r;
#endif
    }
/*
    fflush(stdout);
*/
    exitrpcudp();
}

static struct sockaddr_in  server_addr_udp;
static struct timeval  calludp_timeout;
static CLIENT *client;

initrpcudp(host, pgnum, vernum)
     char  *host;
     int   pgnum, vernum;
{
  struct hostent  *hp;
  int  socket = RPC_ANYSOCK;

  if ((hp = gethostbyname(host)) == NULL) {
    fprintf(stderr, "Error : Can't get %s hostaddress\n", host);
    exit(-1);
  }
  bcopy(hp->h_addr, (caddr_t)&server_addr_udp.sin_addr, hp->h_length);
  server_addr_udp.sin_family = AF_INET;
  server_addr_udp.sin_port = 0;

  calludp_timeout.tv_sec = 3;
  calludp_timeout.tv_usec = 0;
```



```
  if ((client = clntudp_create(&server_addr_udp, pgnum, vernum,
                               calludp_timeout, &socket)) == NULL) {
    perror("clntudp_create");
    return(-1);
  }

  return(1);
}

callrpcudp(pcnum, inproc, in, outproc, out)
     int   pcnum;
     char  *in, *out;
     xdrproc_t  inproc, outproc;
{
  enum clnt_stat clnt_stat;

  clnt_freeres(client, outproc, out);
  clnt_stat = clnt_call(client, pcnum,
                        inproc, in, outproc, out, calludp_timeout);

  return((int)clnt_stat);
}

exitrpcudp()
{
  clnt_destroy(client);

  return(1);
}
```

```
/*
 * ファイル名 : scaling.c
 *
 * 機能       : 3次元モデルの拡大、縮小、回転関数
 *
 * 履歴       : 1992年10月28日 ; 作成 ; 広瀬  正後
 */
#include        "tool.h"


/*
 * 関数名 : Rotate3()
 *
 * 引数   : なし
 *
 * 戻り値 : なし
 *
 * 機能   : ・各部品を回転させて表示
 *          ・G_buhin にはメニューで選択された部品番号が入っている
 *          ・マウス左ボタンをクリックしたままマウスを移動させることにより回転角 *
 *
 * 履歴   : 1992年10月28日 ; 作成 ; 広瀬  正後
 */
void
ChangeTracker(int hcnt, void *vhm[], int trn, int jiku)
{
    short       xold, yold, xnew, ynew;
    int         i;

    /* 現在のマウス座標を取り込む */
    xold = getvaluator(MOUSEX);
    yold = getvaluator(MOUSEY);

    while(!qtest()) {
        /* 移動後のマウス座標を取り込む */
        xnew = getvaluator(MOUSEX);
        ynew = getvaluator(MOUSEY);

        if (jiku == 0) {
            G_trd[trn][0] += (xnew - xold)/5.0;
            G_trd[trn][1] += (ynew - yold)/5.0;
        } else if (jiku == 1) {
            G_trd[trn][2] += (xnew - xold)/5.0;
            G_trd[trn][1] += (ynew - yold)/5.0;
        } else if (jiku == 2) {
            switch (trn) {
            case ATAMA_TRAK_NO:
                G_trd[trn][5] += (xnew - xold);
                G_trd[trn][3] -= (ynew - yold);
                break;
            case MUNE_TRAK_NO:
                G_trd[trn][4] += (xnew - xold);
                G_trd[trn][5] -= (ynew - yold);
                break;
            case MIGITE_TRAK_NO:
                G_trd[trn][3] += (xnew - xold);
                G_trd[trn][4] -= (ynew - yold);
#ifdef CYBERGLOVE
                G_dgd[14] += (xnew - xold);
#endif
                break;
            case HIDARITE_TRAK_NO:
                G_trd[trn][3] += (xnew - xold);
                G_trd[trn][4] -= (ynew - yold);
#ifdef CYBERGLOVE
                G_dgd[30] += (xnew - xold);
```



```
#endif
                break;
            default:
                break;
            }
        } else if (jiku == 3) {
            switch (trn) {
            case ATAMA_TRAK_NO:
                G_trd[trn][5] += (xnew - xold);
                G_trd[trn][4] -= (ynew - yold);
                break;
            case MUNE_TRAK_NO:
                G_trd[trn][4] += (xnew - xold);
                G_trd[trn][3] -= (ynew - yold);
                break;
            case MIGITE_TRAK_NO:
                G_trd[trn][3] += (xnew - xold);
                G_trd[trn][5] -= (ynew - yold);
#ifdef CYBERGLOVE
度を決定する        G_dgd[15] += (ynew - yold);
#endif
                break;
            case HIDARITE_TRAK_NO:
                G_trd[trn][3] += (xnew - xold);
                G_trd[trn][5] -= (ynew - yold);
#ifdef CYBERGLOVE
                G_dgd[31] += (ynew - yold);
#endif
                break;
            default:
                break;
            }
        } else if (jiku == 4) {
            for (i = 0; i < 10; i++) {
                G_dgd[i] += (xnew - xold);
            }
            for (i = 10; i < 14; i++) {
                G_dgd[i] += (ynew - yold);
            }
        } else if (jiku == 5) {
#ifdef CYBERGLOVE
            for (i = 16; i < 26; i++) {
                G_dgd[i] += (xnew - xold);
            }
            for (i = 26; i < 30; i++) {
                G_dgd[i] += (ynew - yold);
            }
#else
            for (i = 14; i < 24; i++) {
                G_dgd[i] += (xnew - xold);
            }
            for (i = 24; i < 28; i++) {
                G_dgd[i] += (ynew - yold);
            }
#endif
        }

        xold = xnew;
        yold = ynew;

        DrawHuman(hcnt, vhm);
    }
}

/*
 * 関数名 : Scaling()
```

```
 *
 * 引数    : なし
 *
 * 戻り値  : なし
 *
 * 機能    : ・3次元モデルの拡大・縮小・移動表示
 *           ・G_mdlbtn にはメニューで選択された（拡大・縮小）か（移動）かどちらか *
 *           ・マウス中ボタンをクリックしたままマウスを移動させることによりスケー *
 *           ・scale(), translate()により、拡大・縮小・移動を行う
 *
 * 履歴    : 1992年10月28日 ; 作成 ; 広瀬　正俊
 */
void
Scaling(int hcnt, void *vhm[], int mdlbtn)
{
    short       xold, yold, xnew, ynew;
    float       sf, movex, movey;

    /* 現在のマウス座標を取り込む */
    xold = getvaluator(MOUSEX);
    yold = getvaluator(MOUSEY);
    sf = 1.0;

    while (!qtest()) {
        /* 移動後のマウス座標を取り込む */
        xnew = getvaluator(MOUSEX);
        ynew = getvaluator(MOUSEY);

        if (mdlbtn == SCALING) {
            sf += (ynew - yold)/500.0;
            scale(sf, sf, sf);
        } else if (mdlbtn == TRANSLATING) {
            movex = (xnew - xold)/5.0;
            movey = (ynew - yold)/5.0;
            translate(movex, movey, 0.0);
        }

        xold = xnew;
        yold = ynew;

        DrawHuman(hcnt, vhm);
    }
    qreset();
}

void
Rotate3(int hcnt, void *vhm[])
{
    short       xold, yold, xnew, ynew;

    /* 現在のマウス座標を取り込む */
    xold = getvaluator(MOUSEX);
    yold = getvaluator(MOUSEY);

    while(!qtest()) {
        /* 移動後のマウス座標を取り込む */
        xnew = getvaluator(MOUSEX);
        ynew = getvaluator(MOUSEY);

        G_rota += (xnew - xold);
        G_elea -= (ynew - yold);

        xold = xnew;
        yold = ynew;

        DrawHuman(hcnt, vhm);
```



```
    }
}
```

が入っている
ル値、移動量を決定する

```
#include "cyber_glove.h"

xdr_cyber(xdrsp, cdat)
     XDR *xdrsp;
     int cdat[];
{
  return(xdr_vector(xdrsp, cdat, 24, sizeof(int), xdr_int));
}

xdr_cyber_angle(xdrsp, cs)
     XDR *xdrsp;
     struct CYBERSTR  *cs;
{
  return(xdr_vector(xdrsp, cs->joint_angle, 10, sizeof(int), xdr_int) &&
         xdr_vector(xdrsp, cs->joint_space, 4, sizeof(int), xdr_int) &&
         xdr_vector(xdrsp, cs->wrist_angle, 2, sizeof(int), xdr_int));
}

xdr_calib_data(xdrsp, dat)
     XDR *xdrsp;
     int dat[];
{
  return(xdr_vector(xdrsp, dat, 34, sizeof(int), xdr_int));
}
```

```
#include "fastrak.h"


xdr_tracker_sub(xdrsp, rvs)
     XDR *xdrsp;
     PRCSTRUCT *rvs;
{
  return(xdr_int(xdrsp, &rvs->sens_num) &&
         xdr_double(xdrsp, &rvs->x) && xdr_double(xdrsp, &rvs->y) &&
         xdr_double(xdrsp, &rvs->z) && xdr_double(xdrsp, &rvs->a) &&
         xdr_double(xdrsp, &rvs->e) && xdr_double(xdrsp, &rvs->r));
}

xdr_tracker(xdrsp, rv)
     XDR *xdrsp;
     PRCSTRUCT rv[];
{
  return(xdr_vector(xdrsp, rv, ARRAYSIZE, sizeof(PRCSTRUCT), xdr_tracker_sub));
}
```